Kolibri developer documentation

Release 0.16.1.dev0+git.20240424234141

unknown

Apr 24, 2024

CONTENTS

These docs are for software developers wishing to contribute to Kolibri. If you are looking for help installing, configuring and using Kolibri, please refer to the User Guide.

CHAPTER

ONE

WHAT IS KOLIBRI?

Kolibri is the offline learning platform from Learning Equality. It is available for download from our website. The code is hosted on Github and MIT-licensed.

You can ask questions, make suggestions, and report issues in the community forums. If you have found a bug and are comfortable using Github and Markdown, you can create a Github issue following the instructions in the issue template.

CHAPTER

TWO

TABLE OF CONTENTS

2.1 Contributing

2.1.1 Ways to contribute

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

Talk to us

- Get product support in our Community Forums.
- Get development contributions support in Kolibri GitHub Discussions.
- Email us at info@learningequality.org

Translate

Help us translate the application on Crowdin.

Give feedback

You can ask questions, make suggestions, and report issues in the community forums.

If you are proposing a new feature or giving feedback on an existing feature:

- Explain in detail what you're trying to do and why the existing system isn't working for you
- Keep the scope as narrow as possible, to make it easier to understand the specific problem you're trying to address

If you have found a bug and are comfortable using Github and Markdown, you can create a Github issue. Please search the existing issues first to see if it's already been reported.

Please make sure to use the template. Replace the HTML comments (the `<!-- ... -->`) with the information requested.

Write code

See How can I contribute?

Write documentation

If you'd like to help improve Kolibri's User Documentation, see the Kolibri docs Github repo.

You can also help improve our developer documentation that you're reading now! These are in the main Kolibri repo.

Both our developer docs and the user docs are formatted using reStructuredText.

2.1.2 Code of Conduct

Code of Conduct

1. Purpose

A primary goal of Kolibri and KA Lite is to be inclusive to the largest number of contributors, with the most varied and diverse backgrounds possible. As such, we are committed to providing a friendly, safe and welcoming environment for all, regardless of gender, sexual orientation, ability, ethnicity, socioeconomic status, and religion (or lack thereof).

This code of conduct outlines our expectations for all those who participate in our community, as well as the consequences for unacceptable behavior.

We invite all those who participate in Kolibri or KA Lite to help us create safe and positive experiences for everyone.

2. Open Source Citizenship

A supplemental goal of this Code of Conduct is to increase open source citizenship by encouraging participants to recognize and strengthen the relationships between our actions and their effects on our community.

Communities mirror the societies in which they exist and positive action is essential to counteract the many forms of inequality and abuses of power that exist in society.

If you see someone who is making an extra effort to ensure our community is welcoming, friendly, and encourages all participants to contribute to the fullest extent, we also want to know!

3. Expected Behavior

The following behaviors are expected and requested of all community members:

- Participate in an authentic and active way. In doing so, you contribute to the health and longevity of this community.
- Exercise consideration and respect in your speech and actions.
- Attempt collaboration before conflict.
- Refrain from demeaning, discriminatory, or harassing behavior and speech.
- Be mindful of your surroundings and of your fellow participants. Alert community leaders if you notice a dangerous situation, someone in distress, or violations of this Code of Conduct, even if they seem inconsequential.
- Remember that community event venues may be shared with members of the public; please be respectful to all patrons of these locations.

4. Unacceptable Behavior

The following behaviors are considered harassment and are unacceptable within our community:

- Violence, threats of violence or violent language directed against another person.
- Sexist, racist, homophobic, transphobic, ableist or otherwise discriminatory jokes and language.
- Posting or displaying sexually explicit or violent material.
- Posting or threatening to post other people's personally identifying information ("doxing").
- Personal insults, particularly those related to gender, sexual orientation, race, religion, or disability.
- Inappropriate photography or recording.
- Inappropriate physical contact. You should have someone's consent before touching them.
- Unwelcome sexual attention. This includes, sexualized comments or jokes; inappropriate touching, groping, and unwelcomed sexual advances.
- Deliberate intimidation, stalking or following (online or in person).
- Advocating for, or encouraging, any of the above behavior.
- Sustained disruption of community events, including talks and presentations.

5. Consequences of Unacceptable Behavior

Unacceptable behavior from any community member, including sponsors and those with decision-making authority, will not be tolerated.

Anyone asked to stop unacceptable behavior is expected to comply immediately.

If a community member engages in unacceptable behavior, the community organizers may take any action they deem appropriate, up to and including a temporary ban or permanent expulsion from the community without warning (and without refund in the case of a paid event).

6. Reporting Guidelines

If you are subject to or witness unacceptable behavior, or have any other concerns, please notify a community organizer as soon as possible. codeofconduct@learningequality.org.

Reporting Guidelines

Additionally, community organizers are available to help community members engage with local law enforcement or to otherwise help those experiencing unacceptable behavior feel safe. In the context of in-person events, organizers will also provide escorts as desired by the person experiencing distress.

7. Addressing Grievances

If you feel you have been falsely or unfairly accused of violating this Code of Conduct, you should notify Learning Equality with a concise description of your grievance. Your grievance will be handled in accordance with our existing governing policies.

Enforcement Manual

8. Scope

We expect all community participants (contributors, paid or otherwise; sponsors; and other guests) to abide by this Code of Conduct in all community venues–online and in-person–as well as in all one-on-one communications pertaining to community business.

This code of conduct and its related procedures also applies to unacceptable behavior occurring outside the scope of community activities when such behavior has the potential to adversely affect the safety and well-being of community members.

9. Contact info

The Code of Conduct team consists of:

- Laura Danforth (laura@learningequality.org)
- Radina Matic (radina@learningequality.org)
- Richard Tibbles (richard@learningequality.org)

Please write: codeofconduct@learningequality.org

10. License and attribution

This Code of Conduct is distributed under a Creative Commons Attribution-ShareAlike license.

Portions of text derived from the Django Code of Conduct and the Geek Feminism Anti-Harassment Policy.

Retrieved on November 22, 2016 from http://citizencodeofconduct.org/

Reporting Guidelines

If you believe someone is violating the code of conduct we ask that you report it to the Learning Equality by emailing codeofconduct@learningequality.org. All reports will be kept confidential. In some cases we may determine that a public statement will need to be made. If that's the case, the identities of all victims and reporters will remain confidential unless those individuals instruct us otherwise.

If you believe anyone is in physical danger, please notify appropriate law enforcement first. If you are unsure what law enforcement agency is appropriate, please include this in your report and we will attempt to notify them.

If you are unsure whether the incident is a violation, or whether the space where it happened is covered by this Code of Conduct, we encourage you to still report it. We would much rather have a few extra reports where we decide to take no action, rather than miss a report of an actual violation. We do not look negatively on you if we find the incident is not a violation. And knowing about incidents that are not violations, or happen outside our spaces, can also help us to improve the Code of Conduct or the processes surrounding it.

In your report please include:

1. Your contact info (so we can get in touch with you if we need to follow up)
2. Names (real, nicknames, or pseudonyms) of any individuals involved. If there were other witnesses besides you, please try to include them as well.
3. When and where the incident occurred. Please be as specific as possible.
4. Your account of what occurred. If there is a publicly available record (e.g. a mailing list archive or a public Slack logger) please include a link.
5. Any extra context you believe existed for the incident.
6. If you believe this incident is ongoing.
7. Any other information you believe we should have.

What happens after you file a report?

You will receive an email from the Code of Conduct committee acknowledging receipt within 48 hours (we aim to be quicker than that).

The committee will immediately meet to review the incident and determine:

1. What happened.
2. Whether this event constitutes a code of conduct violation.
3. Who the bad actor was.
4. Whether this is an ongoing situation, or if there is a threat to anyone's physical safety.

If this is determined to be an ongoing incident or a threat to physical safety, the committee's immediate priority will be to protect everyone involved. This means we may delay an "official" response until we believe that the situation has ended and that everyone is physically safe.

Once the committee has a complete account of the events they will make a decision as to how to response. Responses may include:

- Nothing (if we determine no violation occurred).
- A private reprimand from the committee to the individual(s) involved.
- A public reprimand.
- An imposed vacation (i.e. asking someone to "take a week off" from a mailing list or Slack).
- A permanent or temporary ban from some or all communication spaces (mailing lists, Slack, etc.)
- A request for a public or private apology.

We'll respond within one week to the person who filed the report with either a resolution or an explanation of why the situation is not yet resolved.

Once we've determined our final action, we'll contact the original reporter to let them know what action (if any) we'll be taking. We'll take into account feedback from the reporter on the appropriateness of our response, but we don't guarantee we'll act on it.

Enforcement Manual

This is the enforcement manual followed by Learning Equality's Code of Conduct Committee. It's used when we respond to an issue to make sure we're consistent and fair. It should be considered an internal document, but we're publishing it publicly in the interests of transparency.

The Code of Conduct Committee

All responses to reports of conduct violations will be managed by a Code of Conduct Committee ("the committee").

Learning Equality's (LE's) core team ("the core") will establish this committee, comprised of at least three members.

How the committee will respond to reports

When a report is sent to the committee, a member will reply with a receipt to confirm that a process of reading your report has started.

See the reporting guidelines for details of what reports should contain. If a report doesn't contain enough information, the committee will obtain all relevant data before acting. The committee is empowered to act on the LE's behalf in contacting any individuals involved to get a more complete account of events.

The committee will then review the incident and determine, to the best of their ability:

- what happened
- whether this event constitutes a code of conduct violation
- who, if anyone, was the bad actor
- whether this is an ongoing situation, and there is a threat to anyone's physical safety

This information will be collected in writing, and whenever possible the committee's deliberations will be recorded and retained (i.e. Slack transcripts, email discussions, recorded voice conversations, etc).

The committee should aim to have a resolution agreed upon within one week. In the event that a resolution can't be determined in that time, the committee will respond to the reporter(s) with an update and projected timeline for resolution.

Acting Unilaterally

If the act is ongoing or involves a threat to anyone's safety (e.g. threats of violence), any committee member may act immediately (before reaching consensus) to end the situation. In ongoing situations, any member may at their discretion employ any of the tools available to the committee, including bans and blocks.

If the incident involves physical danger, any member of the committee may – and should – act unilaterally to protect safety. This can include contacting law enforcement (or other local personnel) and speaking on behalf of Learning Equality.

In situations where an individual committee member acts unilaterally, they must report their actions to the committee for review within 24 hours.

Resolutions

The committee must agree on a resolution by consensus. If the committee cannot reach consensus and deadlocks for over a week, the committee will turn the matter over to the board for resolution.

Possible responses may include:

- Taking no further action (if we determine no violation occurred).
- A private reprimand from the committee to the individual(s) involved. In this case, the committee will deliver that reprimand to the individual(s) over email, cc'ing the committee.
- A public reprimand. In this case, the committee will deliver that reprimand in the same venue that the violation occurred (i.e. in Slack for an Slack violation; email for an email violation, etc.). The committee may choose to publish this message elsewhere for posterity.
- An imposed vacation (i.e. asking someone to "take a week off" from a mailing list or Slack). The committee will communicate this "vacation" to the individual(s). They'll be asked to take this vacation voluntarily, but if they don't agree then a temporary ban may be imposed to enforce this vacation.
- A permanent or temporary ban from some or all Learning Equality spaces (mailing lists, Slack, etc.). The committee will maintain records of all such bans so that they may be reviewed in the future, extended to new Learning Equality fora, or otherwise maintained.
- A request for a public or private apology. The committee may, if it chooses, attach "strings" to this request: for example, the committee may ask a violator to apologize in order to retain his or her membership on a mailing list.

Once a resolution is agreed upon, but before it is enacted, the committee will contact the original reporter and any other affected parties and explain the proposed resolution. The committee will ask if this resolution is acceptable, and must note feedback for the record. However, the committee is not required to act on this feedback.

Finally, the committee will make a report for the core team.

The committee will never publicly discuss the issue; all public statements will be made by the core team.

Conflicts of Interest

In the event of any conflict of interest a committee member must immediately notify the other members, and recuse themselves if necessary.

Attribution

Reporting Guidelines and Enforcement Manual are both distributed under a Creative Commons Attribution-ShareAlike license.

Reporting Guidelines and Enforcement Manual are both derived from the Django' Reporting Guidelines and Django' Enforcement Manual

Changes made to the original doc: Instead of involving a board as DSF has, the core team at Learning Equality is considered. Instead of IRC, we refer to Slack. The Code of Conduct Committee does not have a single chair but acts as a group to make conflicts of interest easier, and to avoid problems in case of absence of the chair person. Instead of interchanging "working group" and "committee" notation, we replaced all occurrences of "working group" and "group" with "committee".

2.1.3 Contributors

Kolibri is copyright Learning Equality and other contributors, and is released under the MIT License.

If you have contributed to Kolibri, feel free to add your name and Github account to this list:

Name	Github user
Eli Dai	66eli77
Adam Stasiw	AdamStasiw
Akshay Mahajan	akshaymahajans
Alan Chen	alanchenz
Alexandros Metaxas	alexMet
Apurva Modi	apurva-modi
Eduard James Aban	arceduardvincent
Arky	arky
Aron Fyodor Asor	aronasorman
Ashmeet Lamba	ashmeet13
Kapya	Aypak
Benjamin Bach	benjaoming
Blaine Jester	bjester
Boni Đukić	bonidjukic
Brian Kwon	br-kwon
Brandon Nguyen	bransgithub
John	BruvaJ
Chao-Wen Tan	chaowentan
Christian Memije	christianmemije
Connor Robertson	conconrob
Cyril Pauya	cpauya
Chris Castle	crcastle
David Garg	davidgarg20
David Hu	divad12
Derek Lobo	dlobo
David Cañas	DXCanas
Dylan McCall	dylanmccall
Mingqi Zhu	EmanekaT
Gerardo Soto	GCodeON
Geoff Rich	geoffrey1218
Hans Gamboa	HansGam
Devon Rueckner	indirectlylit
•	inflrscns
Ivan Savov	ivanistheone
Jamie Alexandre	jamalex
Jason Tame	JasonTame
Jordan Yoshihara	jayoshih
Jessica Aceret	jessicaaceret
Jonathan Boiser	jonboiser
José L. Redrejo Rodríguez	jredrejo
Jessica Aceret	jtamiace
Julián Duque	julianduque
Karla Avila	k2avila
Maxime Brunet	maxbrunet

continues on next page

Table 1 – continued from previous page

Name	Github user
Mrinal Kumar	kmrinal19
Kevin Ollivier	kollivier
Paul Luna	luna215
Lingyi Wang	lyw07
Magali Boizot-Roche	magali-br
•	manuq
Marcella Maki	marcellamaki
Maureen Hernandez	MauHernandez
Michael Gallaspy	MCGallaspy
Leo Lin	mdctleo
Metodi Milev	metodimilevqa
Michael Gamlem III	mgamlem3
Micah Fitch	micahscopes
Michaela Robosova	MisRob
Eduard James Aban	mrpau-eduard
Eugene Oliveros	mrpau-eugene
Julius legaspi	mrpau-julius
Richard Amodia	mrpau-richard
Nick Cannariato	nickcannariato
Jacob Pierce	nucleogenesis
Paul Bussé	paulbusse
Petar Cenov	pcenov
Philip Withnall	pwithnall
Radina Matic	radinamatic
Rafael Aguayo	ralphiee22
Hyun Ahn	rationality6
Rachel Kim	rayykim
Richard Tibbles	rtibbles
Sairina Merino Tsui	sairina
Shanavas M	shanavas786
•	shivangtripathi
Udith Prabhu	udithprabhu
Whitney Zhu	whitzhu
Carol Willing	willingc
Yash Jipkate	YashJipkate
Yixuan Liu	yil039
Jaideep Sharma	camperjett
Allan Otodi	AllanOXDi
Liana Harris	LianaHarris360
Rishi Kejriwal	Kej-r03
Siddhanth Rathod	siddhanthrathod
Akila Induranga	akila-i
Sahajpreet Singh	photon0205
Nishant Shrivastva	shrinishant
Amelia Breault	thanksameeelian
Vikramaditya Singh	Ghat0tkach

continues on next page

Table 1 – continued from previous page

Name	Github user
Kris Katkus	katkuskris
Garvit Singhal	GarvitSinghal47
Adars T S	a6ar55
Shivang Rawat	ShivangRawat30
Alex Vélez	AlexVelezLl
Mazen Oweiss	moweiss
Eshaan Aggarwal	EshaanAgg
Nikhil Sharma	ThEditor

2.2 Getting started

First of all, thank you for your interest in contributing to Kolibri! The project was founded by volunteers dedicated to helping make educational materials more accessible to those in need, and every contribution makes a difference. The instructions below should get you up and running the code in no time!

2.2.1 Prerequisites

Most of the steps below require entering commands into your Terminal, so you should expect to become comfortable with this if you're not already.

If you encounter issues:

- Searching online is often effective: chances are high that someone else encountered similar issues in the past
- Please let us know if our docs can be improved, either by filing an issue or submitting a PR!

Note: Theoretically, Windows can be used to develop Kolibri, but we haven't done much testing with it. If you're running Windows, you are likely to encounter some issues with this guide, and we'd appreciate any help improving these docs for Windows developers!

Git and GitHub

1. Install and set up Git on your computer. Try this tutorial if you need more practice with Git!
2. Sign up and configure your GitHub account if you don't have one already.
3. Fork the main Kolibri repository. This will make it easier to submit pull requests. Read more details about forking from GitHub.
4. **Important**: Install and set up the Git LFS extension.

Tip: Register your SSH keys on GitHub to avoid having to repeatedly enter your password

Checking out the code

First, clone your Kolibri fork to your local computer. In the command below, replace `$USERNAME` with your own GitHub username:

```
git clone git@github.com:$USERNAME/kolibri.git
```

Next, initialize Git LFS:

```
cd kolibri  # Enter the Kolibri directory
git lfs install
```

Finally, add the Learning Equality repo as a remote called *upstream*. That way you can keep your local checkout updated with the most recent changes:

```
git remote add upstream git@github.com:learningequality/kolibri.git
git fetch --all  # Check if there are changes upstream
git checkout -t upstream/develop # Checkout the development branch
```

Python and Pip

To develop on Kolibri, you'll need:

- Python 3.6+ (Kolibri doesn't currently support Python 3.12.0 or higher)
- pip

Managing Python installations can be quite tricky. We *highly* recommend using pyenv or if you are more comfortable using a package manager, then package managers like Homebrew on Mac or `apt` on Debian for this.

To install pyenv see the detailed instructions here *Installing pyenv*.

Warning: Never modify your system's built-in version of Python

Python virtual environment

You should use a Python virtual environment to isolate the dependencies of your Python projects from each other and to avoid corrupting your system's Python installation.

There are many ways to set up Python virtual environments: You can use pyenv-virtualenv as shown in the instructions below; you can also use Virtualenv, Virtualenvwrapper Pipenv, Python 3 venv, Poetry etc.

Note: Most virtual environments will require special setup for non-Bash shells such as Fish and ZSH.

To setup and start using pyenv-virtualenv, follow the instructions here *Using pyenv-virtualenv*.

Once pyenv-virtualenv is installed, you can use the following commands to set up and use a virtual environment from within the Kolibri repo:

```
pyenv virtualenv 3.9.9 kolibri-py3.9  # can also make a python 2 environment
pyenv activate kolibri-py3.9  # activates the virtual environment
```

Now, any commands you run will target your virtual environment rather than the global Python installation. To deactivate the virtualenv, simply run:

```
pyenv deactivate
```

(Note that you'll want to leave it activated for the remainder of the setup process)

Warning: Never install project dependencies using `sudo pip install ...`

Environment variables

Environment variables can be set in many ways, including:

- adding them to a `~/.bash_profile` file (for Bash) or a similar file in your shell of choice
- using a `.env` file for this project, loaded with Pipenv
- setting them temporarily in the current Bash session using `EXPORT` or similar (not recommended except for testing)

There are two environment variables you should plan to set:

- `KOLIBRI_RUN_MODE` is **required**.

  This variable is sent to our pingback server (private repo), and you must set it to something besides an empty string. This allows us to filter development work out of our usage statistics. There are also some special testing behaviors that can be triggered for special strings, as described elsewhere in the developer docs and integration testing Gherkin scenarios.

  For example, you could add this line at the end of your `~/.bash_profile` file:

  ```
  export KOLIBRI_RUN_MODE="dev"
  ```

- `KOLIBRI_HOME` is optional.

  This variable determines where Kolibri will store its content and databases. It is useful to set if you want to have multiple versions of Kolibri running simultaneously.

Install Python dependencies

To install Kolibri project-specific dependencies make sure you're in the `kolibri` directory and your Python virtual environment is active. Then run:

```
# required
pip install -r requirements.txt --upgrade
pip install -r requirements/dev.txt --upgrade
pip install -e .

# optional
pip install -r requirements/test.txt --upgrade
pip install -r requirements/docs.txt --upgrade
```

Note that the `--upgrade` flags above can usually be omitted to speed up the process.

Install Node.js, Yarn and other dependencies

1. Install Node.js (version 18.x is required)
2. Install Yarn
3. Install non-python project-specific dependencies

For a more detailed guide to using nodeenv see *Using nodeenv*.

The Python project-specific dependencies installed above will install `nodeenv`, which is a useful tool for using specific versions of Node.js and other Node.js tools in Python environments. To setup Node.js and Yarn within the Kolibri project environment, ensure your Python virtual environment is active, then run:

```
# node.js, npm, and yarn
# If you are setting up the release-v0.15.x branch or earlier:
nodeenv -p --node=10.17.0
# If you are setting up the develop branch:
nodeenv -p --node=18.19.0
npm install -g yarn

# other required project dependencies
yarn install
```

Database setup

To initialize the database run the following command:

```
kolibri manage migrate
```

2.2.2 Running the server

Development server

To start up the development server and build the client-side dependencies, use the following command:

```
yarn run devserver
```

This will take some time to build the front-end assets, after which you should be able to access the server at `http://127.0.0.1:8000/`.

Alternatively, you can run the devserver with hot reload enabled using:

```
yarn run devserver-hot
```

Tip: Running the development server to compile all client-side dependencies can take up a lot of system resources. To limit the specific frontend bundles that are built and watched, you can pass keywords to either of the above commands to only watch those.

```
yarn run devserver-hot learn
```

Would build all assets that are not currently built, and run a devserver only watching the Learn plugin.

```
yarn run devserver core,learn
```

Would run the devserver not in hot mode, and rebuild the core Kolibri assets and the Learn plugin.

For a complete reference of the commands that can be run and what they do, inspect the `scripts` section of the root *./package.json* file.

Warning: Some functionality, such as right-to-left language support, is broken when hot-reload is enabled

Tip: If you get an error similar to "Node Sass could not find a binding for your current environment", try running `npm rebuild node-sass`

Production server

In production, content is served through Whitenoise. Frontend static assets are pre-built:

```
# first build the assets
yarn run build

# now, run the Django production server
kolibri start
```

Now you should be able to access the server at `http://127.0.0.1:8080/`.

Kolibri has support for being run as a `Type=notify` service under systemd. When doing so, it is recommended to run `kolibri start` with the `--skip-update` option, and to run `kolibri configure setup` separately beforehand to handle database migrations and other one-time setup steps. This avoids the `kolibri start` command timing out under systemd if migrations are happening.

Separate servers

If you are working mainly on backend code, you can build the front-end assets once and then just run the Python devserver. This may also help with multi-device testing over a LAN.

```
# first build the front-end assets
yarn run build

# now, run the Django devserver
yarn run python-devserver
```

You can also run the Django development server and webpack devserver independently in separate terminal windows. In the first terminal you can start the django development server:

```
yarn run python-devserver
```

and in the second terminal, start the webpack build process for frontend assets:

```
yarn run frontend-devserver
```

Running in App Mode

Some of Kolibri's functionality will differ when being run as a mobile app. In order to run the development server in that "app mode" context, you can use the following commands.

```
# run the Python "app mode" server and the frontend server together:
yarn run app-devserver

# you may also run the python "app mode" server by itself
# this will require you to run the frontend server in a separate terminal
yarn run app-python-devserver
```

This will run the script located at `integration_testing/scripts/run_kolibri_app_mode.py`. There you may change the port, register app capabilities (ie, `os_user`) and make adjustments to meet your needs.

When the app development server is started, you will see a message with a particular URL that you will need to use in order to initialize your browser session properly. Once your browser session has been initialized for use in the app mode, your browser session will remain in this mode until you clear your cookies, even if you've started your normal Kolibri development server.

```
[app-python-devserver] Kolibri running at: http://127.0.0.1:8000/app/api/initialize/
↪6b91ec2b697042c2b360235894ad2632
```

2.2.3 Editor configuration

We have a project-level *.editorconfig* file to help you configure your text editor or IDE to use our internal conventions.

Check your editor to see if it supports EditorConfig out-of-the-box, or if a plugin is available.

2.2.4 Vue development tools

Vue.js devtools (Legacy) is a browser plugin that is very helpful when working with Vue.js components and Vuex. Kolibri is using Vue 2, so be sure to find the "Legacy" plugin as the latest version of the extension is for Vue 3.

To ensure a more efficient workflow, install appropriate editor plugins for Vue.js, ESLint, and stylelint.

2.2.5 Sample resources and data

Once you have the server running, proceed to import some channels and resources. To quickly import all available and supported Kolibri resource types, use the token `nakav-mafak` for the Kolibri QA channel (~350MB).

Now you can create users, classes, lessons, etc manually. To auto-generate some sample user data you can also run:

```
kolibri manage generateuserdata
```

2.2.6 Linting and auto-formatting

Manual linting and formatting

Linting and code auto-formatting are done by Prettier and Black.

You can manually run the auto-formatters for the frontend using:

```
yarn run lint-frontend:format
```

Or to check the formatting without writing changes, run:

```
yarn run lint-frontend
```

The linting and formatting for the backend is handled using `pre-commit` below.

Pre-commit hooks

A full set of linting and auto-formatting can also be applied by pre-commit hooks. The pre-commit hooks are identical to the automated build check by Travis CI in Pull Requests.

pre-commit is used to apply a full set of checks and formatting automatically each time that `git commit` runs. If there are errors, the Git commit is aborted and you are asked to fix the error and run `git commit` again.

Pre-commit is already installed as a development dependency, but you also need to enable it:

```
pre-commit install
```

To run all pre-commit checks in the same way that they will be run on our Github CI servers, run:

```
pre-commit run --all-files
```

Tip: As a convenience, many developers install linting and formatting plugins in their code editor (IDE). Installing ESLint, Prettier, Black, and Flake8 plugins in your editor will catch most (but not all) code-quality checks.

Tip: Pre-commit can have issues running from alternative Git clients like GitUp. If you encounter problems while committing changes, run `pre-commit uninstall` to disable pre-commit.

Warning: If you do not use any linting tools, your code is likely fail our server-side checks and you will need to update the PR in order to get it merged.

2.2.7 Design system

We have a large number of reusable patterns, conventions, and components built into the application. Review the Kolibri Design System to get a sense for the tools at your disposal, and to ensure that new changes stay consistent with established UI patterns.

2.2.8 Updating documentation

First, install some additional dependencies related to building documentation output:

```
pip install -r requirements/docs.txt
pip install -r requirements/build.txt
```

To make changes to documentation, edit the `rst` files in the `kolibri/docs` directory and then run:

```
make docs
```

You can also run the auto-build for faster editing from the `docs` directory:

```
cd docs
sphinx-autobuild --port 8888 . _build
```

Now you should be able to preview the docs at `http://127.0.0.1:8888/`.

2.2.9 Automated testing

Kolibri comes with a Javascript test suite based on Jest. To run all front-end tests:

```
yarn run test
```

Kolibri comes with a Python test suite based on pytest. To run all back-end tests:

```
pytest
```

To run specific tests only, you can add the filepath of the file. To further filter either by TestClass name or test method name, you can add *-k* followed by a string to filter classes or methods by. For example, to only run a test named `test_admin_can_delete_membership` in kolibri/auth/test/test_permissions.py:

```
pytest kolibri/auth/test/test_permissions -k test_admin_can_delete_membership
```

To only run the whole class named `MembershipPermissionsTestCase` in kolibri/auth/test/test_permissions.py:

```
pytest kolibri/auth/test/test_permissions -k MembershipPermissionsTestCase
```

For more advanced usage, logical operators can also be used in wrapped strings, for example, the following will run only one test, named `test_admin_can_delete_membership` in the `MembershipPermissionsTestCase` class in kolibri/auth/test/test_permissions.py:

```
pytest kolibri/auth/test/test_permissions -k "MembershipPermissionsTestCase and test_
↪admin_can_delete_membership"
```

You can also use `tox` to setup a clean and disposable environment:

```
tox -e py3.4  # Runs tests with Python 3.4
```

To run Python tests for all environments, use simply `tox`. This simulates what our CI also does on GitHub PRs.

Note: `tox` reuses its environment when it is run again. If you add anything to the requirements, you will want to either delete the *.tox* directory, or run `tox` with the `-r` argument to recreate the environment

2.2.10 Manual testing

All changes should be thoroughly tested and vetted before being merged in. Our primary considerations are:

- Performance
- Accessibility
- Compatibility
- Localization
- Consistency

For more information, see the next section on *Manual testing & QA*.

2.2.11 Submitting a pull request

Here's a very simple scenario. Below, your remote is called `origin`, and Learning Equality is `le`.

First, create a new local working branch:

```
# checkout the upstream develop branch
git checkout le/develop
# make a new feature branch
git checkout -b my-awesome-changes
```

After making changes to the code and committing them locally, push your working branch to your fork on GitHub:

```
git push origin my-awesome-changes
```

Go to Kolibri's GitHub page, and create a the new pull request.

Note: Please fill in all the applicable sections in the PR template and DELETE unecessary headings

Another member of the team will review your code, and either ask for updates on your part or merge your PR to Kolibri codebase. Until the PR is merged you can push new commits to your branch and add updates to it.

Learn more about our *Development workflow* and *Release process*

2.2.12 Development using Docker

Engineers who are familiar with Docker can start a Kolibri instance without setting up the full JavaScript and Python development environments on the host machine.

For more information, see the *docker* directory and the `docker-*` commands in the *Makefile*.

Development server

Start the Kolibri devserver running inside a container:

```
# only needed first time
make docker-build-base

# takes a few mins to run pip install -e + webpack build
make docker-devserver
```

Building a pex file

Note: The easiest way to obtain a pex file is to submit a Github PR and download the built assets from buildkite.

If you want to build and run a pex from the Kolibri code in your current local source files without relying on the github and the buildkite integration, you can run the following commands to build a pex file:

```
make docker-whl
```

The pex file will be generated in the `dist/` directory. You can run this pex file using the production server approach described below.

Production server

You can start a Kolibri instance running any pex file by setting the appropriate environment variables in your local copy of *docker/env.list* then running the commands:

```
# only needed first time
make docker-build-base

# run demo server
make docker-demoserver
```

The choice of pex file can be controlled by setting environment variables in the file *./docker/env.list*:

- `KOLIBRI_PEX_URL`: Download URL or the string `default`
- `DOCKERMNT_PEX_PATH`: Local path such as `/docker/mnt/nameof.pex`

2.3 Tech stack overview

Kolibri is a web application built primarily using Python on the server-side and JavaScript on the client-side.

Note that since Kolibri is still in development, the APIs are subject to change, and a lot of code is still in flux.

2.3.1 Server

The server is a Django application, and contains only pure-Python (3.6+) dependencies at run-time. It is responsible for:

- Interfacing with the database (either SQLite or PostgreSQL)
- Authentication and permission middleware
- Routing and handling of API calls, using the Django REST Framework
- Top-level URL routing between high-level sections of the application
- Serving basic HTML wrappers for the UI with data bootstrapped into the page
- Serving additional client assets such as fonts and images

2.3.2 Client

The frontend user interface is built using Vue and uses ES6 syntax transpiled by Bublé. The client is responsible for:

- Compositing and rendering the UI
- Managing client-side state using Vuex
- Interacting with the server through the API

2.3.3 Developer docs

Documentation is formatted using reStructuredText and the output is compiled by Sphinx and hosted on Read the Docs.

Additionally, information about the design and implementation of Kolibri might be found on Google Drive, Github, Trello, Slack, InVision, mailing lists, office whiteboards, and lurking in the fragmented collective consciousness of our team and contributors.

2.3.4 Build infrastructure

We use a combination of both Node.js and Python scripts to transform our source code as-written to the code that is run in a browser. This process involves webpack, plus a number of both custom and third-party extensions.

Preparation of client-side resources involves:

- ES6 to ES5
- Transforming Vue.js component files (*.vue) into JS and CSS
- SCSS to CSS
- Auto-prefixing CSS
- Bundling multiple JS dependencies into single files
- Minifying and compressing code

- Bundle resources such as fonts and images
- Generating source maps
- Providing mechanisms for decoupled "Kolibri plugins" to interact with each other and asynchronously load dependencies

The *Makefile* contains the top-level commands for building Python distributions, in particular wheel files (`make dist`) and pex files (`make pex`).

The builds are automated using buildkite, whose top-level configuration lives in the Kolibri repo. Other platform distributions such as Windows, Debian, and Android are built from the wheel files and maintained in their own repositories.

2.3.5 Automated testing

We use a number of mechanisms to help encourage code quality and consistency. Most of these are run automatically on Github pull requests, and developers should run them locally too.

- pre-commit is run locally on `git commit` and enforces a variety of code conventions
- We use EditorConfig to help developers set their editor preferences
- tox is used to run our test suites under a range of Python and Node environment versions
- `sphinx-build -b linkcheck` checks the validity of documentation links
- pytest runs our Python unit tests. We also leverage the Django test framework.
- In addition to building client assets, webpack runs linters on client-side code: ESLint for ES6 JavaScript, Stylelint for SCSS, and HTMLHint for HTML and Vue.js components.
- Client-side code is tested using Jest
- codecov reports on the test coverage
- We have Sentry clients integrated (off by default) for automated error reporting

2.4 How To Guides

These guides are step by step guides for common tasks in getting started and working on Kolibri.

2.4.1 Installing pyenv

Prerequisites

Git installed.

Install

First check to see if you already have `pyenv` installed by running this in a terminal window:

```
pyenv
```

If it is already installed, either update it using `pyenv update` or using the package manager that you used to install it.

If it is not installed you can install it using the following command, pasted into a new terminal window:

```
curl https://pyenv.run | bash
```

The output of the command tells you to add certain lines to your startup files for your terminal sessions. Follow the PyEnv setup instructions copied below - if you are unsure which section to follow, you are probably using a **bash** shell.

- For **bash**:

  Stock Bash startup files vary widely between distributions/operating systems in terms of which of them source which startup files, under what circumstances, in what order and what additional configuration they perform. As such, the most reliable way to get Pyenv in all environments is to append Pyenv configuration commands to both `.bashrc` (for interactive shells) and the profile file that Bash would use (for login shells).

  First, add the commands to `~/.bashrc`:

  ```
  echo 'export PYENV_ROOT="$HOME/.pyenv"' >> ~/.bashrc
  echo 'command -v pyenv >/dev/null || export PATH="$PYENV_ROOT/bin:$PATH"' >> ~/.
  ↪bashrc
  echo 'eval "$(pyenv init -)"' >> ~/.bashrc
  echo 'eval "$(pyenv virtualenv-init -)"' >> ~/.bashrc
  ```

  Then, if you have `~/.profile`, `~/.bash_profile` or `~/.bash_login`, add the commands there as well. If you have none of these, add them to `~/.profile`.

  – to add to `~/.profile`:

    ```
    echo 'export PYENV_ROOT="$HOME/.pyenv"' >> ~/.profile
    echo 'command -v pyenv >/dev/null || export PATH="$PYENV_ROOT/bin:$PATH"' >> ~/.
    ↪profile
    echo 'eval "$(pyenv init -)"' >> ~/.profile
    echo 'eval "$(pyenv virtualenv-init -)"' >> ~/.profile
    ```

  – to add to `~/.bash_profile`:

    ```
    echo 'export PYENV_ROOT="$HOME/.pyenv"' >> ~/.bash_profile
    echo 'command -v pyenv >/dev/null || export PATH="$PYENV_ROOT/bin:$PATH"' >> ~/.
    ↪bash_profile
    echo 'eval "$(pyenv init -)"' >> ~/.bash_profile
    echo 'eval "$(pyenv virtualenv-init -)"' >> ~/.bash_profile
    ```

- For **Zsh**:

  ```
  echo 'export PYENV_ROOT="$HOME/.pyenv"' >> ~/.zshrc
  echo 'command -v pyenv >/dev/null || export PATH="$PYENV_ROOT/bin:$PATH"' >> ~/.
  ↪zshrc
  echo 'eval "$(pyenv init -)"' >> ~/.zshrc
  echo 'eval "$(pyenv virtualenv-init -)"' >> ~/.zshrc
  ```

If you wish to get Pyenv in noninteractive login shells as well, also add the commands to `~/.zprofile` or `~/.zlogin`.

- For **Fish shell**:

  Execute this interactively:

  ```
  set -Ux PYENV_ROOT $HOME/.pyenv
  set -U fish_user_paths $PYENV_ROOT/bin $fish_user_paths
  ```

  And add this to `~/.config/fish/config.fish`:

  ```
  pyenv init - | source
  ```

  Bash warning: There are some systems where the `BASH_ENV` variable is configured to point to `.bashrc`. On such systems, you should almost certainly put the `eval "$(pyenv init -)"` line into `.bash_profile`, and **not** into `.bashrc`. Otherwise, you may observe strange behaviour, such as `pyenv` getting into an infinite loop. See #264 for details.

  Proxy note: If you use a proxy, export `http_proxy` and `https_proxy`, too.

Installation of pyenv on Windows

1. Run PowerShell terminal as Administrator
2. Run the following installation command in the PowerShell terminal :

```
Invoke-WebRequest -UseBasicParsing -Uri "https://raw.githubusercontent.com/pyenv-win/
↪pyenv-win/master/pyenv-win/install-pyenv-win.ps1" -OutFile "./install-pyenv-win.ps1"; &
↪"./install-pyenv-win.ps1"
```

If you are getting any **UnauthorizedAccess** error, run:

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope LocalMachine
```

press Y not A, to execute Policy Change for this power shell only.

then run the previous command again

Restart your shell

For the `PATH` changes to take effect, run the following command.

```
exec "$SHELL"
```

This will give no visual indicator, but the `pyenv` command should now work in the terminal.

To check if pyenv is installed correctly, type:

```
pyenv version
```

Install Python build dependencies

Install Python build dependencies before attempting to install a new Python version.

You can now begin using Pyenv.

Installing a Python Version with PyEnv

Use the `pyenv` command in your terminal to install a recent version of Python:

```
pyenv install 3.9.9
```

The PyEnv installation wiki provides a list of common issues when installing Python versions on different operating systems.

You can now activate this version of Python just for this shell session:

```
pyenv shell 3.9.9
```

Now if you check the Python version it should be 3.9.9:

```
python --version
```

See the PyEnv documentation for more detailed usage of the `pyenv` command.

2.4.2 Using pyenv-virtualenv

Virtual Environments

Virtual environments allow a developer to have an encapsulated Python environment, using a specific version of Python, and with dependencies installed in a way that only affect the virtual environment. This is important as different projects or even different versions of the same project may have different dependencies, and virtual environments allow you to switch between them seamlessly and explicitly.

Using `pyenv virtualenv` with pyenv

To create a virtualenv for the Python version used with pyenv, run `pyenv virtualenv`, specifying the Python version you want and the name of the virtualenv directory. For example, because we can make a virtual environment for Kolibri using Python 3.9.9:

```
$ pyenv virtualenv 3.9.9 kolibri-py3.9
```

If you get 'command not found' or a similar error, and pyenv virtualenv is not installed, please follow the installation instructions.

will create a virtualenv based on Python 3.9.9 under `$(pyenv root)/versions` in a folder called `kolibri-py3.9`.

List existing virtualenvs

`pyenv virtualenvs` shows you the list of existing virtualenvs and `conda` environments.

```
$ pyenv virtualenvs
  3.9.9/envs/kolibri-py3.9 (created from /home/youuuu/.pyenv/versions/3.9.9)
  kolibri-py3.9 (created from /home/youuuu/.pyenv/versions/3.9.9)
```

There are two entries for each virtualenv, and the shorter one is just a symlink.

Activate virtualenv

If you want a virtual environment to always activate when you enter a certain directory, you can use the ``*pyenv local*` <https://github.com/pyenv/pyenv/blob/master/COMMANDS.md#pyenv-local>`_ command.

```
pyenv local kolibri-py3.9
```

Now whenever you enter the directory, the virtual environment will be activated.

You can also activate and deactivate a pyenv virtualenv manually:

```
pyenv activate kolibri-py3.9
pyenv deactivate
```

Delete existing virtualenv

Removing the directories in `$(pyenv root)/versions` and `$(pyenv root)/versions/{version}/envs` will delete the virtualenv, or you can run:

```
pyenv uninstall kolibri-py3.9
```

You can also delete existing virtualenvs by using `virtualenv-delete` command, e.g. you can run:

```
pyenv virtualenv-delete kolibri-py3.9
```

This will delete virtualenv called `kolibri-py3.9`.

For more information on use of virtual environments see the pyenv-virtualenv documentation.

2.4.3 Using nodeenv

Instructions

Once you've created a python virtual environment, you can use `nodeenv` to install particular versions of node.js within the environment. This allows you to use a different node.js version in the virtual environment than what's available on your host, keep multiple virtual enviroments with different versions of node.js, and to install node.js "global" modules that are only available within the virtual environment.

First make sure your virtual environment is activated. With pyenv you can do this with:

```
$ pyenv activate kolibri-py3.9
```

If nodeenv is not already installed in your virtual environment, you can install it using this command:

```
$ pip install nodeenv
```

If you don't already know what version you need to install, the first step is to determine the latest node.js version for a major release version. You can use `nodeenv` to list out all versions:

```
$ nodeenv -l
```

but this lists out everything. Alternatively, here's a one line bash function that can be used to determine that version:

```
$ function latest-node() { curl -s "https://nodejs.org/dist/latest-v$1.x/" | egrep -m 1 -
↪o "$1\.[0-9]+\.[0-9]+" | head -1; }
$ latest-node 18
18.19.0
```

Once you've determined the version, you can install it:

```
$ nodeenv --python-virtualenv --node 18.19.0
 * Install prebuilt node (18.19.0) ..... done.
 * Appending data to /home/bjester/Projects/learningequality/kolibri/venv/bin/activate
 * Appending data to /home/bjester/Projects/learningequality/kolibri/venv/bin/activate.
↪fish
```

You'll notice in the output above, the installation modifies the virtual environment activation scripts. Reloading the virtual environment will ensure everything works correctly.

```
$ pyenv deactivate
$ pyenv activate kolibri-py3.9
$ npm install -g yarn # success
```

2.4.4 Rebasing a Pull Request

On certain occasions, it might be necessary to redirect a pull request from the develop branch to the latest release branch, such as `release-v*` (e.g., `release-v0.16.x` when working on version 0.16), or vice versa. This guide outlines the steps for rebasing a feature branch related to your pull request while maintaining a clean commit history.

The demonstration centers on the process of rebasing a feature branch that is directed towards the `develop` branch in your pull request, transitioning it to the most recent release branch, identified as `release-v*`. If the need arises to rebase your pull request in the opposite direction—from `release-v*` to `develop` you can follow the same steps, just adjusting the branch names as indicated in the guide below.

- Make sure you have local versions of the `develop` branch and the `release-v*` branch.
- Ensure that both branches are up to date. For this guide, we'll assume they are named `develop` and `release-v*`, respectively.

Locally, checkout your feature branch and run the following rebase command:

```
git rebase --onto release-v* develop
```

This command will rebase your current feature branch onto `release-v*`, removing any commits that are already present in `develop`.

After completing the rebase, you will need to force push to update your remote branch. Use the following command:

```
git push --force
```

Caution: Handle force-pushes with care.

2.4.5 Running another Kolibri instance alongside the development server

This guide will walk you through the process of setting up and running another instance of Kolibri alongside your development server using the `pex` executable.

Introduction

As Kolibri's features continue to expand into remote content browsing, it's often necessary to test and experiment with another Kolibri instance running alongside your development server. One effective approach is to use the `pex` executable. This workflow is straightforward and can be employed independently of ZeroTier or even internet network access. By following these steps, you can effectively simulate real-world scenarios and enhance your development workflow.

Steps

- **Locate the .pex executable:**
  - Navigate to the Kolibri GitHub repository.
  - Click on the "Actions" tab at the top of the repository.
  - Select the "Kolibri Build Assets for Pull Request" option from the sidebar.
  - Select a workflow build from the list.
  - Scroll down the workflow build page to find the "Artifacts" section. In this section, you will find the `.pex` file that you need to download.
- **Save and unzip the .pex file:**

  Save the downloaded `.pex` file to a suitable location on your machine. Unzip the downloaded `pex` file to a folder where you want to run the additional Kolibri instance from.
- **Run another Kolibri instance:**

  First, make sure you are using Python version <= 3.9.

  Then, open your terminal and navigate to the folder where you unzipped the `pex` file. Use the following command to start another Kolibri instance:

  ```
  KOLIBRI_HOME="<foldername>" python <filename>.pex start
  ```

  Replace `<filename>` with the actual filename of the `pex` executable and replace `<foldername>` with the desired name for the folder that will store the settings and data for this instance.

  Be sure to choose a meaningful folder name and avoid leaving it blank to ensure it doesn't overwrite your default `.kolibri`directory.

  Note: You don't need to create the folder beforehand; it will be automatically generated if not already present when you run the command.
- **Complete initial setup:**

  In the terminal, you'll find the URL of the new Kolibri instance. Open the URL in your browser and complete the initial device setup as you would for a regular Kolibri instance. Additionally, import a few resources from desired channels.

- **Run your development server:**

  Once the additional Kolibri instance is up and running, start your development server as usual. You should now see the new device on your network.

- **Stop the other Kolibri instance:**

  When you're done testing, you can stop the additional Kolibri instance using the following command:

  ```
  python <filename>.pex stop
  ```

  This will gracefully shut down the instance.

2.4.6 Running Kolibri with local Kolibri Design System

Kolibri uses components from Kolibri Design System (KDS). KDS is installed in Kolibri as a usual npm dependency.

It is sometimes useful to run Kolibri development server linked to local KDS repository, for example to confirm that a KDS update fixes bug observed in Kolibri, when developing new KDS feature in support of Kolibri feature, etc.

For this purpose, Kolibri provides `devserver-with-kds` command that will run the development server with Kolibri using local KDS:

```
yarn run devserver-with-kds <kds-path>
```

where `<kds-path>` is the path of the local `kolibri-design-system` repository.

It is recommended to use an absolute KDS path as some developers observed problems when running the command with a relative path.

2.5 Frontend architecture

2.5.1 Single-page Apps

The Kolibri frontend is made of a few high-level "app" plugins, which are single-page JS applications (conventionally *app.js*) with their own base URL and a single root Vue.js component. Examples of apps are 'Learn' and 'User Management'. Apps are independent of each other, and can only reference components and styles from within themselves and from core.

Each app is implemented as a Kolibri plugin (see *Kolibri plugin architecture*), and is defined in a subdirectory of *kolibri/plugins*.

On the Server-side, the `kolibri_plugin.py` file describes most of the configuration for the single-page app. In particular, this includes the base Django HTML template to return (with an empty <body>), the URL at which the app is exposed, and the javascript entry file which is run on load.

On the client-side, the app creates a single `KolibriModule` object in the entry file (conventionally *app.js*) and registers this with the core app, a global variable called `kolibriCoreAppGlobal`. The Kolibri Module then mounts single root component to the HTML returned by the server, which recursively contains all additional components, html and logic.

Defining a new Kolibri module

Note: This section is mostly relevant if you are creating a new app or plugin. If you are just creating new components, you don't need to do this.

A Kolibri Module is initially defined in Python by sub-classing the `WebpackBundleHook` class (in `kolibri.core.webpack.hooks`). The hook defines the JS entry point (conventionally called *app.js*) where the `KolibriModule` subclass is instantiated, and where events and callbacks on the module are registered. These are defined in the `events` and `once` properties. Each defines key-value pairs of the name of an event, and the name of the method on the `KolibriModule` object. When these events are triggered on the Kolibri core JavaScript app, these callbacks will be called. (If the `KolibriModule` is registered for asynchronous loading, the Kolibri Module will first be loaded, and then the callbacks called when it is ready. See *Frontend build pipeline* for more information.)

All apps should extend the `KolibriModule` class found in *kolibri/core/assets/src/kolibri_module.js*.

The `ready` method will be automatically executed once the Module is loaded and registered with the Kolibri Core App. By convention, JavaScript is injected into the served HTML *after* the `<rootvue>` tag, meaning that this tag should be available when the `ready` method is called, and the root component (conventionally in *vue/index.vue*) can be mounted here.

Creating a side nav entry

If you want to expose your new single page app as a top level navigation item in the sidebar nav, then it is necessary to create a nav item in your plugin. This is implemented as a hook, which is a combination of the WebpackBundleHook and a navigation hook. So it allows the creation of a navigation item frontend bundle, and signalling that this should be included as a navigation item. Here is an example of it in use.

```python
from kolibri.core.hooks import NavigationHook
from kolibri.plugins.hooks import register_hook

@register_hook
class ExampleNavItem(NavigationHook):
    bundle_id = "side_nav"
```

For more information on using *bundle_id* and connecting it to the relevant Javascript entry point read the documentation on the *Frontend build pipeline*. The entry point for the nav item should minimally do the following:

```
<template>

  <CoreMenuOption
    :label="$tr('label')"
    :link="url"
    icon="learn"
  />

</template>


<script>

  import CoreMenuOption from 'kolibri.coreVue.components.CoreMenuOption';
  import navComponents from 'kolibri.utils.navComponents';
```

(continues on next page)

(continued from previous page)

```
  import urls from 'kolibri.urls';

  const component = {
    name: 'ExampleSideNavEntry',
    components: {
      CoreMenuOption,
    },
    computed: {
      url() {
        return urls['kolibri:kolibri.plugins.example:example']();
      },
    },
    priority: 5,
    $tr: {
      label: 'Example',
    },
  };

  navComponents.register(component);

  export default component;

</script>
```

This will create a navigation component which will be registered to appear in the navigation side bar.

Content renderers

A special kind of Kolibri Module is dedicated to rendering particular content types. All content renderers should extend the `ContentRendererModule` class found in *kolibri/core/assets/src/content_renderer_module.js*. In addition, rather than subclassing the `WebpackBundleHook` class, content renderers should be defined in the Python code using the `ContentRendererHook` class defined in `kolibri.content.hooks`. In addition to the standard options for the `WebpackBundleHook`, the `ContentRendererHook` also requires a `presets` tuple listing the format presets that it will render.

Kolibri Content hooks

Hooks for managing the display and rendering of content.

`class kolibri.core.content.hooks.`**`ContentNodeDisplayHook`**`(*args, **kwargs)`

A hook that registers a capability of a plugin to provide a user interface for a content node. When subclassed, this hook should expose a method that accepts a ContentNode instance as an argument, and returns a URL where the interface to interacting with that node for the user is exposed. If this plugin cannot produce an interface for this particular content node then it may return None.

`class kolibri.core.content.hooks.`**`ContentRendererHook`**`(*args, **kwargs)`

An inheritable hook that allows special behaviour for a frontend module that defines a content renderer.

`render_to_page_load_async_html()`

Generates script tag containing Javascript to register a content renderer.

Returns

HTML of a script tag to insert into a page.

The ContentRendererModule class has one required property getRendererComponent which should return a Vue component that wraps the content rendering code. This component will be passed files, file, itemData, preset, itemId, answerState, allowHints, extraFields, interactive, lang, showCorrectAnswer, defaultItemPreset, availableFiles, defaultFile, supplementaryFiles, thumbnailFiles, contentDirection, and contentIsRtl props, defining the files associated with the piece of content, and other required data for rendering. These will be automatically mixed into any content renderer component definition when loaded. For more details of these props see the Content Renderer documentation.

In order to log data about users viewing content, the component should emit startTracking, updateProgress, and stopTracking events, using the Vue $emit method. startTracking and stopTracking are emitted without any arguments, whereas updateProgress should be emitted with a single value between 0 and 1 representing the current proportion of progress on the content.

```
this.$emit('startTracking');
this.$emit('stopTracking');
this.$emit('updateProgress', 0.25);
```

For content that has assessment functionality three additional props will be passed: itemId, answerState, and showCorrectAnswer. itemId is a unique identifier for that content for a particular question in the assessment, answerState is passed to prefill an answer (one that has been previously given on an exam, or for a coach to preview a learner's given answers), showCorrectAnswer is a Boolean that determines if the correct answer for the question should be automatically prefilled without user input - this will only be activated in the case that answerState is falsy - if the renderer is asked to fill in the correct answer, but is unable to do so, it should emit an answerUnavailable event.

The answer renderer should also define a checkAnswer method in its component methods, this method should return an object with the following keys: correct, answerState, and simpleAnswer - describing the correctness, an object describing the answer that can be used to reconstruct it within the renderer, and a simple, human readable answer. If no valid answer is given, null should be returned. In addition to the base content renderer events, assessment items can also emit a hintTaken event to indicate that the user has taken a hint in the assessment, an itemError event to indicate that there has been an error in rendering the requested question corresponding to the itemId, and an interaction event that indicates a user has interacted with the assessment.

```
{
  methods: {
    checkAnswer() {
      return {
        correct: true,
        answerState: {
          answer: 81,
          working: '3^2 = 3 * 3',
        },
        simpleAnswer: '81',
      };
    },
  },
};
```

2.5.2 Layout of frontend code

Frontend code and assets are generally contained in one of two places: either in one of the plugin subdirectories (under *kolibri/plugins*) or in *kolibri/core*, which contains code shared across all plugins as described below.

Within these directories, there should be an *assets* directory with *src* and *test* under it. Most assets will go in *src*, and tests for the components will go in *test*.

For example:

```
kolibri/
  core/                            # core (shared) items
    assets/
      src/
        CoreBase.vue               # global base template, used by plugins
        CoreModal.vue              # example of another shared component
        core-global.scss           # globally defined styles, included in head
        core-theme.scss            # style variable values
        font-noto-sans.css         # embedded font
      test/
        ...                        # tests for core assets
  plugins/
    learn                          # learn plugin
      assets/
        src/
          views/
            LearnIndex.vue         # root view
            SomePage.vue           # top-level client-side page
            AnotherPage/           # top-level client-side page
              index.vue
              Child.vue            # child component used only by parent
            Shared.vue             # shared across this plugin
          app.js                   # instantiate learn app on client-side
          router.js
          store.js
        test/
          app.js
    management/
      assets/
        src/
          views/UserPage.vue       # nested-view
          views/ManagementIndex.vue # root view
          app.js                   # instantiate mgmt app on client-side
        test/
          app.js
```

In the example above, the *views/AnotherPage/index.vue* file in *learn* can use other assets in the same directory (such as *Child.vue*), components in *views* (such as *Shared.vue*), and assets in core (such as variables in *core-theme.scss*). However it cannot use files in other plugin directories (such as *management*).

Note: For many development scenarios, only files in these directories need to be touched.

There is also a lot of logic and configuration relevant to frontend code loading, parsing, testing, and linting. This includes webpack, NPM, and integration with the plugin system. This is somewhat scattered, and includes logic in *frontend_build/...*, *package.json*, *kolibri/core/webpack/...*, and other locations. Much of this functionality is described in other sections of the docs (such as *Frontend build pipeline*), but it can take some time to understand how it all hangs together.

2.5.3 Shared core functionality

Kolibri provides a set of shared "core" functionality – including components, styles, and helper logic, and libraries – which can be re-used across apps and plugins.

JS libraries and Vue components

The following libraries and components are available globally, in all module code:

- `vue` - the Vue.js object
- `vuex` - the Vuex object
- `logging` - our wrapper around the loglevel logging module
- `CoreBase` - a shared base Vue.js component (*CoreBase.vue*)

And **many** others. The complete specification for commonly shared modules can be found in `kolibri/core/assets/src/core-app/apiSpec.js`. This object defines which modules are imported into the core object. These can then be imported throughout the codebase - e.g.:

```
import Vue from 'kolibri.lib.vue';
import CoreBase from 'kolibri.coreVue.components.CoreBase';
```

Adding additional globally-available objects is relatively straightforward due to the *plugin and webpack build system*.

To expose something in the core app, add the module to the object in `apiSpec.js`, scoping it to the appropriate property for better organization - e.g.:

```
components: {
  CoreTable,
},
utils: {
  navComponents,
},
```

These modules would now be available for import anywhere with the following statements:

```
import CoreTable from 'kolibri.coreVue.components.CoreTable';
import navComponents from 'kolibri.utils.navComponents';
```

Note: In order to avoid bloating the core api, only add modules that need to be used in multiple plugins.

Styling

To help enforce style guide specs, we provide global variables that can be used throughout the codebase. This requires including `@import '~kolibri-design-system/lib/styles/definitions';` within a SCSS file or a component's `<style>` block. This exposes all variables in `definitions.scss`.

Dynamic core theme

Vuex state is used to drive overall theming of the application, in order to allow for more flexible theming (either for accessibility or cosmetic purposes). All core colour styles are defined in Javascript variables kept in Vuex state, which are then applied inline to elements using Vue.js style bindings from Vuex getters.

There are two cases where dynamic styles cannot be directly applied to DOM elements: - inline styles cannot apply pseudo-classes (e.g. ':hover', ':focus', '::before') - styles applied during Vue transitions

For these cases, it's necessary to define a "computed class" using the `$computedClass` function. This returns an auto-generated class name which can be used like a standard CSS class name. Under the hood, this uses Aphrodite to create unique classes for each set of inputs given, so be careful not to abuse this feature!

In order to apply a style using a computed class, define a style object as a computed property, similarly to how you might for a Vue.js style binding. Pseudo-selectors can be encoded within this object:

```
import themeMixin from 'kolibri.coreVue.mixins.themeMixin';

export default {
  mixins: [themeMixin],
  computed: {
    pseudoStyle() {
      return {
        ':hover': {
          backgroundColor: this.$themeTokens.primaryDark,
        },
      };
    },
  },
};
```

Then, within the template code, this can be applied to an element or component using a Vue.js class binding, and using the `$computedClass` method, referencing this style object:

```
<div :class="$computedClass(pseudoStyle)">I'm going to get a white background when you␣
↪hover on me!</div>
```

To use computed classes for Vue.js transitions, you can use the `{event}-class` properties as options on the `<transition>` or `<transition-group>` special component, and the `$computedClass` method can be used again:

```
<transition-group :move-class="$computedClass(pseudoSelector)">
  <div>While moving I'll have the hover style applied!</div>
</transition-group>
```

Bootstrapped data

The kolibriCoreAppGlobal object is also used to bootstrap data into the JS app, rather than making unnecessary API requests.

For example, we currently embellish the kolibriCoreAppGlobal object with a urls object. This is defined by Django JS Reverse and exposes Django URLs on the client side. This will primarily be used for accessing API Urls for synchronizing with the REST API. See the Django JS Reverse documentation for details on invoking the Url.

Additional functionality

These methods are also publicly exposed methods of the core app:

```
kolibriCoreAppGlobal.register_kolibri_module_async   // Register a Kolibri module for␣
↪asynchronous loading.
kolibriCoreAppGlobal.register_kolibri_module_sync    // Register a Kolibri module once␣
↪it has loaded.
kolibriCoreAppGlobal.stopListening                   // Unbind an event/callback pair␣
↪from triggering.
kolibriCoreAppGlobal.emit                            // Emit an event, with optional␣
↪args.
```

2.5.4 Vue components

We leverage Vue.js components as the primary building blocks for our UI. For general UI development work, this is the most common tool a developer will use. It would be prudent to read through the Vue.js guide thoroughly.

Each component contains HTML with dynamic Vue.js directives, styling which is scoped to that component (written using SCSS), and logic which is also scoped to that component (all code, including that in Vue components should be written using Bublé compatible ES2015 JavaScript).

Components allow us to define new custom tags that encapsulate a piece of self-contained, re-usable UI functionality. When composed together, they form a tree structure of parents and children. Each component has a well-defined interface used by its parent component, made up of input properties, events and content slots. Components should never reference their parent.

Read through the *Frontend code conventions* for further guidelines on writing components.

Design system

Our design system contains reusable patterns and components that should be used whenever possible to maintain UI consistency and avoid duplication of effort.

SVG Icons

Icons in Kolibri should be accessed through the <KIcon> component. The icons available can be found and searched at Kolibri Design System.

Each icon is associated with a token, which is passed to <KIcon> and other Kolibri Design System components which accept an icon or iconAfter prop such as KIconButton.

```
<!--
  embed https://material.io/resources/icons/?search=add_circl&icon=add_circle_outline&
↪style=baseline
-->
<KIcon :icon="add" />
```

2.5.5 Frontend code conventions

Establishing code conventions is important in order to keep a more consistent codebase. Therefore the goal for the tools and principles below is to help ensure any committed code is properly aligned with the conventions.

For design conventions, see the Kolibri Design System.

Linting and auto formatting

Many of our conventions are enforced through various linters including ESLint, ESLint Vue plugin, stylelint, and HTMLHint. The enforced rules are located in the `.eslintrc.js`, `.stylelintrc.js`, and `.htmlhintrc` files located at the root of the project.

Also available are options and tools that enable auto-formatting of `.vue`, `.js`, `.scss`, and `.py` files to conform to code conventions. To facilitate this, we use Black to auto-format `.py` files, and Prettier to auto-format the others. Auto-formatting runs by default while running the dev server, otherwise be sure to run the dev server with `-warn` as described in *Getting started* to prevent it from auto-formatting.

In addition, `pre-commit` hooks can be installed to perform linting and auto-formatting. To enable the hooks, be sure to follow the directions described in *Getting started*.

You can also install the appropriate editor plugins for the various linters to see linting warnings/errors inline.

Vue.js components

- Make sure to follow the official Vue.js style guide when creating Vue components.
- Keep components stateless and declarative as much as possible
- For simple components, make *SomeComonent.vue*. For more complex components, make *SomeComponent/index.vue* and add private sub-components
- All user-visible app text should be internationalized. See *Internationalization* for details
- Avoid direct DOM references and Vue component "lifecycle events" except in special cases
- Props, slots, and Vuex state/getters for communicating down the view hierarchy
- Events and Vuex actions for communicating up the view hierarchy
- If possible, use *<template/>* for conditionals to avoid extra unnecessary nested elements.

Styling anti-patterns

- **Adding unnecessary new rules** - whenever possible, delete code to fix issues
- **Unscoped styles** - if absolutely necessary, use deep selectors to style component children. SCSS supports `/deep/`
- **Classes referenced in javascript** - if absolutely necessary, use ref instead (also an anti-pattern)
- **References by ID** - use a `class` instead
- **HTML tag selectors** - define a `class` instead
- **Floats or flexbox for layout** - use `KGrid` instead
- **Media queries** - use `responsive-window` or `responsive-element`
- **Nested selectors** - make a sub-component instead (more reading here and here)
- **Dynamically-generated class names** - avoid patterns which fail the grep test
- **Complex pre-processor functionality** - use Vue computed styles instead
- **Hard-coded values** - rely on variables defined in the core theme
- **Left or right alignment on user-generated text** - use `dir="auto"` instead for RTL support

2.5.6 Vuex

We use the Vuex library to manage state. Generally Vuex should only store data that needs to persist / be accessed between views. If this is not necessary, than local component data is a better place to store the data.

To be continued…

2.5.7 HTML5 API

In order to effectively and safely host embedded HTML5 apps as a first class content type in Kolibri, we use the standard IFrame Sandbox functionality and serve HTML5 apps from a separate origin. This allows for HTML5 apps to run arbitrary Javascript, without concerns about accessing privileged user data, as the separate origin will prevent leakage of the session authentication into the sandboxed context.

Standard Web APIs

This shared origin does mean that every HTML5 app running in Kolibri is sharing the same origin - for standard Web APIs like cookies, local storage, and IndexedDB, this poses an issue, as it is possible that multiple HTML5 apps might overwrite each other’s data.

To handle this eventuality, and to provide an enhanced user experience across multiple devices we shim these APIs in the context of the sandbox. Cookies and LocalStorage are persisted across the IFrame boundary, meaning that if a user interacts with an HTML5 app and it sets data to cookies and local storage, then if the user subsequently returns to the same HTML5 app, the cookies and local storage values from the previous session will be restored and available.

IndexedDB is also shimmed, but due to the very large amount of data that can be stored in IndexedDB, and the fact that it is often used for the local caching of file based assets (by the Unity framework, for example) this data is transmitted out of the IFrame sandbox. Instead the databases for IndexedDB are namespaced, in order to prevent clashes between IndexedDB storage from multiple HTML5 apps - however, this does mean that any data persisted to IndexedDB will only be preserved within the same browser only.

SCORM

A large number of educational web content relies on the SCORM API to log data about learner interactions. In order to support this, Kolibri embeds a *SCORM* namespace on *window.parent* within the HTML5 app context. This is the standard place for SCORM API to be located, so any existing content that is SCORM compatible can be used without modification in this context. Currently, only SCORM 1.2 is supported by this interface, and there are no plans as yet to support the sequencing standard introduced by SCORM 2004. More information about SCORM 1.2 and the API it exposes is available at the SCORM website.

xAPI

A more general purpose, but not as widely used, standard for logging interactions about learning content is xAPI. In order to provide preliminary support for this standard, Kolibri exposes a *window.xAPI* object in the HTML5 app context. This API offers a set of methods that allow for using xAPI equivalent actions via a Promise based API. The methods available are loosely based on the XAPIWrapper Javascript library API, but limits its support to sending and querying statements, state, activity profiles, and agents. At the moment, the primary use case for this API is internal, it is used to log data from H5P content interactions.

Custom Navigation

The purpose of the `kolibri.js` extension of our HTML5 API is to allow a sandboxed HTML5 app to safely request the main Kolibri application's data.

External/partner product teams can create HTML5 applications that are fully embeddable within Kolibri and can read Kolibri content data, which they otherwise wouldn't be able to access. This opens up possibilities for creative ways in which learners can engage with content, because partners can create any type of app they want. The app could be something completely new, developed for a content source that we are adding to the platform, or it could be a branded, offline recreation of a partner's existing learning app that previously would not have been able to exist on Kolibri.

When a user has permissions to access a custom channel, and they click on it in the main learn tab, rather than viewing *normal Kolibri*, they will experience a full-screen HTML5 app. One *out-of-the-box* user interaction is the `navigateTo()` function, which opens a modal that displays a content node. For other data fetching requests, the app, not Kolibri, has the responsibilty of determining what to do with that data.

Basic API

Access the API from within an HTML5 app by using `window.kolibri.[function]`

Functions:

```
/**
* Type definition for Language metadata
* @typedef {Object} Language
* @property {string} id - an IETF language tag
* @property {string} lang_code - the ISO 639-1 language code
* @property {string} lang_subcode - the regional identifier
* @property {string} lang_name - the name of the language in that language
* @property {('ltr'|'rtl'|)} lang_direction - Direction of the language's script,
* top to bottom is not supported currently
*/

/**
```

(continues on next page)

(continued from previous page)

```
* Type definition for ContentNode metadata
* @typedef {Object} ContentNode
* @property {string} id - unique id of the ContentNode
* @property {string} channel_id - unique channel_id of the channel that the ContentNode␣
↪is in
* @property {string} content_id - identifier that is common across all instances of this␣
↪resource
* @property {string} title - A title that summarizes this ContentNode for the user
* @property {string} description - detailed description of the ContentNode
* @property {string} author - author of the ContentNode
* @property {string} thumbnail_url - URL for the thumbnail for this ContentNode,
* this may be any valid URL format including base64 encoded or blob URL
* @property {boolean} available - Whether the ContentNode has all necessary files for␣
↪rendering
* @property {boolean} coach_content - Whether the ContentNode is intended only for coach␣
↪users
* @property {Language} lang - The primary language of the ContentNode
* @property {string} license_description - The description of the license, which may be␣
↪localized
* @property {string} license_name - The human readable name of the license, localized
* @property {string} license_owner - The name of the person or organization that holds␣
↪copyright
* @property {number} num_coach_contents - Number of coach contents that are descendants␣
↪of this
* @property {string} parent - The unique id of the parent of this ContentNode
* @property {number} sort_order - The order of display for this node in its channel
* if depth recursion was not deep enough
*/

/**
* Type definition for PageResults array
* @property {ContentNode[]} results - the array of ContentNodes for this page
* This will be updated to a Pagination Object once pagination is implemented
*/

/**
* Type definition for Theme options
* properties TBD
* @typedef {Object} Theme
*/

/**
* Type definition for NavigationContext
* This can have arbitrary properties as defined
* by the navigating app that it uses to resume its state
* Should be able to be encoded down to <1600 characters using
* an encoding function something like 'encode context' above
* @typedef {Object} NavigationContext
* @property {string} node_id - The current node_id that is being displayed,
* custom apps should handle this as it may be used to
* generate links externally to jump to this state
*/
```

(continues on next page)

(continued from previous page)

```
/*
* Method to query contentnodes from Kolibri and return
* an array of matching metadata
* @param {Object} options - The different options to filter by
* @param {string} [options.parent] - id of the parent node to filter by, or 'self'
* @param {string} [options.ids] - an array of ids to filter by
* @return {Promise<PageResult>} - a Promise that resolves to an array of ContentNodes
*/
getContentByFilter(options)

/*
* Method to query a single contentnode from Kolibri and return
* a metadata object
* @param {string} id - id of the ContentNode
* @return {Promise<ContentNode>} - a Promise that resolves to a ContentNode
*/
getContentById(id)

/*
* Method to search for contentnodes on Kolibri and return
* an array of matching metadata
* @param {Object} options - The different options to search by
* @param {string} [options.keyword] - search term for key word search
* @param {string} [options.under] - id of topic to search under, or 'self'
* @return {Promise<PageResult>} - a Promise that resolves to an array of ContentNodes
*/
searchContent(options)

/*
* Method to set a default theme for any content rendering initiated by this app
* @param {Theme} options - The different options for custom themeing
* @param {string} [options.appBarColor] - Color for app bar atop the renderer
* @param {string} [options.textColor] - Color for the text or icon
* @param {string} [options.backdropColor] - Color for modal backdrop
* @param {string} [options.backgroundColor] - Color for modal background
* @return {Promise} - a Promise that resolves when the theme has been applied
*/
themeRenderer(options)

/*
* Method to allow navigation to or rendering of a specific node
* has optional parameter context that can update the URL for a custom context.
* When this is called for a resource node in the custom navigation context
* this will launch a renderer overlay to maintain the current state, and update the
* query parameters for the URL of the custom context to indicate the change
* If called for a topic in a custom context or outside of a custom context
* this will simply prompt navigation to that node in Kolibri.
* @param {string} nodeId - id of the parent node to navigate to
* @param {NavigationContext=} context - optional context describing the state update
* if node_id is missing from the context, it will be automatically filled in by this␣
↪method
```

(continues on next page)

(continued from previous page)

```
* @return {Promise} - a Promise that resolves when the navigation has completed
*/
navigateTo(nodeId, context)

/*
* Method to allow updating of stored state in the URL
* @param {NavigationContext} context - context describing the state update
* @return {Promise} - a Promise that resolves when the context has been updated
*/
updateContext(context)

/*
* Method to request the current context state
* @return {Promise<NavigationContext>} - a Promise that resolves
* when the context has been updated
*/
getContext()

/*
* Method to return the current version of Kolibri and hence the API available.
* @return {Promise<string>} - A version string
*/
getVersion()
```

2.5.8 Adding dependencies

Dependencies are tracked using `yarn` - see the docs here.

We distinguish development dependencies from runtime dependencies, and these should be installed as such using `yarn add --dev [dep]` or `yarn add [dep]`, respectively. Your new dependency should now be recorded in *package.json*, and all of its dependencies should be recorded in *yarn.lock*.

Individual plugins can also have their own package.json and yarn.lock for their own dependencies. Running `yarn install` will also install all the dependencies for each activated plugin (inside a node_modules folder inside the plugin itself). These dependencies will only be available to that plugin at build time. Dependencies for individual plugins should be added from within the root directory of that particular plugin.

To assist in tracking the source of bloat in our codebase, the command `yarn run bundle-stats` is available to give a full readout of the size that uglified packages take up in the final Javascript code.

In addition, a plugin can have its own webpack config, specified inside the `buildConfig.js` file for plugin specific webpack configuration (loaders, plugins, etc.). These options will be merged with the base options using `webpack-merge`.

2.5.9 Unit testing

Unit testing is carried out using Jest. All JavaScript code should have unit tests for all object methods and functions.

Tests are written in JavaScript, and placed in the 'assets/test' folder. An example test is shown below:

```
var assert = require('assert');

var SearchModel = require('../src/search/search_model.js');

describe('SearchModel', function() {
  describe('default result', function() {
    it('should be empty an empty array', function () {
      var test_model = new SearchModel();
      assert.deepEqual(test_model.get("result"), []);
    });
  });
});
```

Vue.js components can also be tested. The management plugin contains an example (*kolibri/plugins/management/assets/test/management.js*) where the component is bound to a temporary DOM node, changes are made to the state, and assertions are made about the new component structure.

2.5.10 Frontend build pipeline

Asset pipelining is done using Webpack - this allows the use of require to import modules - as such all written code should be highly modular, individual files should be responsible for exporting a single function or object.

There are two distinct entities that control this behaviour - a Kolibri Hook on the Python side, which manages the registration of the frontend code within Django and a `buildConfig.js` file for the webpack configuration. The format of the `buildConfig.js` is relatively straight forward, and the Kolibri Hook and the `buildConfig.js` are connected by a single shared `bundle_id` specified in both:

```
@register_hook
class LearnNavItem(NavigationHook):
    bundle_id = "side_nav"


@register_hook
class LearnAsset(webpack_hooks.WebpackBundleHook):
    bundle_id = "app"
```

```
module.exports = [
  {
    bundle_id: 'app',
    webpack_config: {
      entry: './assets/src/app.js',
    },
  },
  {
    bundle_id: 'side_nav',
    webpack_config: {
      entry: './assets/src/views/LearnSideNavEntry.vue',
```

(continues on next page)

(continued from previous page)

```
    },
  },
];
```

The two specifications are connected by the shared specification of the `bundle_id`. Minimally an `entry` value for the `webpack_config` object is required, but any other valid webpack configuration options may be passed as part of the object - they will be merged with the default Kolibri webpack build.

Kolibri has a system for synchronously and asynchronously loading these bundled JavaScript modules which is mediated by a small core JavaScript app, `kolibriCoreAppGlobal`. Kolibri Modules define to which events they subscribe, and asynchronously registered Kolibri Modules are loaded by `kolibriCoreAppGlobal` only when those events are triggered. For example if the Video Viewer's Kolibri Module subscribes to the *content_loaded:video* event, then when that event is triggered on `kolibriCoreAppGlobal` it will asynchronously load the Video Viewer module and re-trigger the *content_loaded:video* event on the object the module returns.

Synchronous and asynchronous loading is defined by the template tag used to import the JavaScript for the Kolibri Module into the Django template. Synchronous loading merely inserts the JavaScript and CSS for the Kolibri Module directly into the Django template, meaning it is executed at page load.

This can be achieved in two ways using template tags.

The first way is simply by using the `webpack_asset` template tag defined in *kolibri/core/webpack/templatetags/webpack_tags.py*.

The second way is if a Kolibri Module needs to load in the template defined by another plugin or a core part of Kolibri, a template tag and hook can be defined to register that Kolibri Module's assets to be loaded on that page. An example of this is found in the `base.html` template using the `frontend_base_assets` tag, the hook that the template tag uses is defined in *kolibri/core/hooks.py*.

Asynchronous loading can also, analogously, be done in two ways. Asynchronous loading registers a Kolibri Module against `kolibriCoreAppGlobal` on the frontend at page load, but does not load, or execute any of the code until the events that the Kolibri Module specifies are triggered. When these are triggered, the `kolibriCoreAppGlobal` will load the Kolibri Module and pass on any callbacks once it has initialized. Asynchronous loading can be done either explicitly with a template tag that directly imports a single Kolibri Module using `webpack_base_async_assets`.

For some parts of the build system, we pre-build assets and commit them to the repository, when we essentially vendoring a built version of an external library. We do this for both the Khan Academy Perseus renderer, which we build a version of and commit to the repository, and the H5P Javascript files. Both have their own build processes that configured within the yarn workspaces for each.

The Perseus build currently draws from the Learning Equality fork of the Perseus repository here we have made specific updates to Perseus, as it is no longer open sourced by Khan Academy. We have also made some edits and updates that make our build process easier and more streamlined. To run the build process to rebuild perseus dist bundle from the head of the default branch of our fork, run `yarn workspace kolibri-perseus-viewer run build-perseus`. This will update all the relevant files and leave a diff to commit after it has finished. This should be committed and submitted as a pull request to update the code.

2.6 Backend architecture

2.6.1 Content database module

This is a core module found in `kolibri/core/content`.

Concepts and Definitions

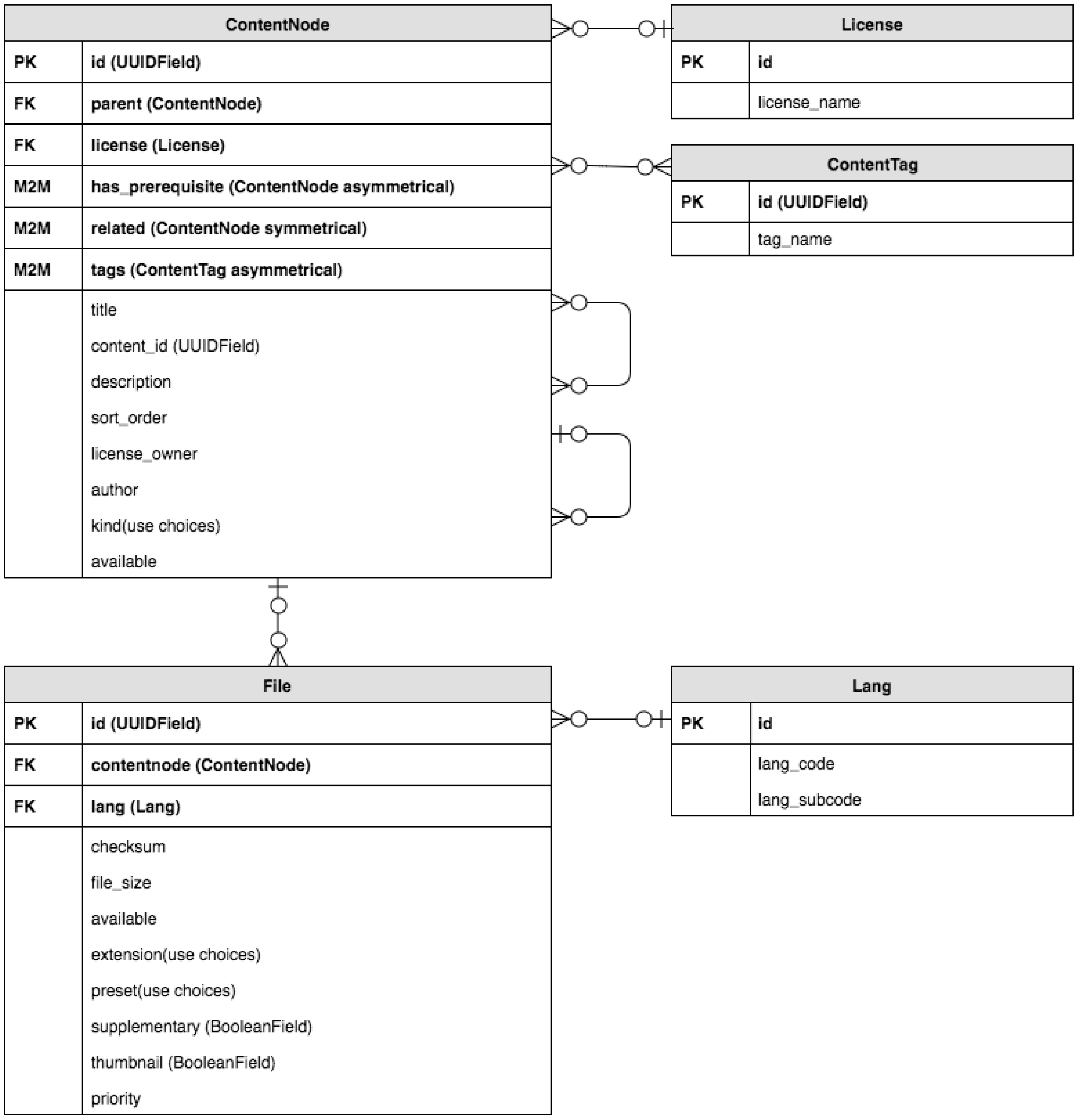
ContentNode

High-level abstraction for representing different content kinds, such as Topic, Video, Audio, Exercise, and Document, and can be easily extended to support new content kinds. With multiple ContentNode objects, it supports grouping, arranging them in tree structure, and symmetric and asymmetric relationship between two ContentNode objects.

File

Model that stores details about a source file such as the language, size, format, and location.

ContentDB diagram



- PK = Primary Key
- FK = Foreign Key
- M2M = ManyToManyField

ContentTag

This model is used to establish a filtering system for all ContentNode objects.

ChannelMetadata

Model in each content database that stores the database readable names, description and author for each channel.

ChannelMetadataCache

This class stores the channel metadata cached/denormed into the default database.

Implementation details and workflows

To achieve using separate databases for each channel and be able to switch channels dynamically, the following data structure and utility functions have been implemented.

ContentDBRoutingMiddleware

This middleware will be applied to every request, and will dynamically select a database based on the channel_id. If a channel ID was included in the URL, it will ensure the appropriate content DB is used for the duration of the request. (Note: *set_active_content_database* is thread-local, so this shouldn't interfere with other parallel requests.)

For example, this is how the client side dynamically requests data from a specific channel:

```
>>> localhost:8000/api/content/<channel_1_id>/contentnode
```

this will respond with all the contentnode data stored in database <channel_1_id>.sqlite3

```
>>> localhost:8000/api/content/<channel_2_id>/contentnode
```

this will respond with all the contentnode data stored in database <channel_2_id>.sqlite3

get_active_content_database

A utility function to retrieve the temporary thread-local variable that *using_content_database* sets

set_active_content_database

A utility function to set the temporary thread-local variable

using_content_database

A decorator and context manager to do queries on a specific content DB.

Usage as a context manager:

```
from models import ContentNode

with using_content_database("nalanda"):
    objects = ContentNode.objects.all()
    return objects.count()
```

Usage as a decorator:

```
from models import ContentNode

@using_content_database('nalanda')
def delete_all_the_nalanda_content():
    ContentNode.objects.all().delete()
```

ContentDBRouter

A router that decides what content database to read from based on a thread-local variable.

ContentNode

`ContentNode` is implemented as a Django model that inherits from two abstract classes, MPTTModel and ContentDatabaseModel.

- django-mptt's MPTTModel allows for efficient traversal and querying of the ContentNode tree.
- `ContentDatabaseModel` is used as a marker so that the content_db_router knows to query against the content database only if the model inherits from ContentDatabaseModel.

The tree structure is established by the `parent` field that is a foreign key pointing to another ContentNode object. You can also create a symmetric relationship using the `related` field, or an asymmetric field using the `is_prerequisite` field.

File

The `File` model also inherits from `ContentDatabaseModel`.

To find where the source file is located, the class method `get_url` uses the `checksum` field and `settings.CONTENT_STORAGE_DIR` to calculate the file path. Every source file is named based on its MD5 hash value (this value is also stored in the `checksum` field) and stored in a namespaced folder under the directory specified in `settings.CONTENT_STORAGE_DIR`. Because it's likely to have thousands of content files, and some filesystems cannot handle a flat folder with a large number of files very well, we create namespaced subfolders to improve the performance. So the eventual file path would look something like:

`[CONTENT_STORAGE_DIR]/content/storage/9/8/9808fa7c560b9801acccf0f6cf74c3ea.mp4`

Content constants

A Python module that stores constants for the `kind` field in ContentNode model and the `preset` field and `extension` field in File model.

Workflows

There are two workflows that handle content navigation and content rendering:

- Content navigation
  1. Start with a ContentNode object.
  2. Get the associated File object that has the `thumbnail` field being True.
  3. Get the thumbnail image by calling this File's `get_url` method.
  4. Determine the template using the `kind` field of this ContentNode object.
  5. Renders the template with the thumbnail image.
- Content rendering
  1. Start with a ContentNode object.
  2. Retrieve a queryset of associated File objects that are filtered by the preset.
  3. Use the `thumbnail` field as a filter on this queryset to get the File object and call this File object's `get_url` method to get the source file (the thumbnail image)
  4. Use the `supplementary` field as a filter on this queryset to get the "supplementary" File objects, such as caption (subtitle), and call these File objects' `get_url` method to get the source files.
  5. Use the `supplementary` field as a filter on this queryset to get the essential File object. Call its `get_url` method to get the source file and use its `extension` field to choose the content player.
  6. Play the content.

API methods

`class kolibri.core.content.api.`**`BaseContentNodeMixin`**

A base mixin for viewsets that need to return the same format of data serialization for ContentNodes. Also used for public ContentNode endpoints!

`class kolibri.core.content.api.`**`BaseContentNodeTreeViewset`**`(*args, **kwargs)`

`retrieve``(request, pk=None)`

A nested, paginated representation of the children and grandchildren of a specific node

GET parameters on request can be: depth - a value of either 1 or 2 indicating the depth to recurse the tree, either 1 or 2 levels if this parameter is missing it will default to 2. next__gt - a value to return child nodes with a lft value greater than this, if missing defaults to None

The pagination object returned for "children" will have this form: results - a list of serialized children, that can also have their own nested children attribute. more - a dictionary or None, if a dictionary, will have an id key that is the id of the parent object for these children, and a params key that is a dictionary of the required query parameters to query more children for this parent - at a minimum this will include next__gt and depth, but may also include other query parameters for filtering content nodes.

The "more" property describes the "id" required to do URL reversal on this endpoint, and the params that should be passed as query parameters to get the next set of results for pagination.

Parameters

- `request` – request object
- `pk` – id parent node

Returns

an object representing the parent with a pagination object as "children"

`class kolibri.core.content.api.ChannelMetadataViewSet(*args, **kwargs)`

`dispatch(request, *args, **kwargs)`

.dispatch() is pretty much the same as Django's regular dispatch, but with extra hooks for startup, finalize, and exception handling.

`class kolibri.core.content.api.CharInFilter(*args, **kwargs)`

`class kolibri.core.content.api.ContentNodeBookmarksViewset(*args, **kwargs)`

`get_queryset()`

Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using *self.queryset*.

This method should always be used rather than accessing *self.queryset* directly, as *self.queryset* gets evaluated only once, and those results are cached for all subsequent requests.

You may want to override this if you need to provide different querysets depending on the incoming request.

(Eg. return a list of items that is specific to the user)

`pagination_class`

alias of `ValuesViewsetLimitOffsetPagination`

`class kolibri.core.content.api.ContentNodeGranularViewset(**kwargs)`

`get_queryset()`

Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using *self.queryset*.

This method should always be used rather than accessing *self.queryset* directly, as *self.queryset* gets evaluated only once, and those results are cached for all subsequent requests.

You may want to override this if you need to provide different querysets depending on the incoming request.

(Eg. return a list of items that is specific to the user)

`get_serializer_context()`

Extra context provided to the serializer class.

`serializer_class`

alias of `ContentNodeGranularSerializer`

`class kolibri.core.content.api.ContentNodeProgressViewset(**kwargs)`

`get_queryset()`

Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using *self.queryset*.

This method should always be used rather than accessing *self.queryset* directly, as *self.queryset* gets evaluated only once, and those results are cached for all subsequent requests.

You may want to override this if you need to provide different querysets depending on the incoming request.

(Eg. return a list of items that is specific to the user)

`pagination_class`

alias of *OptionalPagination*

class `kolibri.core.content.api.ContentNodeSearchViewset`(**args*, ***kwargs*)

`initial`(*request*, **args*, ***kwargs*)

Runs anything that needs to occur prior to calling the method handler.

`search`(*value*, *max_results*, *filter=True*)

Implement various filtering strategies in order to get a wide range of search results. When filter is used, this object must have a request attribute having a 'query_params' QueryDict containing the filters to be applied

class `kolibri.core.content.api.ContentNodeTreeViewset`(**args*, ***kwargs*)

`dispatch`(*request*, **args*, ***kwargs*)

.dispatch() is pretty much the same as Django's regular dispatch, but with extra hooks for startup, finalize, and exception handling.

`retrieve`(*request*, *pk=None*)

A nested, paginated representation of the children and grandchildren of a specific node

GET parameters on request can be: depth - a value of either 1 or 2 indicating the depth to recurse the tree, either 1 or 2 levels if this parameter is missing it will default to 2. next__gt - a value to return child nodes with a lft value greater than this, if missing defaults to None

The pagination object returned for "children" will have this form: results - a list of serialized children, that can also have their own nested children attribute. more - a dictionary or None, if a dictionary, will have an id key that is the id of the parent object for these children, and a params key that is a dictionary of the required query parameters to query more children for this parent - at a minimum this will include next__gt and depth, but may also include other query parameters for filtering content nodes.

The "more" property describes the "id" required to do URL reversal on this endpoint, and the params that should be passed as query parameters to get the next set of results for pagination.

Parameters

- **`request`** – request object
- **`pk`** – id parent node

Returns

an object representing the parent with a pagination object as "children"

class `kolibri.core.content.api.ContentNodeViewset`(**args*, ***kwargs*)

`descendants`(*request*)

Returns a slim view all the descendants of a set of content nodes (as designated by the passed in ids). In addition to id, title, kind, and content_id, each node is also annotated with the ancestor_id of one of the ids that are passed into the request. In the case where a node has more than one ancestor in the set of content nodes requested, duplicates of that content node are returned, each annotated with one of the ancestor_ids for a node.

`dispatch`(*request*, **args*, ***kwargs*)

.dispatch() is pretty much the same as Django's regular dispatch, but with extra hooks for startup, finalize, and exception handling.

pagination_class

alias of *OptionalContentNodePagination*

recommendations_for(*request*, ***kwargs*)

Recommend items that are similar to this piece of content.

class kolibri.core.content.api.**ContentRequestViewset**(**args*, ***kwargs*)

get_queryset()

Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using *self.queryset*.

This method should always be used rather than accessing *self.queryset* directly, as *self.queryset* gets evaluated only once, and those results are cached for all subsequent requests.

You may want to override this if you need to provide different querysets depending on the incoming request.

(Eg. return a list of items that is specific to the user)

pagination_class

alias of *OptionalPageNumberPagination*

serializer_class

alias of `ContentDownloadRequestSerializer`

class kolibri.core.content.api.**FileViewset**(***kwargs*)

get_queryset()

Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using *self.queryset*.

This method should always be used rather than accessing *self.queryset* directly, as *self.queryset* gets evaluated only once, and those results are cached for all subsequent requests.

You may want to override this if you need to provide different querysets depending on the incoming request.

(Eg. return a list of items that is specific to the user)

pagination_class

alias of *OptionalPageNumberPagination*

serializer_class

alias of `FileSerializer`

class kolibri.core.content.api.**InternalContentNodeMixin**

A mixin for all content node viewsets for internal use, whereas BaseContentNodeMixin is reused for public API endpoints also.

class kolibri.core.content.api.**OptionalContentNodePagination**

class kolibri.core.content.api.**OptionalPageNumberPagination**

Pagination class that allows for page number-style pagination, when requested. To activate, the *page_size* argument must be set. For example, to request the first 20 records: *?page_size=20&page=1*

class kolibri.core.content.api.**OptionalPagination**

class kolibri.core.content.api.**RemoteChannelViewSet**(***kwargs*)

dispatch(*request*, **args*, ***kwargs*)

.dispatch() is pretty much the same as Django's regular dispatch, but with extra hooks for startup, finalize, and exception handling.

list(*request*, **args*, ***kwargs*)

Gets metadata about all public channels on kolibri studio.

retrieve(*request*, *pk=None*)

Gets metadata about a channel through a token or channel id.

class `kolibri.core.content.api.`**RemoteViewSet**(**args*, ***kwargs*)

class `kolibri.core.content.api.`**UUIDInFilter**(**args*, ***kwargs*)

class `kolibri.core.content.api.`**UserContentNodeViewset**(**args*, ***kwargs*)

A content node viewset for filtering on user specific fields.

get_queryset()

Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using *self.queryset*.

This method should always be used rather than accessing *self.queryset* directly, as *self.queryset* gets evaluated only once, and those results are cached for all subsequent requests.

You may want to override this if you need to provide different querysets depending on the incoming request.

(Eg. return a list of items that is specific to the user)

pagination_class

alias of *`OptionalPagination`*

`kolibri.core.content.api.`**metadata_cache**(*view_func*, *cache_key_func=<function get_cache_key>*)

Decorator to apply an Etag sensitive page cache

`kolibri.core.content.api.`**no_cache_on_method**(*view_func*)

Decorator to disable caching for a particular method

API endpoints

request specific content:

```
>>> localhost:8000/api/content/<channel_id>/contentnode/<content_id>
```

search content:

```
>>> localhost:8000/api/content/<channel_id>/contentnode/?search=<search words>
```

request specific content with specified fields:

```
>>> localhost:8000/api/content/<channel_id>/contentnode/<content_id>/?fields=pk,title,
↪kind
```

request paginated contents

```
>>> localhost:8000/api/content/<channel_id>/contentnode/?page=6&page_size=10
```

request combines different usages

```
>>> localhost:8000/api/content/<channel_id>/contentnode/?fields=pk,title,kind,instance_
↪id,description,files&page=6&page_size=10&search=wh
```

2.6.2 Users, auth, and permissions module

This is a core module found in `kolibri/core/auth`.

Models

We have four main abstractions: Users, Collections, Memberships, and Roles.

Users represent people, like students in a school, teachers for a classroom, or volunteers setting up informal installations. A `FacilityUser` belongs to a particular facility, and has permissions only with respect to other data that is associated with that facility. `FacilityUser` accounts (like other facility data) may be synced across multiple devices.

Collections form a hierarchy, with Collections able to belong to other Collections. Collections are subdivided into several pre-defined levels (`Facility` > `Classroom` > `LearnerGroup`).

A `FacilityUser` (but not a `DeviceOwner`) can be marked as a member of a `Collection` through a `Membership` object. Being a member of a Collection also means being a member of all the Collections above that Collection in the hierarchy.

Another way in which a `FacilityUser` can be associated with a particular `Collection` is through a `Role` object, which grants the user a role with respect to the `Collection` and all the collections below it. A `Role` object also stores the "kind" of the role (currently, one of "admin" or "coach"), which affects what permissions the user gains through the `Role`.

`class kolibri.core.auth.models.`**`AbstractFacilityDataModel`**`(*args, **kwargs)`

Base model for Kolibri "Facility Data", which is data that is specific to a particular `Facility`, such as `FacilityUsers`, `Collections`, and other data associated with those users and collections.

Parameters

- **`id`** (*`UUIDField`*) – Id
- **`_morango_dirty_bit`** (*`BooleanField`*) – morango dirty bit
- **`_morango_source_id`** (*`CharField`*) – morango source id
- **`_morango_partition`** (*`CharField`*) – morango partition
- **`dataset_id`** (ForeignKey to ~) – Dataset

`cached_related_dataset_lookup``(related_obj_name)`

Attempt to get the dataset_id either from the cache or the actual related obj instance.

Parameters
`related_obj_name` – string representing the name of the related object on this model

Returns
the dataset_id associated with the related obj

`calculate_source_id``()`

Should return a string that uniquely defines the model instance or *None* for a random uuid.

`clean_fields``(*args, **kwargs)`

Cleans all fields and raises a ValidationError containing a dict of all validation errors if any occur.

`ensure_dataset`(**args*, ***kwargs*)

If no dataset has yet been specified, try to infer it. If a dataset has already been specified, to prevent inconsistencies, make sure it matches the inferred dataset, otherwise raise a `KolibriValidationError`. If we have no dataset and it can't be inferred, we raise a `KolibriValidationError` exception as well.

`full_clean`(**args*, ***kwargs*)

Calls clean_fields, clean, and validate_unique, on the model, and raises a `ValidationError` for any errors that occurred.

`infer_dataset`(**args*, ***kwargs*)

This method is used by *ensure_dataset* to "infer" which dataset should be associated with this instance. It should be overridden in any subclass of `AbstractFacilityDataModel`, to define a model-specific inference.

`save`(**args*, ***kwargs*)

Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

`class kolibri.core.auth.models.AdHocGroup`(**args*, ***kwargs*)

An `AdHocGroup` is a collection kind that can be used in an assignment to create a group that is specific to a single `Lesson` or `Exam`.

Parameters

- `id` (*UUIDField*) – Id
- `_morango_dirty_bit` (*BooleanField*) – morango dirty bit
- `_morango_source_id` (*CharField*) – morango source id
- `_morango_partition` (*CharField*) – morango partition
- `dataset_id` (ForeignKey to ~) – Dataset
- `name` (*CharField*) – Name
- `parent_id` (ForeignKey to ~) – Parent
- `kind` (*CharField*) – Kind

`exception DoesNotExist`

`exception MultipleObjectsReturned`

`classmethod deserialize`(*dict_model*)

Returns an unsaved class object based on the valid properties passed in.

`get_classroom`()

Gets the `AdHocGroup`'s parent `Classroom`.

Returns
A `Classroom` instance.

`save`(**args*, ***kwargs*)

Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

class kolibri.core.auth.models.**Classroom**(*id*, *_morango_dirty_bit*, *_morango_source_id*, *_morango_partition*, *dataset*, *name*, *parent*, *kind*)

Parameters

- **id** (*UUIDField*) – Id
- **_morango_dirty_bit** (*BooleanField*) – morango dirty bit
- **_morango_source_id** (*CharField*) – morango source id
- **_morango_partition** (*CharField*) – morango partition
- **dataset_id** (ForeignKey to ~) – Dataset
- **name** (*CharField*) – Name
- **parent_id** (ForeignKey to ~) – Parent
- **kind** (*CharField*) – Kind

exception **DoesNotExist**

exception **MultipleObjectsReturned**

get_facility()

Gets the Classroom's parent Facility.

Returns
A Facility instance.

get_individual_learners_group()

Returns a QuerySet of AdHocGroups.

:return A AdHocGroup QuerySet.

get_learner_groups()

Returns a QuerySet of LearnerGroups associated with this Classroom.

Returns
A LearnerGroup QuerySet.

save(**args*, ***kwargs*)

Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

class kolibri.core.auth.models.**Collection**(**args*, ***kwargs*)

Collections are hierarchical groups of FacilityUsers, used for grouping users and making decisions about permissions. FacilityUsers can have roles for one or more Collections, by way of obtaining Roles associated with those Collections. Collections can belong to other Collections, and user membership in a Collection is conferred through Memberships. Collections are subdivided into several pre-defined levels.

Parameters

- **id** (*UUIDField*) – Id
- **_morango_dirty_bit** (*BooleanField*) – morango dirty bit
- **_morango_source_id** (*CharField*) – morango source id
- **_morango_partition** (*CharField*) – morango partition
- **dataset_id** (ForeignKey to ~) – Dataset

- **name** (*CharField*) – Name
- **parent_id** (ForeignKey to ~) – Parent
- **kind** (*CharField*) – Kind

`exception DoesNotExist`

`exception MultipleObjectsReturned`

add_member(*user*)

Create a `Membership` associating the provided user with this `Collection`. If the `Membership` object already exists, just return that, without changing anything.

Parameters
user – The `FacilityUser` to add to this `Collection`.

Returns
The `Membership` object (possibly new) that associates the user with the `Collection`.

add_role(*user*, *role_kind*)

Create a `Role` associating the provided user with this collection, with the specified kind of role. If the Role object already exists, just return that, without changing anything.

Parameters

- **user** – The `FacilityUser` to associate with this `Collection`.
- **role_kind** – The kind of role to give the user with respect to this `Collection`.

Returns
The `Role` object (possibly new) that associates the user with the `Collection`.

calculate_partition()

Should return a string specifying this model instance's partition, using *self.ID_PLACEHOLDER* in place of its own ID, if needed.

clean_fields(**args*, ***kwargs*)

Cleans all fields and raises a ValidationError containing a dict of all validation errors if any occur.

get_admins()

Returns users who have the admin role for this immediate collection.

get_coaches()

Returns users who have the coach role for this immediate collection.

infer_dataset(**args*, ***kwargs*)

This method is used by *ensure_dataset* to "infer" which dataset should be associated with this instance. It should be overridden in any subclass of `AbstractFacilityDataModel`, to define a model-specific inference.

remove_member(*user*)

Remove any `Membership` objects associating the provided user with this `Collection`.

Parameters
user – The `FacilityUser` to remove from this `Collection`.

Returns
`True` if a `Membership` was removed, `False` if there was no matching `Membership` to remove.

`remove_role`(*user*, *role_kind*)

Remove any `Role` objects associating the provided user with this `Collection`, with the specified kind of role.

Parameters

- **user** – The `FacilityUser` to dissociate from this `Collection` (for the specific role kind).
- **role_kind** – The kind of role to remove from the user with respect to this `Collection`.

`save`(**args*, ***kwargs*)

Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

class kolibri.core.auth.models.**DatasetCache**

class kolibri.core.auth.models.**Facility**(*id*, *_morango_dirty_bit*, *_morango_source_id*, *_morango_partition*, *dataset*, *name*, *parent*, *kind*)

Parameters

- **id** (*UUIDField*) – Id
- **_morango_dirty_bit** (*BooleanField*) – morango dirty bit
- **_morango_source_id** (*CharField*) – morango source id
- **_morango_partition** (*CharField*) – morango partition
- **dataset_id** (ForeignKey to ~) – Dataset
- **name** (*CharField*) – Name
- **parent_id** (ForeignKey to ~) – Parent
- **kind** (*CharField*) – Kind

exception **DoesNotExist**

exception **MultipleObjectsReturned**

`ensure_dataset`(**args*, ***kwargs*)

If no dataset has yet been specified, try to infer it. If a dataset has already been specified, to prevent inconsistencies, make sure it matches the inferred dataset, otherwise raise a `KolibriValidationError`. If we have no dataset and it can't be inferred, we raise a `KolibriValidationError` exception as well.

`get_classrooms`()

Returns a QuerySet of Classrooms under this Facility.

Returns
A Classroom QuerySet.

`infer_dataset`(**args*, ***kwargs*)

This method is used by *ensure_dataset* to "infer" which dataset should be associated with this instance. It should be overridden in any subclass of `AbstractFacilityDataModel`, to define a model-specific inference.

`save`(**args*, ***kwargs*)

Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

`class kolibri.core.auth.models.FacilityDataSyncableModel(*args, **kwargs)`

Parameters

- **id** (*UUIDField*) – Id
- **_morango_dirty_bit** (*BooleanField*) – morango dirty bit
- **_morango_source_id** (*CharField*) – morango source id
- **_morango_partition** (*CharField*) – morango partition

`class kolibri.core.auth.models.FacilityDataset(*args, **kwargs)`

`FacilityDataset` stores high-level metadata and settings for a particular `Facility`. It is also the model that all models storing facility data (data that is associated with a particular facility, and that inherits from `AbstractFacilityDataModel`) foreign key onto, to indicate that they belong to this particular `Facility`.

Parameters

- **id** (*UUIDField*) – Id
- **_morango_dirty_bit** (*BooleanField*) – morango dirty bit
- **_morango_source_id** (*CharField*) – morango source id
- **_morango_partition** (*CharField*) – morango partition
- **description** (*TextField*) – Description
- **location** (*CharField*) – Location
- **preset** (*CharField*) – Preset
- **learner_can_edit_username** (*BooleanField*) – Learner can edit username
- **learner_can_edit_name** (*BooleanField*) – Learner can edit name
- **learner_can_edit_password** (*BooleanField*) – Learner can edit password
- **learner_can_sign_up** (*BooleanField*) – Learner can sign up
- **learner_can_delete_account** (*BooleanField*) – Learner can delete account
- **learner_can_login_with_no_password** (*BooleanField*) – Learner can login with no password
- **show_download_button_in_learn** (*BooleanField*) – Show download button in learn
- **extra_fields** (*JSONField*) – Extra fields
- **registered** (*BooleanField*) – Registered

`exception DoesNotExist`

`exception MultipleObjectsReturned`

`calculate_partition()`

Should return a string specifying this model instance's partition, using *self.ID_PLACEHOLDER* in place of its own ID, if needed.

`calculate_source_id()`

Should return a string that uniquely defines the model instance or *None* for a random uuid.

`full_facility_import`

Returns True if this user is a member of a facility that has been fully imported.

`save`(**args*, ***kwargs*)

Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

`class kolibri.core.auth.models.FacilityUser`(**args*, ***kwargs*)

`FacilityUser` is the fundamental object of the auth app. These users represent the main users, and can be associated with a hierarchy of `Collections` through `Memberships` and `Roles`, which then serve to help determine permissions.

Parameters

- **password** (*CharField*) – Password
- **last_login** (*DateTimeField*) – Last login
- **id** (*UUIDField*) – Id
- **_morango_dirty_bit** (*BooleanField*) – morango dirty bit
- **_morango_source_id** (*CharField*) – morango source id
- **_morango_partition** (*CharField*) – morango partition
- **dataset_id** (ForeignKey to ~) – Dataset
- **username** (*CharField*) – Required. 254 characters or fewer.
- **full_name** (*CharField*) – Full name
- **date_joined** (*DateTimeTzField*) – Date joined
- **facility_id** (ForeignKey to ~) – Facility
- **gender** (*CharField*) – Gender
- **birth_year** (*CharField*) – Birth year
- **id_number** (*CharField*) – Id number

`exception DoesNotExist`

`exception MultipleObjectsReturned`

`calculate_partition()`

Should return a string specifying this model instance's partition, using *self.ID_PLACEHOLDER* in place of its own ID, if needed.

`can_create_instance`(*obj*)

Checks whether this user (self) has permission to create a particular model instance (obj).

This method should be overridden by classes that inherit from `KolibriAbstractBaseUser`.

In general, unless an instance has already been initialized, this method should not be called directly; instead, it should be preferred to call `can_create`.

Parameters
`obj` – An (unsaved) instance of a Django model, to check permissions for.

Returns
`True` if this user should have permission to create the object, otherwise `False`.

Return type
bool

`can_delete`(*obj*)

Checks whether this user (self) has permission to delete a particular model instance (obj).

This method should be overridden by classes that inherit from KolibriAbstractBaseUser.

Parameters
`obj` – An instance of a Django model, to check permissions for.

Returns
`True` if this user should have permission to delete the object, otherwise `False`.

Return type
bool

`property can_manage_content`

bool(x) -> bool

Returns True when the argument x is true, False otherwise. The builtins True and False are the only two instances of the class bool. The class bool is a subclass of the class int, and cannot be subclassed.

`can_read`(*obj*)

Checks whether this user (self) has permission to read a particular model instance (obj).

This method should be overridden by classes that inherit from `KolibriAbstractBaseUser`.

Parameters
`obj` – An instance of a Django model, to check permissions for.

Returns
`True` if this user should have permission to read the object, otherwise `False`.

Return type
bool

`can_update`(*obj*)

Checks whether this user (self) has permission to update a particular model instance (obj).

This method should be overridden by classes that inherit from KolibriAbstractBaseUser.

Parameters
`obj` – An instance of a Django model, to check permissions for.

Returns
`True` if this user should have permission to update the object, otherwise `False`.

Return type
bool

`classmethod deserialize`(*dict_model*)

Returns an unsaved class object based on the valid properties passed in.

`filter_readable`(*queryset*)

Filters a queryset down to only the elements that this user should have permission to read.

Parameters
`queryset` – A `QuerySet` instance that the filtering should be applied to.

Returns
Filtered `QuerySet` including only elements that are readable by this user.

`full_facility_import`

Returns True if this user is a member of a facility that has been fully imported.

`has_role_for_collection`(*kinds*, *coll*)

Determine whether this user has (at least one of) the specified role kind(s) in relation to the specified `Collection`.

Parameters

- **kinds** (*string from kolibri.core.auth.constants.role_kinds.**) – The kind (or kinds) of role to check for, as a string or iterable.
- **coll** – The target `Collection` for which this user has the roles.

Returns

`True` if this user has the specified role kind with respect to the target `Collection`, otherwise `False`.

Return type

bool

`has_role_for_user`(*kinds*, *user*)

Determine whether this user has (at least one of) the specified role kind(s) in relation to the specified user.

Parameters

- **user** – The user that is the target of the role (for which this user has the roles).
- **kinds** (string from `kolibri.core.auth.constants.role_kinds.*`) – The kind (or kinds) of role to check for, as a string or iterable.

Returns

`True` if this user has the specified role kind with respect to the target user, otherwise `False`.

Return type

bool

`infer_dataset`(**args*, ***kwargs*)

This method is used by *ensure_dataset* to "infer" which dataset should be associated with this instance. It should be overridden in any subclass of `AbstractFacilityDataModel`, to define a model-specific inference.

`is_member_of`(*coll*)

Determine whether this user is a member of the specified `Collection`.

Parameters

coll – The `Collection` for which we are checking this user's membership.

Returns

`True` if this user is a member of the specified `Collection`, otherwise False.

Return type

bool

`property is_staff`

bool(x) -> bool

Returns True when the argument x is true, False otherwise. The builtins True and False are the only two instances of the class bool. The class bool is a subclass of the class int, and cannot be subclassed.

`property is_superuser`

bool(x) -> bool

Returns True when the argument x is true, False otherwise. The builtins True and False are the only two instances of the class bool. The class bool is a subclass of the class int, and cannot be subclassed.

`property session_data`

Data that is added to the session data at login and during session updates.

`class kolibri.core.auth.models.KolibriAbstractBaseUser(*args, **kwargs)`

Our custom user type, derived from `AbstractBaseUser` as described in the Django docs. Draws liberally from `django.contrib.auth.AbstractUser`, except we exclude some fields we don't care about, like email.

This model is an abstract model, and is inherited by `FacilityUser`.

Parameters

- **password** (*CharField*) – Password
- **last_login** (*DateTimeField*) – Last login
- **username** (*CharField*) – Required. 254 characters or fewer.
- **full_name** (*CharField*) – Full name
- **date_joined** (*DateTimeTzField*) – Date joined

`can_create(Model, data)`

Checks whether this user (self) has permission to create an instance of Model with the specified attributes (data).

This method defers to the `can_create_instance` method, and in most cases should not itself be overridden.

Parameters

- **Model** – A subclass of `django.db.models.Model`
- **data** – A `dict` of data to be used in creating an instance of the Model

Returns
`True` if this user should have permission to create an instance of Model with the specified data, else `False`.

Return type
bool

`can_create_instance(obj)`

Checks whether this user (self) has permission to create a particular model instance (obj).

This method should be overridden by classes that inherit from `KolibriAbstractBaseUser`.

In general, unless an instance has already been initialized, this method should not be called directly; instead, it should be preferred to call `can_create`.

Parameters
obj – An (unsaved) instance of a Django model, to check permissions for.

Returns
`True` if this user should have permission to create the object, otherwise `False`.

Return type
bool

`can_delete(obj)`

Checks whether this user (self) has permission to delete a particular model instance (obj).

This method should be overridden by classes that inherit from KolibriAbstractBaseUser.

Parameters
obj – An instance of a Django model, to check permissions for.

Returns
True if this user should have permission to delete the object, otherwise False.

Return type
bool

`can_read`(*obj*)

Checks whether this user (self) has permission to read a particular model instance (obj).

This method should be overridden by classes that inherit from `KolibriAbstractBaseUser`.

Parameters
obj – An instance of a Django model, to check permissions for.

Returns
True if this user should have permission to read the object, otherwise False.

Return type
bool

`can_update`(*obj*)

Checks whether this user (self) has permission to update a particular model instance (obj).

This method should be overridden by classes that inherit from KolibriAbstractBaseUser.

Parameters
obj – An instance of a Django model, to check permissions for.

Returns
True if this user should have permission to update the object, otherwise False.

Return type
bool

`filter_readable`(*queryset*)

Filters a queryset down to only the elements that this user should have permission to read.

Parameters
queryset – A `QuerySet` instance that the filtering should be applied to.

Returns
Filtered `QuerySet` including only elements that are readable by this user.

`has_role_for`(*kinds*, *obj*)

Helper function that defers to `has_role_for_user` or `has_role_for_collection` based on the type of object passed in.

`has_role_for_collection`(*kinds*, *coll*)

Determine whether this user has (at least one of) the specified role kind(s) in relation to the specified `Collection`.

Parameters

- **kinds** (*string from kolibri.core.auth.constants.role_kinds.**) – The kind (or kinds) of role to check for, as a string or iterable.
- **coll** – The target `Collection` for which this user has the roles.

Returns
True if this user has the specified role kind with respect to the target `Collection`, otherwise False.

Return type
bool

`has_role_for_user`(*kinds*, *user*)

Determine whether this user has (at least one of) the specified role kind(s) in relation to the specified user.

Parameters

- **user** – The user that is the target of the role (for which this user has the roles).
- **kinds** (string from `kolibri.core.auth.constants.role_kinds.*`) – The kind (or kinds) of role to check for, as a string or iterable.

Returns
`True` if this user has the specified role kind with respect to the target user, otherwise `False`.

Return type
bool

`is_member_of`(*coll*)

Determine whether this user is a member of the specified `Collection`.

Parameters
coll – The `Collection` for which we are checking this user's membership.

Returns
`True` if this user is a member of the specified `Collection`, otherwise False.

Return type
bool

`property session_data`

Data that is added to the session data at login and during session updates.

`class kolibri.core.auth.models.KolibriAnonymousUser`

Custom anonymous user that also exposes the same interface as KolibriAbstractBaseUser, for consistency.

Parameters

- **password** (*CharField*) – Password
- **last_login** (*DateTimeField*) – Last login
- **full_name** (*CharField*) – Full name
- **date_joined** (*DateTimeTzField*) – Date joined

`can_create_instance`(*obj*)

Checks whether this user (self) has permission to create a particular model instance (obj).

This method should be overridden by classes that inherit from `KolibriAbstractBaseUser`.

In general, unless an instance has already been initialized, this method should not be called directly; instead, it should be preferred to call `can_create`.

Parameters
obj – An (unsaved) instance of a Django model, to check permissions for.

Returns
`True` if this user should have permission to create the object, otherwise `False`.

Return type
bool

`can_delete`(*obj*)

Checks whether this user (self) has permission to delete a particular model instance (obj).

This method should be overridden by classes that inherit from KolibriAbstractBaseUser.

Parameters
`obj` – An instance of a Django model, to check permissions for.

Returns
`True` if this user should have permission to delete the object, otherwise `False`.

Return type
bool

`can_read`(*obj*)

Checks whether this user (self) has permission to read a particular model instance (obj).

This method should be overridden by classes that inherit from `KolibriAbstractBaseUser`.

Parameters
`obj` – An instance of a Django model, to check permissions for.

Returns
`True` if this user should have permission to read the object, otherwise `False`.

Return type
bool

`can_update`(*obj*)

Checks whether this user (self) has permission to update a particular model instance (obj).

This method should be overridden by classes that inherit from KolibriAbstractBaseUser.

Parameters
`obj` – An instance of a Django model, to check permissions for.

Returns
`True` if this user should have permission to update the object, otherwise `False`.

Return type
bool

`filter_readable`(*queryset*)

Filters a queryset down to only the elements that this user should have permission to read.

Parameters
`queryset` – A `QuerySet` instance that the filtering should be applied to.

Returns
Filtered `QuerySet` including only elements that are readable by this user.

`has_role_for_collection`(*kinds*, *coll*)

Determine whether this user has (at least one of) the specified role kind(s) in relation to the specified `Collection`.

Parameters

- `kinds` (*string from kolibri.core.auth.constants.role_kinds.**) – The kind (or kinds) of role to check for, as a string or iterable.
- `coll` – The target `Collection` for which this user has the roles.

Returns
`True` if this user has the specified role kind with respect to the target `Collection`, otherwise `False`.

Return type
bool

`has_role_for_user`(*kinds*, *user*)

Determine whether this user has (at least one of) the specified role kind(s) in relation to the specified user.

Parameters

- **`user`** – The user that is the target of the role (for which this user has the roles).
- **`kinds`** (string from `kolibri.core.auth.constants.role_kinds.*`) – The kind (or kinds) of role to check for, as a string or iterable.

Returns
`True` if this user has the specified role kind with respect to the target user, otherwise `False`.

Return type
bool

`is_member_of`(*coll*)

Determine whether this user is a member of the specified `Collection`.

Parameters
`coll` – The `Collection` for which we are checking this user's membership.

Returns
`True` if this user is a member of the specified `Collection`, otherwise False.

Return type
bool

`property session_data`

Data that is added to the session data at login and during session updates.

`class kolibri.core.auth.models.LearnerGroup`(*id*, *_morango_dirty_bit*, *_morango_source_id*, *_morango_partition*, *dataset*, *name*, *parent*, *kind*)

Parameters

- **`id`** (*`UUIDField`*) – Id
- **`_morango_dirty_bit`** (*`BooleanField`*) – morango dirty bit
- **`_morango_source_id`** (*`CharField`*) – morango source id
- **`_morango_partition`** (*`CharField`*) – morango partition
- **`dataset_id`** (ForeignKey to ~) – Dataset
- **`name`** (*`CharField`*) – Name
- **`parent_id`** (ForeignKey to ~) – Parent
- **`kind`** (*`CharField`*) – Kind

`exception DoesNotExist`

`exception MultipleObjectsReturned`

get_classroom()

Gets the LearnerGroup's parent Classroom.

Returns

A Classroom instance.

save(**args*, ***kwargs*)

Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

class kolibri.core.auth.models.**Membership**(**args*, ***kwargs*)

A FacilityUser can be marked as a member of a Collection through a Membership object. Being a member of a Collection also means being a member of all the Collections above that Collection in the tree (i.e. if you are a member of a LearnerGroup, you are also a member of the Classroom that contains that LearnerGroup, and of the Facility that contains that Classroom).

Parameters

- **id** (*UUIDField*) – Id
- **_morango_dirty_bit** (*BooleanField*) – morango dirty bit
- **_morango_source_id** (*CharField*) – morango source id
- **_morango_partition** (*CharField*) – morango partition
- **dataset_id** (ForeignKey to ~) – Dataset
- **user_id** (ForeignKey to ~) – User
- **collection_id** (TreeForeignKey to ~) – Collection

exception **DoesNotExist**

exception **MultipleObjectsReturned**

calculate_partition()

Should return a string specifying this model instance's partition, using *self.ID_PLACEHOLDER* in place of its own ID, if needed.

calculate_source_id()

Should return a string that uniquely defines the model instance or *None* for a random uuid.

infer_dataset(**args*, ***kwargs*)

This method is used by *ensure_dataset* to "infer" which dataset should be associated with this instance. It should be overridden in any subclass of AbstractFacilityDataModel, to define a model-specific inference.

save(**args*, ***kwargs*)

Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

class kolibri.core.auth.models.**Role**(**args*, ***kwargs*)

A FacilityUser can have a role for a particular Collection through a Role object, which also stores the "kind" of the Role (currently, one of "admin" or "coach"). Having a role for a Collection also implies having that role for all sub-collections of that Collection (i.e. all the Collections below it in the tree).

Parameters

- `id` (*UUIDField*) – Id
- `_morango_dirty_bit` (*BooleanField*) – morango dirty bit
- `_morango_source_id` (*CharField*) – morango source id
- `_morango_partition` (*CharField*) – morango partition
- `dataset_id` (ForeignKey to ~) – Dataset
- `user_id` (ForeignKey to ~) – User
- `collection_id` (TreeForeignKey to ~) – Collection
- `kind` (*CharField*) – Kind

exception DoesNotExist

exception MultipleObjectsReturned

calculate_partition()

Should return a string specifying this model instance's partition, using *self.ID_PLACEHOLDER* in place of its own ID, if needed.

calculate_source_id()

Should return a string that uniquely defines the model instance or *None* for a random uuid.

infer_dataset(**args*, ***kwargs*)

This method is used by *ensure_dataset* to "infer" which dataset should be associated with this instance. It should be overridden in any subclass of `AbstractFacilityDataModel`, to define a model-specific inference.

save(**args*, ***kwargs*)

Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

Concepts and Definitions

Facility

All user data (accounts, logs, ratings, etc) in Kolibri are associated with a particular "Facility". A Facility is a grouping of users who are physically co-located, and who generally access Kolibri from the same server on a local network, for example in a school, library, or community center. Collectively, all the data associated with a particular Facility are referred to as a "Facility Dataset".

Users

Kolibri's users are instances of the `FacilityUser` model, which derives from Django's `AbstractBaseUser`. A user `FacilityUser` is associated with a particular `Facility`, and the user's account and data may be synchronized across multiple devices. A `FacilityUser` may be made into a superuser, with permissions to modify any data on her own device. However, normally a `FacilityUser` only has permissions for some subset of data from their own Facility Dataset (as determined in part by the roles they possess; see below).

Collections

Collections are hierarchical groups of users, used for grouping users and making decisions about permissions. Users can have roles for one or more Collections, by way of obtaining Roles associated with those Collections. Collections can belong to other Collections, and user membership in a collection is conferred through Membership. Collections are subdivided into several pre-defined levels: Facility, Classroom, and LearnerGroup, as illustrated here:

In this illustration, Facility X contains two Classrooms, Class A and Class B. Class A contains two LearnerGroups, Group Q and Group R.

Membership

A `FacilityUser` can be marked as a member of a `Collection` through a `Membership` object. Being a member of a Collection requires first being a member of all the Collections above that Collection in the hierarchy. Thus, in the illustration below, Alice is directly associated with Group Q through a `Membership` object, which makes her a member of Group Q. As Group Q is contained within Class A, which is contained within Facility X, must also be a member of both those collections.

Note also that a `FacilityUser` is always implicitly a member of the `Facility` with which it is associated, even if it does not have any `Membership` objects.

Roles

Another way in which a `FacilityUser` can be associated with a particular `Collection` is through a `Role` object, which grants the user a role with respect to the `Collection` and all the collections below it. A `Role` object stores the "kind" of the role (currently, one of "admin", "coach", or "assignable coach"), which affects what permissions the user gains through the `Role`.

To illustrate, consider the example in the following figure:

The figure shows a Role object linking Bob with Class A, and the Role is marked with kind "coach", which we can informally read as "Bob is a coach for Class A". We consider user roles to be "downward-transitive" (meaning if you have a role for a collection, you also have that role for descendents of that collection). Thus, in our example, we can say that "Bob is also a coach for Group Q". Furthermore, as Alice is a member of Class A, we can say that "Bob is a coach for Alice".

A user can be assigned certain roles for different collection types:

- `Facility` collections: admin, coach, or assignable coach roles
- `Classroom` collections: coach roles
- `LearnerGroup` and `AdHocGroup` collections: no roles

Role-Based Permissions

As a lot of Facility Data in Kolibri is associated with a particular `FacilityUser`, for many objects we can concisely define a requesting user's permissions in terms of his or her roles for the object's associated User. For example, if a `ContentLog` represents a particular `FacilityUser`'s interaction with a piece of content, we might decide that another `FacilityUser` can view the `ContentLog` if she is a coach (has the coach role) for the user. In our scenario above, this would mean that Bob would have read permissions for a `ContentLog` for which "user=Alice", by virtue of having the coach role for Alice.

Some data may not be related to a particular user, but rather with a `Collection` (e.g. the `Collection` object itself, settings for a `Collection`, or content assignments for a `Collection`). Permissions for these objects can be defined in terms of the role the requesting User has with respect to the object's associated Collection. So, for example, we might allow Bob to assign content to Class A on the basis of him having the "coach" role for Class A.

Permission Levels

As we are constructing a RESTful API for accessing data within Kolibri, the core actions for which we need to define permissions are the CRUD operations (Create, Read, Update, Delete). As Create, Update, and Delete permissions often go hand in hand, we can collectively refer to them as "Write Permissions".

Implementation details

Collections

A `Collection` is implemented as a Django model that inherits from django-mptt's MPTTModel, which allows for efficient traversal and querying of the collection hierarchy. For convenience, the specific types of collections – `Facility`, `Classroom`, and `LearnerGroup` – are implemented as _proxy models of the main `Collection` model. There is a `kind` field on `Collection` that allows us to distinguish between these types, and the `ModelManager` for the proxy models returns only instances of the matching kind.

From a `Collection` instance, you can traverse upwards in the tree with the `parent` field, and downwards via the `children` field (which is a reverse `RelatedManager` for the `parent` field):

```
>>> my_classroom.parent
<Collection: "Facility X" (facility)>

>>> my_facility.children.all()
[<Collection: "Class A" (classroom)>, <Collection: "Class B" (classroom)>]
```

Note that the above methods (which are provided by `MPTTModel`) return generic `Collection` instances, rather than specific proxy model instances. To retrieve parents and children as appropriate proxy models, use the helper methods provided on the proxy models, e.g.:

```
>>> my_classroom.get_facility()
<Facility: Facility X>

>>> my_facility.get_classrooms()
[<Classroom: Class A>, <Classroom: Class B>]
```

Facility and FacilityDataset

The `Facility` model (a proxy model for `Collection`, as described above) is special in that it has no `parent`; it is the root of a tree. A `Facility` model instance, and all other Facility Data associated with the `Facility` and its `FacilityUsers`, inherits from `AbstractFacilityDataModel`, which has a `dataset` field that foreign keys onto a common `FacilityDataset` instance. This makes it easy to check, for purposes of permissions or filtering data for synchronization, which instances are part of a particular Facility Dataset. The `dataset` field is automatically set during the `save` method, by calling the `infer_dataset` method, which must be overridden in every subclass of `AbstractFacilityDataModel` to return the dataset to associate with that instance.

Efficient hierarchy calculations

In order to make decisions about whether a user has a certain permission for an object, we need an efficient way to retrieve the set of roles the user has in relation to that object. This involves traversing the Role table, Collection hierarchy, and possibly the Membership table. Because we require explicit representation of membership at each level in the hierarchy, we can rely solely on the transitivity of role permissions in order to determine the role that a user has with respect to some data.

Managing Roles and Memberships

User and `Collection` models have various helper methods for retrieving and modifying roles and memberships:

- To get all the members of a collection (including those of its descendant collections), use `Collection.get_members()`.
- To add or remove roles/memberships, use the `add_role`, `remove_role`, `add_member`, and `remove_member` methods of `Collection` (or the additional convenience methods, such as `add_admin`, that exist on the proxy models).
- To check whether a user is a member of a `Collection`, use `KolibriAbstractBaseUser.is_member_of`
- To check whether a user has a particular kind of role for a collection or another user, use the `has_role_for_collection` and `has_role_for_user` methods of `KolibriAbstractBaseUser`.
- To list all role kinds a user has for a collection or another user, use the `get_roles_for_collection` and `get_roles_for_user` methods of `KolibriAbstractBaseUser`.

Encoding Permission Rules

We need to associate a particular set of rules with each model, to specify the permissions that users should have in relation to instances of that model. While not all models have the same rules, there are some broad categories of models that do share the same rules (e.g. ContentInteractionLog, ContentSummaryLog, and UserSessionLog – collectively, "User Log Data"). Hence, it is useful to encapsulate a permissions "class" that can be reused across models, and extended (through inheritance) if slightly different behavior is needed. These classes of permissions are defined as Python classes that inherit from kolibri.auth.permissions.base.BasePermissions, which defines the following overridable methods:

- The following four Boolean (True/False) permission checks, corresponding to the "CRUD" operations: - `user_can_create_object` - `user_can_read_object` - `user_can_update_object` - `user_can_delete_object`
- The queryset-filtering `readable_by_user_filter` method, which takes in a user and returns a Django `Q` object that can be used to filter to just objects that should be readable by the user.

Associating permissions with models

A model is associated with a particular permissions class through a "permissions" attribute defined on the top level of the model class, referencing an instance of a Permissions class (a class that subclasses `BasePermissions`). For example, to specify that a model `ContentSummaryLog` should draw its permissions rules from the `UserLogPermissions` class, modify the model definition as follows:

```
class ContentSummaryLog(models.Model):

    permissions = UserLogPermissions()

    <remainder of model definition>
```

Specifying role-based permissions

Defining a custom Permissions class and overriding its methods allows for arbitrary logic to be used in defining the rules governing the permissions, but many cases can be covered by more constrained rule specifications. In particular, the rules for many models can be specified in terms of the role- based permissions system described above. A built-in subclass of `BasePermissions`, called `RoleBasedPermissions`, makes this easy. Creating an instance of `RoleBasedPermissions` involves passing in the following parameters:

- Tuples of role kinds that should be granted each of the CRUD permissions, encoded in the following parameters: `can_be_created_by`, `can_be_read_by`, `can_be_updated_by`, `can_be_deleted_by`.
- The `target_field` parameter that determines the "target" object for the role-checking; this should be the name of a field on the model that foreign keys either onto a `FacilityUser` or a `Collection`. If the model we're checking permissions for is itself the target, then `target_field` may be `"."`.

An example, showing that read permissions should be granted to a coach or admin for the user referred to by the model's "user" field. Similarly, write permissions should only be available to an admin for the user:

```
class UserLog(models.Model):

    permissions = RoleBasedPermissions(
        target_field="user",
        can_be_created_by=(role_kinds.ADMIN,),
        can_be_read_by=(role_kinds.COACH, role_kinds.ADMIN),
        can_be_updated_by=(role_kinds.ADMIN,),
        can_be_deleted_by=(role_kinds.ADMIN,),
    )

    <remainder of model definition>
```

Built-in permissions classes

Some common rules are encapsulated by the permissions classes in `kolibri.auth.permissions.general`. These include:

- `IsOwn`: only allows access to the object if the object belongs to the requesting user (in other words, if the object has a specific field, `field_name`, that foreign keys onto the user)
- `IsFromSameFacility`: only allows access to object if user is associated with the same facility as the object
- `IsSelf`: only allows access to the object if the object *is* the user

A general pattern with these provided classes is to allow an argument called `read_only`, which means that rather than allowing both write (create, update, delete) and read permissions, they will only grant read permission. So, for example, `IsFromSameFacility(read_only=True)` will allow any user from the same facility to read the model, but not to write to it, whereas `IsFromSameFacility(read_only=False)` or `IsFromSameFacility()` would allow both.

Combining permissions classes

In many cases, it may be necessary to combine multiple permission classes together to define the ruleset that you want. This can be done using the Boolean operators `|` (OR) and `&` (AND). So, for example, `IsOwn(field_name="user") | IsSelf()` would allow access to the model if either the model has a foreign key named "user" that points to the user, or the model is *itself* the user model. Combining two permission classes with `&`, on the other hand, means both classes must return `True` for a permission to be granted. Note that permissions classes combined in this way still support the `readable_by_user_filter` method, returning a queryset that is either the union (for `|`) or intersection (`&`) of the querysets that were returned by each of the permissions classes.

Checking permissions

Checking whether a user has permission to perform a CRUD operation on an object involves calling the appropriate methods on the `KolibriAbstractBaseUser` (`FacilityUser` or `DeviceOwner`) instance. For instance, to check whether request.user has delete permission for `ContentSummaryLog` instance log_obj, you could do:

```
if request.user.can_delete(log_obj):
    log_obj.delete()
```

Checking whether a user can create an object is slightly different, as you may not yet have an instance of the model. Instead, pass in the model class and a `dict` of the data that you want to create it with:

```
data = {"user": request.user, "content_id": "qq123"}
if request.user.can_create(ContentSummaryLog, data):
    ContentSummaryLog.objects.create(**data)
```

To efficiently filter a queryset so that it only includes records that the user should have permission to read (to make sure you're not sending them data they shouldn't be able to access), use the `filter_readable` method:

```
all_results = ContentSummaryLog.objects.filter(content_id="qq123")
permitted_results = request.user.filter_readable(all_results)
```

Note that for the `DeviceOwner` model, these methods will simply return `True` (or unfiltered querysets), as device owners are considered superusers. For the `FacilityUser` model, they defer to the permissions encoded in the `permission` object on the model class.

Using Kolibri permissions with Django REST Framework

There are two classes that make it simple to leverage the permissions system described above within a Django REST Framework `ViewSet`, to restrict permissions appropriately on API endpoints, based on the currently logged-in user.

`KolibriAuthPermissions` is a subclass of `rest_framework.permissions.BasePermission` that defers to our `KolibriAbstractBaseUser` permissions interface methods for determining which object-level permissions to grant to the current user:

- Permissions for 'POST' are based on `request.user.can_create`

- Permissions for 'GET', 'OPTIONS' and 'HEAD' are based on `request.user.can_read` (Note that adding `KolibriAuthPermissions` only checks object-level permissions, and does not filter queries made against a list view; see `KolibriAuthPermissionsFilter` below)
- Permissions for 'PUT' and 'PATCH' are based on `request.user.can_update`
- Permissions for 'DELETE' are based on `request.user.can_delete`

`KolibriAuthPermissions` is a subclass of `rest_framework.filters.BaseFilterBackend` that filters list views to include only records for which the current user has read permissions. This only applies to 'GET' requests.

For example, to use the Kolibri permissions system to restrict permissions for an API endpoint providing access to a `ContentLog` model, you would do the following:

```python
from kolibri.auth.api import KolibriAuthPermissions, KolibriAuthPermissionsFilter

class FacilityViewSet(viewsets.ModelViewSet):
    permission_classes = (KolibriAuthPermissions,)
    filter_backends = (KolibriAuthPermissionsFilter,)
    queryset = ContentLog.objects.all()
    serializer_class = ContentLogSerializer
```

2.6.3 User log module

This is a core module found in `kolibri/core/logger`.

Models

This app provides the core functionality for tracking user engagement with content and the Kolibri app.

It stores:

- details of users' interactions with content
- summaries of those interactions
- interactions with the software in general

Eventually, it may also store user feedback on the content and the software.

class `kolibri.core.logger.models.`**`AttemptLog`**(**args*, ***kwargs*)

This model provides a summary of a user's interactions with a question in a content node. (Think of it like a ContentNodeAttemptLog to distinguish it from ExamAttemptLog and BaseAttemptLog)

Parameters

- **id** (*UUIDField*) – Id
- **_morango_dirty_bit** (*BooleanField*) – morango dirty bit
- **_morango_source_id** (*CharField*) – morango source id
- **_morango_partition** (*CharField*) – morango partition
- **dataset_id** (ForeignKey to ~) – Dataset
- **item** (*CharField*) – Item
- **start_timestamp** (*DateTimeTzField*) – Start timestamp
- **end_timestamp** (*DateTimeTzField*) – End timestamp

- **completion_timestamp** (*DateTimeTzField*) – Completion timestamp
- **time_spent** (*FloatField*) – (in seconds)
- **complete** (*BooleanField*) – Complete
- **correct** (*FloatField*) – Correct
- **hinted** (*BooleanField*) – Hinted
- **answer** (*JSONField*) – Answer
- **simple_answer** (*CharField*) – Simple answer
- **interaction_history** (*JSONField*) – Interaction history
- **user_id** (ForeignKey to ~) – User
- **error** (*BooleanField*) – Error
- **masterylog_id** (ForeignKey to ~) – Masterylog
- **sessionlog_id** (ForeignKey to ~) – Sessionlog

`exception DoesNotExist`

`exception MultipleObjectsReturned`

`infer_dataset`(**args*, ***kwargs*)

This method is used by *ensure_dataset* to "infer" which dataset should be associated with this instance. It should be overridden in any subclass of `AbstractFacilityDataModel`, to define a model-specific inference.

`class kolibri.core.logger.models.BaseAttemptLog`(**args*, ***kwargs*)

This is an abstract model that provides a summary of a user's interactions with a particular item/question in an assessment/exercise/exam

Parameters

- **id** (*UUIDField*) – Id
- **_morango_dirty_bit** (*BooleanField*) – morango dirty bit
- **_morango_source_id** (*CharField*) – morango source id
- **_morango_partition** (*CharField*) – morango partition
- **dataset_id** (ForeignKey to ~) – Dataset
- **item** (*CharField*) – Item
- **start_timestamp** (*DateTimeTzField*) – Start timestamp
- **end_timestamp** (*DateTimeTzField*) – End timestamp
- **completion_timestamp** (*DateTimeTzField*) – Completion timestamp
- **time_spent** (*FloatField*) – (in seconds)
- **complete** (*BooleanField*) – Complete
- **correct** (*FloatField*) – Correct
- **hinted** (*BooleanField*) – Hinted
- **answer** (*JSONField*) – Answer
- **simple_answer** (*CharField*) – Simple answer

- **interaction_history** (*JSONField*) – Interaction history
- **user_id** (ForeignKey to ~) – User
- **error** (*BooleanField*) – Error

class kolibri.core.logger.models.**BaseLogModel**(**args*, ***kwargs*)

Parameters

- **id** (*UUIDField*) – Id
- **_morango_dirty_bit** (*BooleanField*) – morango dirty bit
- **_morango_source_id** (*CharField*) – morango source id
- **_morango_partition** (*CharField*) – morango partition
- **dataset_id** (ForeignKey to ~) – Dataset

calculate_partition()

Should return a string specifying this model instance's partition, using *self.ID_PLACEHOLDER* in place of its own ID, if needed.

infer_dataset(**args*, ***kwargs*)

This method is used by *ensure_dataset* to "infer" which dataset should be associated with this instance. It should be overridden in any subclass of `AbstractFacilityDataModel`, to define a model-specific inference.

class kolibri.core.logger.models.**BaseLogQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

filter_by_content_ids(*content_ids*, *content_id_lookup='content_id'*)

Filter a set of logs by content_id, using content_ids from the provided list or queryset.

filter_by_topic(*topic*, *content_id_lookup='content_id'*)

Filter a set of logs by content_id, using content_ids from all descendants of specified topic.

class kolibri.core.logger.models.**ContentSessionLog**(**args*, ***kwargs*)

This model provides a record of interactions with a content item within a single visit to that content page.

Parameters

- **id** (*UUIDField*) – Id
- **_morango_dirty_bit** (*BooleanField*) – morango dirty bit
- **_morango_source_id** (*CharField*) – morango source id
- **_morango_partition** (*CharField*) – morango partition
- **dataset_id** (ForeignKey to ~) – Dataset
- **user_id** (ForeignKey to ~) – User
- **content_id** (*UUIDField*) – Content id
- **visitor_id** (*UUIDField*) – Visitor id
- **channel_id** (*UUIDField*) – Channel id
- **start_timestamp** (*DateTimeTzField*) – Start timestamp
- **end_timestamp** (*DateTimeTzField*) – End timestamp
- **time_spent** (*FloatField*) – (in seconds)

- **progress** (*FloatField*) – Progress
- **kind** (*CharField*) – Kind
- **extra_fields** (*JSONField*) – Extra fields

exception DoesNotExist

exception MultipleObjectsReturned

save(**args*, ***kwargs*)

Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

class kolibri.core.logger.models.**ContentSummaryLog**(**args*, ***kwargs*)

This model provides an aggregate summary of all recorded interactions a user has had with a content item over time.

Parameters

- **id** (*UUIDField*) – Id
- **_morango_dirty_bit** (*BooleanField*) – morango dirty bit
- **_morango_source_id** (*CharField*) – morango source id
- **_morango_partition** (*CharField*) – morango partition
- **dataset_id** (ForeignKey to ~) – Dataset
- **user_id** (ForeignKey to ~) – User
- **content_id** (*UUIDField*) – Content id
- **channel_id** (*UUIDField*) – Channel id
- **start_timestamp** (*DateTimeTzField*) – Start timestamp
- **end_timestamp** (*DateTimeTzField*) – End timestamp
- **completion_timestamp** (*DateTimeTzField*) – Completion timestamp
- **time_spent** (*FloatField*) – (in seconds)
- **progress** (*FloatField*) – Progress
- **kind** (*CharField*) – Kind
- **extra_fields** (*JSONField*) – Extra fields

exception DoesNotExist

exception MultipleObjectsReturned

calculate_source_id()

Should return a string that uniquely defines the model instance or *None* for a random uuid.

save(**args*, ***kwargs*)

Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

`class kolibri.core.logger.models.`**`ExamAttemptLog`**`(`*`*args`*`, `*`**kwargs`*`)`

This model provides a summary of a user's interactions with a question in an exam

Parameters

- **`id`** (*`UUIDField`*) – Id
- **`_morango_dirty_bit`** (*`BooleanField`*) – morango dirty bit
- **`_morango_source_id`** (*`CharField`*) – morango source id
- **`_morango_partition`** (*`CharField`*) – morango partition
- **`dataset_id`** (ForeignKey to ~) – Dataset
- **`item`** (*`CharField`*) – Item
- **`start_timestamp`** (*`DateTimeTzField`*) – Start timestamp
- **`end_timestamp`** (*`DateTimeTzField`*) – End timestamp
- **`completion_timestamp`** (*`DateTimeTzField`*) – Completion timestamp
- **`time_spent`** (*`FloatField`*) – (in seconds)
- **`complete`** (*`BooleanField`*) – Complete
- **`correct`** (*`FloatField`*) – Correct
- **`hinted`** (*`BooleanField`*) – Hinted
- **`answer`** (*`JSONField`*) – Answer
- **`simple_answer`** (*`CharField`*) – Simple answer
- **`interaction_history`** (*`JSONField`*) – Interaction history
- **`user_id`** (ForeignKey to ~) – User
- **`error`** (*`BooleanField`*) – Error
- **`examlog_id`** (ForeignKey to ~) – Examlog
- **`content_id`** (*`UUIDField`*) – Content id

`exception` **`DoesNotExist`**

`exception` **`MultipleObjectsReturned`**

`calculate_partition``()`

Should return a string specifying this model instance's partition, using *self.ID_PLACEHOLDER* in place of its own ID, if needed.

`infer_dataset``(`*`*args`*`, `*`**kwargs`*`)`

This method is used by *ensure_dataset* to "infer" which dataset should be associated with this instance. It should be overridden in any subclass of `AbstractFacilityDataModel`, to define a model-specific inference.

`class kolibri.core.logger.models.`**`ExamLog`**`(`*`*args`*`, `*`**kwargs`*`)`

This model provides a summary of a user's interactions with an exam, and serves as an aggregation point for individual attempts on questions in that exam.

Parameters

- **`id`** (*`UUIDField`*) – Id
- **`_morango_dirty_bit`** (*`BooleanField`*) – morango dirty bit

- `_morango_source_id` (*CharField*) – morango source id
- `_morango_partition` (*CharField*) – morango partition
- `dataset_id` (ForeignKey to ~) – Dataset
- `exam_id` (ForeignKey to ~) – Exam
- `user_id` (ForeignKey to ~) – User
- `closed` (*BooleanField*) – Closed
- `completion_timestamp` (*DateTimeTzField*) – Completion timestamp

`exception DoesNotExist`

`exception MultipleObjectsReturned`

`calculate_partition()`

Should return a string specifying this model instance's partition, using *self.ID_PLACEHOLDER* in place of its own ID, if needed.

`calculate_source_id()`

Should return a string that uniquely defines the model instance or *None* for a random uuid.

`class kolibri.core.logger.models.GenerateCSVLogRequest(*args, **kwargs)`

This model provides a record of a user's request to generate session and summary log files

Parameters

- `id` (*AutoField*) – Id
- `facility_id` (ForeignKey to ~) – Facility
- `selected_start_date` (*DateTimeTzField*) – Selected start date
- `selected_end_date` (*DateTimeTzField*) – Selected end date
- `date_requested` (*DateTimeTzField*) – Date requested
- `log_type` (*CharField*) – Log type

`exception DoesNotExist`

`exception MultipleObjectsReturned`

`class kolibri.core.logger.models.MasteryLog(*args, **kwargs)`

This model provides a summary of a user's engagement with an assessment within a mastery level

Parameters

- `id` (*UUIDField*) – Id
- `_morango_dirty_bit` (*BooleanField*) – morango dirty bit
- `_morango_source_id` (*CharField*) – morango source id
- `_morango_partition` (*CharField*) – morango partition
- `dataset_id` (ForeignKey to ~) – Dataset
- `user_id` (ForeignKey to ~) – User
- `summarylog_id` (ForeignKey to ~) – Summarylog
- `mastery_criterion` (*JSONField*) – Mastery criterion
- `start_timestamp` (*DateTimeTzField*) – Start timestamp

- `end_timestamp` (*DateTimeTzField*) – End timestamp
- `completion_timestamp` (*DateTimeTzField*) – Completion timestamp
- `mastery_level` (*IntegerField*) – Mastery level
- `complete` (*BooleanField*) – Complete
- `time_spent` (*FloatField*) – (in seconds)

`exception DoesNotExist`

`exception MultipleObjectsReturned`

`calculate_source_id()`

Should return a string that uniquely defines the model instance or *None* for a random uuid.

`infer_dataset`(**args*, ***kwargs*)

This method is used by *ensure_dataset* to "infer" which dataset should be associated with this instance. It should be overridden in any subclass of `AbstractFacilityDataModel`, to define a model-specific inference.

`class kolibri.core.logger.models.UserSessionLog`(**args*, ***kwargs*)

This model provides a record of a user session in Kolibri.

Parameters

- `id` (*UUIDField*) – Id
- `_morango_dirty_bit` (*BooleanField*) – morango dirty bit
- `_morango_source_id` (*CharField*) – morango source id
- `_morango_partition` (*CharField*) – morango partition
- `dataset_id` (ForeignKey to ~) – Dataset
- `user_id` (ForeignKey to ~) – User
- `channels` (*TextField*) – Channels
- `start_timestamp` (*DateTimeTzField*) – Start timestamp
- `last_interaction_timestamp` (*DateTimeTzField*) – Last interaction timestamp
- `pages` (*TextField*) – Pages
- `device_info` (*CharField*) – Device info

`exception DoesNotExist`

`exception MultipleObjectsReturned`

`classmethod update_log`(*user*, *os_info=None*, *browser_info=None*)

Update the current UserSessionLog for a particular user.

ua_parser never defaults the setting of os.family and user_agent.family It uses the value 'other' whenever the values are not recognized or the parsing fails. The code depends on this behaviour.

Concepts and definitions

All logs use MorangoDB to synchronize their data across devices.

Content session logs

These models provide a high-level record that a user interacted with a content item for some contiguous period of time. This generally corresponds to the time between when a user navigates to the content and when they navigate away from it.

Specifically, it encodes the channel that the content was in, the id of the content, which user it was, and time-related date. It may also encode additional data that is specific to the particular content type in a JSON blob.

As a typical use case, a ContentSessionLog object might be used to record high-level information about how long a user engaged with an exercise or a video during a single viewing. More granular interaction information about what happened within the session may be stored in another model such as an attempt log, below.

Content summary logs

These models provide an aggregate summary of all interactions of a user with a content item. It encodes the channel that the content was in, the id of the content, and information such as cummulative time spent. It may also encode additional data specific to the particular content type in a JSON blob.

As a typical use case, a ContentSummaryLog object might be used to provide summary data about the state of completion of a particular exercise, video, or other content.

When a new ContentSessionLog is saved, the associated ContentSummaryLog is updated at the same time. This means that the ContentSummaryLog acts as an aggregation layer for the progress of a particular piece of content.

To implement this, a content viewer app would define the aggregation function that summarizes session logs into the summary log. While this could happen in the frontend, it would probably be more efficient for this to happen on the server side.

These logs will use MorangoDB to synchronize their data across devices - in the case where two summary logs from different devices conflict, then the aggregation logic would be applied across all interaction logs to create a consolidated summary log.

Attempt logs

These models store granular information about a user's interactions with individual components of some kind of assessment. There are two subclasses: AttemptLog which tracks attempts at questions within exercises, and ExamAttemptLog which tracks attempts at questions within exams.

Exam logs

These models provide information about when users took exams.

User session logs

These models provide a record of a user session in Kolibri. It encodes the channels interacted with, the length of time engaged, and the specific pages visited.

Concretely, a UserSessionLog records which pages a user visits and how long the user is logged in for.

Implementation details

Permissions

See *Encoding Permission Rules*.

2.6.4 Kolibri plugin architecture

The behavior of Kolibri can be extended using plugins. The following is a guide to developing plugins.

Enabling and disabling plugins

Non-core plugins can be enabled or disabled using the `kolibri plugin` commands.

How plugins work

From a user's perspective, plugins are enabled and disabled through the command line interface or through a UI. Users can also configure a plugin's behavior through the main Kolibri interface.

From a developer's perspective, plugins are wrappers around Django applications, listed in `ACTIVE_PLUGINS` on the kolibri config object. They are initialized before Django's app registry is initialized and then their relevant Django apps are added to the `INSTALLED_APPS` of kolibri.

Loading a plugin

In general, a plugin should **never** modify internals of Kolibri or other plugins without using the hooks API or normal conventional Django scenarios.

Note: Each app in `ACTIVE_PLUGINS` in the kolibri conf is searched for the special `kolibri_plugin` module.

Everything that a plugin does is expected to be defined through `<myapp>/kolibri_plugin.py`.

Kolibri Hooks API

What are hooks

Hooks are classes that define *something* that happens at one or more places where the hook is looked for and applied. It means that you can "hook into a component" in Kolibri and have it do a predefined and parameterized *thing*. For instance, Kolibri could ask all its plugins who wants to add something to the user settings panel, and its then up to the plugins to inherit from that specific hook and feed back the parameters that the hook definition expects.

The consequences of a hook being applied can happen anywhere in Kolibri. Each hook is defined through a class inheriting from `KolibriHook`. But how the inheritor of that class deals with plugins using it, is entirely up to each specific implementation and can be applied in templates, views, middleware - basically everywhere!

That's why you should consult the class definition and documentation of the hook you are adding plugin functionality with.

We have two different types of hooks:

Abstract hooks
: Are definitions of hooks that are implemented by *implementing hooks*. These hooks are Python abstract base classes, and can use the @abstractproperty and @abstractmethod decorators from the abc module in order to define which properties and methods their descendant registered hooks should implement.

Registered hooks
: Are concrete hooks that inherit from abstract hooks, thus embodying the definitions of the abstract hook into a specific case. If the abstract parent hook has any abstract properties or methods, the hook being registered as a descendant must implement those properties and methods, or an error will occur.

So what's "a hook"?
: Simply referring to "a hook" is okay, it can be ambiguous on purpose. For instance, in the example, we talk about "a navigation hook". So we both mean the abstract definition of the navigation hook and everything that is registered for the navigation.

Where can I find hooks?

All Kolibri core applications and plugins alike should *by convention* define their abstract hooks inside `<myapp>/hooks.py`. Thus, to see which hooks a Kolibri component exposes, you can refer to its `hooks` module.

Note: Defining abstract hooks in `<myapp>/hooks.py` isn't mandatory, but *loading* a concrete hook in `<myapp>/kolibri_plugin.py` is.

Warning: Do not define abstract and registered hooks in the same module. Or to put it in other words: Always put registered hooks in `<myapp>/kolibri_plugin.py`. The registered hooks will only be initialized for use by the Kolibri plugin registry if they are registered inside the kolibri_plugin.py module for the plugin.

In which order are hooks used/applied?

This is entirely up to the registering class. By default, hooks are applied in the same order that the registered hook gets registered! While it could be the case that plugins could be enabled in a certain order to get a specific ordering of hooks - it is best not to depend on this behaviour as it could result in brittleness.

An example of a plugin using a hook

Note: The example shows a NavigationHook which is simplified for the sake of readability. The actual implementation in Kolibri will differ.

Example implementation

Here is an example of how to use a hook in `myplugin.kolibri_plugin.py`:

```
from kolibri.core.hooks import NavigationHook
from kolibri.plugins.hooks import register_hook

@register_hook
class MyPluginNavItem(NavigationHook):
    bundle_id = "side_nav"
```

The decorator `@register_hook` signals that the wrapped class is intended to be registered against any abstract KolibriHook descendants that it inherits from. In this case, the hook being registered inherits from NavigationHook, so any hook registered will be available on the `NavigationHook.registered_hooks` property.

Here is the definition of the abstract NavigationHook in kolibri.core.hooks:

```
from kolibri.core.webpack.hooks import WebpackBundleHook
from kolibri.plugins.hooks import define_hook


@define_hook
class NavigationHook(WebpackBundleHook):
    pass
```

As can be seen from above, to define an abstract hook, instead of using the `@register_hook` decorator, the `@define_hook` decorator is used instead, to signal that this instance of inheritance is not intended to register anything against the parent `WebpackBundleHook`. However, because of the inheritance relationship, any hook registered against `NavigationHook` (like our example registered hook above), will also be registered against the `WebpackBundleHook`, so we should expect to see our plugin's nav item listed in the `WebpackBundleHook.registered_hooks` property as well as in the `NavigationHook.registered_hooks` property.

Usage of the hook

The hook can then be used to collect all the information from the hooks, as per this usage of the NavigationHook in kolibri/core/kolibri_plugin.py:

```
from kolibri.core.hooks import NavigationHook

...
    def navigation_tags(self):
        return [
            hook.render_to_page_load_sync_html()
            for hook in NavigationHook.registered_hooks
        ]
```

Each registered hook is iterated over and its appropriate HTML for rendering into the frontend are returned. When iterating over registered_hooks the returned objects are each instances of the hook classes that were registered.

Warning: Do not load registered hook classes outside of a plugin's kolibri_plugin. Either define them there directly or import the modules that define them. Hook classes should all be seen at load time, and placing that logic in kolibri_plugin guarantees that things are registered correctly.

Defining a plugin

A plugin must have a Python module inside it called kolibri_plugin, inside this there must be an object subclassed from KolibriPluginBase - here is a minimal example:

```
from kolibri.plugins import KolibriPluginBase

class ExamplePlugin(KolibriPluginBase):
    pass
```

The Python module that contains this kolibri_plugin module can now be enabled and disabled as a plugin. If the module path for the plugin is kolibri.plugins.example_plugin then it could be enabled by:

```
kolibri plugin enable kolibri.plugins.example_plugin
```

The above command can be passed multiple plugin names to enable at once. If Kolibri is running, it needs to be restarted for the change to take effect.

Similarly, to disable the plugin the following command can be used:

```
kolibri plugin disable kolibri.plugins.example_plugin
```

To exactly set the currently enabled plugins (disabling all other plugins, and enabling the ones specified) you can do this:

```
kolibri plugin apply kolibri.plugins.learn kolibri.plugins.default_theme
```

This will disable all other plugins and only enable kolibri.plugins.learn and *kolibri.plugins.default_theme*`.

Creating a plugin

Plugins can be standalone Django apps in their own right, meaning they can define templates, models, new urls, and views just like any other app. Any activated plugin is added to the `INSTALLED_APPS` setting of Django, so any models, templates, or templatetags defined in the conventional way for Django inside an app will work inside of a Kolibri plugin.

In addition, Kolibri exposes some additional functionality that allows for the core URLs, Django settings, and Kolibri options to be extended by a plugin. These are set

```
class ExamplePlugin(KolibriPluginBase):
    untranslated_view_urls = "api_urls"
    translated_view_urls = "urls"
    options = "options"
    settings = "settings"
```

These are all path references to modules within the plugin itself, so options would be accessible on the Python module path as `kolibri.plugins.example_plugin.options`.

`untranslated_view_urls`, `translated_view_urls` should both be standard Django urls modules in the plugin that expose a `urlpatterns` variable - the first will be mounted as API urls - with no language prefixing, the second will be mounted with language prefixing and will be assumed to contain language specific content.

`settings` should be a module containing Django settings that should be added to the Kolibri settings. This should not be used to override existing settings (and an error will be thrown if it is used in this way), but rather as a way for plugins to add additional settings to the Django settings. This is particularly useful when a plugin is being used to wrap a Django library that requires its own settings to define its behaviour - this module can be used to add these extra settings in a way that is encapsulated to the plugin.

`options` should be a module that exposes a variable `options_spec` which defines Kolibri options specific to this plugin. For more information on how to configure these, see the base Kolibri options specification in *kolibri/utils/options.py*. These values can then be set either by environment variables or by editing the `options.ini` file in the `KOLIBRI_HOME` directory. These options values can also be used inside the settings module above, to provide customization of plugin specific behaviour. These options cannot clash with existing Kolibri options defined in `kolibri.utils.options`, except in order to change the default value of a Kolibri option - attempting to change any other value of a core Kolibri option will result in a Runtime Error.

A very common use case for plugins is to implement a single page app or other Kolibri module for adding frontend functionality using Kolibri Javascript code. Each of these Javascript bundles are defined in the *kolibri_plugin.py* file by subclassing the `WebpackBundleHook` class to define each frontend Kolibri module. This allows a webpack built Javascript bundle to be cross-referenced and loaded into Kolibri. For more information on developing frontend code for Kolibri please see *Frontend architecture*.

Learn plugin example

View the source to learn more!

2.6.5 Kolibri backend tasks system

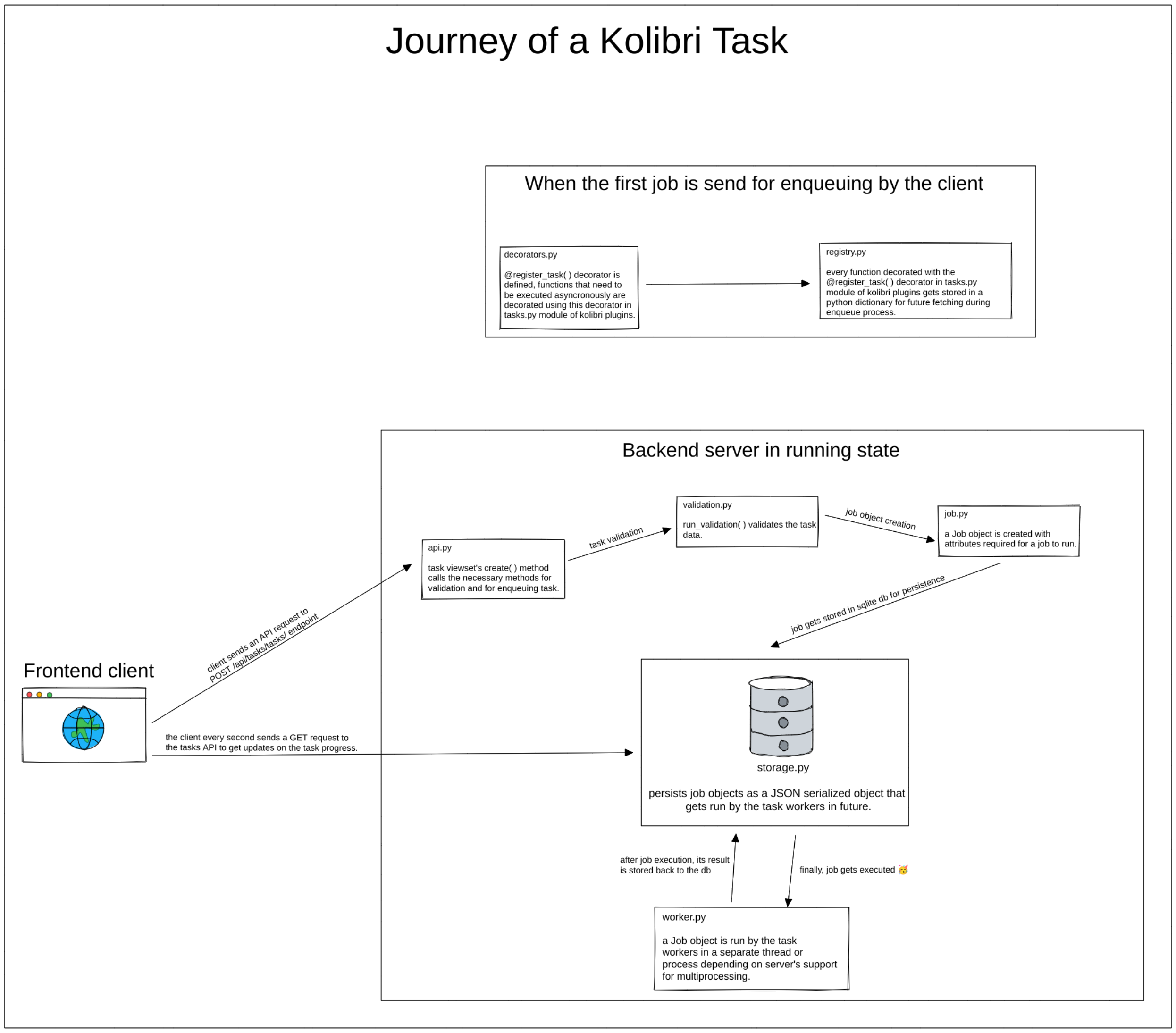
Kolibri plugins and Django apps can use the backend tasks system to run time consuming processes asynchronously outside of the HTTP request-response cycle. This frees the HTTP server for client use.

The kolibri task system is implemented as a core Django app on `kolibri.core.tasks`.

Kolibri backend tasks system flow diagram

The following diagram explains how a task travels from the frontend client to the different parts of the backend task system. It aims to give a high level understanding of the backend tasks system.

You should download the following image to be able to zoom it in your image viewer. You can download by right clicking on following image and select "save image as" option.


Journey of a Kolibri Task
When the first job is send for enqueuing by the client
decorators.py
@register_task( ) decorator is defined, functions that need to be executed asyncronously are decorated using this decorator in tasks.py module of kolibri plugins.
registry.py
every function decorated with the @register_task( ) decorator in tasks.py module of kolibri plugins gets stored in a python dictionary for future fetching during enqueue process.
Backend server in running state
api.py
task viewset's create( ) method calls the necessary methods for validation and for enqueuing task.
task validation
validation.py
run_validation( ) validates the task data.
job object creation
job.py
a Job object is created with attributes required for a job to run.
job gets stored in sqlite db for persistence
Frontend client
client sends an API request to POST /api/tasks/tasks/ endpoint
the client every second sends a GET request to the tasks API to get updates on the task progress.
storage.py
persists job objects as a JSON serialized object that gets run by the task workers in future.
after job execution, its result is stored back to the db
finally, job gets executed 🥳
worker.py
a Job object is run by the task workers in a separate thread or process depending on server's support for multiprocessing.

Defining tasks via `@register_task` decorator

When Kolibri starts, the task backend searches for a module named `tasks.py` in every Django app and imports them, which results in the registration of tasks defined within.

When the `tasks.py` module gets run, functions decorated with `@register_task` decorator gets registered in the `JobRegistry`.

The `@register_task` decorator is implemented in `kolibri.core.tasks.decorators`. It registers the decorated function as a task to the task backend system.

Kolibri plugins and kolibri's Django apps can pass several arguments to the decorator based on their needs.

- `job_id (string)`: job's id.
- `queue (string)`: queue in which the job should be enqueued.
- `validator (callable)`: validator for the job. The details of how validation works is described later.
- `priority (5 or 10)`: priority of the job. It can be `"HIGH"` (5) or `"REGULAR"``(``10`). `"REGULAR"` priority is for tasks that can wait for some time before it actually starts executing. Tasks that are shown to users in the task manager should use `"REGULAR"` priority. `"HIGH"` priority is used for tasks that need execution as soon as possible. These are often short-lived tasks that temporarily block user interaction using a loading animation (for example, tasks that import channel metadata before browsing).
- `cancellable (boolean)`: whether the job is cancellable or not.
- `track_progress (boolean)`: whether to track progress of the job or not.
- `permission_classes (Django Rest Framework's permission classes)`: a list of DRF permissions user should have in order to enqueue the job.

Example usage

The below code sample shows how we can use the `@register_task` decorator to register a function as a task.

We will refer to below sample code in the later sections also.

```python
from rest_framework import serializers

from kolibri.core.tasks.decorators import register_task
from kolibri.core.tasks.job import Priority
from kolibri.core.tasks.permissions import IsSuperAdmin
from kolibri.core.tasks.validation import JobValidator

class AddValidator(JobValidator):
    a = serializers.IntegerField()
    b = serializers.IntegerField()

    def validate(self, data):
        if data['a'] + data['b'] > 100:
            raise serializers.ValidationError("Sum of a and b should be less than 100")
        job_data = super(AddValidator, self).validate(data)
        job_data["extra_metadata"].update({"user": "kolibri"})
        return job_data

@register_task(job_id="02", queue="maths", validator=AddValidator, priority=Priority.
↪HIGH, cancellable=False, track_progress=True, permission_classes=[IsSuperAdmin])
```

(continues on next page)

(continued from previous page)

```
def add(a, b):
    return a + b
```

Enqueuing tasks via the `POST /api/tasks/tasks/` API endpoint

To enqueue a task that is registered with the `@register_task` decorator we use `POST /api/tasks/tasks/` endpoint method defined on `kolibri.core.tasks.api.BaseViewSet.create`.

The request payload for `POST /api/tasks/tasks/` API endpoint should have:

- `"type" (required)` having value as string representing the dotted path to the function registered via the `@register_task` decorator.
- other key value pairs as per client's choice.

A valid request payload can be:

```
{
  "type": "kolibri.core.content.tasks.add",
  "a": 45,
  "b": 49
}
```

A successful response looks like this:

```
{
  "status": "QUEUED",
  "exception": "",
  "traceback": "",
  "percentage": 0,
  "id": 1,
  "cancellable": False,
  "clearable": False,
}
```

When we send a request to `POST /api/tasks/tasks/` API endpoint, first, we validate the payload. The request payload **must** have a `"type"` parameter as string and the user should have the permissions mentioned on the `permission_classes` argument of decorator. If the user has permissions then we proceed.

Then, we check whether the registered task function has a validator associated with it or not. If it has a validator, it gets run. The return value of the validator must be a dictionary that conforms to the function signature of the Job object. The dictionary returned by the validator is passed to a Job object to be enqueued. By default, any key value pairs in the request object that are registered as input fields on the validator will be passed to the function as kwargs. If no fields are defined on the validator, or no validator is registered, then the function will receive no arguments.

We can add `extra_metadata` in the returning dictionary of validator function to set extra metadata for the job. If the validator raises any exception, our API endpoint method will re raise it. Keep in mind that `extra_metadata` is **not** passed to the task function as an argument.

For example, if the validator returns a dictionary like:

```
{
  "kwargs" : {
      "a": req_data["a"],
      "b": req_data["b"],
```

(continues on next page)

(continued from previous page)

```
  },
  "extra_metadata": {
    "user": "kolibri"
  }
}
```

The task function will receive `a` and `b` as keyword arguments.

Once the validator is run and no exceptions are raised, we enqueue the `"task"` function. Depending on the `priority` of the task, the worker pool will run the task.

We can also enqueue tasks in bulk. The frontend just have to send a list of tasks, like:

```
[
  {
    "type": "kolibri.core.content.tasks.add",
    "a": 45,
    "b": 49
  },
  {
    "type": "kolibri.core.content.tasks.add",
    "a": 20,
    "b": 52
  },
  {
    "type": "kolibri.core.content.tasks.subtract",
    "a": 10,
    "b": 59
  }
]
```

The tasks backend will iterate over this list and it will perform the operations of a task on every `"type"` function – checking permissions, running the validator and enqueuing the task function.

The response will be a list of enqueued jobs like:

```
[
  {
    "status": "QUEUED",
    "exception": "",
    "traceback": "",
    "percentage": 0,
    "id": "e05ad2b3-eae8-4e29-9f00-b16accfee3e2",
    "cancellable": False,
    "clearable": False,
  },
  {
    "status": "QUEUED",
    "exception": "",
    "traceback": "",
    "percentage": 0,
    "id": "329f0fe0-bfb0-47f8-9e33-0468ef9805e5",
    "cancellable": False,
    "clearable": False,
```

(continues on next page)

(continued from previous page)

```
  },
  {
    "status": "QUEUED",
    "exception": "",
    "traceback": "",
    "percentage": 0,
    "id": "895a881a-6825-4be0-8bd4-0e8db40ab324",
    "cancellable": False,
    "clearable": False,
  }
]
```

However, if any task fails validation, all tasks in the request will be rejected. Validation happens prior to enqueuing, so tasks will not be partially started in the bulk case.

2.6.6 Distribution build pipeline

The Kolibri Package build pipeline looks like this:

```
                        Git release branch
                                |
                                |
                               / \
                              /   \
Python dist, online dependencies   \
   `python setup.py bdist_wheel`    \
               /                     \
              /                 Python dist, bundled dependencies
       Upload to PyPi         `python setup.py bdist_wheel --static`
      Installable with                  \
    `pip install kolibri`                \
                                    Upload to PyPi
                                    Installable with
                              `pip install kolibri-static`
                                /           |          \
                               /            |           \
                         Windows         Android       Debian
                        installer          APK        installer
```

Make targets

- To build a wheel file, run `make dist`
- To build a pex file, run `make pex` after `make dist`
- Builds for additional platforms are triggered from buildkite based on *.buildkite/pipeline.yml*

More on version numbers

Note: The content below is pulled from the docstring of the `kolibri.utils.version` module.

We follow semantic versioning 2.0.0 according to semver.org but for Python distributions and in the internal string representation in Python, you will find a PEP-440 flavor.

- `1.1.0` (Semver) = `1.1.0` (PEP-440).
- `1.0.0-alpha1` (Semver) = `1.0.0a1` (PEP-440).

Here's how version numbers are generated:

- `kolibri.__version__` is automatically set, runtime environments use it to decide the version of Kolibri as a string. This is especially something that PyPi and setuptools use.
- `kolibri.VERSION` is a tuple containing major, minor, and patch version information, it's set in `kolibri/__init__.py`
- `kolibri/VERSION` is a file containing the exact version of Kolibri for a distributed environment - when it exists, as long as its major, minor, and patch versions are compatible with `kolibri.VERSION` then it is used as the version. If these versions do not match, an AssertionError will be thrown.
- `git describe --tags` is a command run to fetch tag information from a git checkout with the Kolibri code. The information is used to validate the major components of `kolibri.VERSION` and to add a suffix (if needed). This information is stored permanently in `kolibri/VERSION` before shipping any built asset by calling `make writeversion` during `make dist` etc.

This table shows examples of kolibri.VERSION and git data used to generate a specific version:

Release type	`kolibri.VERSION`	Git data	Examples
Final	(1, 2, 3)	Final tag: e.g. v1.2.3	1.2.3
dev release (alpha0)	(1, 2, 3)	timestamp of latest commit + hash	1.2.3.dev0+git.123.f1234567
alpha1+	(1, 2, 3)	Alpha tag: e.g. v1.2.3a1	Clean head: 1.2.3a1, 4 changes since tag: 1.2.3a1.dev0+git.4.f1234567
beta1+	(1, 2, 3)	Beta tag: e.g. v1.2.3b1	Clean head: 1.2.3b1, 5 changes since tag: 1.2.3b1.dev0+git.5.f1234567
rc1+ (release candidate)	(1, 2, 3)	RC tag: e.g. v1.2.3rc1	Clean head: 1.2.3rc1, Changes since tag: 1.2.3rc1.dev0+git.f1234567

Built assets: `kolibri/VERSION` is auto-generated with `make writeversion` during the build process. The file is read in preference to git data in order to prioritize swift version resolution in an installed environment.

Release order example 1.2.3 release:

- `VERSION = (1, 2, 3)` throughout the development phase, this results in a lot of `1.2.3.dev0+git1234abcd` with no need for git tags.
- `VERSION = (1, 2, 3)` for the first alpha release, a git tag v1.2.3a0 is made.

Warning: Do not import anything from the rest of Kolibri in this module, it's crucial that it can be loaded without the settings/configuration/django stack.

If you wish to use `version.py` in another project, raw-copy the contents of this file. You cannot import this module in other distributed package's `__init__`, because `setup.py` cannot depend on the import of other packages at install-time (which is when the version is generated and stored).

Warning: Tagging is known to break after rebasing, so in case you rebase a branch after tagging it, delete the tag and add it again. Basically, `git describe --tags` detects the closest tag, but after a rebase, its concept of distance is misguided.

2.6.7 Upgrading

Warning: These instructions are under development

Upgrade paths

Kolibri can be automatically upgraded forwards. For instance, you can upgrade from `0.1->0.2` and `0.1->0.7`. For changes in the database schema we use Django database migrations to manage these changes and make updates on upgrade. When changes are made to model schema during development, then these migrations can be generated by executing `kolibri manage makemigrations`.

This will trigger the Django management command that will inspect the current model schema, the current migrations, and generate new migrations to cover any discrepancies. For some migrations, manual editing will be required to ensure compatibility with Python 2 and 3 - this normally happens for Django Model fields that take a `choices` keyword argument, where the choices are strings. The strings should have no prefix (`u` or `b`) and the migration should contain `from __future__ import unicode_literals` as an import.

We also use the upgrade functionality triggered during the CLI initialization to copy in new copies of static files that are used in the frontend app. These upgrades are only triggered for a subset of our CLI commands - start, services, manage, shell. Ones that ultimately start Django processes. In examining `cli.py` - those commands that are instantiated using the `cls=KolibriDjangoCommand` keyword argument will trigger this update behaviour.

As well as database migrations, there are also sometimes additional fixes that are put into Kolibri in order to facilitate moving between versions. This may be for bug fixing or efficiency purposes. These are sometimes carried out outside of migrations in order to leverage the full Kolibri code base, which can be restricted inside the contexts of Django data migrations.

In order to implement these upgrades, a decorator is available in `kolibri.core.upgrade`, `version_upgrade`. An toy example is shown below.

```python
import logging
from kolibri.core.upgrade import version_upgrade

logger = logging.getLogger(__name__)

@version_upgrade(old_version="<0.6.4", new_version=">=1.0.0")
def big_leap_upgrade():
    logger.warning("You've just upgraded from a very old version to a very new version!")
```

If placed into a file named `upgrade.py` either in a core app that is part of the `INSTALLED_APPS` Django setting, or is in an activated Kolibri plugin, this upgrade will be picked up and run any time an upgrade happens from a Kolibri version older than `0.6.4` to a Kolibri version equal to or newer than `1.0.0`.

2.6.8 Learning facility data syncing

Kolibri features the capability to synchronize facility data between Kolibri instances, which supports its hybrid, distance, and offline learning applications. Each Kolibri instance is able to sync partitioned datasets (a learning facility) in a peer-to-peer manner. To enable this functionality, Learning Equality developed a pure python database replication engine for Django, called Morango (repository, documentation). Morango has several important features:

- A certificate-based authentication system to protect privacy and integrity of data
- A change-tracking system to support calculation of differences between databases across low-bandwidth connections
- A set of constructs to support data partitioning

The auth module found in `kolibri/core/auth` contains most of the Kolibri specific code that powers this feature.

The `sync` management command

The `sync` management command inside the auth module uses Morango's tooling to manage facility syncs between itself and other Kolibri devices, as well as Kolibri Data Portal.

Integrating with a sync

There are two primary ways in which Kolibri plugins may integrate with a sync:

a) Adding a Morango sync operation, which may execute at any stage of a sync

b) Adding a hook functions, which may execute before or after a sync transfer

When considering these two options, you should consider the following:

a) If the integration is vital to features being developed, a Morango sync operation should be implemented. This brings the benefit of providing integrity with the corresponding synced data, such that both are atomically applied.

b) If the integration isn't vital and is fail-tolerant, a sync hook function is the ideal choice as their execution does not impede the sync in any way.

Morango sync operations

A Morango operation is can be injected into any stage of a sync transfer, which include the following: `INITIALIZING`, `SERIALIZING`, `QUEUING`, `TRANSFERRING`, `DEQUEUING`, `DESERIALIZING`, and `CLEANUP`. Morango uses Django settings to manage which operations occur during each stage, but Kolibri builds upon by specifying one `KolibriSyncOperation` (code) that invokes each operation registered by Kolibri plugins.

Here's an example of a Kolibri plugin adding a custom sync operations:

```python
from morango.sync.operations import LocalOperation

from kolibri.core.auth.hooks import FacilityDataSyncHook
from kolibri.core.auth.sync_operations import KolibriSyncOperationMixin
from kolibri.plugins.hooks import register_hook


class CustomCleanupOperation(KolibriSyncOperationMixin, LocalOperation):
    priority = 5

```

(continues on next page)

(continued from previous page)

```
    def handle_initial(self, context):
        """
        :type context: morango.sync.context.LocalSessionContext
        """
        # CUSTOM CODE HERE

@register_hook
class MyPluginSyncHook(FacilityDataSyncHook):
    cleanup_operations = [CustomCleanupOperation()]
```

Sync hook functions

Sync hook functions utilize the same class as above, `FacilityDataSyncHook`, but instead may defined `pre_transfer` or `post_transfer` methods.

Here's an example of a Kolibri plugin adding a custom hooks:

```
from kolibri.core.auth.hooks import FacilityDataSyncHook
from kolibri.plugins.hooks import register_hook

@register_hook
class MyPluginSyncHook(FacilityDataSyncHook):
    def pre_transfer(
        self,
        dataset_id,
        local_is_single_user,
        remote_is_single_user,
        single_user_id,
        context,
    ):
        """
        Invoked before the initialization stage
        :type dataset_id: str
        :type local_is_single_user: bool
        :type remote_is_single_user: bool
        :type single_user_id: str
        :type context: morango.sync.context.LocalSessionContext
        """
        # CUSTOM CODE HERE

    def post_transfer(
        self,
        dataset_id,
        local_is_single_user,
        remote_is_single_user,
        single_user_id,
        context,
    ):
        """
        Invoked at after the cleanup stage
        :type dataset_id: str
```

(continues on next page)

(continued from previous page)

```
        :type local_is_single_user: bool
        :type remote_is_single_user: bool
        :type single_user_id: str
        :type context: morango.sync.context.LocalSessionContext
        """
        # CUSTOM CODE HERE
```

2.7 Server/client communication

2.7.1 Server API

The Kolibri server represents data as Django Models. These models are defined in `models.py` files, which can be found in the folders of the different Django apps/plugins.

In Django, Model data are usually exposed to users through webpages that are generated by the Django server. To make the data available to the Kolibri client, which is a single-page app, the Models are exposed as JSON data through a REST API provided by the Django REST Framework (DRF). It's important to remark that Kolibri limits the content types the DRF api support to be only `application/json` or `multipart/form-data`. This limitation is set at `kolibri/core/negotiation.py`.

In the `api.py` files, Django REST framework ViewSets are defined which describe how the data is made available through the REST API. Each ViewSet also requires a defined Serializer, which describes the way in which the data from the Django model is serialized into JSON and returned through the REST API. Additionally, optional filters can be applied to the ViewSet which will allow queries to filter by particular features of the data (for example by a field) or by more complex constraints, such as which group the user associated with the data belongs to. Permissions can be applied to a ViewSet, allowing the API to implicitly restrict the data that is returned, based on the currently logged in user.

The default DRF use of Serializers for serialization to JSON tends to encourage the adoption of non-performant patterns of code, particularly ones that use DRF Serializer Method Fields, which then do further queries on a per model basis inside the method. This can easily result in the N + 1 query problem, whereby the number of queries required scales with the number of entities requested in the query. To make this and other performance issues less of a concern, we have created a special ValuesViewset class defined at `kolibri/core/api.py`, which relies on queryset annotation and post query processing in order to serialize all the relevant data. In addition, to prevent the inflation of full Django models into memory, all queries are done with a *values* call resulting in lower memory overhead.

Finally, in the `api_urls.py` file, the ViewSets are given a name (through the `basename` keyword argument), which sets a particular URL namespace, which is then registered and exposed when the Django server runs. Sometimes, a more complex URL scheme is used, as in the content core app, where every query is required to be prefixed by a channel id (hence the <`channel_id`> placeholder in that route's regex pattern)

Listing 1: api_urls.py

```
router = routers.SimpleRouter()
router.register("channel", ChannelMetadataViewSet, basename="channel")

router.register(r"contentnode", ContentNodeViewset, basename="contentnode")
router.register(
    r"contentnode_tree", ContentNodeTreeViewset, basename="contentnode_tree"
)
router.register(
    r"contentnode_search", ContentNodeSearchViewset, basename="contentnode_search"
```

(continues on next page)

(continued from previous page)

```
)
router.register(r"file", FileViewset, basename="file")
router.register(
    r"contentnodeprogress", ContentNodeProgressViewset, basename="contentnodeprogress"
)
router.register(
    r"contentnode_granular",
    ContentNodeGranularViewset,
    basename="contentnode_granular",
)
router.register(r"remotechannel", RemoteChannelViewSet, basename="remotechannel")

urlpatterns = [url(r"^", include(router.urls))]
```

To explore the server REST APIs, visit */api_explorer/* on the Kolibri server while running with developer settings.

2.7.2 Client resource layer

To access this REST API in the frontend Javascript code, an abstraction layer has been written to reduce the complexity of inferring URLs, caching resources, and saving data back to the server.

Resources

In order to access a particular REST API endpoint, a Javascript Resource has to be defined, an example is shown here

Listing 2: channel.js

```
import { Resource } from 'kolibri.lib.apiResource';

export default new Resource({
  name: 'channel',
});
```

Here, the `name` property is set to `'channel'` in order to match the `basename` assigned to the `/channel` endpoint in *api_urls.py*.

If this resource is part of the core app, it can be added to a global registry of resources inside `kolibri/core/assets/src/api-resources/index.js`. Otherwise, it can be imported as needed, such as in the coach reports module.

Models

The instantiated Resource can then be queried for client side representations of particular information. For a representation of a single server side Django model, we can request a Model from the Resource, using `fetchModel`

```
// corresponds to resource address /api/content/contentnode/<id>
const modelPromise = ContentNodeResource.fetchModel(id);
```

The argument is the database id (primary key) for the model.

We now have a reference for the promise to fetch data fron the server. To read the data, we must resolve the promise to an object representing the data

```
modelPromise.then((data) => {
  logging.info('This is the model data: ', data);
});
```

The `fetchModel` method returns a `Promise` which resolves when the data has been successfully retrieved. This may have been due to a round trip call to the REST API, or, if the data has already been previously returned, then it will skip the call to the REST API and return a cached copy of the data.

If it is important to get data that has not been cached, you can call the `fetchModel` method with a force parameter

```
ContentNodeResource.fetchModel(id, { force: true }).then((data) => {
  logging.info('This is definitely the most up to date model data: ', data);
});
```

Collections

For particular views on a data table (which could range from 'show me everything' to 'show me all content nodes with titles starting with "p"') - Collections are used. Collections are a cached view onto the data table, which are populated by Models - so if a Model that has previously been fetched from the server by a Collection is requested from `getModel`, it is already cachced.

```
// corresponds to /api/content/contentnode/?popular=1
const collectionPromise = ContentNodeResource.fetchCollection({ getParams: { popular: 1 }
↪ });
```

The getParams option defines the GET parameters that are used to define the filters to be applied to the data and hence the subset of the data that the Collection represents.

We now have a reference for the promise to fetch data fron the server. To read the data, we must resolve the promise to an array of the returned data objects

```
collectionPromise.then((dataArray) => {
  logging.info('This is the model data: ', dataArray);
});
```

The `fetchCollection` method returns a `Promise` which resolves when the data has been successfully retrieved. This may have been due to a round trip call to the REST API, or, if the data has already been previously returned, then it will skip the call to the REST API and return a cached copy of the data.

If it is important to get data that has not been cached, you can call the `fetch` method with a force parameter

```
ContentNodeResource.fetchCollection({ getParams: { popular: 1 }, force: true }).
↪then((dataArray) => {
  logging.info('This is the model data: ', dataArray);
});
```

2.7.3 Data flow

2.8 Development workflow

2.8.1 Git workflow

At a high level, we follow the 'Gitflow' model. Some helpful references:

- Atlassian tutorial
- Original description

2.8.2 Pull requests

Submissions

Be sure to follow the instructions shown in the Github PR template when you create a new PR.

In particular, **please use the labels** "Needs review", "Work in progress", and "Needs updates" mutually exclusively to communicate the state of the PR.

Developers maintain their own clones of the Learning Equality Kolibri repo in their personal Github accounts, and submit pull requests back to the LE repo.

Every pull request will require some combination of manual testing, code review, automated tests, gherkin stories, and UI design review. Developers must fully test their own code before requesting a review, and then closely follow the template and checklist that appears in the PR description. All automated tests must pass.

Unit tests and gherkin stories should be written to ensure coverage of critical, brittle, complicated, or otherwise risky paths through the code and user experience. Intentional, thoughtful coverage of these critical paths is more important than global percentage of code covered.

Try to keep PRs as self-contained as possible. The bigger the PR, the more challenging it is to review, and the more likely that merging will be blocked by various issues. If your PR is not being reviewed in a timely manner, reach out to stakeholders and politely remind them that you're waiting for a review.

Some additional guidelines:

- Submitters should fully test their code *before* asking for a review
- If the PR is languishing, feel free to prod team members for review
- Try to keep the PR up-to-date with the target branch
- Make sure to use the checkboxes in the PR template

Git history

Within the Kolibri repo, we have the following primary rule:

> *Never* rewrite history on shared branches.

History has been rewritten if a force push is required to update the remote. This will occur from e.g. amending commits, squashing commits, and rebasing a branch.

Some additional git history guidance:

- Be encouraged to rewrite history on personal branches so that your git commits tell a story
- Avoid noisy, meaningless commits such as "fixed typo". Squash these prior to submitting a PR
- When possible, make each commit a self-contained change that plays nicely with `git bisect`
- Once a PR code review has occurred, avoid squashing subsequent changes as this makes it impossible to see what changes were made since the code review
- Don't worry too much about a "clean" commit history. It's better to have some messy commits than to waste an hour than debugging a rebase-gone-wrong

Code Reviews

When reviewing PRs, keep feedback focused on critical changes. Lengthy conversations should be moved to a real-time chat when possible. Be polite, respectful, and constructive. We highly recommend following the guidance in this blog post.

Some general guidelines:

- Reviewers should actually run and test the PR
- When giving opinions, clarify whether the comment is meant to be a "blocking" comment or if it is just a conversation
- Pre-existing issues or other cleanup suggestions are can be opened as new issues, or mentioned as "non-blocking" comments
- Code formatting comments should be rare because we use Prettier and Black

Finally, if you see a very trivial but important necessary change, the reviewer can commit the change directly to a pull request branch. This can greatly speed up the process of getting a PR merged. Pushing commits to a submitter's branch should only be done for non-controversial changes or with the submitter's permission.

Note: When pushing to another user's branch, you may get an error like:

```
Authentication required:  You must have push access to verify locks
```

This is due to a Git LFS bug. Try disabling lock verification using the `lfs.[remote].locksverify` setting, or simply running `rm -rf .git/hooks/pre-push`.

Note: Remember to keep the "Needs review", "Work in progress", and "Needs updates" mutually exclusive and up-to-date.

Merging PRs

Who should merge PRs, and when?

First, all automated checks need to pass before merging. Then…

- If there is just one reviewer and they approve the changes, the reviewer should merge the PR immediately
- If there are multiple reviewers or stakeholders, the last one to approve can merge
- The reviewer might approve the PR, but also request minor changes such as a typo fix or variable name update. The submitter can then make the change and merge it themselves, with the understanding that the new changes will be limited in scope
- Stale reviews should be dismissed by the PR submitter when the feedback has been addressed

Copyright and licensing

The project as a whole is released under the MIT license, and copyright on its code is held by multiple parties including Learning Equality.

Individual files, such as code copied in from other projects may be under a different license, if that license is compatible.

Similarly, files from Kolibri may end up being copied into other projects.

For these reasons, copyright and license data may be listed explicitly at the top of some files. For example:

```
# Copyright 2023 Ann Contributor
# SPDX-License-Identifier: MIT
```

This format is machine readable and complies with the REUSE specification for software licensing.

For files where the license is not explicitly stated, the overall project license applies.

2.8.3 Development phases

We have the following release types:

- **Final**
  - Public releases
  - Info: major, minor, patch
  - PEP-440: `1.2.3`
  - Git tag: `v1.2.3` on a release branch
- **Beta**
  - Final integration testing, string freeze, and beta release candidates
  - High level of risk-aversion in PRs
  - Info: major, minor, patch, beta
  - PEP-440: `1.2.3b4`
  - Git tag: `v1.2.3-beta4` on a release branch
- **Alpha**
  - Initial testing releases

  - Avoid broken builds in PRs
  - Info: major, minor, patch, alpha
  - PEP-440: `1.2.3a4`
  - Git tag: `v1.2.3-alpha4` on the develop branch
- **Dev**
  - Feature branches, PRs, or other git commits
  - Info: major, minor, patch, commit
  - Experimental work is OK

Within the Learning Equality Kolibri repository:

- The `develop` branch is our current development branch, and the default target for PRs
- Release branches named like `release-v1.2.x` (for example). This will track all patch releases within the 1.2.x minor release line. Distinct releases are tracked as tags like `v1.2.3`
- We sometimes create feature branches for changes that are long-running, collaborative, and disruptive. These should be kept up-to-date with `develop` by merging, not rebasing.

If a change needs to be introduced to an older release, target the oldest release branch that we want the change made in. Then that change will need to be merged into all subsequent releases, one-at-a-time, until it eventually gets back to `develop`.

2.8.4 Github labels

We use a wide range of labels to help organize issues and pull requests in the Kolibri repo.

Priority

These are used to sort issues and sometimes PRs by priority if *and only if* the item is assigned a milestone. Every issue in a milestone ought to have a priority label.

Only 'critical' items are strictly blockers for a release, but typically all important items should be expected to make it in, too. Priority within a release is generally assigned by a core Learning Equality team member.

- **P0 - critical**
- **P1 - important**
- **P2 - normal**
- **P3 - low**

Changelog

The **changelog** label is used on PRs or issues to generate 'more details' links in the *Release Notes*.

Work-in-progress

The **work-in-progress** label is helpful if you have a PR open that's not ready for review yet.

Development category

Labels prefixed with **DEV:** are used to help organize issues (and sometimes PRs) by area of responsibility or scope of domain knowledge necessary.

TODO items

Labels prefixed with **TODO:** help flag items that need some action before the issue or PR can be fully resolved.

Organizational Tags

Labels prefixed with **TAG:** are general-purpose, and are used to help organize issues and PRs.

2.9 Build system and workflow

2.9.1 Frequently asked questions

- How does the build system work overall?

  The `Kolibri Python Package` pipeline creates a `whl` file and triggers every other pipeline. Those pipelines download the `whl` (or the `.deb`, in the pi image's case) from the associated build in `Kolibri Python Package` to build their own installer. See below for more detail.

  One key thing to remember: each build is associated with a specific commit. Builds happen as a Github "check" for each commit, and can be found on the associated commit inside of its "checks" links - denoted by a green checkmark (or a red X in the case of a failed build).

- How do I access builds for a PR?

  The easiest way to do so would be to go through Github. Navigate to the PR page. If it's an open PR, the "Checks" section should be salient - one of those checks should be Buildkite's, and that will take you to the associated build page.

  If it's a closed PR, click on the "Commits" section. Every commit that has a green checkmark to its right is a commit that had checks run against it. Find the commit you're interested in and click on the checkmark to get the link to the associated build.

  If you want one of the installers in a build, click on the link to the build pipeline triggered for that installer. From that installer build's page, you should be able to unblock the "Block" step and start the build of the specified installer. See below for more detail.

- How do I trigger builds that are blocked?

  "Block" steps can be unblocked via the Builkite GUI, on the build page. Clicking on the "Block" step should present you with a confirmation modal, asking if you want to proceed.

  Most "secondary" pipelines - installer pipelines - have a "Block" step as their first, so that the build won't run unless the parent `Kolibri Python Package` build is a release.

- How do I access builds for tagged releases?

The best way to do so is, again, from Github. Navigate to the "Releases" page on Github. On the left of the Release name, you should see the short-SHA of a Github commit. This short SHA is also a link. Clicking on it will take you to the Github page for that commit. Underneath the commit message, you should find a green checkmark (or a red "X" if it didn't build properly).

Clicking on the symbol should link to the build page you're looking for.

- How do I find builds for specific tags?

  The best way to do so is from Github. Navigate to the "Tags" page on Github. On the left of the Tag name, you should see the short-SHA of a Github commit. This short SHA is also a link. Clicking on it will take you to the Github page for that commit. Underneath the commit message, you should find a green checkmark (or a red "X" if it didn't build properly). Clicking on the symbol should link to the build page you're looking for.

2.9.2 Design goals

The build pipeline currently uses Buildkite as its build system. Buildkite is flexible in that the build queue is hosted on their servers, while the build agents (the servers that actually run the build scripts) are self hosted.

The design goals of the pipeline in its current iteration are chiefly:

- Continuously integrate authors' changes into a built package/installer/app (asset)
- Provide *timely* alerts to all authors of Pull Requests (PRs) and Releases to Kolibri-related projects on Github (GH)
- Make those assets available to testers and developers
- In the event of a release, make those assets available on the corresponding release page in GH

These goals are described at a high level, and carry some implicit meaning. These implications translated to some more concrete goals in the pipeline's most recent iteration:

- For the sake of speed:
  - Automatically build as few assets as possible on a per-PR basis.
- For the the sake of convenience:
  - Allow for testers/developers to request additional assets without human intervention.
- For the sake of resilience:
  - Have more than one build agent, distributed geographically.

There is certainly overlap in those goals. For example, a faster build translates to a more convenient release process for our release managers, who must ensure that assets build after tagging a release.

2.9.3 Overview of Buildkite

Before describing the current architecture, it might be helpful to provide context by giving an overview of how Buildkite works.

This entire section will not be describing any of LE's build pipelines in particular - only how Buildkite manages pipelines.

Without diving too deep (LINK please visit their official documentation if you'd like more detail), the Buildkite product has 2 main components, (illustrated on this page (LINK to agent page)):

1. The Buildkite Agent API

2. The Buildkite Agent Daemon

API and vocabulary

The API is hosted on Buildkite servers. It's primary purpose is to receive build *steps* in the form of (mostly) YAML, and distribute them as *jobs* to Agents.

A *step*, in this context, is a YAML-formatted instruction - in all of our product repositories, our steps live inside of `.buildkite/pipeline.yml`. It's the serial form of instructions for Buildkite.

A *job* is the instantiation of a step, or the de-serialized form of a step. They aren't always running, but are used as references for the processes involved in running the commands dictated by the step. Jobs are assigned to Agents, and can run on any of the Agents connected to our account's Buildkite API.

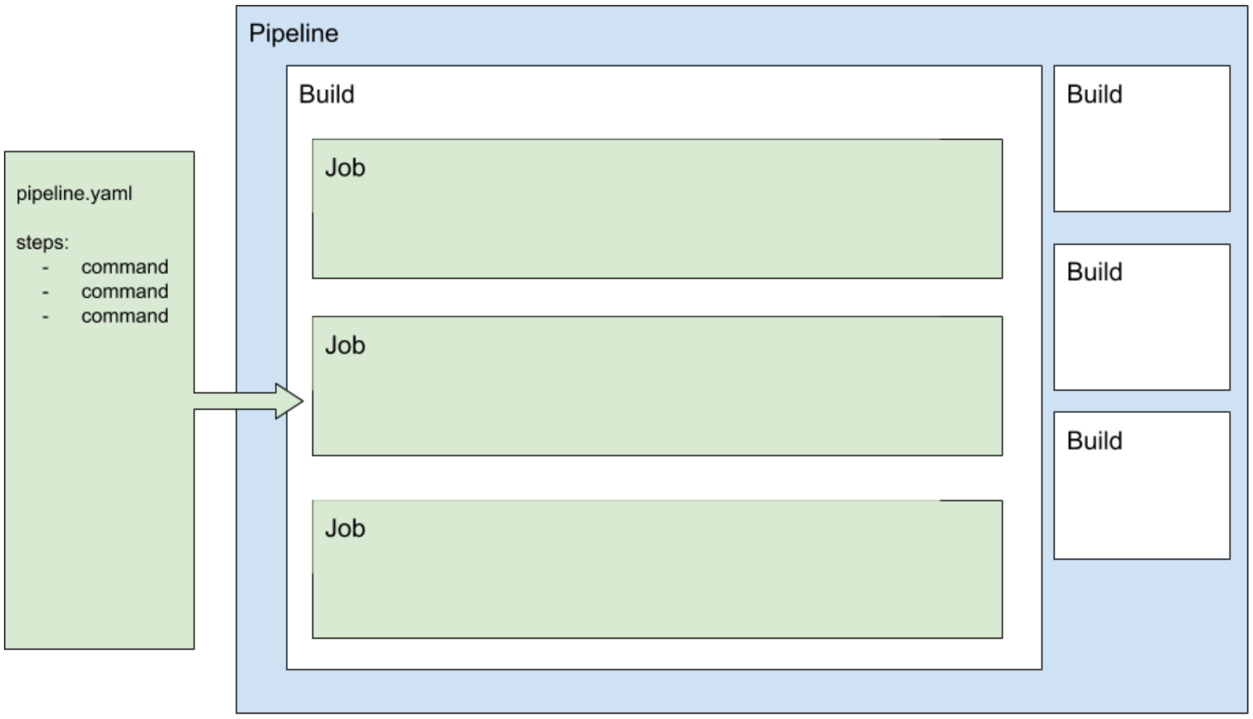
A *build* can be considered a container for jobs. After the `pipeline.yml` file is de-serialized, all jobs are added to a *build* before being delegated to an agent.

Apart from the job allocation functionality, Buildkite conveniently provides us with:

- A Webhook server/client
- A web UI
- Ephemeral asset hosting

  A *pipeline* can be thought of as a sort of container for builds (each build must belong to a pipeline), as well as a housing mechanism for the features described above; those settings can all be configured on a per-pipeline basis.

Here's a visual of what concept is a property of which:



Github integration

The Webhook client functionality is critical, as it allows us to integrate with Github.

Github alerts Buildkite that a new PR, commit, or tag has been created via webhook. This spurs the Buildkite servers to create a job, instructing the Agent to pull the GH repo and send Buildkite the steps it needs to be carried out.

The step that defines this job cannot not be defined inside of the `pipeline.yaml` file commited to the repository. This *must* be defined on Buildkite's servers using their web GUI.

Agent

The agent is hosted on LE servers. Some of these servers are physically located in the LE physical office, and others are physically located in a cloud provider's server farm.

All of these servers have a Buildkite Agent application installed as a background service. It's primary purpose is to receive jobs from the Buildkite API and execute them.

Apart from the obvious authentication components that are required to access the API, the agent provides us with:

- An agent-level hooks system
- The ability to completely self-manage our build environments and secrets

The value of self hosted

Many build systems provide a free tier of hosting. In the best of those cases, you provide them a Docker image that they then deploy. Your jobs run inside of that image. The mechanism with which secrets (envars and files) are passed to these systems vary wildly.

We could probably make those systems work if need be. By self hosting, however, we completely control various facets of the build pipeline:

- Secrets
  - Where they live
  - How they're stored or downloaded
  - Their form (envar vs JSON file, etc.)
- Complete control of our dependencies, down to the OS/Kernel.
- The ability to invest in the one-time-cost (as opposed to the ongoing cost of cloud-provided hosting) of physical hardware , customized to our workload.
  - "Hybrid Cloud" setups - where the bulk of the workload is on-premises, with some off-premises secondary workloads.

2.9.4 Learning Equality's pipelines

There is one pipeline per installer, each is configured to listen to a different GH repository. :

- `Kolibri Python Package`
  - https://github.com/learningequality/kolibri
- `Kolibri MacOS`
  - https://github.com/learningequality/kolibri-installer-mac

- `Kolibri Android Installer`
  - https://github.com/learningequality/kolibri-installer-android
- `Kolibri Debian`
  - https://github.com/learningequality/kolibri-installer-debian
- `Kolibri Windows`
  - https://github.com/learningequality/kolibri-installer-windows
- `Kolibri Raspian Image`
  - https://github.com/learningequality/pi-gen

This implies a few things:

- A manually triggered build (clicking on the "New Build" button on Buildkite) will pull from a specific repository.
- An automatically triggered build will pull from the same repository, given that the webhook has been set and the triggers are properly configured
- After pulling the repository, each pipeline assumes that there is a file defining steps in the repository it just downloaded. By default, it will use `.buildkite/pipeline.json`. This can be changed, but we don't do that in any of our pipelines.

With one exception, each pipeline's sole concern is to build the asset it is named for, then upload it to the appropriate destinations. The exception, and the "appropriate destinations", will be explained below.

Pipeline orchestration

Presently, the `Kolibri Python Package` Pipeline carries more responsibility than the rest.

Whereas the other pipelines' responsibilities stop at building and uploading their installer, `Kolibri Python Package` acts as the "kick off" point for the other installers. Being the only pipeline listening to the Kolibri repository on Github for changes, it is the only pipeline triggered by those changes.

After building the `.whl` and `.pex` in a single step, the `Kolibri Python Package` proceeds to trigger the other installers, most of which rely on the `.whl` file (The single exception is `Kolibri Raspbian Image`, which relies on the `.deb` installer).

These *trigger steps* live inside of the `Kolibri Python Package`, but send metadata to each of the other pipelines and trigger an entirely new build in each one.

Block steps

These triggered builds are created simultaneously; this does not mean that the jobs belonging to the builds are assigned simultaneously. The very first thing a new build does is pull the repository and de-serialize the steps living inside the `.buildkite` folder.

For non-release builds, each Build's first step is a "Block" step - this kind of step does not create a job. At this point, the Build is "finished". That is, finished for now: the build will progress once user input confirming procession has been received.

The "finished" signal on the triggered builds report back to the `Kolibri Python Package` pipeline, indicating it as "complete" even if no build has been run.

This allows for efficiency: Time won't be wasted waiting for every single installer to be built for non-release pipelines. If a developer *wants* one of the other installers, they may navigate to the appropriate pipeline and unblock the step.

Release builds

In the case that this build belongs to a release-tagged Git commit, a few conditions are triggered:

1. The “Upload Release Artifact” step, conditional based on the existence of a Git tag, now exists at the end of the artifact build steps (Both standard and triggered builds).
2. The Block step at the start of each “child” pipeline, conditional based on the existence of a Git tag, ceases to exist. This means that all of the triggered builds, in each of the triggered pipelines, will run and generate a artifact. The longest of these is the Raspbian image.

2.10 Release process

Kolibri releases are tracked in Notion. This page contains:

- A ‘Kolibri releases’ tracker database
- A set of templates in the tracker database for Major/Minor and Final/pre-releases
- Checkslists of release steps
- Additional guidance on performing release steps

We also maintain a small set of release process automation scripts which automate some processes.

2.11 Internationalization

As a platform intended for use around the world, Kolibri has a strong mandate for translation and internationalization. As such, it has been designed with technologies to enable this built in.

2.11.1 Writing localized strings

For strings in the frontend, we are using Vue-Intl, an in house port of React-intl. Strings are collected during the build process, and bundled into exported JSON files.

Messages will be discovered for any registered plugins and loaded into the page if that language is set as the Django language. All language setting for the frontend are based off the current Django language for the HTTP request.

.vue files

Within Kolibri .vue components, messages are defined in the `<script>` section as attributes of the component definition:

```
export default {
  name: 'componentName',
  $trs: {
    msgId1: 'Message text 1',
    msgId2: 'Message text 2',
  },
};
```

The component names and message IDs should all be camelCase.

User visible strings can be used anywhere in the .vue file using `$tr('msgId')` (in the template) or `this.$tr('msgId')` (in the script).

An example Vue component would then look like this

```
<template>
  <div>
    <!-- puts 'Hello world' in paragraph -->
    <p>{{ $tr('helloWorld') }}</p>
  </div>
</template>


<script>

  export default {
    name: 'someComponent',
    mounted() {
      // prints 'Hello world' to console
      console.log(this.$trs('helloWorld'));
    },
    $trs: {
      helloWorld: 'Hello world',
    },
  };

</script>
```

.js files

In order to translate strings in Javascript source files, the namespace and messages are defined like this:

```
import { createTranslator } from 'kolibri.utils.i18n';
const name = 'someModule';
const messages = {
  helloWorld: 'Hello world',
};
const translator = createTranslator(name, messages);
```

Then messages are available from the `$tr` method on the translator object:

```
console.log(translator.$tr('helloWorld'));
```

common*String modules

A pattern we use in order to avoid having to define the same string across multiple Vue or JS files is to define "common" strings translator. These common translators are typically used within plugins for strings common to that plugin alone. However, there is also a "core" set of common strings available to be used throughout the application.

In order to avoid bloating the common modules, we typically will not add a string we are duplicating to a common module unless it is being used across three or more files.

Common strings modules should typically have a translator created using the `createTranslator` function in which strings are defined - these can then be used in the setup function of a component to expose specific strings as functions:

```
import commonStringsModule from '../common/commonStringsModule';


export default {
  name: 'someComponent',
  setup() {
    const { myCoolString$, stringWithArgument$ } = commonStringsModule;
    return {
      myCoolString$, stringWithArgument$
    };
  },
};
```

```
<template>
  <div>
    <p>{{ myCoolString$() }}</p>
    <p>{{ stringWithArgument$({ count: 4 }) }}</p>
  </div>
</template>
```

Previously, this has been handled via mixins, which has required this additional complexity in the modules. You may see modules that include the translator object and the following:

- An exported function that accepts a `string` and an `object` - which it then passes to the `$tr()` function to get a string from the translator in the module.
- An exported Vue mixin that exposes the exported function as a `method`. This allows Vue components to use the mixin and have the exported function to get a translated string readily at hand easily.

ICU message syntax

All frontend translations can be parameterized using ICU message syntax. Additional documentation is available on crowdin.

This syntax can be used to do things like inject variables, pluralize words, and localize numbers.

Dynamic values are passed into translation strings as named arguments in an object. For example:

```
export default {
  name: 'anothetComponent',
  mounted() {
    // outputs 'Henry read 2 stories'
    console.log(this.$tr('msg', {name: 'Henry', count: 2}));
  },
```

(continues on next page)

(continued from previous page)

```
  $trs: {
    msg: '{name} read {count} {count, plural, one {story} other {stories}}',
  },
};
```

.py files

For any user-facing strings in python files, we are using standard Django tools (`gettext` and associated functions). See the Django i18n documentation for more information.

2.11.2 RTL language support

Kolibri has full support for right-to-left languages, and all functionality should work equally well when displayed in both LTR and RTL languages.

There are a number of important considerations to take into account with RTL content. Material Design has an excellent article that covers most important topics at a high level.

Warning: Right-to-left support is broken when running the development server with hot reloading enabled (`yarn run devserver-hot`)

Text alignment

Alignment of application text (i.e. text using `$tr` syntax) is mostly handled "automagically" by the RTLCSS framework. This means that application text should have CSS applied to it as though it is written in English. For example, if you want the text aligned left for LTR languages and right for RTL, simply use `text-align: left`. This will be automatically flipped to `text-align: right` by the webpack plugin. Since the application is only ever viewed in one language at a time, RTLCSS can apply these changes to all CSS at once.

On the other hand, alignment of user-generated text (from databases or from content) is inherently unknown beforehand. Therefore all user-generated text must have `dir="auto"` set on a parent DOM node. This can get especially complicated when LTR and RTL content are mixed inline bidirectionally. Read more about the Unicode Bidirectional algorithm.

A rule of thumb for inline bidirectional text:

- if user-generated text is on its own in a block-level DOM element, it should be aligned based on the text's language using `dir="auto"` on the block-level element.
- if user-generated text is displayed inline with application text (such as "App Label: user text"), it should be aligned using CSS `text-align` on the block-level element, and `dir="auto"` on a `span` wrapping the inline user text.

Behavior

Occasionally it is necessary to perform different logic depending on the directionalty of the the currently-selected language. For example, the handling of a button that changes horizontal scroll position would need to flip direction.

In the frontend, we provide a `isRtl` property attached to every Vue instance. For example, you could write Vue methods like:

```
previous() {
  if (this.isRtl) this.scrollRight();
  else this.scrollLeft();
},
next() {
  if (this.isRtl) this.scrollLeft();
  else this.scrollRight();
},
```

If you need to get the current language directionality on the backend, you can use `django.utils.translation.get_language_bidi`.

Iconography

Choosing whether or not to mirror icons in RTL languages is a subtle decision. Some icons should be flipped, but not others. From the Material guidelines:

> *anything that relates to time should be depicted as moving from right to left. For example, forward points to the left, and backwards points to the right*

It is recommended to use the `KIcon` component when possible, as this will handle RTL flipping for you and apply it when appropriate, as well as taking care of other details:

```
<KIcon icon="forward" />
```

If `KIcon` does not have the icon you need or is not usable for some reason, we also provide a global CSS class `rtl-icon` which will flip the icon. This can be applied conditionally with the `isRtl` property, e.g.:

```
<img src="forward.png" :class="{ 'is-rtl': isRtl }" alt="" />
```

Content rendererers

User interfaces that are tightly coupled to embedded content, such as the ‘next page’ and ‘previous page’ buttons in a book, need to be flipped to match the language direction of that content. UIs that are not tightly integrated with the content should match the overall application language, not the content.

Information about content language direction is available in the computed props `contentDirection` and `contentIsRtl` from `kolibri.coreVue.mixins.contentRendererMixin`. These can be used to change styling and directionality dynamically, similar to the application-wide `isRtl` value.

In situations where we are using third-party libraries it might be necessary to flip the entire content renderer UI automatically using the RTLCSS framework rather than make targeted changes to the DOM. To handle these cases, it’s possible to dynamically load the correct CSS webpack bundle using a promise:

```
export default {
  name: 'SomeContentRenderer',
  created() {
```

(continues on next page)

(continued from previous page)

```
    // load alternate CSS
    this.cssPromise = this.$options.contentModule.loadDirectionalCSS(this.
↪contentDirection);
  },
  mounted() {
    this.cssPromise.then(() => {
      // initialize third-party library when the vue is mounted AND the CSS is loaded
    });
  },
};
```

2.11.3 Crowdin workflow

We use the Crowdin platform to enable third parties to translate the strings in our application.

Note that you have to specify branch names for most commands.

Note: These notes are only for the Kolibri application. For translation of user documentation, please see the kolibri-docs repository.

Note: The Kolibri Crowdin workflow relies on the project having the "Duplicate strings" setting set to "Show – translators will translate each instance separately". If this is not set, the workflow will not function as expected!

Prerequisites

The tooling requires a minimum Python version of 3.7 and the dependencies in `requirements/fonts.txt` installed.

You'll need to have GNU `gettext` available on your path. You may be able to install it using your system's package manager.

Note: If you install `gettext` on Mac with Homebrew, you may need to add the binary to your path manually

Finally, ensure you have an environment variable `CROWDIN_API_KEY`. You can generate your Crowdin API key by navigating to your Crowdin account settings page.

Extracting and uploading sources

Typically, strings will be uploaded when a new release branch is cut from `develop`, signifying the beginning of string freeze and the `beta` releases. (See *Release process*.)

Before translators can begin working on the strings in our application, they need to be uploaded to Crowdin. Translations are maintained in release branches on Crowdin in the Crowdin kolibri project.

This command will extract front- and backend strings and upload them to Crowdin, and may take a while:

```
make i18n-upload branch=[release-branch-name]
```

The branch name will typically look something like: `release-v0.8.x`

Pre-translation

After running the `i18n-upload` command above, the newly created branch should have some percentage of strings in supported languages shown as both translated and approved. These strings are the *exact* matches from the previous release, meaning that both the string IDs and the English text is exactly the same.

At this point, it is often desirable to apply some form of pre-translation to the remaining strings using Crowdin's "translation memory" functionality. There are two ways to do this: with and without auto-approval.

To run pre-translation without auto-approval **(recommended)**:

```
make i18n-pretranslate branch=[release-branch-name]
```

Or to run pre-translation with auto-approval:

```
make i18n-pretranslate-approve-all branch=[release-branch-name]
```

Warning: The exact behavior of Crowdin's translation memory is not specified. Given some English phrase, it is not always possible to predict what suggested translation it will make. Therefore, auto-approval be used with caution.

Transferring screenshots

Every release, we need to transfer screenshots on the platform from the previous branch to the new branch, as this is the only way to persist screenshots across branches. To do this run:

```
make i18n-transfer-screenshots branch=[release-branch-name] source=[previous-release-
↪branch-name]
```

This will match all screenshots by their Kolibri message ID to persist screenshots across releases.

Reviewing screenshots

Every release, we need to review screenshots on the platform to ensure they are up to date. To generate a report of all the screenshots for a particular branch run:

```
make i18n-screenshot-report branch=[release-branch-name]
```

This will generate an HTML report that can be browsed to double check screenshots against the source English strings, with a link to the string on Crowdin to update the screenshot if needed.

Downloading translations to Kolibri

As translators work on Crowdin, we will periodically retrieve the latest updates and commit them to Kolibri's codebase. In the process, we'll also update the custom fonts that are generated based on the translated application text.

First, you need to download source fonts from Google. In order to do this, run:

```
make i18n-download-source-fonts
```

Next, we download the latest translations from Crowdin and rebuild a number of dependent files which will be checked in to git. Do this using the command below. **It can take a long time!**

```
make i18n-download branch=[release-branch-name]
```

This will do a number of things for you:

- Rebuild the crowdin project (note that builds can only happen once every 30 minutes, as per the Crowdin API)
- Download and update all translations for the currently supported languages
- Run Django's `compilemessages` command
- Regenerate all font and css files
- Regenerate Intl JS files

Check in all the updated files to git and submit them in a PR to the release branch.

Note: Remember about Perseus! Check if files in that repo have changed too, and submit a separate PR. It will be necessary to release a new version and referencing it in Kolibri's `base.txt` requirements file.

2.11.4 Adding a newly supported language

In order to add a new language to Kolibri, the appropriate language information object must be added to the array in `kolibri/locale/language_info.json`.

Warning: Always test a newly added language thoroughly because there are many things that can go wrong. At a minumum, ensure that you can run the development server, switch to the language, and navigate around the app (including Perseus exercises). Additionally, ensure that the fonts are rendered with Noto.

The language must be described using the following keys, with everything in lower case

```
{
  "crowdin_code":   "[Code used to refer to the language on Crowdin]",
  "intl_code":      "[Lowercase code from Intl.js]",
  "language_name":  "[Language name in the target language]",
  "english_name":   "[Language name in English]",
  "default_font":   "[Name of the primary Noto font]"
}
```

- For `crowdin_code`, see Crowdin language codes.
- For `intl_code`, see Supported Intl language codes and make it lowercase.
- For `language_name` and `english_name`, refer to the ISO 639 codes. If necessary, use this backup reference. If the language is a dialect specific to a region, include the name of the region in parentheses after the language name.
- For the `default_font`, we use variants of Noto Sans. Search the Noto database to see which font supports the language you are adding.

If the language doesn't exist in Django, you may get errors when trying to view the language. In this case it needs to be added to `EXTRA_LANG_INFO` in `base.py`.

For the new language to work, the `django.mo` files for the language must also be generated by running `make i18n-download` and committed to the repo.

To test unsupported languages, you can use the *Deployment* section *LANGUAGES* option in the Kolibri options.ini. Either set the value to `all` to activate all languages, or add the specific Intl language code as the value.

Once the language has been fully translated and is ready for use in Kolibri, its Intl language code must be added to the `KOLIBRI_SUPPORTED_LANGUAGES` list in `kolibri/utils/i18n.py`.

Updating font files

We pin our font source files to a particular commit in the Google Noto Fonts github repo.

Google occasionally adds new font files and updates existing ones based on feedback from the community. They're also in the process of converting older-style fonts to their "Phase III" fonts, which are better for us because they can be merged together.

In order to update the version of the repo that we're using to the latest HEAD, run:

```
python packages/kolibri-tools/lib/i18n/fonts.py update-font-manifest
```

You can also specify a particular git hash or tag:

```
python packages/kolibri-tools/lib/i18n/fonts.py update-font-manifest [commit hash]
```

Make sure to test re-generating font files after updating the sources.

Note: We attempt to download fonts from the repo. It is possible that the structure of this repo will change over time, and the download script might need to be updated after changing which version of the repo we're pinned to.

2.11.5 Configuring language options

The languages available in an instance of Kolibri can be configured using a few mechanisms including:

- An environment variable (`KOLIBRI_LANGUAGES`)
- An *options.ini* file (in `LANGUAGES` under `[Deployment]`)
- Overwriting the option in a *kolibri_plugin.py* plugin config file

It takes a comma separated list of `intl_code` language codes. It can also take these special codes:

- `kolibri-supported` will include all languages listed in `KOLIBRI_SUPPORTED_LANGUAGES`
- `kolibri-all` will include all languages defined in *language_info.json*

2.11.6 Auditing strings

Much of our string workflow before developer implementation happens using Ditto. In order to do a full audit of newly added strings from Ditto, a CSV of the newly added strings can be exported from Ditto, and then our internal audit tool can be run to generate a CSV report of any strings that are potentially missing:

```
yarn run auditdittostrings --ditto-file <path to Ditto CSV>
```

This will produce an output CSV file in kolibri/locale/en/LC_MESSAGES/profiles/ditto.csv that contains an audit report on which strings from the Ditto file that are marked as FINAL were not found in the codebase (using an exact match method, so this may produce false positives if strings are not in ICU syntax on Ditto), and also any strings that have been discovered in the codebase to be an exact match - i.e. when we appear to have duplicate strings in our codebase (again, these may be false positives, as some strings may be repeated for different senses).

2.12 Manual testing & QA

2.12.1 General Notes

Accessibility (a11y) testing

Inclusive design benefits all users, and we strive to make Kolibri accessible for all. Testing for accessibility can be challenging, but there are a few features you should check for before submitting your PR:

- Working **keyboard navigation** - everything that user can do with mouse or by touch must also work with the keyboard alone.
- Sufficient color contrast between foreground text/elements and the background.
- Meaningful **text alternative** for all non-decorative images, or an empty `ALT` attribute in case of decorative ones.
- Meaningful **labels** on ALL form or button elements.
- Page has one main **heading** (H1) and consecutive lower heading levels.

Please also visit the *Recommended A11y tools* section of the manual testing documentation

Cross-browser and OS testing

It's vital to ensure that our app works across a wide range of browsers and operating systems, particularly older versions of Windows and Android that are common on old and cheap devices.

In particular, we want to ensure that Kolibri runs on major browsers that match any of the following criteria:

- within the last two versions
- IE 11+
- has at least 1% of global usage stats

Here are some useful options, in order of simplicity:

BrowserStack

BrowserStack is an incredibly useful tool for cross-browser and OS testing. In particular, it's easy to install plugin which forwards `localhost` to a VM running on their servers, which in turn is displayed in your browser.

Amazon Workspaces

In some situations, simply having a browser is not enough. For example, a developer may need to test Windows-specific backend or installer code from another OS. In many situations, a virtual machine is appropriate - however these can be slow to download and run.

Amazon's AWS Workspaces provides a faster alternative. They run Windows VMs in their cloud, and developers can RDP in.

Local Virtual Machines

Workspaces is very useful, but it has limitations: only a small range of OSes are available, and connectivity and provisioning are required.

An alternative is to run the guest operating system inside a virtual machine using e.g. VirtualBox. This also gives more developer flexibility, including e.g. shared directories between the guest and host systems.

There is also a *Testing with Virtual Machines* section, which we hope will help you to use virtual machines.

Hardware

There are some situations where actual hardware is necessary to test the application. This is particularly true when virtualization might prohibit or impede testing features, such as lower-level driver interactions.

Responsiveness to varying screen sizes

We want to ensure that the app looks and behaves reasonably across a wide range of typical screen sizes, from small tablets to large, HD monitors. It is highly recommended to constantly be testing functionality at a range of sizes.

Chrome and Firefox's Developer Tools both have some excellent functionality to simulate arbitrary screen resolutions.

Slow network connections

It's important to simulate end-users network conditions. This will help identify real-world performance issues that may not be apparent on local development machines.

Chrome's Developer Tools have functionality to simulate a variety of network connections, including Edge, 3G, and even offline. An app can be loaded into multiple tabs, each with its own custom network connectivity profile. This will not affect traffic to other tabs.

Within the Chrome Dev Tools, navigate to the Network panel. Select a connection from the drop-down to apply network throttling and latency manipulation. When a Throttle is enabled the panel indicator will show a warning icon. This is to remind you that throttling is enabled when you are in other panels.

For Kolibri, our target audience's network condition can be mimicked by setting connectivity to Regular 3G (100ms, 750kb/s, 250 kb/s).

Performance testing with Django Debug Panel

We have built in support for Django Debug Panel (a Chrome extension that allows tracking of AJAX requests to Django).

To use this, ensure that you have development dependencies installed, and install the Django Debug Panel Chrome Extension. You can then run the development or production servers with the following environment variable set:

```
DJANGO_SETTINGS_MODULE=kolibri.deployment.default.settings.debug_panel
```

This will activate the debug panel, and will display in the Dev tools panel of Chrome. This panel will track all page loads and API requests. However, all data bootstrapping into the template will be disabled, as our data bootstrapping prevents the page load request from being profiled, and also does not profile the bootstrapped API requests.

Generating user data

For manual testing, it is sometimes helpful to have generated user data, particularly for Coach and Admin facing functionality.

In order to do this, a management command is available

```
kolibri manage generateuserdata
```

This will generate user data for each channel on the system. To see available options, use

```
kolibri manage help generateuserdata
```

Examples for Kolibri with imported channels

The command `kolibri manage generateuserdata` (without any arguments) creates 1 facility, with 2 classes, and 20 users each class. It will then create sample data up to maximum of 2 channels. Then it will create 5 lessons per class, 2 exams, and randomize the number of interactions per channel for learners.

Create 2 facilities, with 2 classes per facility, with 20 learners per class.

```
kolibri manage generateuserdata --facilities 2 --classes 2 --users 20
```

Same as above, but prepend their names with "VM1" - useful for testing P2P syncing features.

```
kolibri manage generateuserdata --facilities 2 --classes 2 --users 20 --device-name VM1
```

Create 2 facilities, with 2 classes per facility, with 20 learners per class, 2 interactions per learner.

```
kolibri manage generateuserdata --facilities 2 --classes 2 --users 20 --num-content-
↪items 2
```

Examples for a fresh Kolibri install (no imported channels)

For a fresh Kolibri installation, use this to automatically create superusers and skip on-boarding (setup wizard). The superuser username is `superuser` and password is `password`.

```
kolibri manage generateuserdata --no-onboarding
```

Create 2 facilities, with 2 classes per facility, with 20 learners per class.

```
kolibri manage generateuserdata --facilities 2 --classes 2 --users 20 --no-onboarding
```

Notes

1. If there are existing facilities, it will only create the remaining ones. So if you already have one facility, specifying `--facilities 2` will create one more facility and its subsequent sample data.
2. Use the *–max-channels* option to limit the number of channels for learners to interact with. This saves a lot of time specially on large data samples.
3. The `--no-onboarding` argument creates a super user for each facility with username `superuser` and password `password`.

Collecting client and server errors using Sentry

Sentry clients are available for both backend and frontend error reporting. This can be particularly useful to have running on beta and demo servers in order to catch errors "in the wild".

This behaviour is activated by installing the Kolibri Sentry Plugin. Once installed, the options below become available for configuration.

```
pip install kolibri-sentry-plugin  # might need to run with sudo
```

If you're running Kolibri using a pex file, you'll need to make sure that the pex inherits a Python path with *kolibri_sentry_plugin* available. To do this without inheriting the full system path, run the pex from an active virtual environment with *PEX_INHERIT_PATH=1 python kolibri.pex*.

To set up error reporting, you'll need a Sentry DSN. These are available from your project settings at `https://sentry.io/settings/[org_name]/[project_name]/keys/`

You can set these either in options.ini or as environment variables.

If using options.ini, under a `Debug` header you can use these options:

- `SENTRY_BACKEND_DSN`
- `SENTRY_FRONTEND_DSN`
- `SENTRY_ENVIRONMENT` (optional)

Or if using environment variables:

- `KOLIBRI_DEBUG_SENTRY_BACKEND_DSN`
- `KOLIBRI_DEBUG_SENTRY_FRONTEND_DSN`
- `KOLIBRI_DEBUG_SENTRY_ENVIRONMENT` (optional)

The 'environment' corresponds to a particular installation of Kolibri that we want to track over time - for example, `demo-server`, `beta-server`, or `i18n-server`.

Other information is provided automatically such as the current user, browser info, and locale.

2.12.2 Testing with Virtual Machines

Install VirtualBox

Linux

Download and follow the installation instructions from here: https://www.virtualbox.org/wiki/Linux_Downloads

Windows

Download and follow the Windows installation instructions.

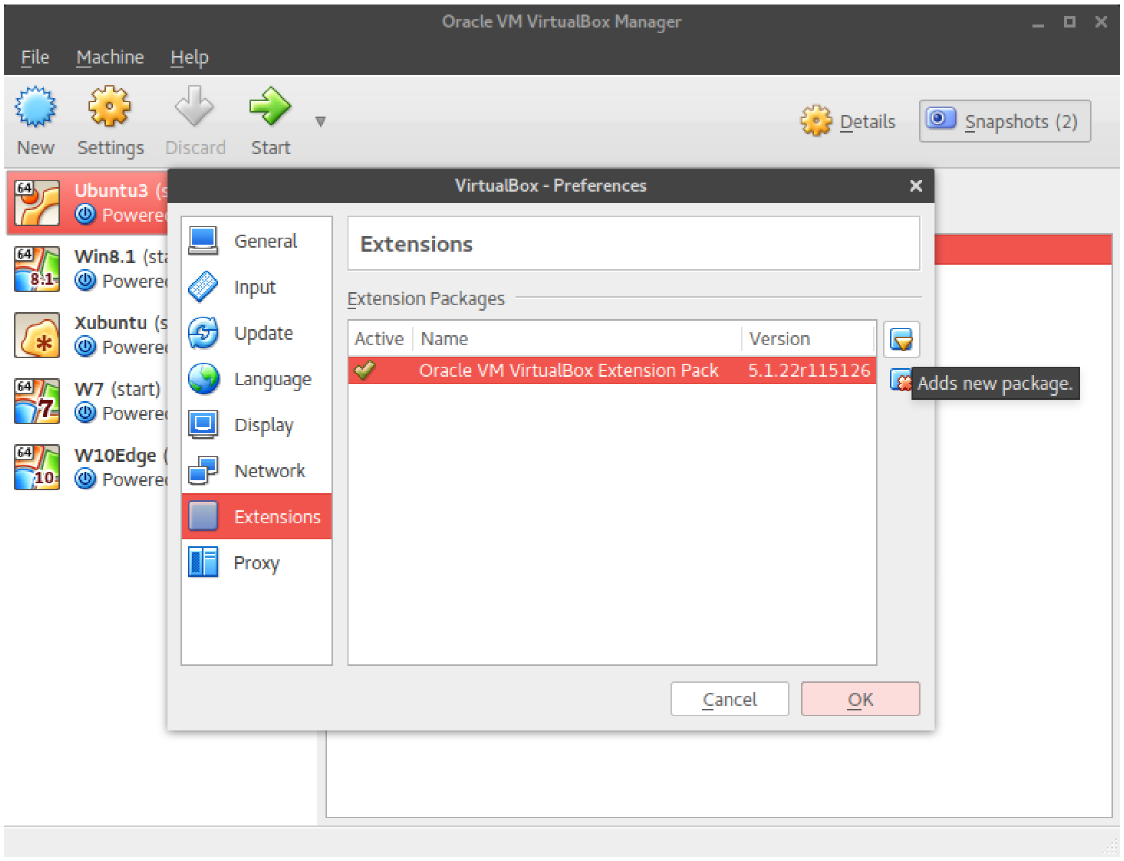
Macintosh

Download and follow the Mac installation instructions.

Install VirtualBox Extension Pack

Independently from which OS you use as host, you need to install the same VB Extension Pack to enable support for USB 2 & 3 devices and other features.

1. Download the VirtualBox Extension Pack.
2. Go to File - Preferences - Extensions in the VirtualBox main menu.
3. Load the extension file in the pane on the right and close the Preferences.



Extensions

Extension Packages

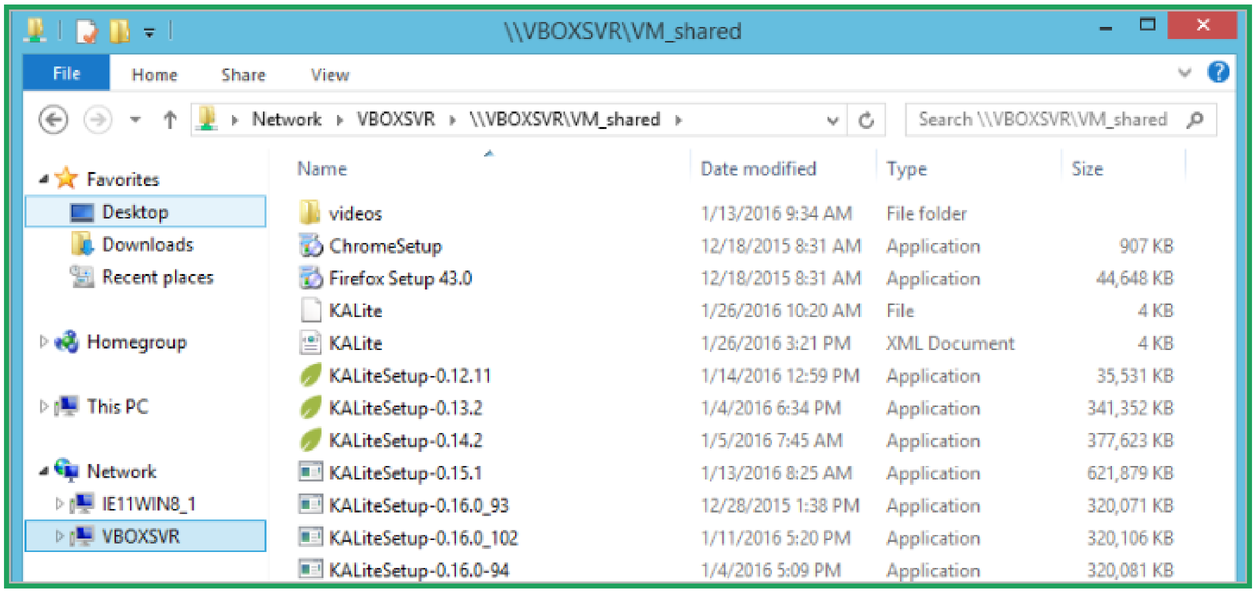
Active	Name	Version
✓	Oracle VM VirtualBox Extension Pack	5.1.22r115126

Kolibri releases

Download the latest Kolibri release installers from GitHub: https://github.com/learningequality/kolibri/releases/latest

Tip: Make a Shared folder for installers and content

Designate one folder on your host OS where you will save all the installers needed for testing in various virtual machines. You could also have save additional installers as browsers (Mozilla Firefox & Google Chrome). After you install VirtualBox and import virtual machines, you will configure each one to share (see) that folder on your host OS as a network folder.



Name	Date modified	Type	Size
videos	1/13/2016 9:34 AM	File folder	
ChromeSetup	12/18/2015 8:31 AM	Application	907 KB
Firefox Setup 43.0	12/18/2015 8:31 AM	Application	44,648 KB
KALite	1/26/2016 10:20 AM	File	4 KB
KALite	1/26/2016 3:21 PM	XML Document	4 KB
KALiteSetup-0.12.11	1/14/2016 12:59 PM	Application	35,531 KB
KALiteSetup-0.13.2	1/4/2016 6:34 PM	Application	341,352 KB
KALiteSetup-0.14.2	1/5/2016 7:45 AM	Application	377,623 KB
KALiteSetup-0.15.1	1/13/2016 8:25 AM	Application	621,879 KB
KALiteSetup-0.16.0_93	12/28/2015 1:38 PM	Application	320,071 KB
KALiteSetup-0.16.0_102	1/11/2016 5:20 PM	Application	320,106 KB
KALiteSetup-0.16.0-94	1/4/2016 5:09 PM	Application	320,081 KB

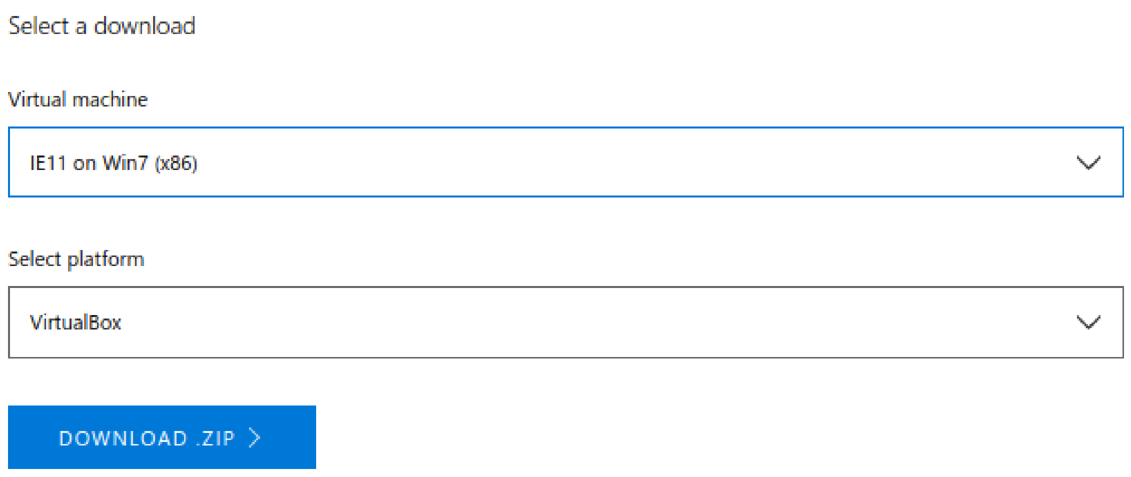
Test Kolibri in Windows guest

Contents

Download virtual machine images

We use Modern IE virtual machine images. Choose the virtual machine (VM) image you wish to test according to your host OS and download them to your computer.

1. Select Windows version virtual machine
2. Select VirtualBox platform


Select a download

Virtual machine

IE11 on Win7 (x86)

Select platform

VirtualBox

Note: We do not support Internet Explorer 10 or below, so make sure to download the VM image with IE11 or above.

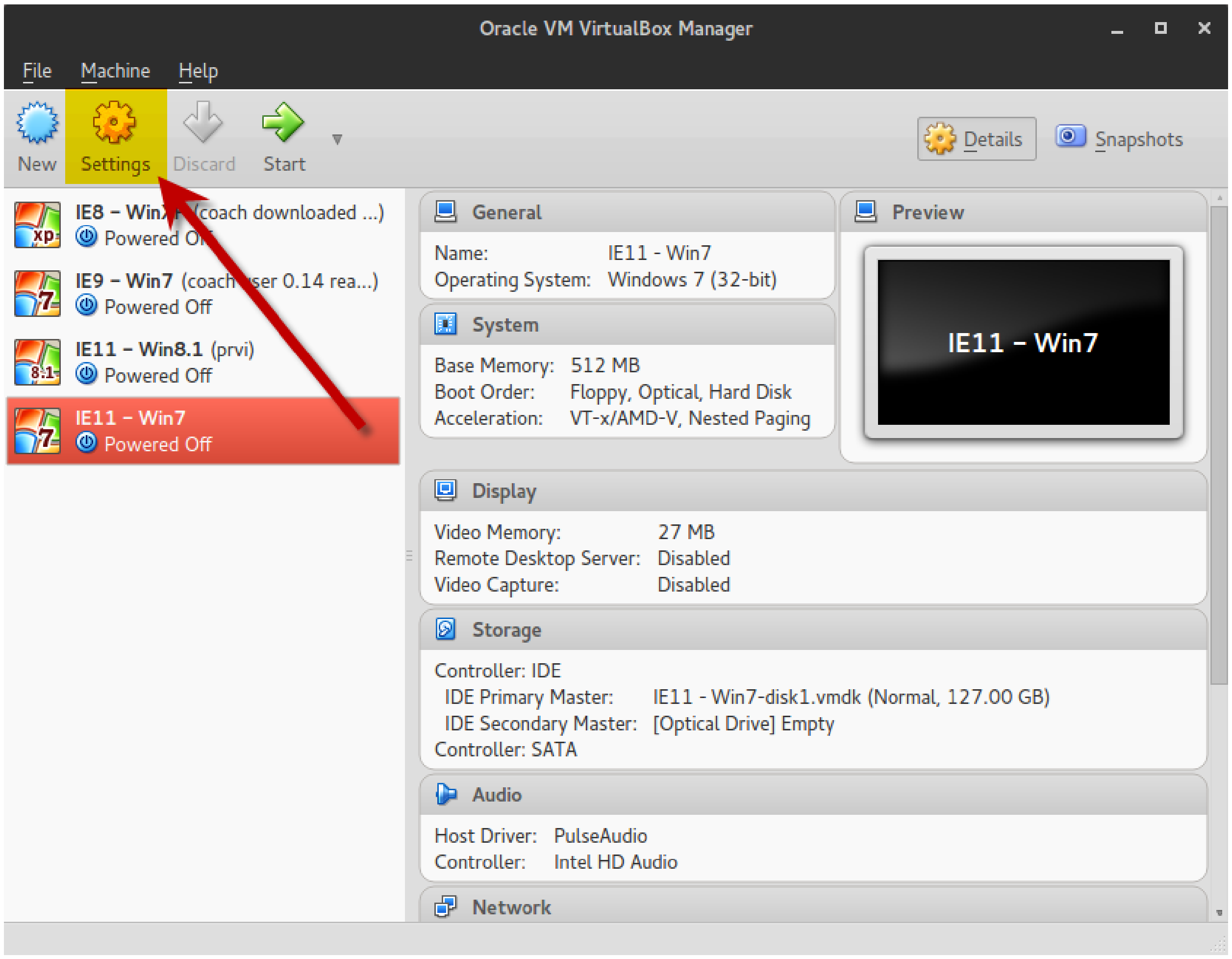
Import VM image into VirtualBox

Extract the contents of downloaded **.zip** file to obtain the corresponding VM image file with **.ova** extension.

Double-click the **.ova** file to import the VM into VirtualBox. You can change some options in the Import window, but it's best if you leave the default values and setup everything in the VirtualBox **Settings** once VM is already imported.

Configure virtual machines for testing

You can have several VMs imported in the VirtualBox Manager Window. Select the VM you want to use for testing and press the **Settings** button to start the configuration:



IE8 – WinXP (coach downloaded ...)
Powered Off

IE9 – Win7 (coach user 0.14 rea...)
Powered Off

IE11 – Win8.1 (prvi)
Powered Off

IE11 – Win7
Powered Off

General

Name: IE11 - Win7
Operating System: Windows 7 (32-bit)

System

Base Memory: 512 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging

Preview

IE11 – Win7

Display

Video Memory: 27 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Storage

Controller: IDE
IDE Primary Master: IE11 - Win7-disk1.vmdk (Normal, 127.00 GB)
IDE Secondary Master: [Optical Drive] Empty
Controller: SATA

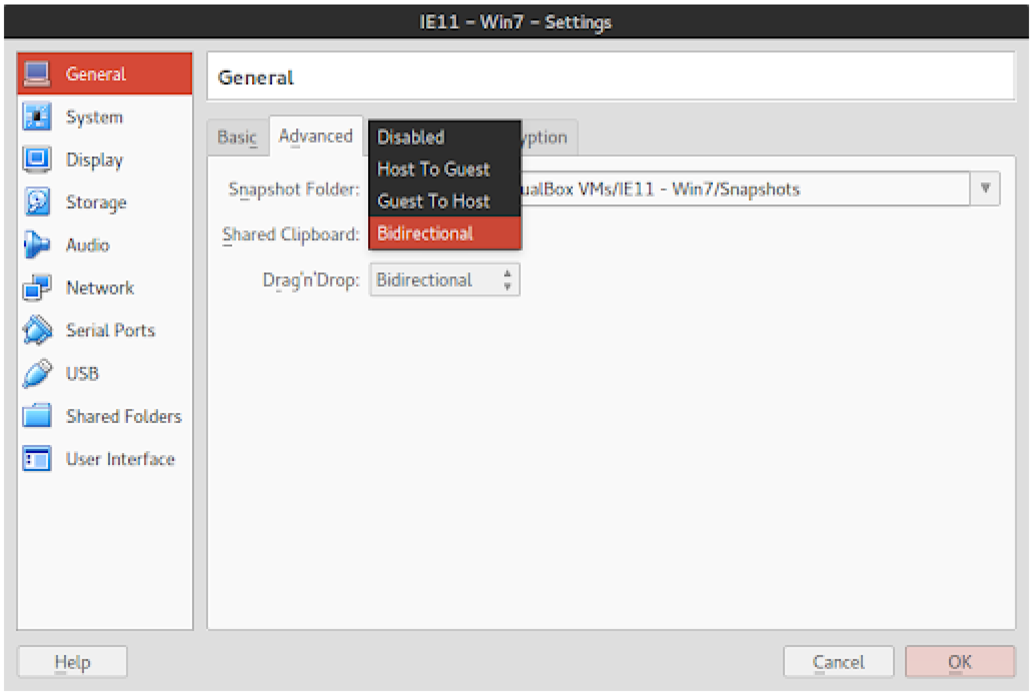
Audio

Host Driver: PulseAudio
Controller: Intel HD Audio

Network

1. Select the **General** pane and go to **Advanced** tab:

   In the **Shared Clipboard** and **Drag'n'Drop** drop-down menus select ***Bidirectional*** - this will allow you to use **Copy & Paste** functions between the VM and your host OS (very useful for copying code and error outputs back and forth! ;)

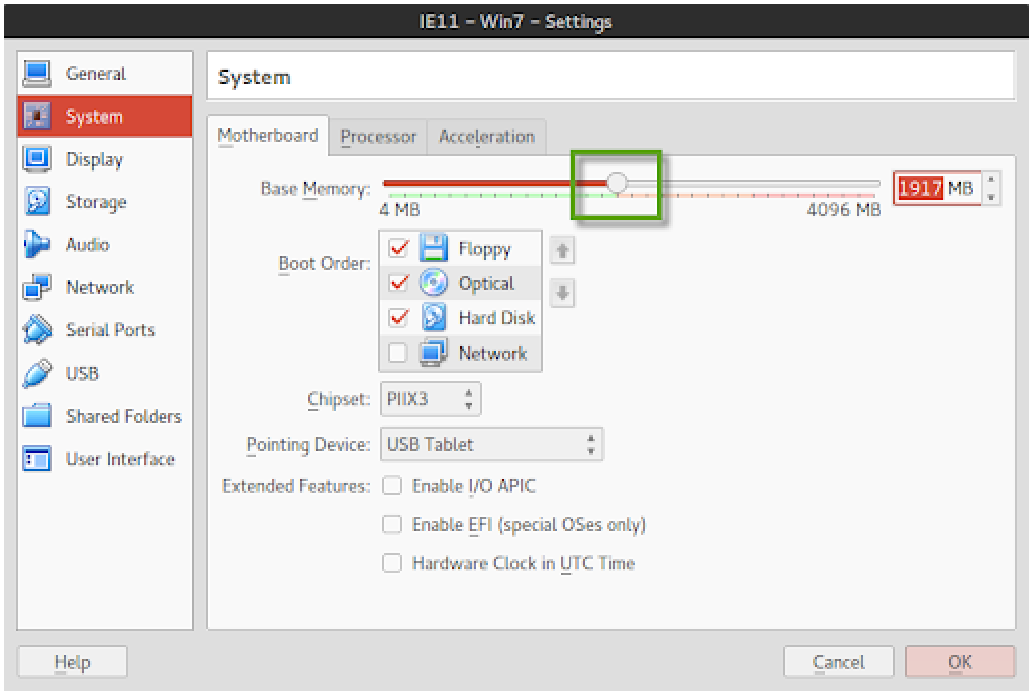


Snapshot Folder: ualBox VMs/IE11 - Win7/Snapshots

Shared Clipboard: Bidirectional

Drag'n'Drop: Bidirectional

2. Select the **System** pane: In the **Motherboard** tab allocate as much RAM as your OS can spare to VM's **Base Memory** with the slider:



Base Memory: 1917 MB

Boot Order: ✓ Floppy, ✓ Optical, ✓ Hard Disk, Network

Chipset: PIIX3

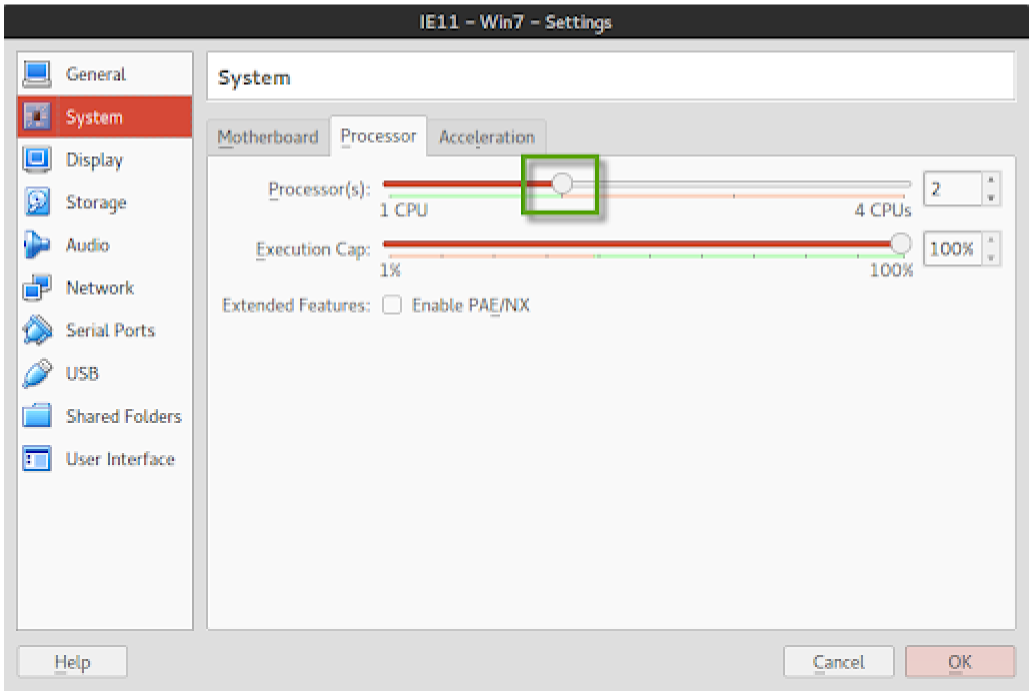
Pointing Device: USB Tablet

Extended Features: Enable I/O APIC

Enable EFI (special OSes only)

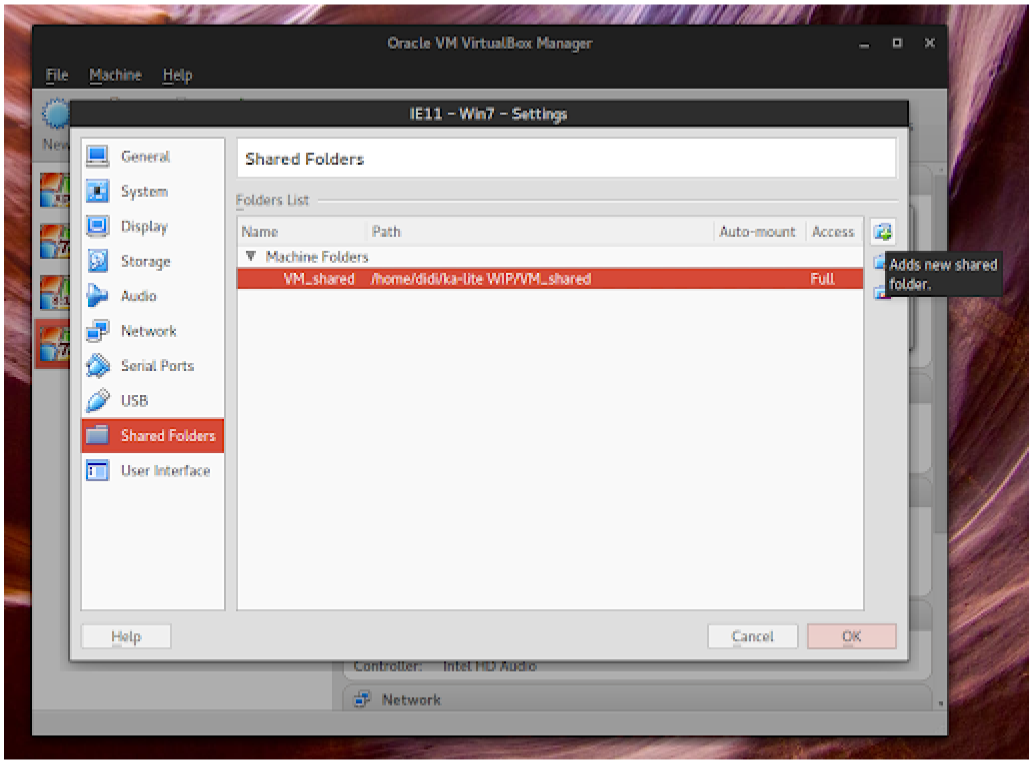
Hardware Clock in UTC Time

In the **Processor** tab allocate as many CPU's as your OS can spare to VM's **Processor(s)** with the slider:



Processor(s): 2
Execution Cap: 100%
Extended Features: Enable PAE/NX

3. In the **Shared Folders** pane add the folder you created previously where you keep Kolibri and other installers on your host machine that you want to make available for virtual machines.



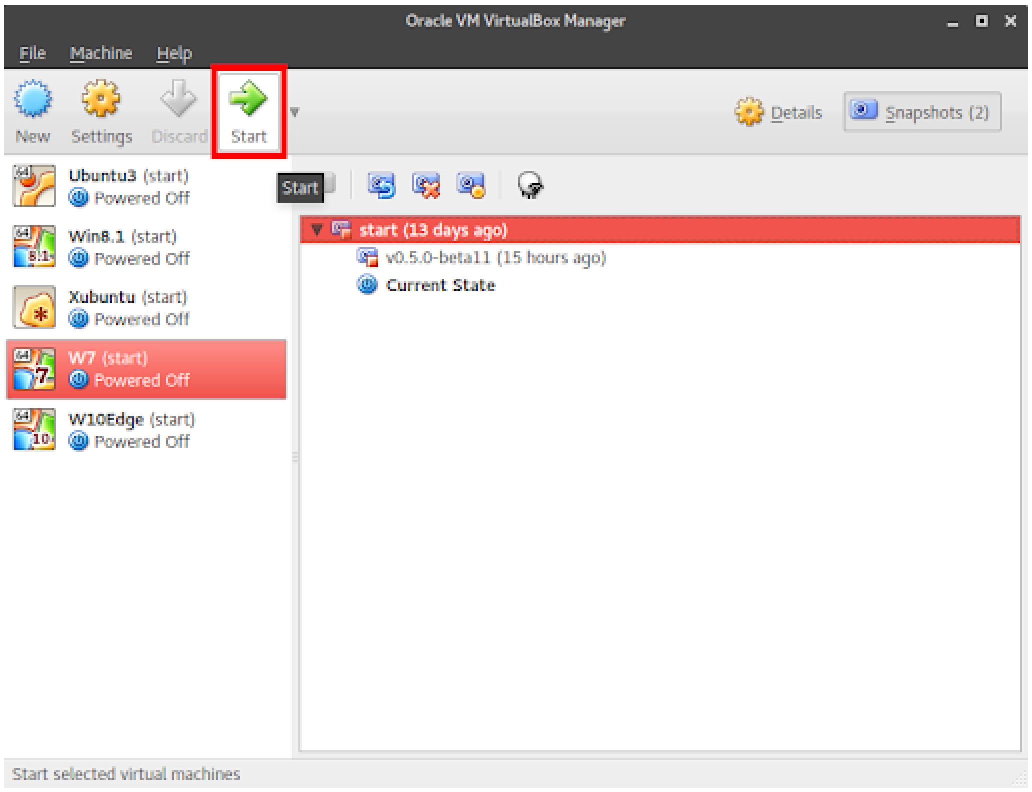
Name	Path	Auto-mount	Access
Machine Folders			
VM_shared	/home/didi/ka-lite WIP/VM_shared		Full

4. Press the **OK** button to save all the changes and close the ***Settings*** window.

   At this point your virtual machine is ready so you can start it and unleash the tester in you!

Start Virtual Machine

Press the green arrow button to start the selected VM.



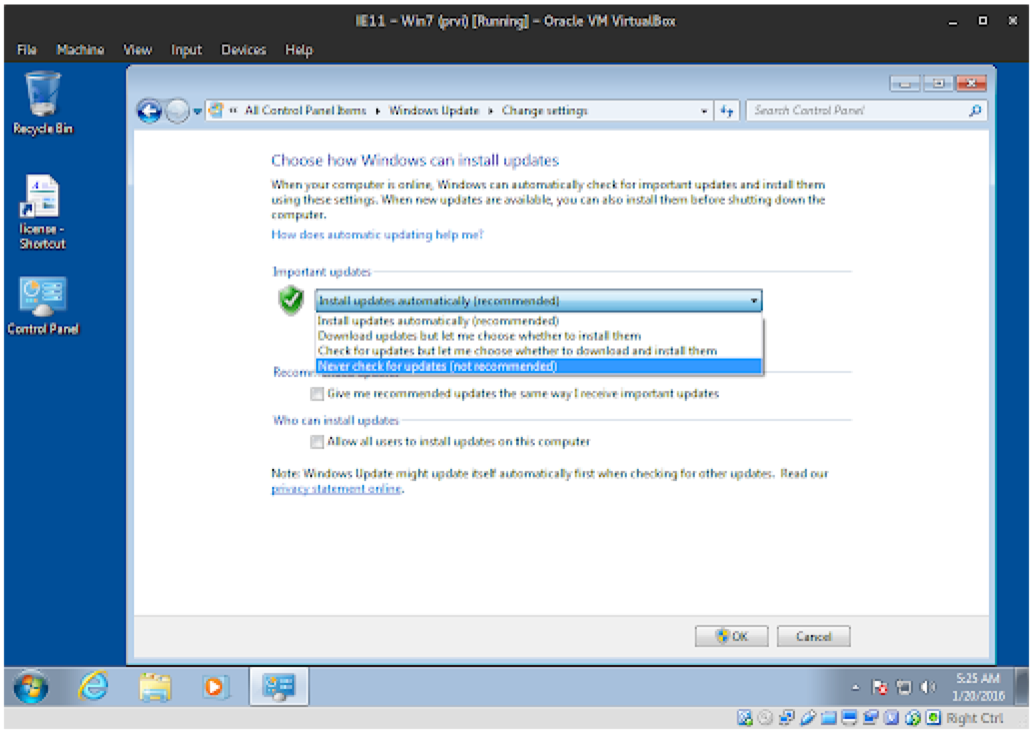
Recommendations for VM tuning prior to Kolibri installation

Manual testing is not a complex process, it usually involves a repetition of predetermined steps in a given testing scenario and recording the results, but it can be time consuming. Unless you are working with VirtualBox on a powerful host computer, VM will run more slowly. The following list of actions to take will help you tune the VM and make it as fast as possible. Apart from allocating as much RAM and processor power as your host OS can spare, you should also perform the following steps:

Disable Windows Update and Modules installer

Modern.ie VMs will come with Windows Update enabled and active by default just as any regular OS. Downloading and installing the updates will require time and occupy VM resources, which will slow down your testing process. Since it is unlikely that Kolibri installation will be somehow affected by VM without the latest Microsoft patches, you should disable Windows Update altogether.

1. Go to **Control Panel > Windows Update > Change settings** and select *Never check for updates* from the drop down menu. Press the **OK** button to save the selection.


IE11 - Win7 (prvi) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Recycle Bin
license - Shortcut
Control Panel
« All Control Panel Items ▸ Windows Update ▸ Change settings
Search Control Panel
OK
Cancel
5:25 AM
1/20/2016
Right Ctrl

Choose how Windows can install updates

When your computer is online, Windows can automatically check for important updates and install them using these settings. When new updates are available, you can also install them before shutting down the computer.

How does automatic updating help me?

Important updates

Install updates automatically (recommended)

Install updates automatically (recommended)
Download updates but let me choose whether to install them
Check for updates but let me choose whether to download and install them
Never check for updates (not recommended)

Recommended updates

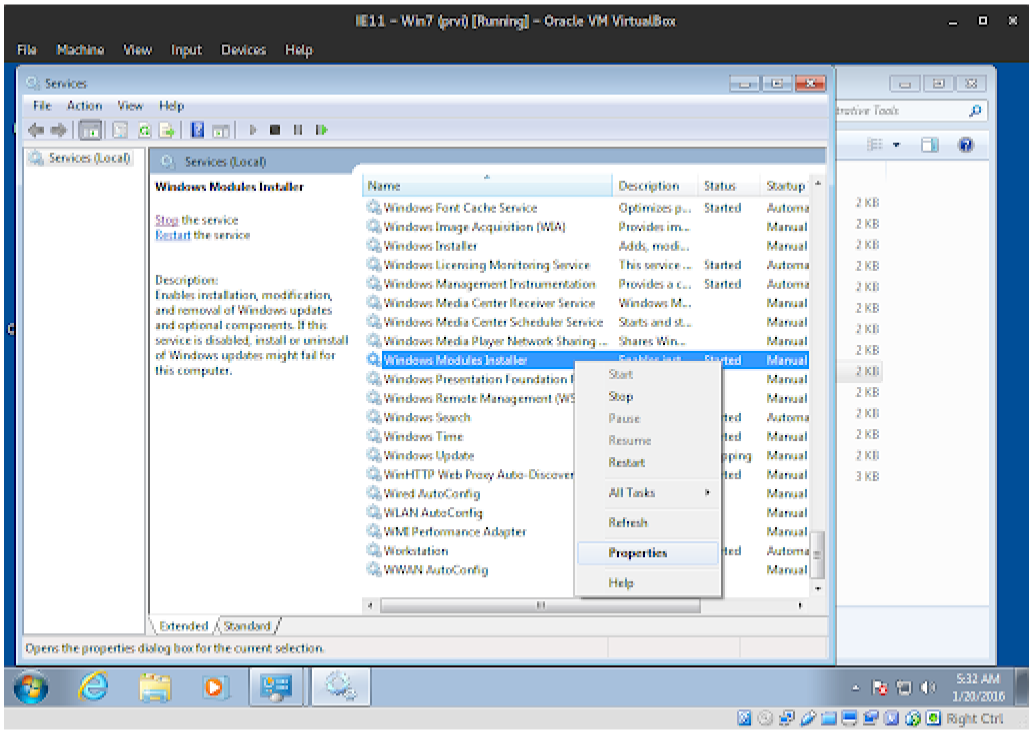
Give me recommended updates the same way I receive important updates

Who can install updates

Allow all users to install updates on this computer

Note: Windows Update might update itself automatically first when checking for other updates. Read our privacy statement online.

2. Unfortunately, previous step does not seem to be quite enough, so the fastest solution is to stop the two culprit services altogether. Go to **Control Panel > Administrative Tools > Services**, locate the **Windows Update** and **Windows Modules Installer** services on the list and right-click on each to open their *Properties* window:



Windows Modules Installer

Stop the service
Restart the service

Description:
Enables installation, modification, and removal of Windows updates and optional components. If this service is disabled, install or uninstall of Windows updates might fail for this computer.

Name	Description	Status	Startup
Windows Font Cache Service	Optimizes p...	Started	Automa
Windows Image Acquisition (WIA)	Provides im...		Manual
Windows Installer	Adds, modi...		Manual
Windows Licensing Monitoring Service	This service ...	Started	Automa
Windows Management Instrumentation	Provides a c...	Started	Automa
Windows Media Center Receiver Service	Windows M...		Manual
Windows Media Center Scheduler Service	Starts and st...		Manual
Windows Media Player Network Sharing ...	Shares Win...		Manual
Windows Modules Installer	Enables inst...	Started	Manual
Windows Presentation Foundation I			Manual
Windows Remote Management (WS			Manual
Windows Search		ted	Automa
Windows Time		ted	Manual
Windows Update		pping	Manual
WinHTTP Web Proxy Auto-Discover		ted	Manual
Wired AutoConfig			Manual
WLAN AutoConfig			Manual
WMI Performance Adapter			Manual
Workstation		ted	Automa
WWAN AutoConfig			Manual

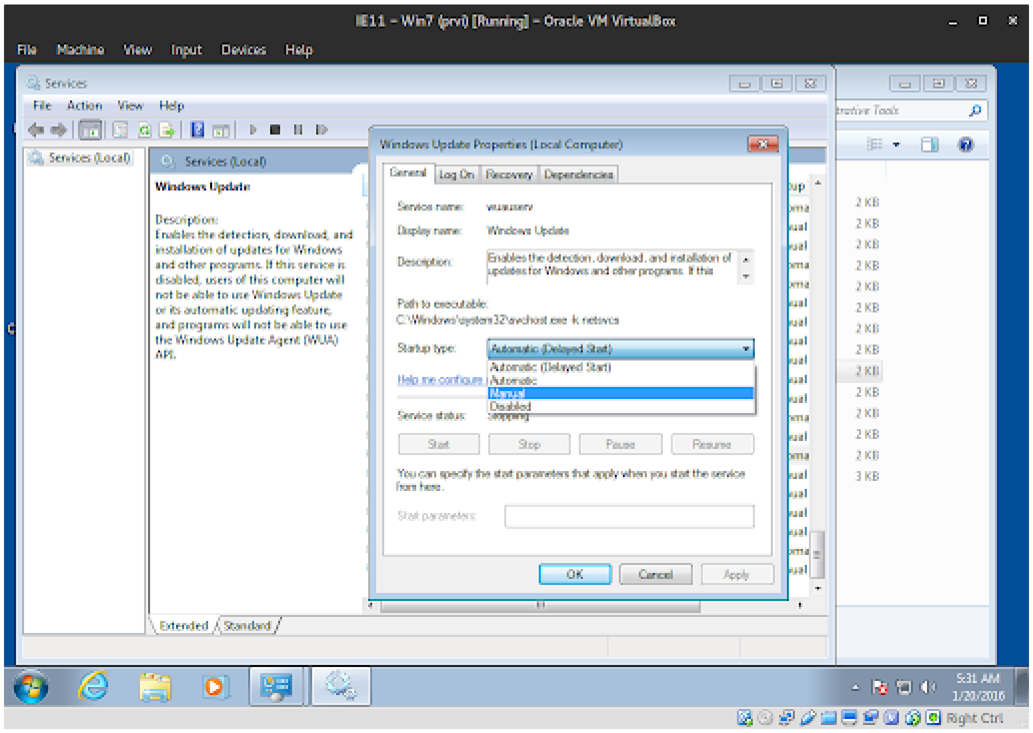
Opens the properties dialog box for the current selection.

a) Press the **Stop** button in the Properties window to stop the service.

b) Select **Manual** from the Startup type drop down menu.

Beware not to select **Disabled** as it may hinder the installation of Python.

c) Press buttons **Apply** and **OK**.


IE11 - Win7 (prvi) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Services
Services (Local)
Extended Standard
General Log On Recovery Dependencies
Start Stop Pause Resume
OK Cancel Apply
5:31 AM
1/20/2016
Right Ctrl

Windows Update

Description:
Enables the detection, download, and installation of updates for Windows and other programs. If this service is disabled, users of this computer will not be able to use Windows Update or its automatic updating feature, and programs will not be able to use the Windows Update Agent (WUA) API.

Windows Update Properties (Local Computer)

Service name: wuauserv

Display name: Windows Update

Description: Enables the detection, download, and installation of updates for Windows and other programs. If this

Path to executable:
C:\Windows\system32\svchost.exe -k netsvcs

Startup type: Automatic (Delayed Start)

- Automatic (Delayed Start)
- Automatic
- Manual
- Disabled

Help me configure

Service status: Stopping

You can specify the start parameters that apply when you start the service from here.

Start parameters:

3. Restart the VM. This way both **Windows Update** and **Windows Modules Installer** services will not hog the resources on your VM anymore and testing will be much faster!

Install additional browsers - Mozilla Firefox & Google Chrome

---

Tip: Keep downloaded browser installers in the same folder designated as Shared with your VMs

---

Apart from Internet Explorer that comes by default with Windows, you should test Kolibri on other browsers like Firefox and Chrome.

It is also recommended that you install basic FF addons & Chrome extensions that will help you work faster and collect better information.

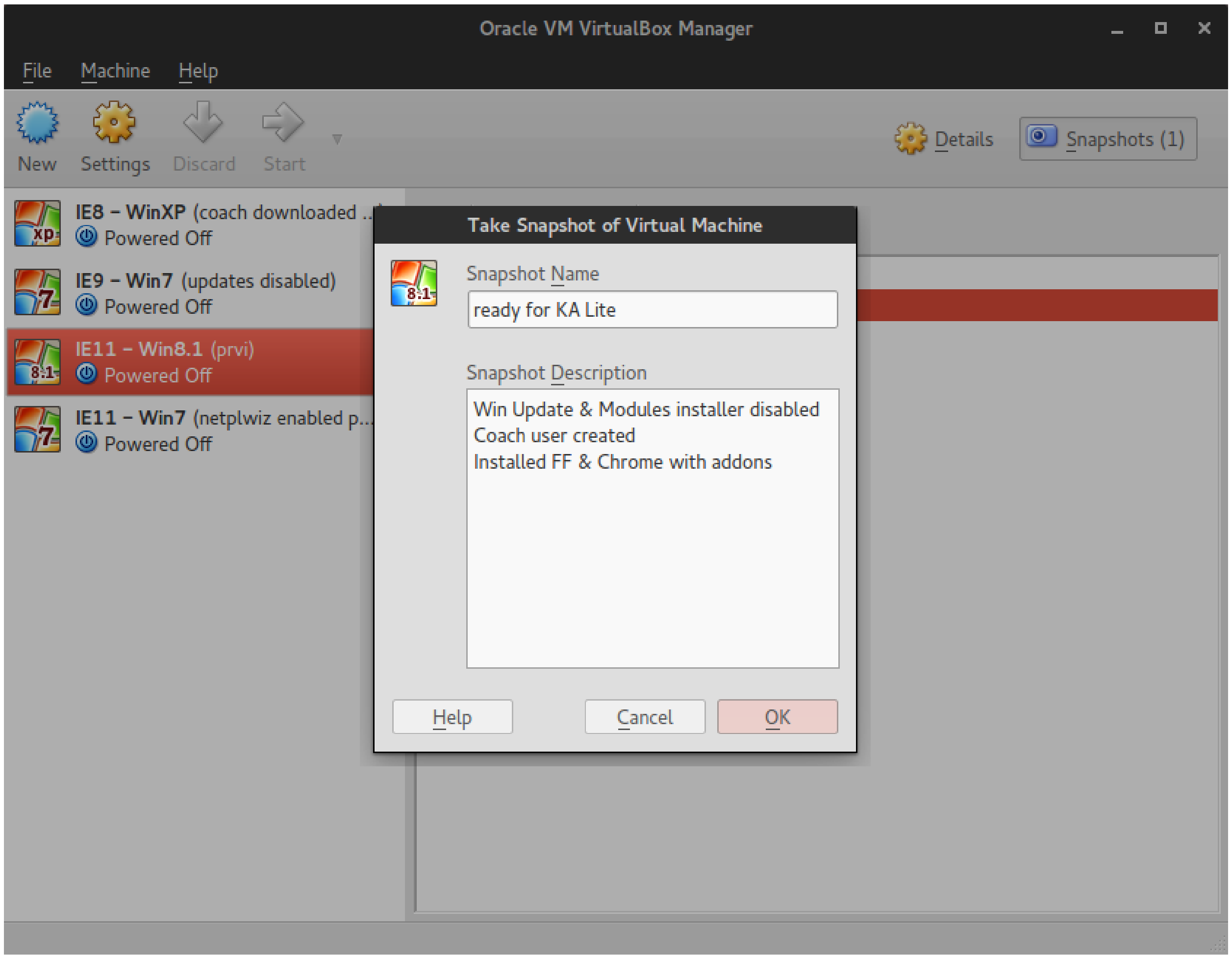
Recommended addons/extensions

TODO

Install Kolibri

Tip: Make a "baseline" snapshot of your VM prior to installing Kolibri the first time

After you've tuned and optimized your VM and installed browsers, make one **Snapshot** before you install Kolibri. This will be your baseline VM snapshot upon which you will install each Kolibri version and restore to it to install the next one.



ready for KA Lite

Win Update & Modules installer disabled
Coach user created
Installed FF & Chrome with addons

Download minimal content

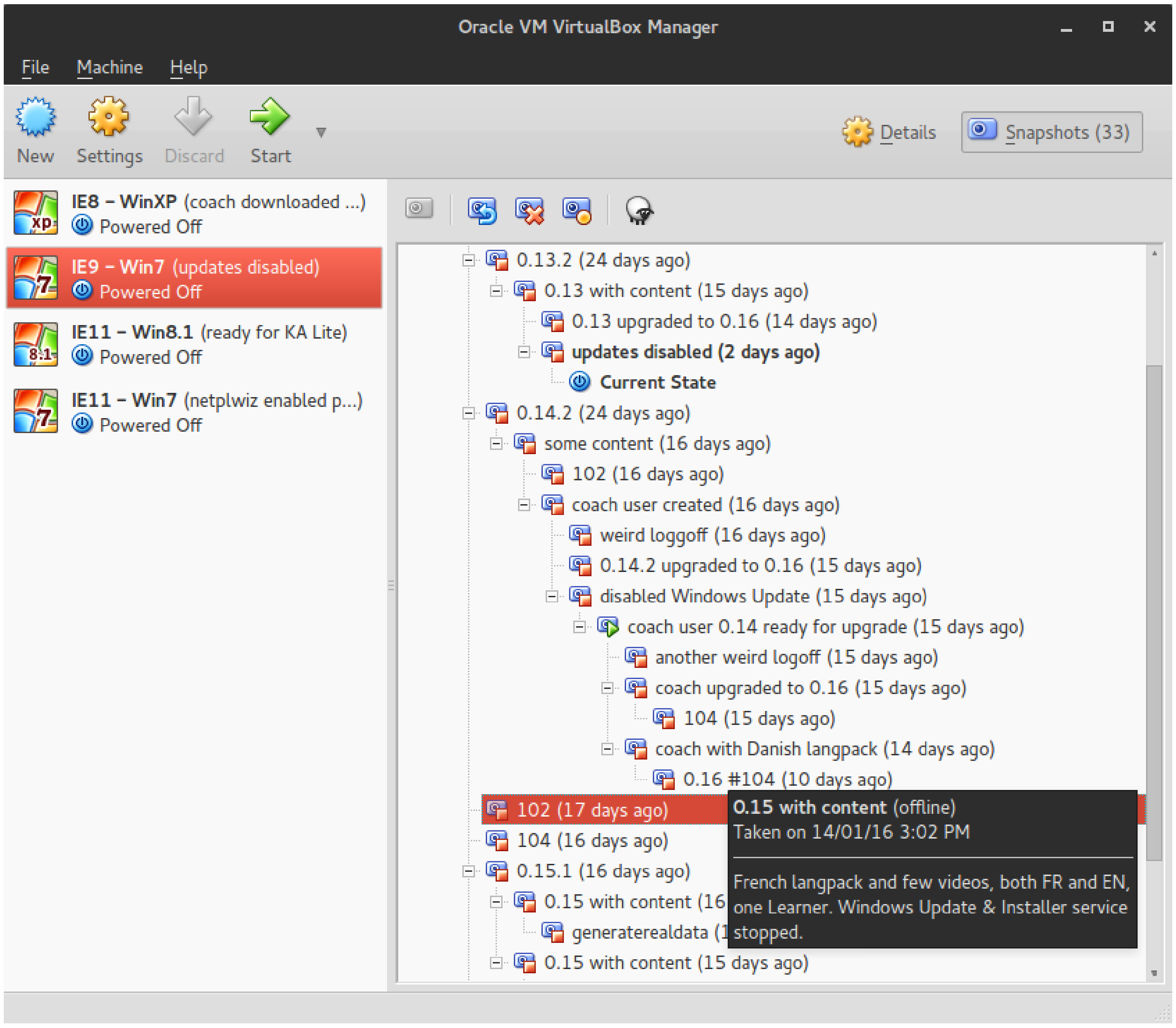
1. Download this small testing channel, or others:
   - `nakav-mafak` - Kolibri QA Channel (~250MB)
2. Use the following command to create Kolibri users for testing.

   ```
   kolibri manage generateuserdata
   ```

   command to create Kolibri users for testing.
3. Login as Coach user and create groups and exams as those are not (yet) available through the previous automatic command.
4. Happy testing!.

Note: When you finish testing one particular release, save it as a VM Snapshot so you can revert to it if necessary.

The goal is to have baseline snapshots for the initial (system "ready") state, and new ones for successive test cases. Since we are in early development phase, you will not have a lot of upgrade scenarios that I used to have while testing KA Lite:



IE8 – WinXP (coach downloaded ...)
Powered Off
IE9 – Win7 (updates disabled)
Powered Off
IE11 – Win8.1 (ready for KA Lite)
Powered Off
IE11 – Win7 (netplwiz enabled p...)
Powered Off

0.13.2 (24 days ago)
0.13 with content (15 days ago)
0.13 upgraded to 0.16 (14 days ago)
updates disabled (2 days ago)
Current State
0.14.2 (24 days ago)
some content (16 days ago)
102 (16 days ago)
coach user created (16 days ago)
weird loggoff (16 days ago)
0.14.2 upgraded to 0.16 (15 days ago)
disabled Windows Update (15 days ago)
coach user 0.14 ready for upgrade (15 days ago)
another weird logoff (15 days ago)
coach upgraded to 0.16 (15 days ago)
104 (15 days ago)
coach with Danish langpack (14 days ago)
0.16 #104 (10 days ago)
102 (17 days ago)
104 (16 days ago)
0.15.1 (16 days ago)
0.15 with content (16
generaterealdata (1
0.15 with content (15 days ago)

0.15 with content (offline)
Taken on 14/01/16 3:02 PM

French langpack and few videos, both FR and EN, one Learner. Windows Update & Installer service stopped.

Tip: Delete unnecessary snapshots because they can occupy a lot of disc space over time.

Use the recommended format for filing Issues on GitHub.

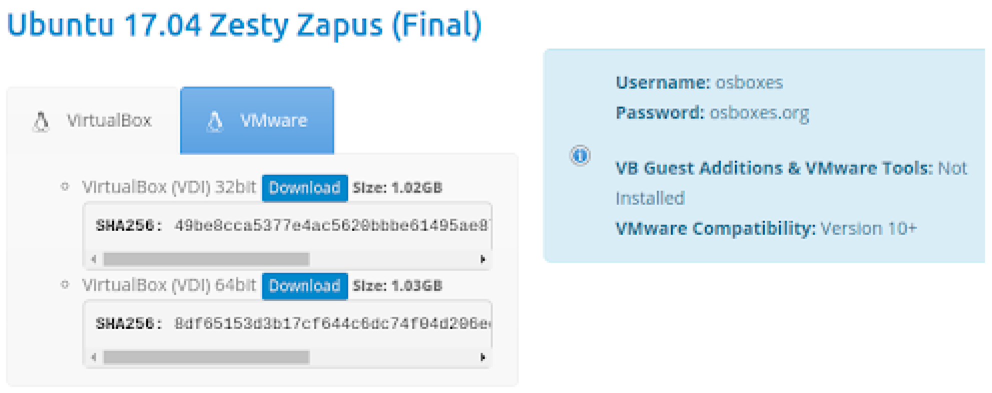
Use browser debugging tools and screenshots to better illustrate the problem.

Test Kolibri in Linux guest

Download virtual machine images

You can download the ISO image and install any Linux distro the usual way, but ready-to-use images to save time it is recommended to and. Choose the virtual machine (VM) image you wish to test according to your host OS and download them to your computer.

1. Select Ubuntu version
2. Select VirtualBox platform
3. Select

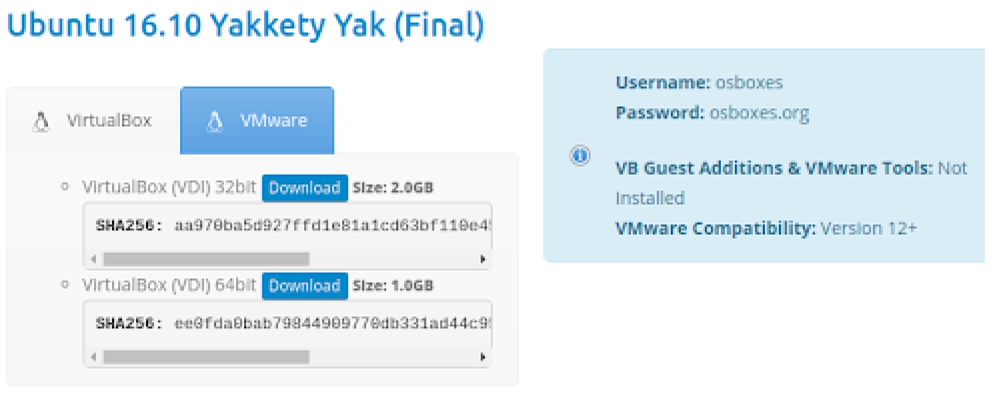


Ubuntu 17.04 Zesty Zapus (Final)

- VirtualBox (VDI) 32bit Download Size: 1.02GB
  `SHA256: 49be8cca5377e4ac5620bbbe61495ae8`
- VirtualBox (VDI) 64bit Download Size: 1.03GB
  `SHA256: 8df65153d3b17cf644c6dc74f04d206e`

Username: osboxes
Password: osboxes.org

VB Guest Additions & VMware Tools: Not Installed
VMware Compatibility: Version 10+



Ubuntu 16.10 Yakkety Yak (Final)

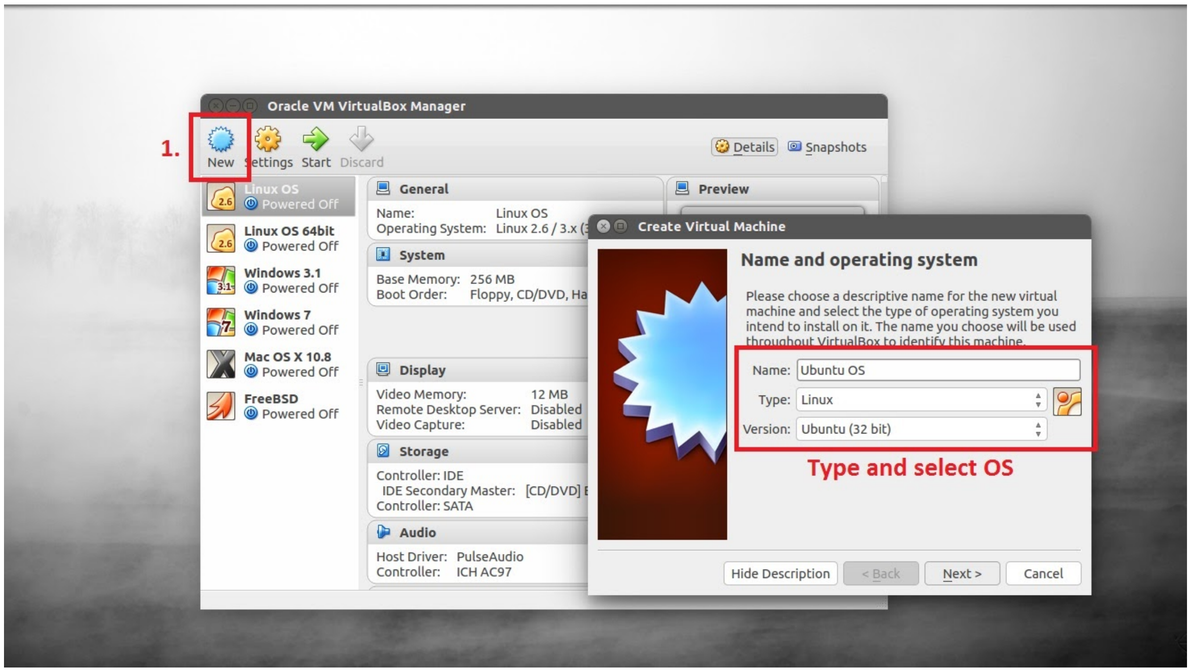
- VirtualBox (VDI) 32bit Download Size: 2.0GB
  `SHA256: aa970ba5d927ffd1e81a1cd63bf110e4`
- VirtualBox (VDI) 64bit Download Size: 1.0GB
  `SHA256: ee0fda0bab79844909770db331ad44c9`

Username: osboxes
Password: osboxes.org

VB Guest Additions & VMware Tools: Not Installed
VMware Compatibility: Version 12+

Import and configure Ubuntu VM image into VirtualBox

1. Extract the contents of downloaded **.7z** file to obtain the corresponding VM image file with **.vdi** extension.
2. Open VirtualBox click on **New** button.
3. Type OS Name, select OS Type and click **Next**.

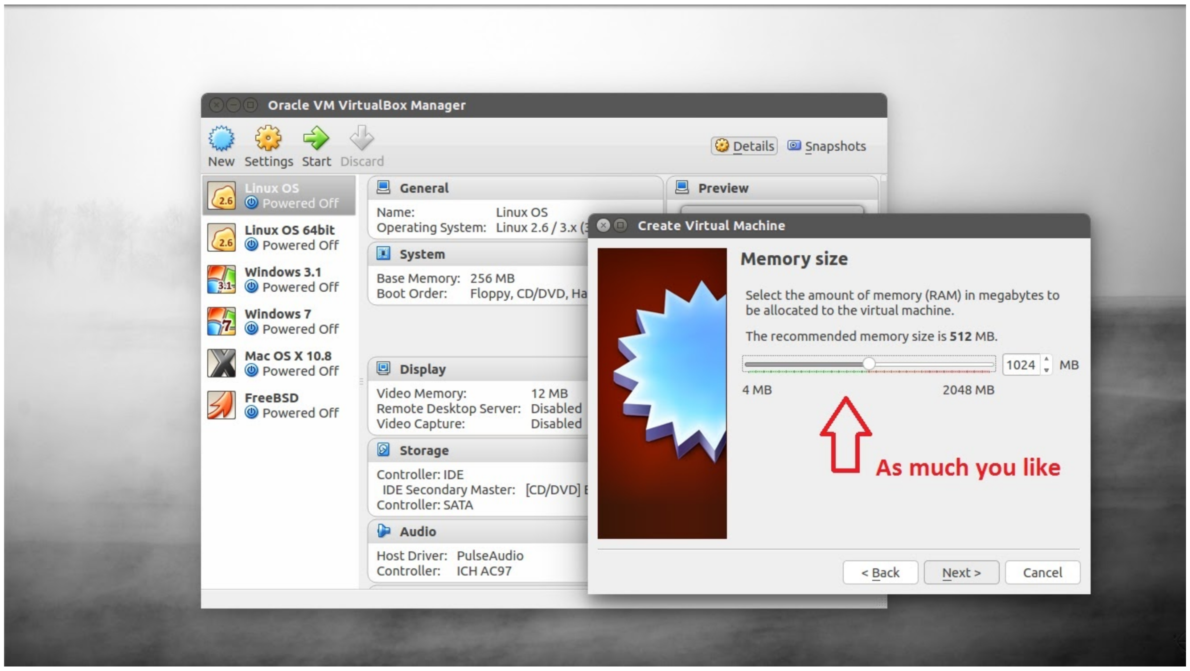


Name and operating system

Please choose a descriptive name for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name: Ubuntu OS
Type: Linux
Version: Ubuntu (32 bit)

4. Set available RAM.



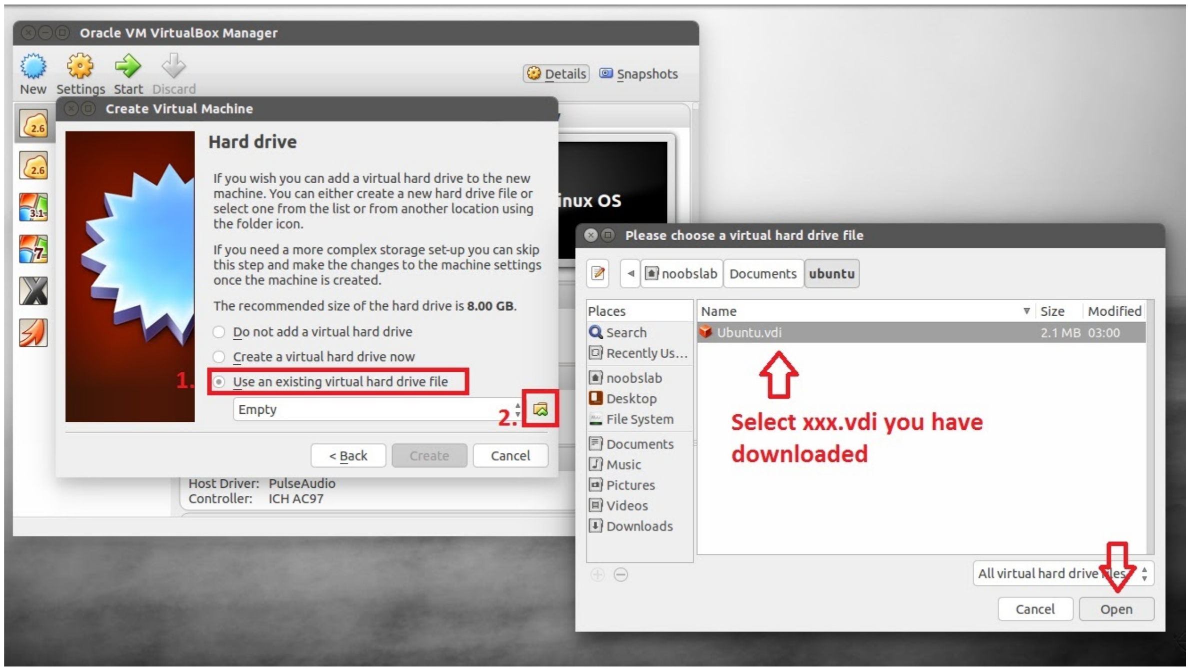
Memory size

Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.

The recommended memory size is **512** MB.

1024 MB

5. Select **Use an existing virtual hard drive file**, browse to where VDI image is located.



Hard drive

If you wish you can add a virtual hard drive to the new machine. You can either create a new hard drive file or select one from the list or from another location using the folder icon.

If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

The recommended size of the hard drive is **8.00 GB**.

- Do not add a virtual hard drive
- Create a virtual hard drive now
- [✓ Use an existing virtual hard drive file]

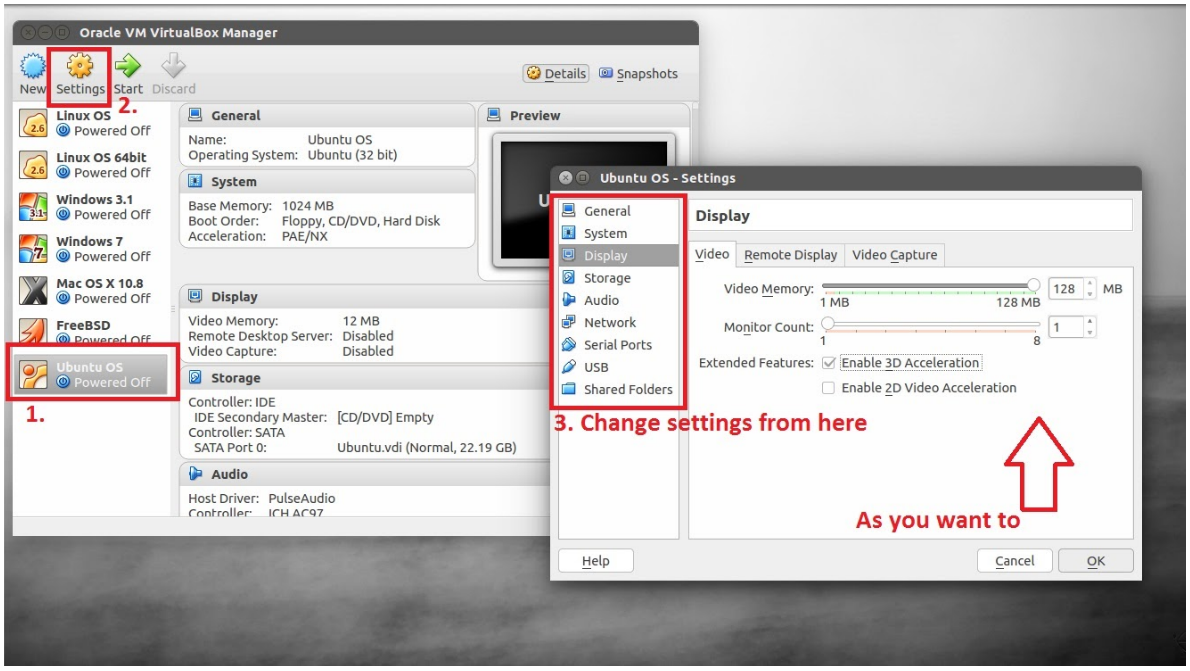
Empty

Host Driver: PulseAudio
Controller: ICH AC97

Name	Size	Modified
Ubuntu.vdi	2.1 MB	03:00

Select xxx.vdi you have downloaded

6. Click on **Settings** and amplify video memory for VM, and **Enable 3D Acceleration**.



Linux OS — Powered Off
Linux OS 64bit — Powered Off
Windows 3.1 — Powered Off
Windows 7 — Powered Off
Mac OS X 10.8 — Powered Off
FreeBSD — Powered Off
Ubuntu OS — Powered Off

General
Name: Ubuntu OS
Operating System: Ubuntu (32 bit)

System
Base Memory: 1024 MB
Boot Order: Floppy, CD/DVD, Hard Disk
Acceleration: PAE/NX

Display
Video Memory: 12 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Storage
Controller: IDE
IDE Secondary Master: [CD/DVD] Empty
Controller: SATA
SATA Port 0: Ubuntu.vdi (Normal, 22.19 GB)

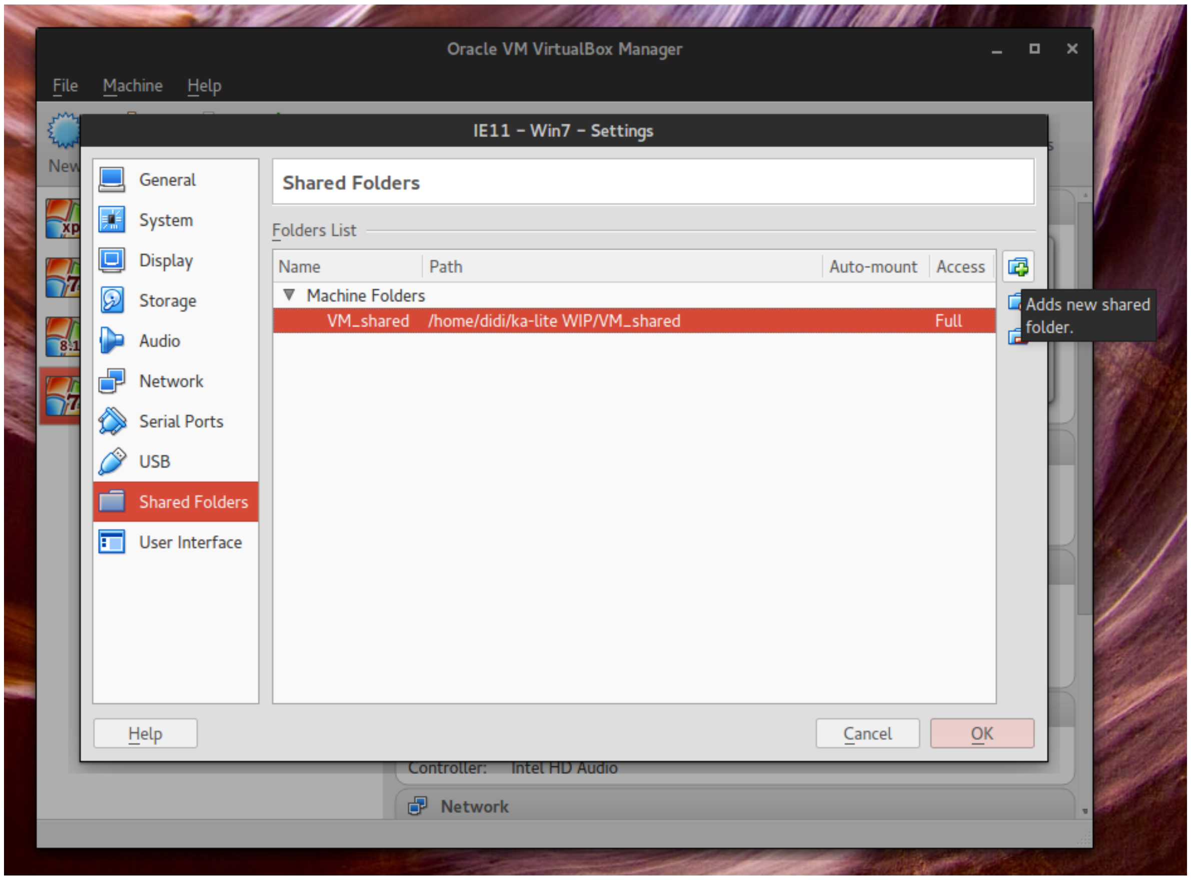
Audio
Host Driver: PulseAudio
Controller: ICH AC97

Display
Video Memory: 128 MB (1 MB – 128 MB)
Monitor Count: 1 (1 – 8)
Extended Features: [✓ Enable 3D Acceleration] Enable 2D Video Acceleration

3. Change settings from here

As you want to

7. In the **Shared Folders** pane add the folder you created previously where you keep Kolibri and other installers on your host machine that you want to make available for virtual machines.



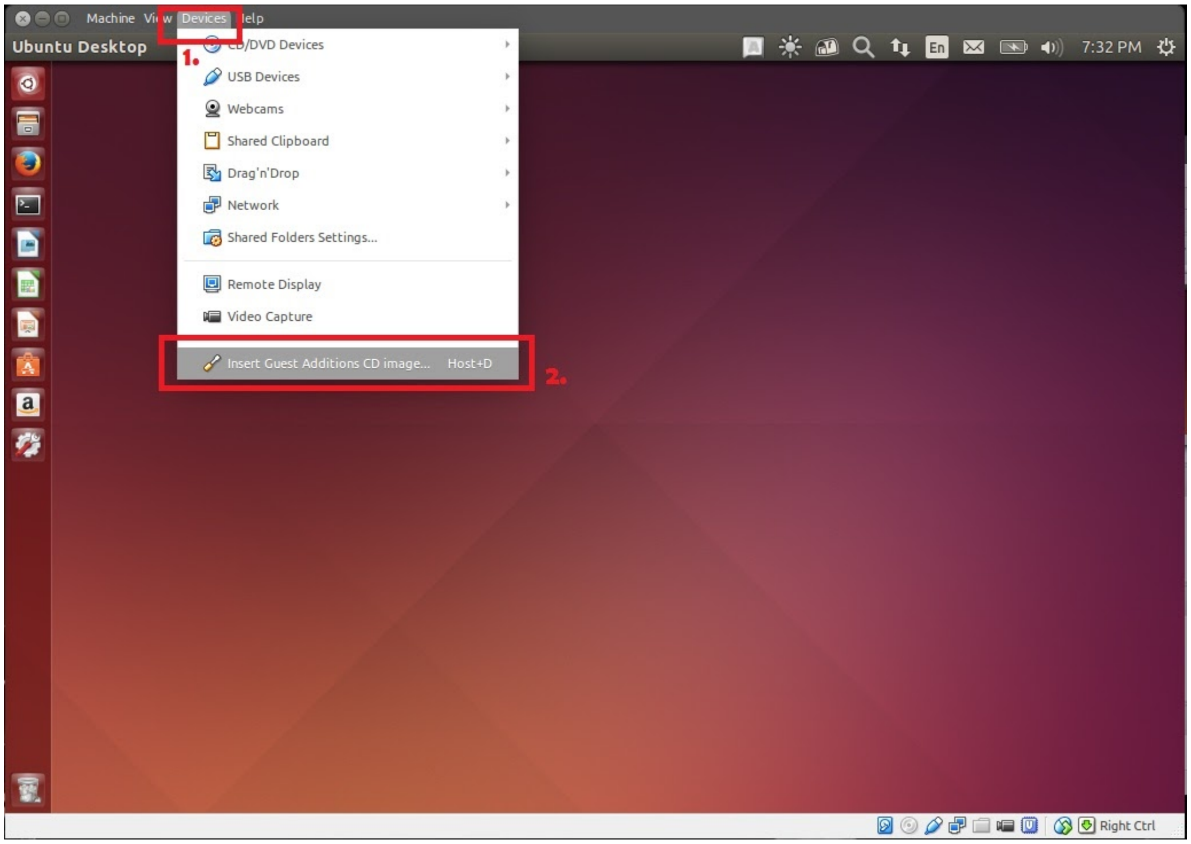
Shared Folders

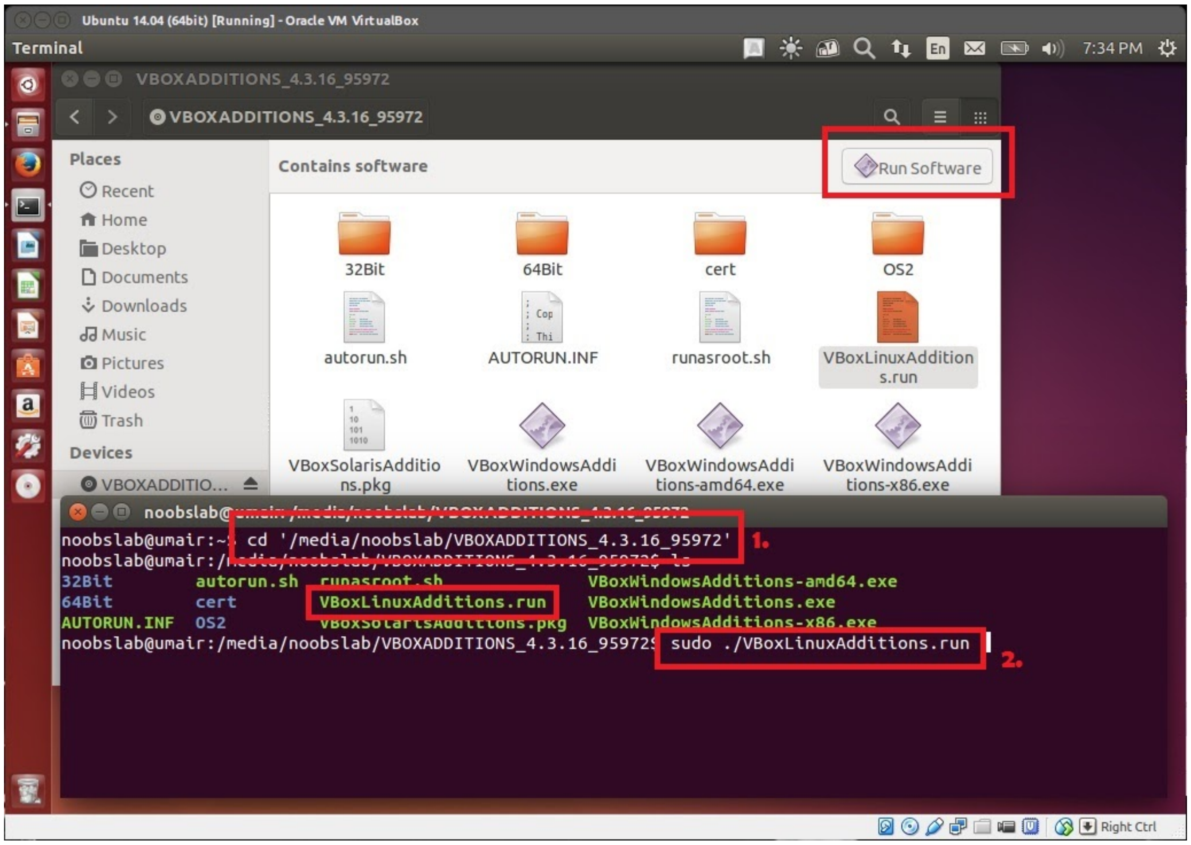
Folders List

Name	Path	Auto-mount	Access
Machine Folders			
VM_shared	/home/didi/ka-lite WIP/VM_shared		Full

Adds new shared folder.

8. Start the newly imported VM.
9. Open **Devices** menu and select **Insert Guest Additions CD image…** option.





```
noobslab@umair:~$ cd '/media/noobslab/VBOXADDITIONS_4.3.16_95972'
noobslab@umair:/media/noobslab/VBOXADDITIONS_4.3.16_95972$ ls
32Bit        autorun.sh  runasroot.sh              VBoxWindowsAdditions-amd64.exe
64Bit        cert        VBoxLinuxAdditions.run    VBoxWindowsAdditions.exe
AUTORUN.INF  OS2         VBoxSolarisAdditions.pkg  VBoxWindowsAdditions-x86.exe
noobslab@umair:/media/noobslab/VBOXADDITIONS_4.3.16_95972$ sudo ./VBoxLinuxAdditions.run
```

10. Add your Ubuntu user in *vboxsf* group to access VirtualBox shared folder in Ubuntu guest. Open Terminal and run:

    ```
    sudo adduser <username> vboxsf
    ```

11. Reboot and you'll be able to find and open the folder shared in VirtualBox **Settings**, under the **Network** in Ubuntu guest.
12. Happy testing!

Test Kolibri in OSX guest

Coming soon!

2.12.3 Testing Kolibri with app plugin enabled

The Kolibri app plugin

The Kolibri app plugin is designed to provide features and behavior with the mobile app user in mind. In order to test or develop Kolibri in this mode, there are commands that can be used to initialize Kolibri as needed.

By running the command: *yarn app-python-devserver* you will start Kolibri in development mode. You can also run *yarn app-devserver* to run the frontend devserver in parallel.

When you start the server with these commands, you will see a message with a URL pointing to *http://127.0.0.1:8000/app/api/initialize/<some token>* - visiting this URL will set your browser so that it can interact with Kolibri as it runs with the app plugin. **You will only have to do this once unless you clear your browser storage.**

2.12.4 Recommended A11y tools

Style Guides

A11Y Style Guide

Living style guide, generated from KSS documented styles…with an accessibility twist.

Firefox Add-ons

WAVE Web Accessibility Extension

When activated, the WAVE extension injects icons and indicators into your page to give feedback about accessibility and to facilitate manual evaluation.

aXe Accessibility Engine

Open-source accessibility testing tool by Deque.

WCAG Contrast checker

Very complete sidebar contrast checker, on top of which is provided a filter to simulate 4 types of colorblindness.

Chrome Extensions

WAVE Evaluation Tool

Web accessibility evaluation tool developed by WebAIM.org.

Accessibility Developer Tools

Adds an Accessibility audit and an Accessibility sidebar pane to the Elements tab of your Chrome Developer Tools.

aXe Accessibility Engine

Open-source accessibility testing tool by Deque.

Accessibility monitor

Continuously monitor accessibility failures in a page as it's being used, rather than a single audit on page load.

NoCoffee vision simulator

Evaluation tool helpful for understanding several vision problems and deficiencies.

Spectrum

Evaluation tool for different types of color vision deficiency.

WCAG Luminosity Contrast Ratio Analyzer

Pick colors, compute contrast, get suggestions & preview with challenged visions.

Bookmarklets/Favelets (browser independent)

tota11y - an accessibility visualization toolkit

Helps visualize the most common accessibility violations (and successes).

HTML_CodeSniffer

Detects violations of both Web Content Accessibility Guidelines (WCAG) 2.0 (all three conformance levels), and the web-related components of the U.S. “Section 508” legislation.

Visual ARIA Bookmarklet

Allows any sighted person to physically see the use of ARIA on public websites.

Jim Thatcher’s Favelets

Several handy a11y assessment favelets/bookmarklets.

Color Contrast tools

Contrast Ratio

By Lea Verou.

Accessibility Color Wheel

Find an accessible color pair and compare contrast with simulation of three types of color deficiency: deuteranopia, protanopia and tritanopia.

Color Safe

Accessible color palettes based on WCAG Guidelines of text and background contrast ratios.

Color Palette Accessibility Evaluator

Online tool for analyzing color combinations that meet WCAG 2 a11y specifications.

Contrast Analyzer

Standalone tool; provides a pass/fail check for WCAG 2.0 contrast criteria; simulates certain visual conditions.

Online A11y validation tools

AChecker

Full HTML, CSS, WCAG & Section 508 online assessment tool.

Functional Accessibility Evaluator 2.0

Sitewide evaluation and reports (requires registration).

Cynthia Says

TENON

Automated A11y testing

Choosing an Automated Accessibility Testing Tool: 13 Questions you should ask

axe-core

The Accessibility Engine for automated testing of HTML-based user interfaces.

pa11ly

Requires Node.js and PhantomJS; Custom Reporters option .

Tanaguru

Very complete website a11y assessment tool; basic and advanced Scenario audits based on Selenium).

2.13 Release Notes

List of the most important changes for each release.

2.13.1 0.16.0

Features

Robust syncing of user data and resources

Support for quick learner setup and independent learners

- Kolibri has a new onboarding experience which allows joining a facility, and streamlines getting started as an independent learner with a rapid "on my own setup" option
- Independent learners can transfer their existing data and learning progress to a facility. ##### Resource discovery
- Assigned lesson and quiz resources are now automatically transferred to learner devices, allowing coaches to dynamically manage learner content, rather than an administrator needing to import all content devices before distribution.
- Administrators and independent learners are now able to view other Kolibri Libraries on their local network and browse their resources, without having to import content. If they are connected to the internet, they will be able to browse resources on the Kolibri Content Library (hosted on Kolibri Studio).
- Administrators can allow learners to download resources from other Kolibri Libraries to their device to view within Kolibri, even when they are no longer on the same network. ##### Support for administrators
- Administrators have a new option to add a PIN on learner-only devices, which allows an administrator easy access to the Device page while preventing learners from inadvertently making changes.
- Administrators are now able to schedule syncing of facility data on a recurring basis at custom intervals.
- When exporting log files, administrators are able to select the date range for the logs. ##### Practice quizzes
- This release supports practice quizzes, which are resources in the format of quizzes that learners can take in preparation for an assessment. They are able to see their score, and retry as many times as they would like, independently. Practice quiz resources are available through the Library, or can be assigned as part of a lesson. The same questions can also be assigned as a coach assigned quiz as a standardized assessment.

Changes

Dev documentation/dev updates

- Updated node version to 18
- Getting started documentation updated
- Updated to Webpack 5
- Created Github actions for build pipeline
- Created Github action to add assets to PRs
- Task API changes have been finalized after initial work in 0.15. Documentation is now updated to describe how to interact with the API and define tasks in plugins.

Architectural changes

- There is a new page architecture that is used across all Kolibri plugins, and the component has been removed. (Selected relevant high level issues and PRs: #9102, #9128, 9134.)
- The Kolibri Process Bus has been updated to support easier composability for custom deployment architectures.
- Conditional promises have been removed.
- To support the new onboarding process for Kolibri, Kolibri apps can now access a capability to provide access controls based on the currently active operating system user.

API Breaking Changes

- Tasks API has now been finalized, previous methods for interacting with tasks that do not use the pluggable Tasks API have been removed.
- The drive info endpoint has been moved the into the device app but functionality remains the same
- The API for coordinating learner only device synchronization within a local area network has been updated to ensure robust and reliable syncing. Any users wishing to use learner only device synchronization must update all Kolibri devices to this newer version

API Additions (non-breaking changes)

- REST API for enabling and disabling plugins
- Add API endpoint and hook driven capability for UI initiated device restart
- Public signup viewset
- Public content metadata endpoints to support granular resource import

Accessibility improvements

- Landmarks have been added and refined across the Library page and its related subpages, for better accessibility. This is a first step in support of more robust accessibility support, particularly in terms of page navigation for screen reader users.

Deprecations

- Support for Python 2.7 will be dropped in the upcoming version, 0.17. Upgrade your Python version to Python 3.6+ to continue working with Kolibri. More recent versions of Python 3 are recommended.
- Support for this Internet Explorer 11 will be dropped in the upcoming version, 0.17. We recommend installing other browsers, such as Mozilla Firefox or Google Chrome, in order to continue working with Kolibri.

Kolibry Design System upgrades

- Kolibri is now using kolibri-design-system v2.0.0 (a major version upgrade!). Please see the KDS release's Github page for documentation and full details about breaking changes and new features.

2.13.2 0.15.12

Added

- Added localization support for Haitian Creole
- Added annotation layer to PDF viewer

Changed

- Updated PID file when the zipcontent server starts

Fixed

- Ensure `startremotecontentimport` and `startdiskcontentimport` pass through the `fail_on_error` option to the importcontent command

2.13.3 0.15.11

Fixed

- Fixed progress tracking edge case where float rounding issues prevent progress reaching 100%

2.13.4 0.15.10

Added

- Add PDF accessibility support for screen readers
- Add support for captions for audio

Fixed

- Fixed overflowing title alignment on content cards
- Improved visible focus outline
- Fixed positioning of transcript layout when language is set to a right-to-left language
- Fixed calculation for number of users displayed on the User Tables

Changed

- Only display the completion modal on the finish event when the resource is also complete

2.13.5 0.15.9

Added

- Specified pre-commit hook python version to 3.10
- Added Python3.11 to supported python versions ### Fixed
- Fixed PDF completion issues
- Fixed learner-facing metadata display of content duration
- Fixed "Mark as complete" functionality to allow learners to mark resources as complete when allowed by the resource
- Disable forward/back buttons on EPUB renderer until locations are properly loaded
- Fix issue that causes learners to skip every other question in an exercise
- Fix searchbox outline
- Fix title spacing in app bar
- Fix bookmark data loading issues that caused inaccurate bookmark display ### Changed
- Changed __init__.py from 5 tuple to 3
- Set a max width on Library main content grid to display properly on extra large monitors
- Remove "All options" from filters in Learn search/filtering side panel
- Switch display of the completion modal to require both completed progress and the resource to be finished
- Add tests to assert totalattempts behaviour
- Display completion modals only on first completion, and allow user to reopen the modal if needed
- Update category search for each level to be searchable
- Update KDS to 1.4.1

2.13.6 0.15.8

Added

- Adds job storage sanity check to ensure that Kolibri will not fail to start if the asynchronous job storage is malformed

Changed

- Logging: remove unused simple formatter, add asctime to color formatter
- Order resume content display by last interaction
- Upgrade morango and lower default sync chunk size through CLI
- Make learners only appear once in reports when assigned from both groups and individually to lessons and quizzes.
- Persist collection tokens when switching between individual and bulk import workflows for channels

Fixed

- CSV Endpoint permissions and error handling
- Adds fix for multiple worker processes duplicating jobs.
- Adds translated string for user kind in the user table
- Check for an array's length to avoid breaking errors
- Fixes Version logic not handling non-tripartite version strings
- Filters out empty nodes, add safety to breaking code
- Prevent controls for the PDF renderer from overlapping content
- Fix quiz completion regression which caused the notification to contain the incorrect score
- height = width in import cards on thumbnail, fix misaligned text
- Update levels to display translated strings, not constant ids

2.13.7 0.15.7

Added

- Integration test gherkin story for automatic device provisioning in https://github.com/learningequality/kolibri/pull/9587

Fixed

- Add content check guard to library page in https://github.com/learningequality/kolibri/pull/9635
- Resolve issues with running morango integration tests on postgres in https://github.com/learningequality/kolibri/pull/9571
- Fix headers in content summary logs by forcing unicode literals in https://github.com/learningequality/kolibri/pull/9602

Changed

- Improve the `importcontent --fail-on-error` option in https://github.com/learningequality/kolibri/pull/9591

2.13.8 0.15.6

Added

- Check node being available on filtered queryset to prevent index error. by @rtibbles in https://github.com/learningequality/kolibri/pull/9539
- Force translations in bulk export/import of user data by @jredrejo in https://github.com/learningequality/kolibri/pull/9557
- Ensure peer import and sync tasks for data and content work with servers using a prefix path by @rtibbles in https://github.com/learningequality/kolibri/pull/9533

Changed

- Changes in 0.15.x to use kolibri with external plugins by @jredrejo in https://github.com/learningequality/kolibri/pull/9543
- Don't use multiprocessing for downloads. by @rtibbles in https://github.com/learningequality/kolibri/pull/9560

Fixed

- Update morango and stop locking sync when db backend is postgres by @bjester in https://github.com/learningequality/kolibri/pull/9556
- Improve facility sync status reporting to users by @MisRob in https://github.com/learningequality/kolibri/pull/9541
- Fix show more of top level resources by @marcellamaki in https://github.com/learningequality/kolibri/pull/9555
- Clean up theme regressions by @rtibbles in https://github.com/learningequality/kolibri/pull/9558
- Move CACHES import into function scope to prevent side effects. by @rtibbles in https://github.com/learningequality/kolibri/pull/9561

2.13.9 0.15.5

Overview

This release fixes a regression with quiz display for non-admins.

Fixed

- Clean up state management for user management page in https://github.com/learningequality/kolibri/pull/9535
- Fix quiz display for non-admins in https://github.com/learningequality/kolibri/pull/9545

2.13.10 0.15.4

Overview

This release of Kolibri includes security fixes to reduce the vulnerability of online Kolibri instances to discovery of user credentials and to sanitize exported CSV files.

Additional changes include small improvements to coach workflows in quiz and lesson workflows and fixing a regression with searching for users during class assignment.

Added

- Restrict exclude coach for to assigned coaches only in https://github.com/learningequality/kolibri/pull/9453
- Content dir argument in https://github.com/learningequality/kolibri/pull/9463

Changed

- Enable “continue” in quiz creation only once exercises selected in https://github.com/learningequality/kolibri/pull/9515
- Update bottom bar text in lesson resources to say save on changes in https://github.com/learningequality/kolibri/pull/9516

Fixed

- add .trim to v-model for username in https://github.com/learningequality/kolibri/pull/9514
- API and CSV fixes in https://github.com/learningequality/kolibri/pull/9523
- Fix missing search results in coach quiz creation in https://github.com/learningequality/kolibri/pull/9522
- Fixed regression: search functionality for assigning coaches and enrolling learners in https://github.com/learningequality/kolibri/pull/#9525

2.13.11 0.15.3

Overview of new features

The goal of this release was to make improvements to the accessibility of Kolibri and to content display. Fixes include improvements to the focus outline that appears for keyboard navigation and fixes to notifications used in screen readers, as well as small improvements to content layout.

Additions and Fixes: Accessibility

- Update firefox bookmarks cards focus outline https://github.com/learningequality/kolibri/pull/9409
- Update side panel focus trapping https://github.com/learningequality/kolibri/pull/9408
- Adds aria labels to immersive toolbar buttons for back and close https://github.com/learningequality/kolibri/pull/9411
- Adds aria-live=polite to the global snackbar component https://github.com/learningequality/kolibri/pull/9410
- Adjust padding for visible focus outline on bottom bar buttons in https://github.com/learningequality/kolibri/pull/9478

Additions and Fixes: Content Display

- Fix pagination issues for facility user page https://github.com/learningequality/kolibri/pull/9422
- Push PDF pages rendering below full screen bar https://github.com/learningequality/kolibri/pull/9439
- Fix X-Axis display for perseus graphs https://github.com/learningequality/kolibri/pull/9446
- Remove shrink ray from TopicsPage content side panel https://github.com/learningequality/kolibri/pull/9449
- Improve icon size in Cagetgory selection modal https://github.com/learningequality/kolibri/pull/8938
- Fix pagination user tables https://github.com/learningequality/kolibri/pull/9450
- Restrict exclude coach for to assigned coaches only https://github.com/learningequality/kolibri/pull/453

Changes

- Ensure all file handlers use utf-8 encoding https://github.com/learningequality/kolibri/pull/9401
- Upgrade morango to v0.6.13 https://github.com/learningequality/kolibri/pull/9445
- 0.14 into 0.15 https://github.com/learningequality/kolibri/pull/9447
- Upgrade KDS to v1.3.1-beta0 https://github.com/learningequality/kolibri/pull/9459

2.13.12 0.15.2

Internationalization and localization

New language support for: Ukrainian

Added

- Additional gherkin scenarios https://github.com/learningequality/kolibri/pull/9130

Changed

- Bump morango to v0.6.10 https://github.com/learningequality/kolibri/pull/9168
- Pin windows installer to 1.5.0 https://github.com/learningequality/kolibri/pull/9200
- Pin django js asset https://github.com/learningequality/kolibri/pull/9163
- Compress HTML files for serving https://github.com/learningequality/kolibri/pull/9197
- Disable mac app pipeline by @rtibbles in https://github.com/learningequality/kolibri/pull/9257
- `SECURE_CONTENT_TYPE_NOSNIFF` set to `True` https://github.com/learningequality/kolibri/pull/9195

Fixed

- Content import, deletion, and `remote_content` settings fixes (#9242, #9337, #9246, #8506)
- Add check for `notification` to avoid il8n error in `CoreBase` https://github.com/learningequality/kolibri/pull/9138
- Redirect for Bookmarks when user is not logged in https://github.com/learningequality/kolibri/pull/9142
- Delete any annotated channelmetadata many to many fields to avoid integrity errors https://github.com/learningequality/kolibri/pull/9141
- Ensure deprovisioning management command deletes DMC https://github.com/learningequality/kolibri/pull/9208
- Fix Python requires to prevent install on incompatible Python versions https://github.com/learningequality/kolibri/pull/9296

2.13.13 0.15.1

Overview of new features

The goals of this release were to fix a bug preventing proper syncing of an individual user's data across multiple devices and to made some small frontend improvements

Added

- Deprecation warnings for Python 3.4 and 3.5
- Added auto-alignment property for text display in cards, based on the language
- Allow untranslated headers in csv imports and correct serialization into json

Changed

- Updated morango to v0.6.8 to support syncing fixes
- Bump zeroconf for fix to properly trigger service update events
- Bump KDS version to v1.3.0
- Updated translations to support minor translation fixes
- Updated gherkin scenarios for new features
- Content API: Change default ordering to combination of "lft" and "id"

Fixed

- Keyboard accessibility/tab navigation focusing for searching and filtering
- Allow for scrolling in side panel, and have side panel always take up full height of page even with 0 results
- Small UI improvements including focus ring spacing, button alignment
- Hide hints column in Perseus renderer when it could not be displayed to improve display on smaller screens
- Handle no xAPI statements existing when calculating H5P and HTML5 progress
- Don't package core node_modules dir
- Refactor card components for consistency and comprehensibility
- Address tech debt around KDS theming colors
- Fixed several front end console errors
- Ensure that we filter by subset_of_users_device on network location API

2.13.14 0.15.0

Internationalization and localization

New language support for: Hausa, Georgian, Indonesian, Mozambican Portuguese, and Greek

Overview of major new features

This release includes a new Learn experience featuring:

- An updated Home page with new layout and interactions
- A new library page featuring a better content browsing, filtering, and search experience
- An update page for browsing individual channels, with new layout and browse/search interactions
- A new bookmarks page and ability to bookmark content within the content renderer
- Sync capabilities for Subset of Users Devices (SoUDs)

Selected high-level technical updates:

- Adding API for SoUD devices, allowing them to request syncing
- Updates to Zeroconf to support SoUD syncing

- Updates to progress tracking
- Consolidation of exam logging
- Fix dataset mismatch between exams and lessons, to allow for syncing
- Adding content metadata search, API, and fields

Fixed

- #8442 Segments SQLite databases to allow concurrent writes to SyncQueue and NetworkLocation models
- #8446 Forces Learner only device sync request retries when server responds with 500+ status code
- #8438 Fixes failure to sync FacilityUser updates when a login has occurred on a Learner only device prior to syncing
- #8438 Fixes failure to sync all updated records when multiple learner only devices have been setup for a single FacilityUser
- #8069 Fix backdrop not being shown while searching resources on mobile
- #8000 Ensure progress_fraction is propagated through resource API
- #7983 Validate usernames during sign-in flow, fix bug in facility settings page
- #7981 Correct the component namespace in the JSON files
- #7953 Fix non-localized numerals
- #7951 Tasks queue cleared on server start
- #7932 Fix DemoBanner focus
- #8174 Fix errors from ContentNodeResource changes
- #8162 Fix dynamic file discovering and serving on Windows
- (#8159, #8132) Fix IE11 compatibility
- #8199 Don't modify lessons when content is deleted
- #8133 Prevent iterable changes size during iteration
- #8121 Error properly on startup
- #8103 Update values viewset implementation and pagination
- #8102 Fix KLabeledIcon UI
- #8101 Stop TextTruncator flash of full text before truncation

Changed

- #8220 Update reference to most recent Kolibri Design System
- #8194 Update data flow docs for accuracy
- #8088 Update DeviceSettingsPage layout. Add labels, tests
- #7936 Change template for personal facility name to "Home facility for {name}"
- #7928 Update memberships, roles, and permissions handling and validation
- #8195 Use a double tap strategy to ensure against zombies

- #8184 Bump morango version to 0.5.6
- #8168 Use consistent "not started" icon and background color in AnswerHistory and AttemptLogList
- #8143 Increase scrolling room for question lists in MultiPanelLayout
- #8130 Replace migration applied check
- #8123 Don't use KResponsiveElementMixin in all ContentCards
- #8592 Fix quiz log syncing

Added

- (#8185, #8595) Add setup wizard for SoUD configuration
- #8229 Add SoUD setup via command line
- (#8202 , #8247 , #8329) Add UI for sync status reporting with notifications for coaches and learners
- (#8192, #8205) Create user sync status tracking, add add permissions to model
- (#8333, #8342, #8345, #8349, #8262) Create queue for SoUD syncing
- #8223 Add notification generation during cleanup stage of sync
- #8222 Add device info versioning
- #8219 Assignment handling within single-user syncing
- #8126 Create API for a subset of user devices to request permission to sync
- #8122 Zeroconf broadcast of SoUD status
- #8165 Initiate auto-syncing from zeroconf
- #8228 Sidechannel loading of assignments
- (#8212, #8215) Create channel-based quizzes, and corresponding gherkin scenarios
- #8095 Add Bookmarks API
- #8084 Allow Kolibri themes to provide a "Photo credit" for the Sign-In page background image
- #8043 Add explicit include_coach_content filter instead of role filter
- (#7989, #8214) Frontend only H5P Rendering and xAPI progress tracking integration
- #7947 Open CSV file with utf-8 encoding in Py3
- #7921 Add content tags to ContentNodeViewset
- #7939 Add endpoint to check for duplicate username and use it to check for existing username while creating an account
- (#8150, #8151) Add learning activity bar component, constants, and icon components
- (#8190, #8180) Add support for multiple learning activities icon, and create related constants
- #8186 Create API endpoint for Tasks backend
- #8177 Return learning_activities and duration from contentnode endpoints
- #8142 Add task decorators and task APIs for functions registered via decorators
- #8138 Add Tree viewset for retrieving nested, paginated views of topic trees
- #8136 Add new card design to AllClassesPage and ClassAssignmentPage and add base card elements

- #8134) Update navigateTo for non-custom HTML5 Apps
- (#8118, #8146) Add @vue-composition-api plugin, and expose through apiSpec, so it is available to all SPAs
- #8117 Add vacuum for morango tables in Postgresql databases
- #8367 Ensure the user will see the welcome modal after login
- #8370 Restart zeroconf after setup
- #8383 filter SoUD devices when scanning the network to import new facilities
- #8385 Do not create accounts in Subset of users devices
- #8411 Upgrade zeroconf
- #8412 Reduce default sync retry interval
- #8413 Reuse kolibriLogin to begin user sessions in the setup wizard
- #8596 Add new icons
- #8742 Allow facility forking and recreation

(Full Release Notes)

(0.15.0 Github milestone)

2.13.15 0.14.7

Internationalization and localization

- Updated localizations

Fixed

- #7766 Content imported by administrators was not immediately available for learners to use
- #7869 Unlisted channels would not appear in list in channel import-workflow after providing token
- #7810 Learners' new passwords were not being validated on the Sign-In page
- #7764 Users' progress on resources was not being properly logged, making it difficult to complete them
- #8003, #8004, #8010 Sign-ins could cause the server to crash if database was locked
- #8003, #7947 Issues downloading CSV files on Windows

Changed

- #7735 Filtering on lists of users returns ranked and approximate matches
- #7733 Resetting a facility's settings respects the preset (e.g. formal, informal, nonformal) chosen for it during setup
- #7823 Improved performance on coach pages for facilities with large numbers of classrooms and groups

(0.14.7 Github milestone)

2.13.16 0.14.6

Fixed

- #7725 On Firefox, text in Khmer, Hindi, Marathi, and other languages did not render properly.
- #7722, #7488 After viewing a restricted page, then signing in, users were not redirected back to the restricted page.
- #7597, #7612 Quiz creation workflow did not properly validate the number of questions

(0.14.6 Github milestone)

2.13.17 0.14.5

(Note: 0.14.4 contained a critical issue and was superseded by 0.14.5)

Changed

- File downloads now run concurrently, taking better advantage of a device's bandwidth and reducing the time needed to import resources from Kolibri Studio or other content sources
- When setting up a new device using the Setup Wizard's "Quick Start" option, the "Allow learners to create accounts" setting is enabled by default.
- The `provisiondevice` management command no longer converts the user-provided facility name to all lower-case
- Markdown descriptions for resources now preserve line breaks from the original source

Fixed

- Multiple bugs when creating, editing, and copying quizzes/lessons
- Multiple bugs when navigating throughout the Coach page
- Multiple bugs specific to Kolibri servers using PostgreSQL
- On Safari, sections of the Facility > Data page would disappear unexpectedly after syncing a facility
- On IE11, it was not possible to setup a new device by importing a facility
- Missing thumbnails on resource cards when searching/browsing in channels
- Numerous visual and accessibility issues
- Facilities could not be renamed if the only changes were to the casing of the name (e.g. changing "Facility" to "FACILITY")

(0.14.5 Github milestone)

2.13.18 0.14.3

(Note: 0.14.0-2 contained regressions and were superseded by 0.14.3)

Fixed

- Some links were opening in new browser windows

(0.14.3 Github milestone)

2.13.19 0.14.2

Fixed

- Prevent SQL checksum related too many variables errors

(0.14.2 Github milestone)

2.13.20 0.14.1

Changed

- Responsive layout for channel cards of Learn Page changed to use horizontal space more efficiently

Fixed

- Resources could not be removed from lessons
- Inaccurate information on Device > Info page when using Debian installer

(0.14.1 Github milestone)

2.13.21 0.14.0

Internationalization and localization

- Added German
- Added Khmer
- CSV data files have localized headers and filenames

Added

- In the Setup Wizard, users can import an existing facility from peer Kolibri devices on the network
- Facility admins can sync facility data with peer Kolibri devices on the network or Kolibri Data Portal
- Facility admins can import and export user accounts to and from a CSV file
- Channels can display a learner-facing “tagline” on Learn channel list
- Device and facility names can now be edited by admins

- Super admins can delete facilities from a device
- Quizzes and lessons can be assigned to individual learners in addition to whole groups or classes
- Super admins can view the Facility and Coach pages for all facilities
- Pingbacks to the telemetry server can now be disabled

Changed

- New card layout for channels on Learn Page is more efficient and displays new taglines
- Simplified setup process when using Kolibri for personal use
- Improved sign-in flow, especially for devices with multiple facilities
- The experience for upgrading channels has been improved with resource highlighting, improved statistics, and more efficient navigation
- Improved icons for facilities, classrooms, quizzes, and other items
- More consistent wording of notifications in the application
- Quizzes and lessons with missing resources are more gracefully handled
- Shut-down times are faster and more consistent

Fixed

- Many visual and user experience issues
- Language filter not working when viewing channels for import/export
- A variety of mobile responsiveness issues have been addressed

(0.14.0 Github milestone)

2.13.22 0.13.3

Changed or fixed

- Fixed: Infinite-loop bug when logging into Kolibri through Internet In A Box (IIAB)
- Fixed: Performance issues and timeouts when viewing large lists of users on the Facility page
- Fixed: Startup errors when Kolibri is installed via `pip` on non-debian-based Linux distributions

Internationalization and localization

- Added Simplified Chinese

(0.13.3 Github milestone)

2.13.23 0.13.2

Changed or fixed

- Fixed: In the Device Page, multiple bugs related to managing channels.
- Fixed: Problems viewing African Storybook content on iPads running iOS 9.

Internationalization and localization

- Added Italian

(0.13.2 Github milestone)

2.13.24 0.13.1

Added

- Python version is shown on the 'Device > Info' page in the 'Advanced' section
- Improved help information when running `kolibri --help` on the command line

Changed or fixed

- Various layout and UX issues, especially some specific to IE11 and Firefox
- 'Device > Info' page not accessible when logged in as a superuser
- Channels unintentionally reordered on 'Device > Channels' when new content is imported
- Video captions flashing in different languages when first opening a video
- Changes to channels updated and republished in Studio not being immediately reflected in Kolibri
- Occasional database blocking errors when importing large collections of content from external drives
- Occasional database corruption due to connections not being closed after operations
- Automatic data restoration for corrupted databases
- Recreate cache.db files when starting the Kolibri server to remove database locks that may not have been cleanly removed in case of an abrupt shut-down.

(0.13.1 Github milestone)

2.13.25 0.13.0

Added

- Improved content management
  - Queues and task manager
  - Granular deletion
  - Improved channel updating
  - Disk usage reporting improvements

    - Auto-discovery of local Kolibri peers

- Demographics collection and reporting
- MacOS app
- High-performance Kolibri Server package for Debian
- Pre-built Raspberry Pi Kolibri image
- Video transcripts
- Downloadable and printable coach reports
- New device settings
- "Skip to content" keyboard link

Changed or fixed

- Preserve 'unlisted' status on channels imported from token
- Allow duplicate channel resources to be maintained independently
- Auto-refresh learner assignemnt view
- Unclean shutdowns on very large databases, due to prolonged database cleanup
- Facility admin performance improvements
- Jittering modal scrollbars
- Updated side-bar styling
- Improved form validation behavior
- Improved learner quiz view
- Improved keyboard accessibility

(0.13.0 Github milestone)

2.13.26 0.12.9

Added

- Improved error reporting in Windows

Changed or fixed

- Database vacuum now works correctly
- Fixes related to network detection
- Improve performance of classroom API endpoint to prevent request timeouts

Internationalization and localization

- Added Korean

(0.12.9 Github milestone)

2.13.27 0.12.8

Changed or fixed

- Fixed: users creating accounts for themselves not being placed in their selected facility
- Fixed: images in Khan Academy exercises not appearing on occasion
- Fixed: "Usage and Privacy" modal not closing when clicking the "Close" button

(0.12.8 Github milestone)

2.13.28 0.12.7

(Note: 0.12.6 contained a regression and was superseded by 0.12.7)

Changed or fixed

- Facility user table is now paginated to improve performance for facilities with large numbers of users.
- Various usability and visual improvements, including improved layout when using a RTL language
- On Windows, `kolibri.exe` is automatically added to the path in the command prompt
- Improved system clean-up when uninstalling on Windows

Internationalization and localization

- Added Latin American Spanish (ES-419)

(0.12.7 Github milestone)

(0.12.6 Github milestone)

2.13.29 0.12.5

- Upgraded Morango to 0.4.6, fixing startup errors for some users.

(0.12.5 Github milestone)

2.13.30 0.12.4

Added

- Device Settings Page - The default language can now be changed under Device > Settings. This is the language that will be used on browsers that have never opened Kolibri before (but can be changed after opening Kolibri using the language selector).
- Coach Reports - Users can preview quizzes and lessons and edit their details from their associated report, without having to go to the "Plan" sub-page.
- Added a `kolibri manage deleteuser` command to remove a user from a server, as well as all other servers synchronized with it.
- Added a new theming system for customizing various colors that appear in Kolibri.

Changed or fixed

- EPUB documents with large tables are displayed in a single-column, scrollable format to improve their readability.
- EPUB viewer now saves font and theme settings between sessions.
- Quiz creation workflow only places unique questions in a quiz, removing duplicates that may appear in a topic tree.
- Title and name headers are consistently accompanied by icons in Kolibri symbol system to help orient the user.

(0.12.4 Github milestone)

2.13.31 0.12.3

Changed or fixed

- Improved handling of partially-download or otherwise corrupted content databases
- Fixed regression where users could not change their passwords in the Profile page
- Improved PostgreSQL support
- Added fixes related to coach tools

(0.12.3 Github milestone)

2.13.32 0.12.2

Added

- Coaches can edit lessons from the Coach > Reports page
- Coaches can preview and edit quiz details from the Coach > Reports and Plan pages

Changed or fixed

- Coaches can edit quiz and lesson details and statuses in the same user interface

2.13.33 0.12.2

Added

- Dynamic selection for CherryPy thread count based on available server memory

Changed or fixed

- Alignment of coach report icons when viewed in right-to-left languages corrected
- Fixes to loading of some HTML5 apps
- Lessons are now correctly scoped to their classes for learners

Internationalization and localization

- Added Gujarati
- Fixed missing translations in coach group management

(0.12.2 Github milestone)

2.13.34 0.12.1

Added

- Initial support for uwsgi serving mode.

Changed or fixed

- Fixed 0.12.0 regression in HTML5 rendering that affected African Storybooks and some other HTML5 content.
- Fixed 0.12.0 regression that prevented some pages from loading properly on older versions of Safari/iOS.

Internationalization and localization

- Added Burmese

(0.12.1 Github milestone)

2.13.35 0.12.0

Added

- Coach Dashboard - added regularly updating notifications and new information architecture for the coach interface, to provide actionable feedback for coaches about learner progress
- New capability for sandboxed HTML5 app content to utilize sessionStorage, localStorage and cookies, with the latter two restored between user sessions
- Support for enrolling learners in multiple groups in a class
- Management command to reorder channels to provide more customized display in learn

Changed or fixed

- Exams are now known as Quizzes
- Quizzes with content from deleted channels will now show an error message when a learner or coach is viewing the problems in the quiz or quiz report
- Lessons with content from deleted channels will have those contents automatically removed. If you have created lessons with deleted content prior to 0.12, learner playlists and coach reports for those lessons will be broken. To fix the lesson, simply view it as a coach under Coach > Plan, and it will be fixed and updated automatically
- Changes the sub-navigation to a Material Design tabs-like experience
- Make facility log exporting a background process for a better user experience when downloading large logs
- Allow appbar to move off screen when scrolling on mobile, to increase screen real estate
- Kolibri now supports for iOS Safari 9.3+
- Validation is now done in the 'provisiondevice' command for the username of the super admin user being created
- Disable import and export buttons while a channel is being downloaded to prevent accidental clicks
- Prevent quizzes and lessons in the same class from being created with the same name
- Update quiz and lesson progress for learners without refreshing the page
- Improved focus rings for keyboard navigation
- Coach content no longer appears in recommendations for non-coach users
- The Kolibri loading animation is now beautiful, and much quicker to load
- Icons and tables are now more standardized across Kolibri, to give a more consistent user experience
- Enable two high contrast themes for EPUB rendering for better accessibility
- Supports accessing Kolibri through uwsgi

Internationalization and localization

- Languages: English, Arabic, Bengali, Bulgarian, Chinyanja, Farsi, French, Fulfulde Mboororoore, Hindi, Marathi, Portuguese (Brazilian), Spanish, Swahili, Telugu, Urdu, Vietnamese, and Yoruba

(0.12.0 Github milestone)

2.13.36 0.11.1

Added

- Support for RTL EPubs
- Support for Python 3.7

Changed or fixed

- Fullscreen renderer mode now works in Chrome 71
- Account sign up now works when guest access is disabled
- Navigating in and out of exercise detail views is fixed
- Misleading exam submission modal text is now more accurate
- Browsing content tree in exam creation is now faster
- Unavailable content in coach reports is now viewable
- Content import errors are handled better
- Added command to restore availability of content after bad upgrade

Internationalization and localization

- Added Fufulde Mboroore

(0.11.1 Github milestone)

2.13.37 0.11.0

Added

- Support for EPUB-format electronic books
- Upgrades to exam and lesson creation, including search functionality and auto-save
- New error handling and reporting functionality
- Channel import from custom network locations
- Setting for enabling or disabling guest access
- Basic commands to help with GDPR compliance
- Privacy information to help users and admins understand how their data is stored

Changed or fixed

- Improvements to rendering of some pages on smaller screens
- Improvements to search behavior in filtering and handling of large result sets
- Improvements to the setup wizard based on user feedback and testing
- Improvements to user management, particularly for admins and super admins
- Fix: Allow usernames in non-latin alphabets
- Fix: Drive listing and space availability reporting
- Auto-refresh in coach reports
- Added more validation to help with log-in
- Security: upgraded Python cryptography and pyopenssl libraries for CVE-2018-10903

Internationalization and localization

- Languages: English, Arabic, Bengali, Bulgarian, Chinyanja, Farsi, French, Hindi, Marathi, Portuguese (Brazilian), Spanish, Swahili, Telugu, Urdu, Vietnamese, and Yoruba
- Improved consistency of language across the application, and renamed "Superuser" to "Super admin"
- Many fixes to translation and localization
- Consistent font rendering across all languages

(0.11.0 Github milestone)

2.13.38 0.10.3

Internationalization and localization

- Added Mexican Spanish (es_MX) and Bulgarian (bg)

Fixed

- Upgrade issue upon username conflict between device owner and facility user
- Channel import listing of USB devices when non-US locale
- Counts for coach-specific content would in some cases be wrongly displayed

(0.10.3 Github milestone)

2.13.39 0.10.2

- Performance improvements and bug fixes for content import
- Exam creation optimizations

(0.10.2 Github milestone)

2.13.40 0.10.1

- Bug fix release
- Several smaller UI fixes
- Fixes for SSL issues on low-spec devices / unstable connectivity
- Compatibility fixes for older system libraries

(0.10.1 Github milestone)

2.13.41 0.10.0

- Support for coach-specific content
- Content import/export is more reliable and easier to use
- Search has improved results and handles duplicate items
- Display of answer history in learner exercises is improved
- Login page is more responsive
- Windows-specific improvements and bug fixes
- New Kolibri configuration file
- Overall improved performance
- Auto-play videos
- Various improvements to PDF renderer
- Command to migrate content directory location
- Languages: English, Arabic, Bengali, Chinyanja, Farsi, French, Hindi, Kannada, Marathi, Burmese, Portuguese (Brazilian), Spanish, Swahili, Tamil, Telugu, Urdu, Yoruba, and Zulu

(0.10.0 Github milestone)

2.13.42 0.9.3

- Compressed database upload
- Various bug fixes

(0.9.3 Github milestone)

2.13.43 0.9.2

- Various bug fixes
- Languages: English, Arabic, Bengali, Chinyanja, Farsi, French, Hindi, Marathi, Portuguese (Brazilian), Spanish, Swahili, Tamil, Telugu, Urdu, Yoruba, and Zulu

(0.9.2 Github milestone)

2.13.44 0.9.1

- Fixed regression that caused very slow imports of large channels
- Adds new 'import users' command to the command-line
- Various consistency and layout updates
- Exercises with an error no longer count as 'correct'
- Fixed issue with password-less sign-on
- Fixed issue with editing lessons
- Various other fixes
- Languages: English, Arabic, Chinyanja, Farsi, French, Hindi, Marathi, Portuguese (Brazilian), Spanish, Swahili, Tamil, Telugu, and Urdu

(0.9.1 Github milestone)

2.13.45 0.9.0

- Consistent usage of 'coach' terminology
- Added class-scoped coaches
- Support for multi-facility selection on login
- Cross-channel exams
- Show correct and submitted answers in exam reports
- Added learner exam reports
- Various bug fixes in exam creation and reports
- Various bug fixes in coach reports
- Fixed logging on Windows
- Added ability for coaches to make copies of exams
- Added icon next to language-switching functionality
- Languages: English, Arabic, Farsi, French, Hindi, Spanish, Swahili, and Urdu

(0.9.0 Github milestone)

2.13.46 0.8.0

- Added support for assigning content using 'Lessons'
- Updated default landing pages in Learn and Coach
- Added 'change password' functionality to 'Profile' page
- Updates to text consistency
- Languages: English, Spanish, Arabic, Farsi, Urdu, French, Haitian Creole, and Burmese
- Various bug fixes

(0.8.0 Github milestone)

2.13.47 0.7.2

- Fix issue with importing large channels on Windows
- Fix issue that prevented importing topic thumbnail files

2.13.48 0.7.1

- Improvements and fixes to installers including Windows & Debian
- Updated documentation

2.13.49 0.7.0

- Completed RTL language support
- Languages: English, Spanish, Arabic, Farsi, Swahili, Urdu, and French
- Support for Python 3.6
- Split user and developer documentation
- Improved lost-connection and session timeout handling
- Added 'device info' administrator page
- Content search integration with Studio
- Granular content import and export

2.13.50 0.6.2

- Consistent ordering of channels in learner views

2.13.51 0.6.1

- Many mobile-friendly updates across the app
- Update French, Portuguese, and Swahili translations
- Upgraded Windows installer

2.13.52 0.6.0

- Cross-channel searching and browsing
- Improved device onboarding experience
- Improved device permissions experience (deprecated 'device owner', added 'superuser' flag and import permission)
- Various channel import/export experience and stability improvements
- Responsive login page
- Dynamic language switching
- Work on integrated living style guide
- Added beta support for right-to-left languages
- Improved handling of locale codes
- Added support for frontend translation outside of Vue components
- Added an open-source 'code of conduct' for contributors
- By default run PEX file in foreground on MacOS
- Crypto optimizations from C extensions
- Deprecated support for HTML in translation strings
- Hide thumbnails from content 'download' button
- Automatic database backup during upgrades. #2365
- … and many other updates and fixes

2.13.53 0.5.3

- Release timeout bug fix from 0.4.8

2.13.54 0.5.2

- Release bug fix from 0.4.7

2.13.55 0.5.1

- Python dependencies: Only bundle, do not install dependencies in system env #2299
- Beta Android support
- Fix 'importchannel' command #2082
- Small translation improvements for Spanish, French, Hindi, and Swahili

2.13.56 0.5.0

- Update all user logging related timestamps to a custom datetime field that includes timezone info
- Added daemon mode (system service) to run `kolibri start` in background (default!) #1548
- Implemented `kolibri stop` and `kolibri status` #1548
- Newly imported channels are given a 'last_updated' timestamp
- Add progress annotation for topics, lazily loaded to increase page load performance
- Add API endpoint for getting number and total size of files in a channel
- Migrate all JS linting to prettier rather than eslint
- Merge audio_mp3_render and video_mp4_render plugins into one single media_player plugin
- KOLIBRI_LISTEN_PORT environment variable for specifying a default for the –port option #1724

2.13.57 0.4.9

- User experience improvements for session timeout

2.13.58 0.4.8

- Prevent session timeout if user is still active
- Fix exam completion timestamp bug
- Prevent exercise attempt logging crosstalk bug
- Update Hindi translations

2.13.59 0.4.7

- Fix bug that made updating existing Django models from the frontend impossible

2.13.60 0.4.6

- Fix various exam and progress tracking issues
- Add automatic sign-out when browser is closed
- Fix search issue
- Learner UI updates
- Updated Hindi translations

2.13.61 0.4.5

- Frontend and backend changes to increase performance of the Kolibri application under heavy load
- Fix bug in frontend simplified login code

2.13.62 0.4.4

- Fix for Python 3 compatibility in Whl, Windows and Pex builds #1797
- Adds Mexican Spanish as an interface language
- Upgrades django-q for bug fixes

2.13.63 0.4.3

- Speed improvements for content recommendation #1798

2.13.64 0.4.2

- Fixes for morango database migrations

2.13.65 0.4.1

- Makes usernames for login case insensitive #1733
- Fixes various issues with exercise rendering #1757
- Removes wrong CLI usage instructions #1742

2.13.66 0.4.0

- Class and group management
- Learner reports #1464
- Performance optimizations #1499
- Anonymous exercises fixed #1466
- Integrated Morango, to prep for data syncing (will require fresh database)
- Adds Simplified Login support as a configurable facility flag

2.13.67 0.3.3

- Turns video captions on by default

2.13.68 0.3.2

- Updated translations for Portuguese and Kiswahili in exercises.
- Updated Spanish translations

2.13.69 0.3.1

- Portuguese and Kaswihili updates
- Windows fixes (mimetypes and modified time)
- VF sidebar translations

2.13.70 0.3.0

- Add support for nested URL structures in API Resource layer
- Add Spanish and Swahili translations
- Improve pipeline for translating plugins
- Add search back in
- Content Renderers use explicit new API rather than event-based loading

2.13.71 0.2.0

- Add authentication for tasks API
- Temporarily remove 'search' functionality
- Rename 'Learn/Explore' to 'Recommended/Topics'
- Add JS-based 'responsive mixin' as alternative to media queries
- Replace jeet grids with pure.css grids
- Begin using some keen-ui components
- Update primary layout and navigation
- New log-in page
- User sign-up and profile-editing functionality
- Versioning based on git tags
- Client heartbeat for usage tracking
- Allow plugins to override core components
- Wrap all user-facing strings for I18N
- Log filtering based on users and collections
- Improved docs

- Pin dependencies with Yarn
- ES2015 transpilation now Bublé instead of Babel
- Webpack build process compatible with plugins outside the kolibri directory
- Vue2 refactor
- HTML5 app renderer

2.13.72 0.1.1

- SVG inlining
- Exercise completion visualization
- Perseus exercise renderer
- Coach reports

2.13.73 0.1.0 - MVP

- Improved documentation
- Conditional (cancelable) JS promises
- Asset bundling performance improvements
- Endpoint indexing into zip files
- Case-insensitive usernames
- Make plugins more self-contained
- Client-side router bug fixes
- Resource layer smart cache busting
- Loading 'spinner'
- Make modals accessible
- Fuzzy searching
- Usage data export
- Drive enumeration
- Content interaction logging
- I18N string extraction
- Channel switching bug fixes
- Modal popups
- A11Y updates
- Tab focus highlights
- Learn app styling changes
- User management UI
- Task management
- Content import/export

- Session state and login widget
- Channel switching
- Setup wizard plugin
- Documentation updates
- Content downloading

2.13.74 0.0.1 - MMVP

- Page titles
- Javascript logging module
- Responsiveness updates
- A11Y updates
- Cherrypy server
- Vuex integration
- Stylus/Jeet-based grids
- Support for multiple content DBs
- API resource retrieval and caching
- Content recommendation endpoints
- Client-side routing
- Content search
- Video, Document, and MP3 content renderers
- Initial VueIntl integration
- User management API
- Vue.js integration
- Learn app and content browsing
- Content endpoints
- Automatic inclusion of requirements in a static build
- Django JS Reverse with urls representation in kolibriGlobal object
- Python plugin API with hooks
- Webpack build pipeline, including linting
- Authentication, authorization, permissions
- Users, Collections, and Roles

PYTHON MODULE INDEX

INDEX