
Kolibri

Release 0.5

Aug 26, 2020

Contents

1	User Guide	1
2	Developer Guide	63
3	Release Notes	107
4	Contributing	113
5	Credits	117
6	Kolibri	119
7	What is Kolibri?	121
8	How can I use it?	123
9	How can I contribute?	125

1.1 Access Kolibri

1.1.1 Starting Kolibri on Windows

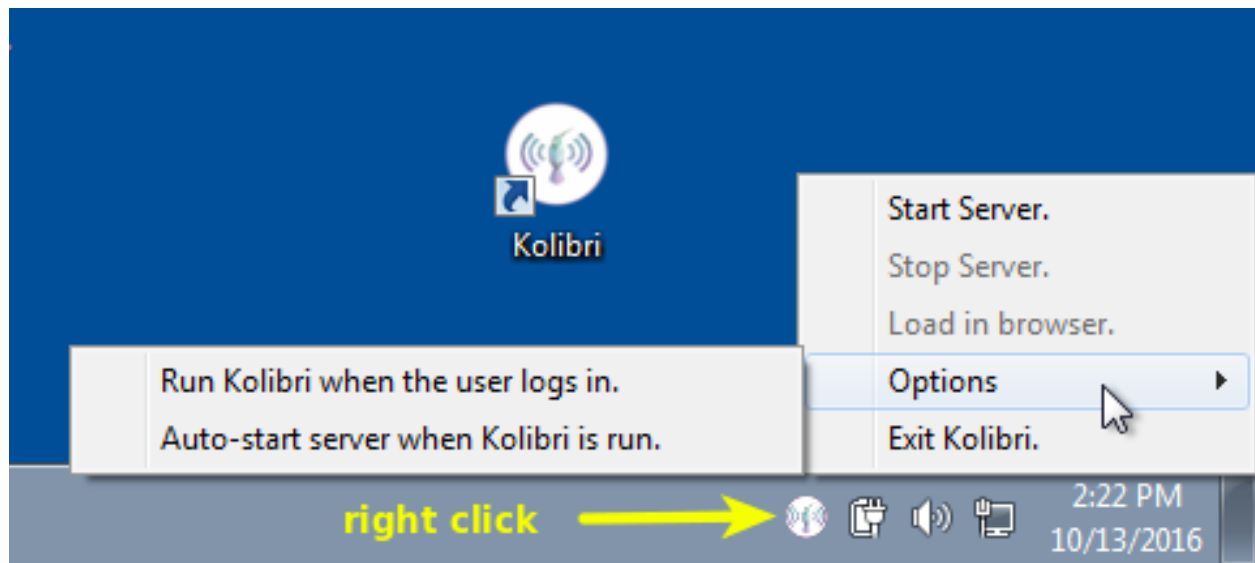
Kolibri has three parts: **Kolibri launcher**, **Kolibri server**, and **Kolibri web app**.

Kolibri launcher is located in the Windows taskbar (usually at bottom right, near the clock), and allows you to start and stop the **Kolibri server**, and configure other settings.

Kolibri server runs as a background process and sends (*serves*) content to the browser.

Kolibri web app displays learning content in the browser.

- Double-click Kolibri desktop shortcut to start **Kolibri launcher**.
- Right click the taskbar icon to open the **Kolibri launcher** menu.



Warning: Starting **Kolibri launcher** will not start **Kolibri server** by default, but you can configure that setting in the launcher options if it suits your needs better.

Kolibri Launcher Menu Options

- Select **Start Server** to start the **Kolibri server**.
 1. You will see the notification message *Kolibri server is starting, please wait...* While server is starting, the options **Start/Stop/Load in browser** will be disabled.
 2. Once **Kolibri server** has started you will see the notification message *Kolibri server is running...* The options **Stop Server** and **Load in browser** will now be available.
- Select **Load in browser** to open the default browser at <http://127.0.0.1:8080> displaying the **Kolibri web app**.
- Select **Stop Server** when you want to stop the **Kolibri server**.
- Select **Options** sub-menu for further configuration.
 1. Activate **Run Kolibri when user logs in** option if you want **Kolibri launcher** to start automatically when you log into the system.
 2. Activate **Auto-start server when Kolibri is run** option if you want the **Kolibri server** to start at the same time as the **Kolibri launcher**.

Note: If you want **Kolibri server** to auto-start as soon as you log into the system, activate both of the above options.

1.1.2 Starting Kolibri on Linux

Warning: Final Kolibri installer for Linux is not released yet, so these steps are *Work in Progress*!

Open the default browser at <http://127.0.0.1:8080> displaying the Kolibri start page.

1.1.3 Accessing Kolibri from Other Devices in the Network

While **Kolibri server** is up and running, other devices (computers, tablets, even mobile phones) in the same **Local Area Network (LAN)** can access its learning contents.

- To access the content on the same device/computer where **Kolibri server** is running, open the browser at the address <http://127.0.0.1:8080/>.
- To access the content from other devices in the same network, you need to know the IP address of one where **Kolibri server** is running. For example, if **Kolibri server** is on a device/computer with IP address **192.168.0.104**, you can access it from others in the same network at the address <http://192.168.0.104:8080>.

Note: Use the `ipconfig` command on Windows or `ipconfig` command on Linux to find out the IP address of the device running the **Kolibri server**.

1.2 Device Owner

1.2.1 Create Device Owner Account

A **Device Owner** is a “super-user” for your Kolibri installation, and a **Facility** is a group of users and their associated data.

In order to manage Kolibri content channels, data and users, you must first create a **Device Owner** account and the **Facility**. The **Device Owner** account registration page appears when you start the server for the first time after the installation of Kolibri and open the browser at <http://127.0.0.1:8080/> (the default URL).



Create Kolibri Device Owner and Facility

Recommended

To use Kolibri, you first need to create a Device Owner account. This account will be used to configure high-level settings for this installation, and create other administrator accounts.

Username

Password

Confirm password

Facility

You also need to create a Facility, which represents your school, training center, or other location where this installation will be used.

Facility name

Create and get started

1. Enter the required information for the **Device Owner** account (username, password, name of the facility). Fields marked with an asterisk (*) are required.
2. Select **Create and get started**.

Adding a New Device Owner

In case you need to create another super-user (e.g. you lost the password for the old one), run the following command (open the `cmd.exe` command prompt in Windows or terminal in linux):

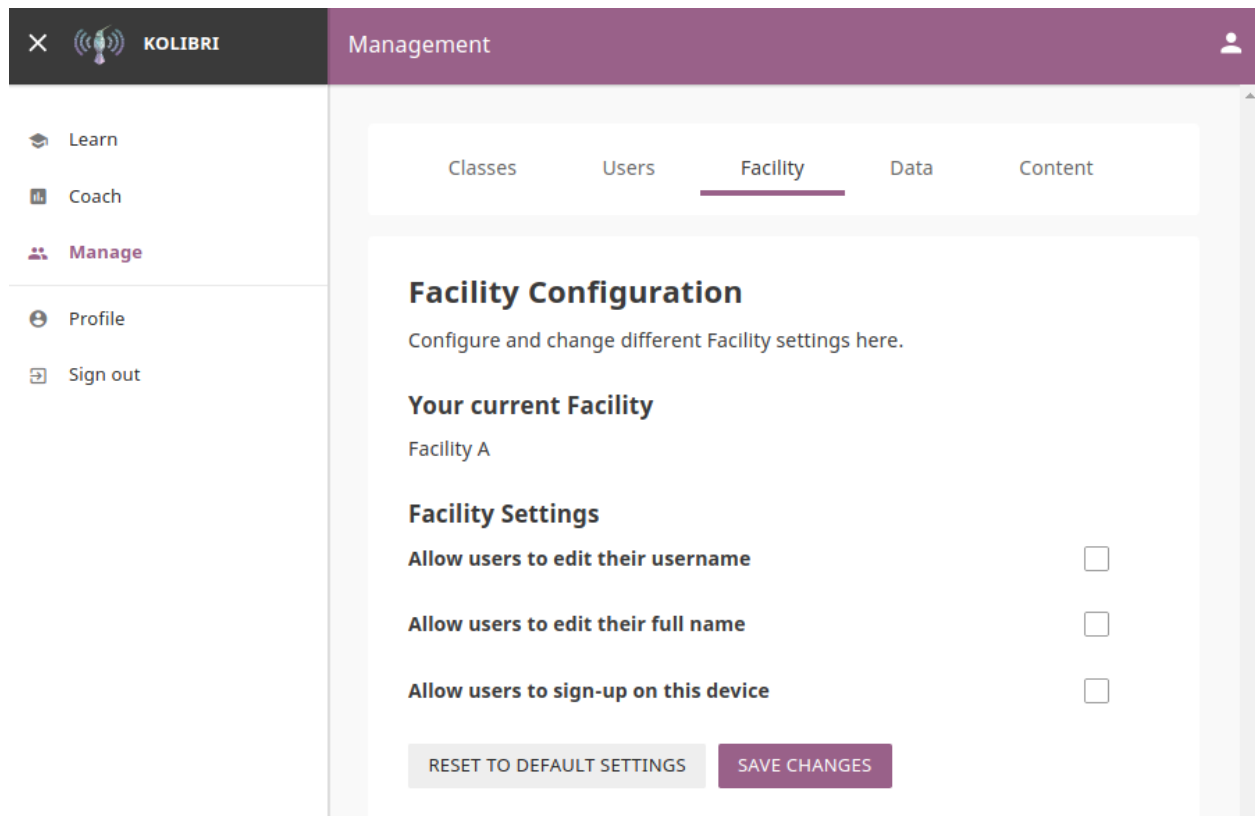

```
kolibri manage createsuperuser
```

You will be prompted to input the **Username** and **Password** and the new **Device Owner** user account will be created.

1.2.2 Manage Facility

You can edit facility configuration settings in Kolibri from the **Facility** tab in your **Manage** dashboard.

1. Activate the options you want to make available for the users of your facility.
2. Click **Save changes** to apply and finish.



Note: To manage facility configuration settings you must be logged-in as **Device Owner** or **Admin**.

1.2.3 Manage Users

You can search for, filter, add, and edit user accounts in Kolibri from the **Users** tab in your **Manage** dashboard.

The screenshot shows the Kolibri Management interface. The top bar indicates the user is signed in as a Device Owner. The sidebar on the left contains links for Learn, Coach, Manage, Profile, and Sign out. The main content area has tabs for Classes, Users, Facility, Data, and Content. The 'Users' tab is selected, showing a list of all users. The list includes columns for Username, Full Name, and Edit. Users listed are: alice, george (Coach), Joe, John, sally (Admin), sarah, and tom (Coach). An 'ADD NEW' button is located in the top right of the user list area.

Username	Full Name	Edit
alice	Alice Ally	
george	George Georger	
Joe	Joe Doe	
John	John Smith	
sally	Sally Salan	
sarah	Sarah Serao	
tom	Tom Tomer	

Note: To manage Kolibri users you must be logged-in as **Device Owner** or **Admin**.

Kolibri User Roles

Kolibri users can have different roles with respective access to features:

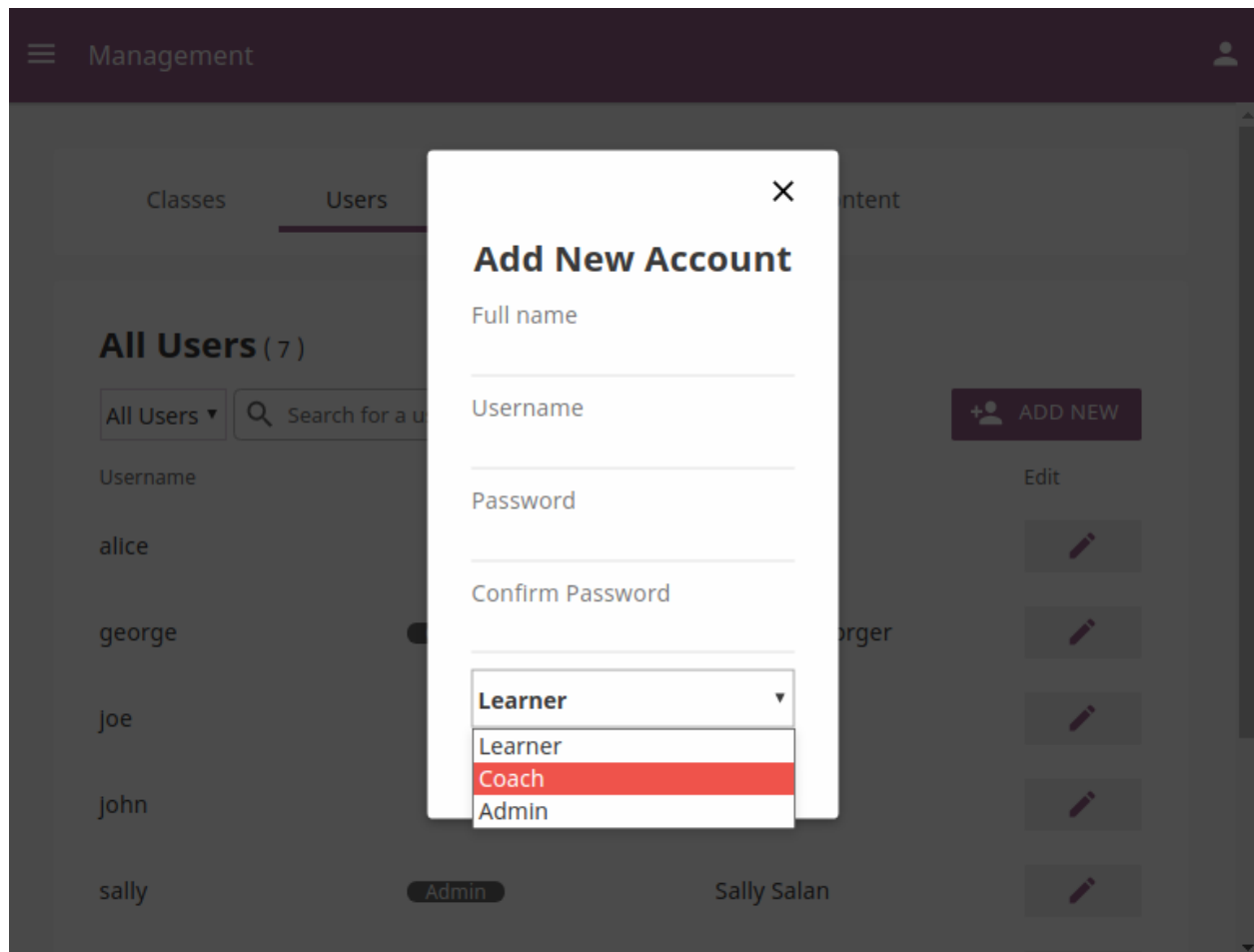
- **Learners** can:
 - View content and have their progress tracked
- **Coaches** can:
 - View content and have their progress tracked
 - View *Coach Reports* to track progress of other users and usage stats for individual exercises
 - Create/Edit/Delete *Groups* in *Classes* and add users to them
 - Create/Edit/Delete *Exams* and assign them to users
- **Admins** can:
 - View content and have their progress tracked
 - View *Coach Reports* to track progress of other users and usage stats for individual exercises

- Create/Edit/Delete other **Admins, Coaches, and Learners**
- Create/Edit/Delete *Groups* in *Classes* and add users to them
- Create/Edit/Delete *Classes* and enroll users in them
- View/Edit *Facility configuration* settings
- Export *Detail* and *Summary* logs usage data
- **Device Owners** can:
 - View content
 - Create/Edit/Delete **Admins, Coaches, and Learners**
 - View/Edit *Facility configuration* settings
 - Export *Detail* and *Summary* logs usage data
 - Import/Export content

Create a New User Account

To create a new user account, follow these steps.

1. Click **Add New** button.
2. Fill in the required information (name, username, password).
3. Select user profile (*Admin, Coach* or *Learner*).
4. Click **Create Account** to add the new user.



Select Users by Type

1. Click **All Users** selector to display user types.
2. Toggle between options to filter the user roster according to type, or leave it as **All Users** to display all.

Users

Data




Content

All Users (7)

All Users ▾
All Users
Admins
Coaches
Learners

+ Add New

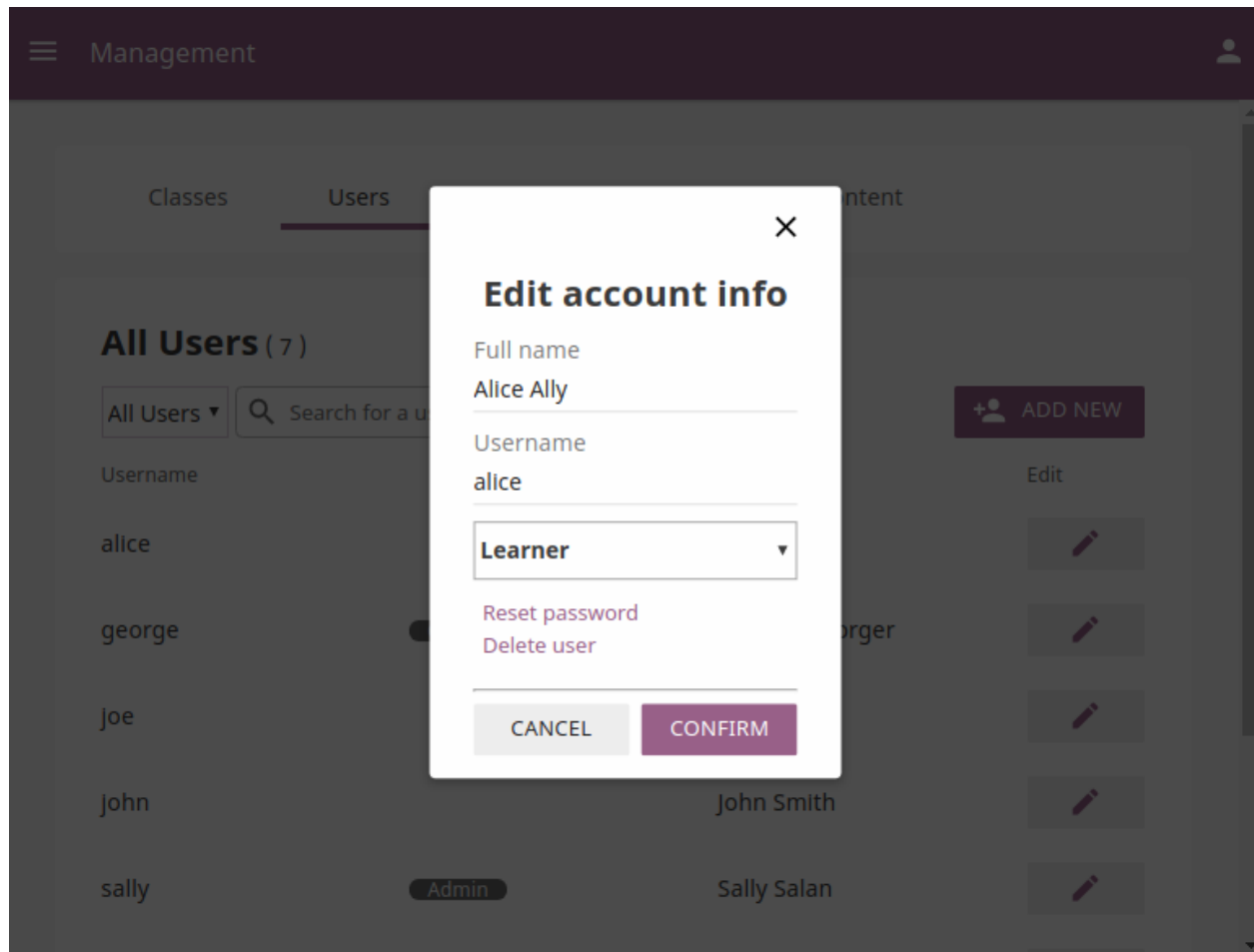
Edit

george		
sally	Coach	
tom	Admin	

Edit User's Account

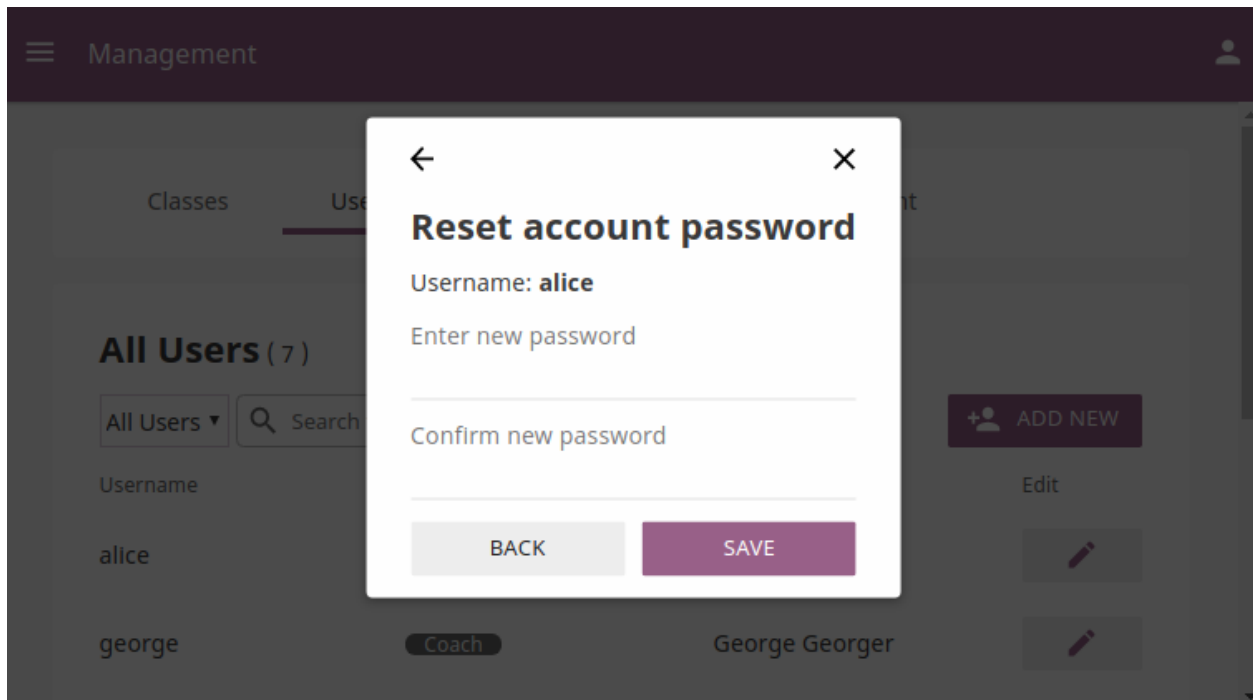
To edit username or the full name account, follow these steps.

1. Click on the **Edit** button (pencil icon) next to the user's name.
2. Edit **Full name** or **Username** in the **Edit account info** window.
3. Click **Confirm** to update the edited information, or **Cancel** to exit without saving.



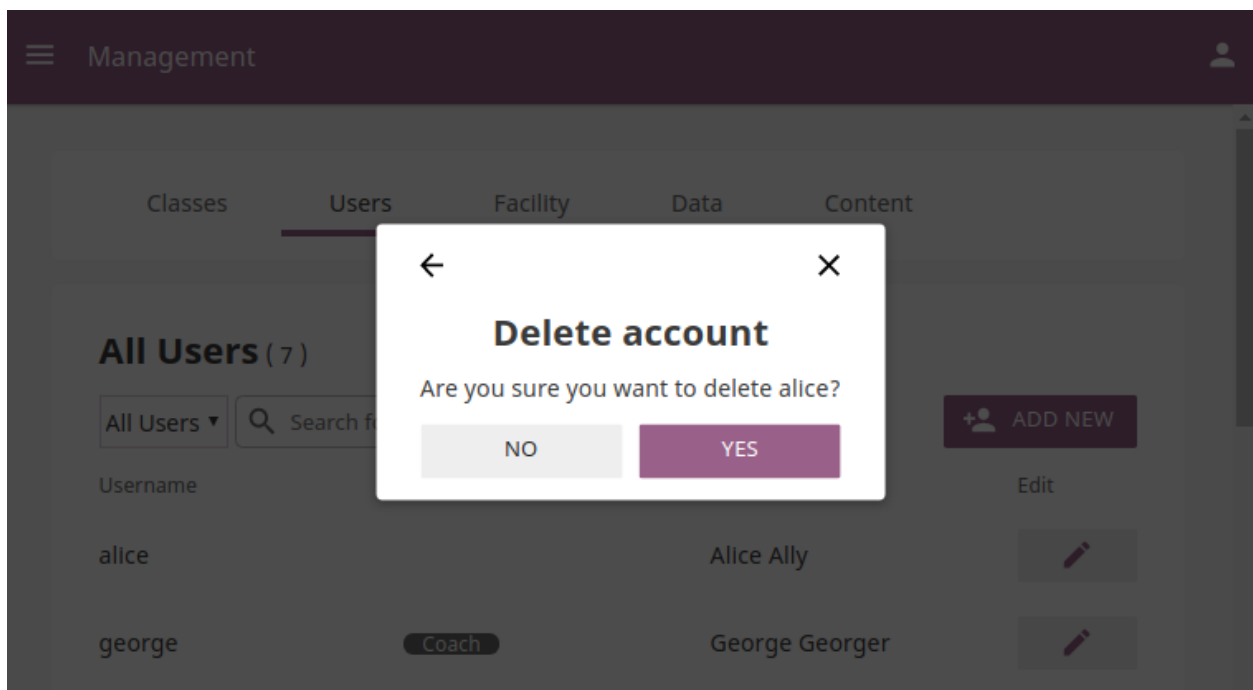
Reset User's Password

1. Click **Reset password** in the **Edit account info** window.
2. Enter the new password in both fields.
3. Click **Save** to confirm, or **Back** to exit without changing the password.



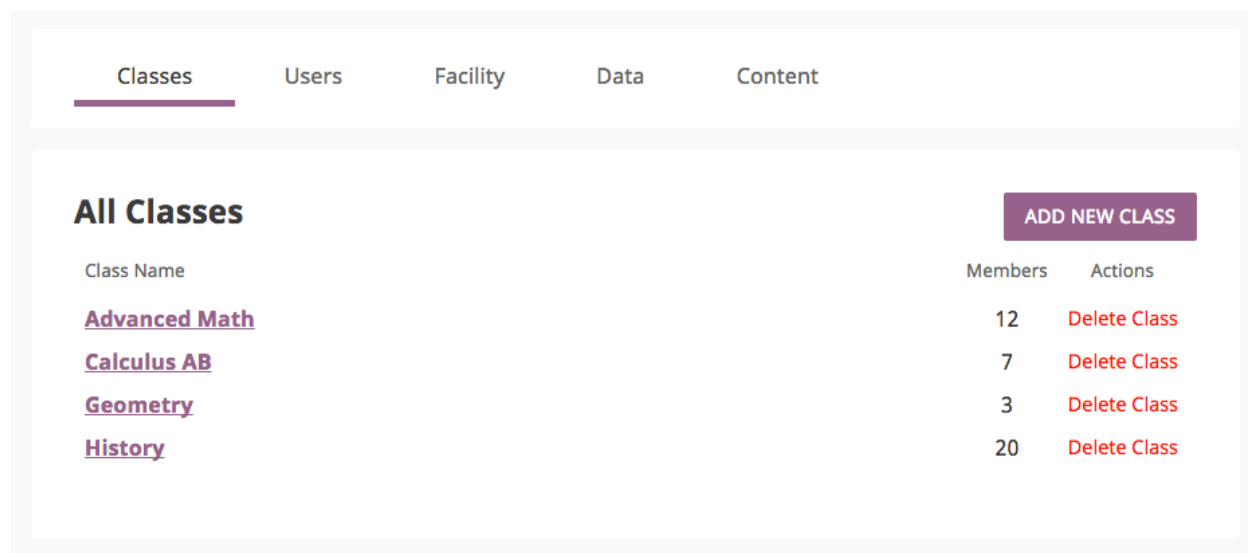
Delete User's Account

1. Click **Delete user** in the **Edit account info** window.
2. Click **Yes** to confirm, or **No** to exit without deleting the account.



1.2.4 Manage Classes

You can view, create and delete classes, as well as search, filter and enroll Kolibri users in them, using the **Classes** tab in your **Manage** dashboard. Default view displays the list of all classes in your facility, with the number of enrolled users for each class.



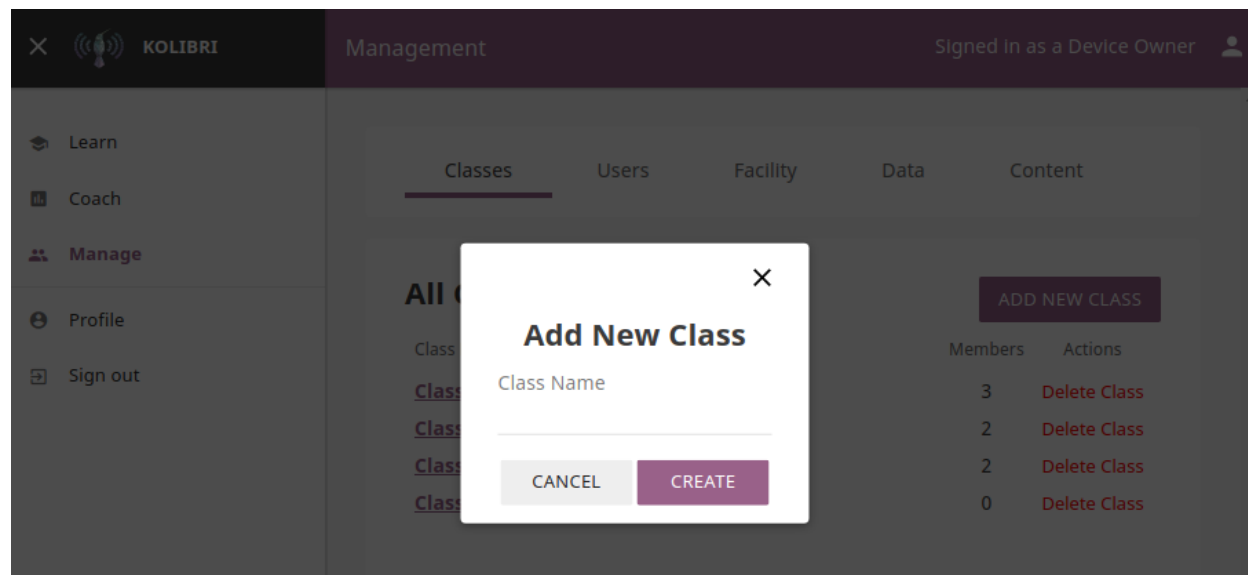
Class Name	Members	Actions
Advanced Math	12	Delete Class
Calculus AB	7	Delete Class
Geometry	3	Delete Class
History	20	Delete Class

Note: To manage Kolibri classes and groups you must be logged-in as **Device Owner** or **Admin**.

Add New Class

To add a new class, follow these steps.

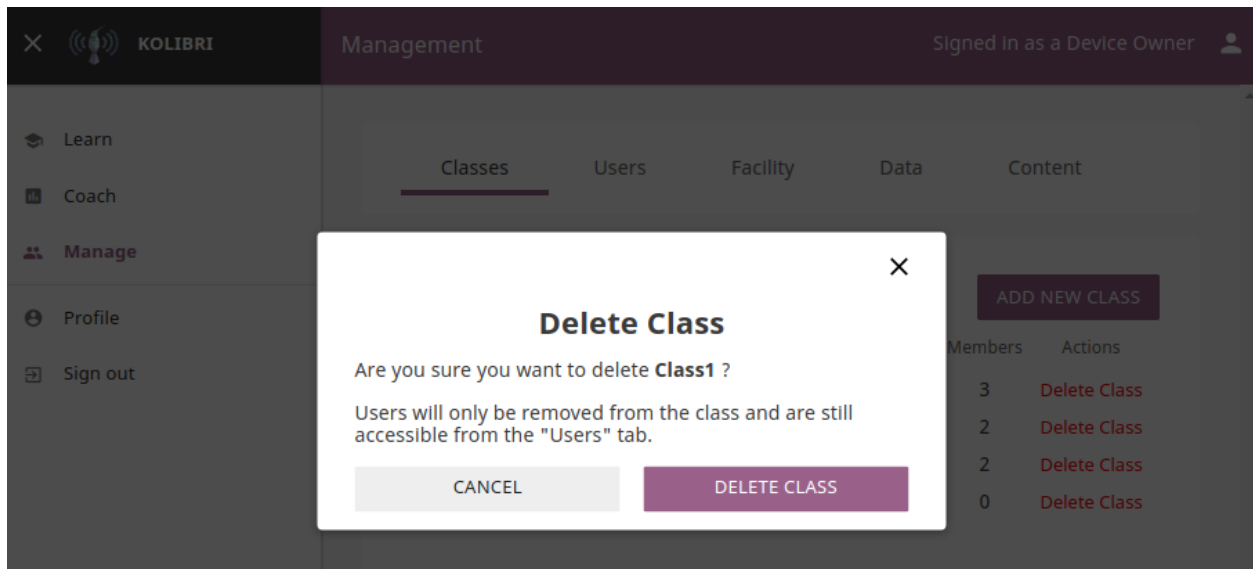
1. Click **Add new class** button.
2. Fill in the class name.
3. Click **Create** to add the new class, or **Cancel** to exit.



Delete Class

To delete class, follow these steps.

1. Click **Delete class** button for the chosen class from the list.
2. Click **Delete class** in the confirmation window to proceed, or **Cancel** to exit without deleting the class.



Note: Users enrolled in the class you are deleting will not be removed from the database.

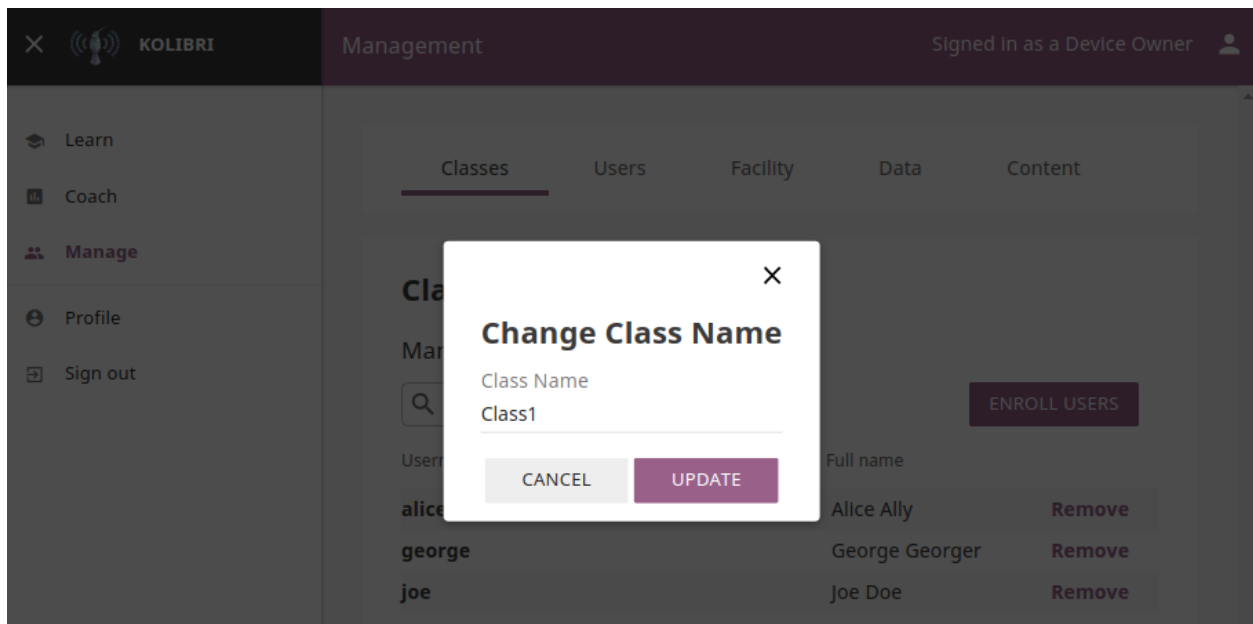
Edit Class and Enroll Users

To edit a class select it from the default view in the **Classes** tab. In the following **Class** view you can change class name, remove currently enrolled users from the class and enroll new ones.

Change Class Name

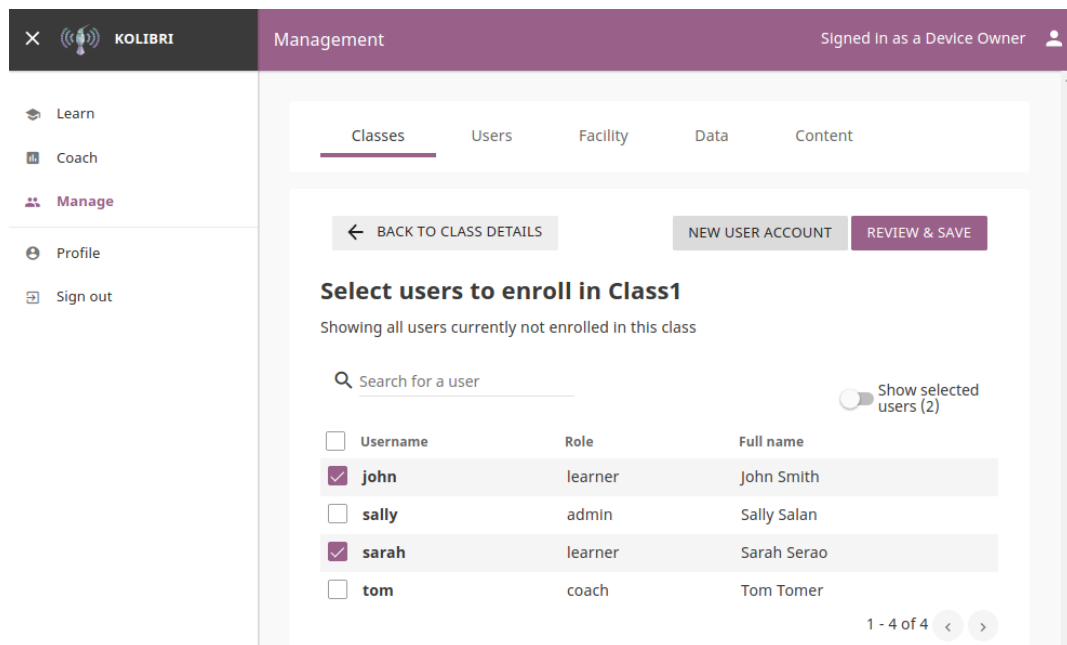
To edit class name, follow these steps.

1. Click on the **Edit** button (pencil icon) next to the class' name.
2. Write the new name in the **Class name** field.
3. Click **Update** to confirm the edited information, or **Cancel** to exit without saving.

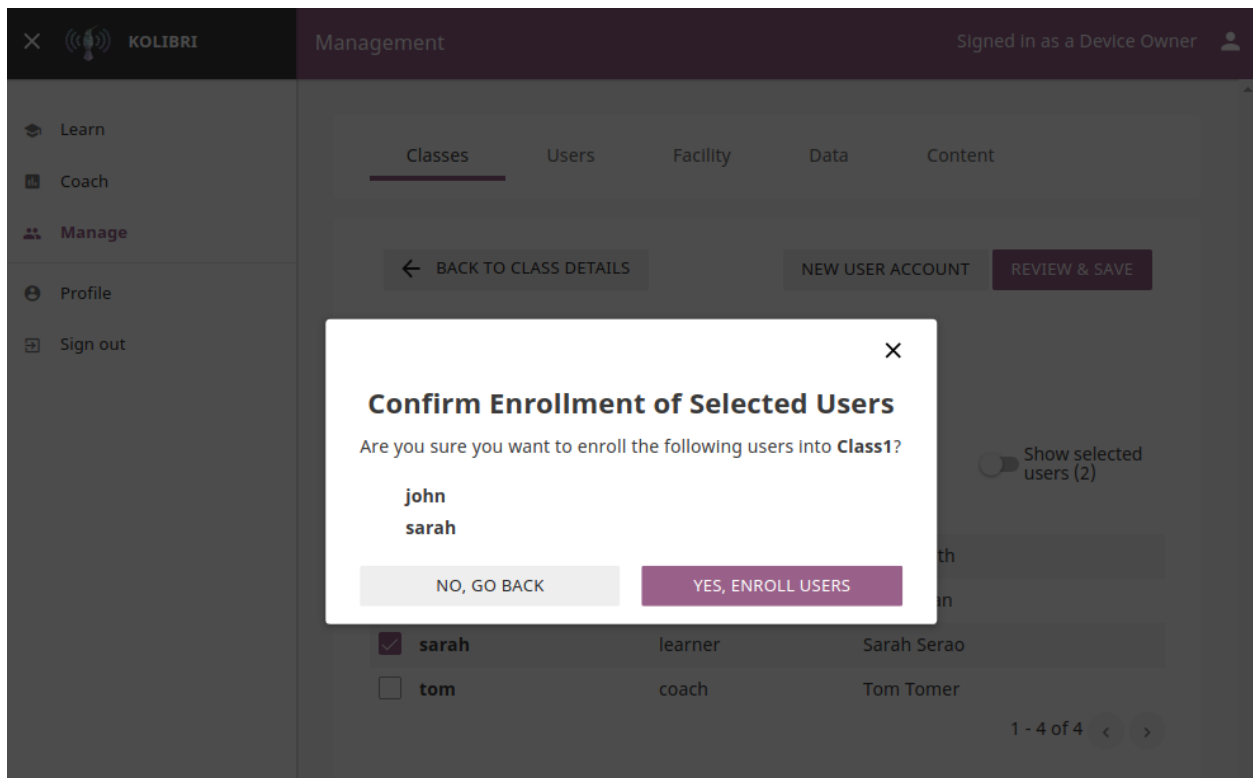


Enroll users to class

1. Click **Enroll users** button.

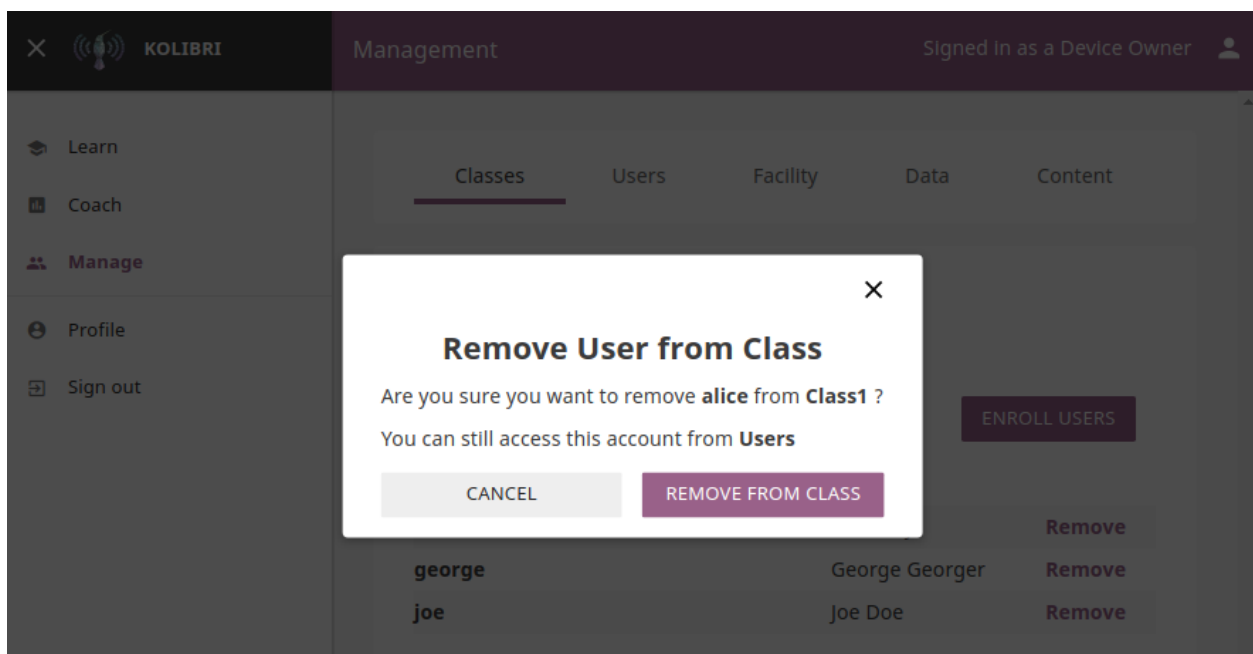


- List in this view contains all the users currently **NOT** enrolled for the selected class.
 - You can search for a specific user by name.
2. Use checkboxes to select all the user in the list, or specific users you want to enroll to class. You can also use the **New user account** button to create a new user AND enroll them at the same time.
 3. Click **Review & save** button.
 4. Click **Yes, enroll users** to confirm, or **No, go back** to exit without enrolling the selected users.



Remove users from class

1. Click **Remove** button for the chosen user.
2. Click **Remove from class** to confirm, or **Cancel** to exit without removing the user.



Note: Users removed from the class will not be deleted from the database, and you can still access their account from the **Users** tab in the **Manage** dashboard.

1.2.5 Manage Data

Note: To manage Kolibri usage data you must be logged-in as **Device Owner** or **Admin**.

You can download Kolibri *Detail* and *Summary* logs usage data and export in the CSV format from the **Data** tab in your **Manage** dashboard.

Users


Data

Export Usage Data

Download CSV (comma-separated value) files containing information about users and their interactions with the content on this device.

Detail Logs


Individual visits to each piece of content.

 Download

Note: When a user views a piece of content, we record how long they spend and the progress they make. Each row in this file records a single visit a user made to a specific piece of content. This includes anonymous usage, when no user is logged in.

Summary Logs

Total time/progress for each piece of content.

 Download

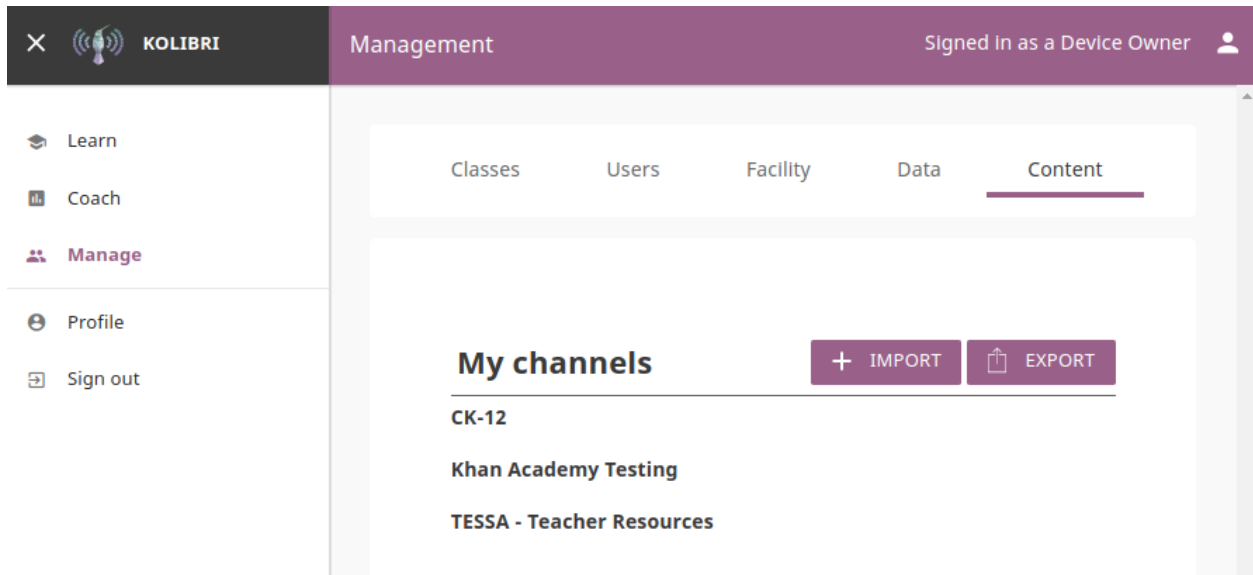
Note: A user may visit the same piece of content multiple times. This file records the total time and progress each user has achieved for each piece of content, summarized across possibly more than one visit. Anonymous usage is not included.

1.2.6 Manage Content

Note: To manage Kolibri content you must be logged-in as **Device Owner**.

Kolibri **Content Channel** is a collection of educational resources (video, audio or document files) prepared and organized by the content curator for their use in Kolibri. Each Kolibri **Content Channel** has its own *Content Channel ID* on [Kolibri content curation server](#) database that you will receive from the content curator who assembled the channel.

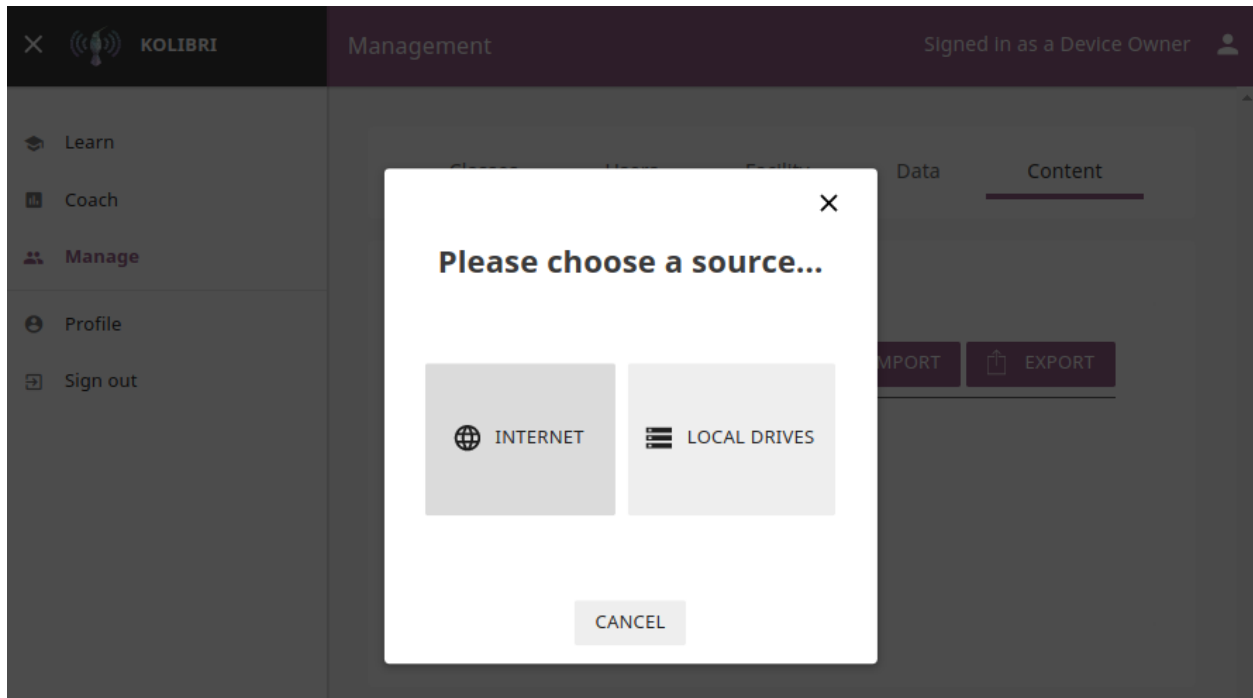
You can import and export **Content Channels** for Kolibri in the **Content** tab.



Import Content Channel to Kolibri

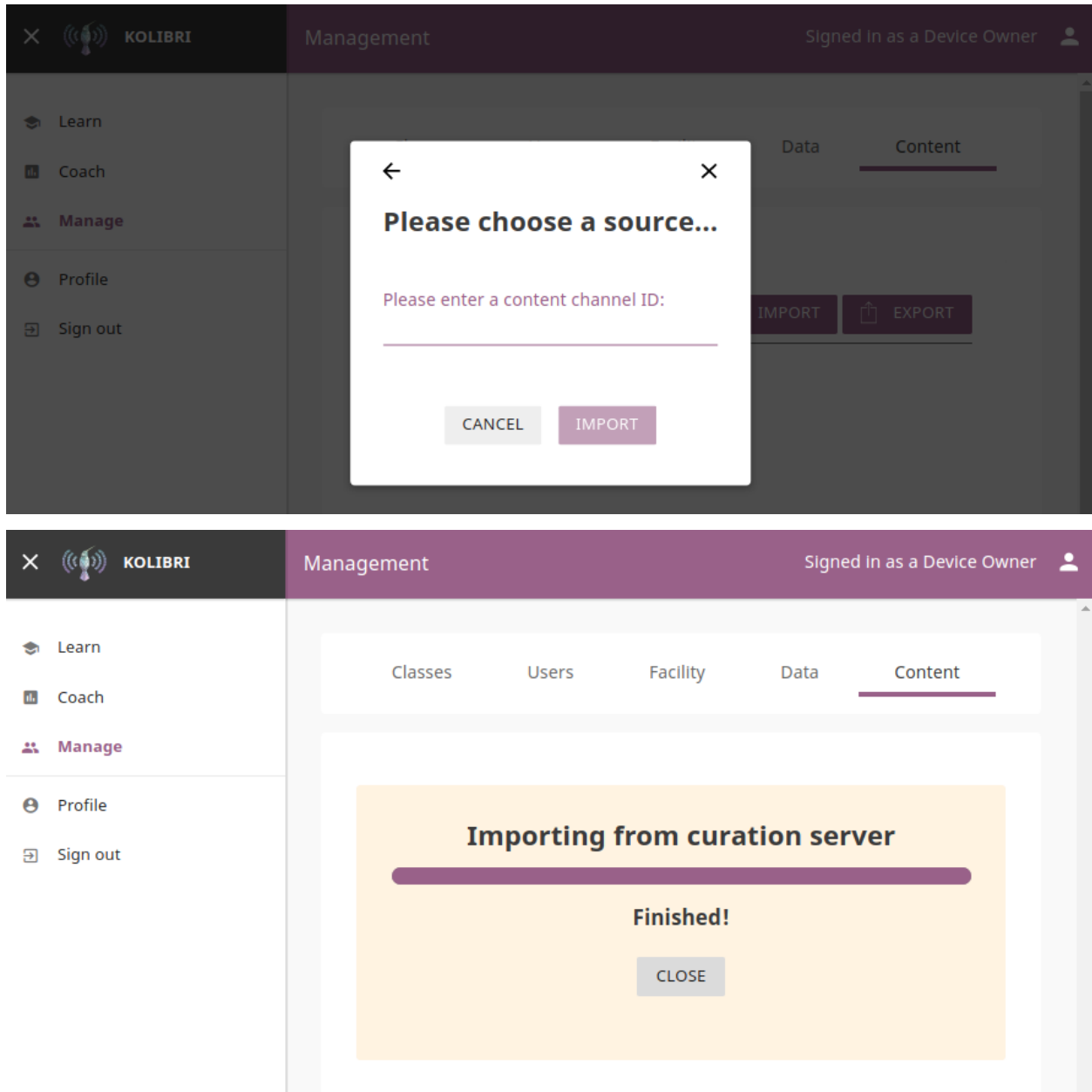
To import **Content Channel** to Kolibri, follow these steps.

1. Click **+ Import** button in **My Channels** pane.
2. Choose the source option (*Internet* or *Local Drives*).



Import Content Channel from the Internet

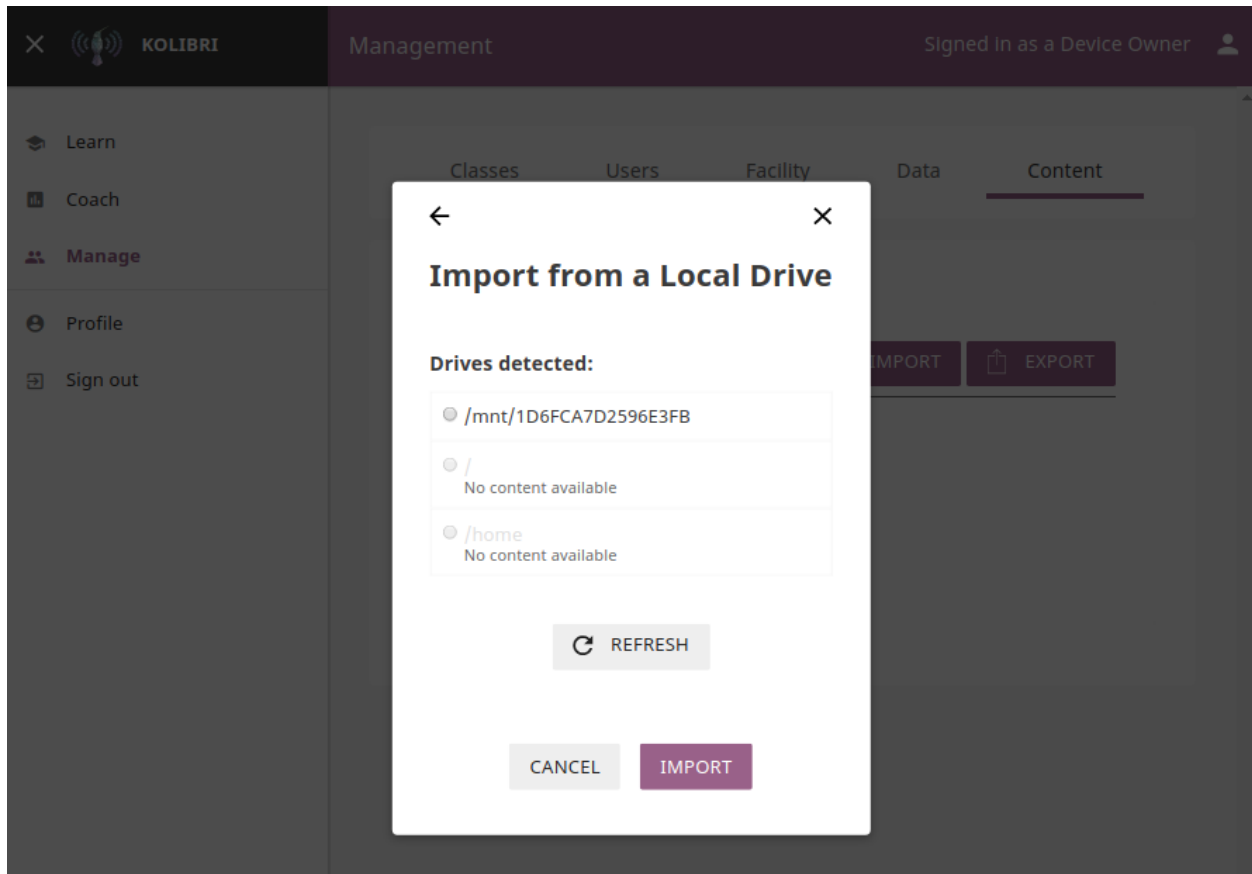
1. Choose option for *Internet*.
2. Enter *Content Database ID* for the desired channel from the content curation server.
3. Click **Import** button.
4. Wait for the content to be downloaded and appear under the **My Channels** heading.



Import Content Channel from a Local Drive

1. Choose option for *Local Drives*.
2. Kolibri will automatically detect the drive(s) with available content files.

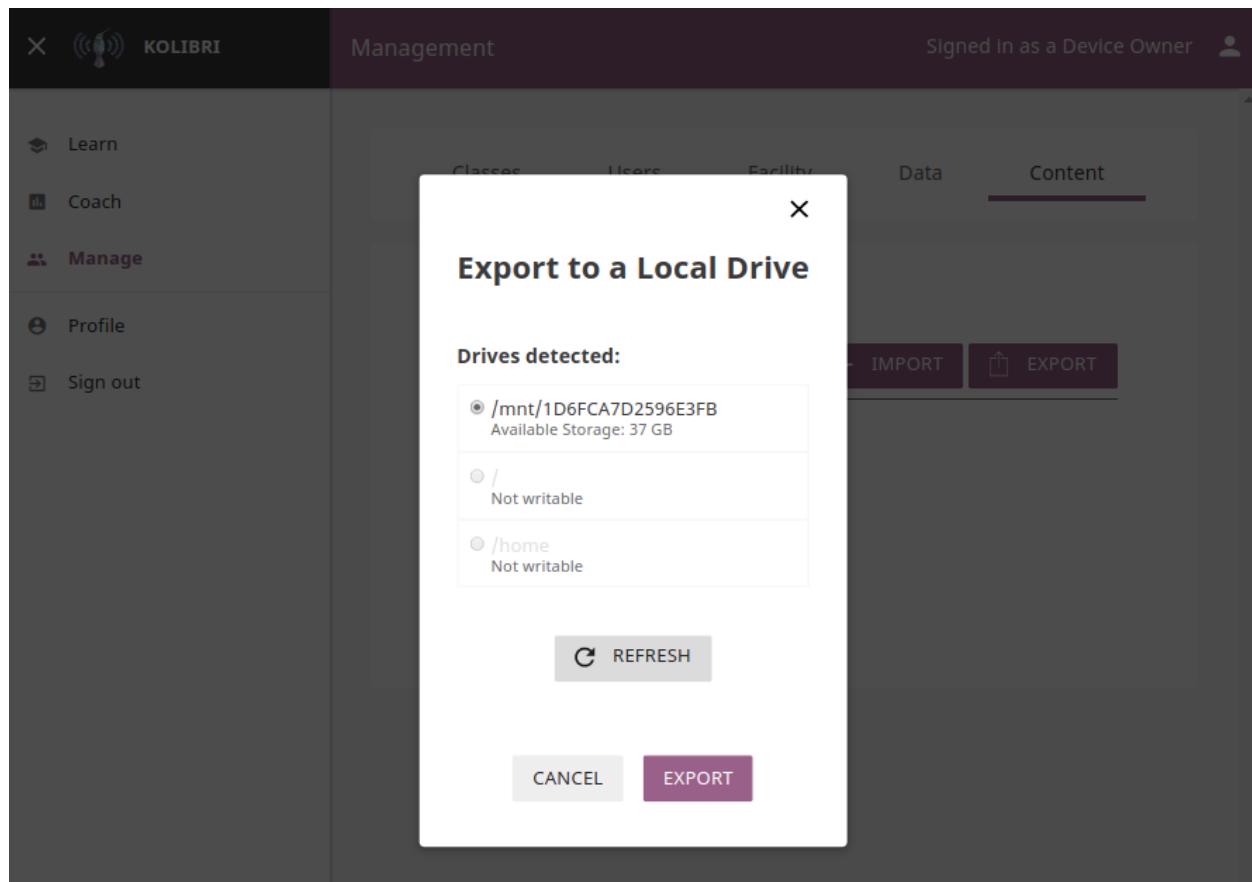
3. Select the drive where the channel content is stored.
4. Click **Import** button.
5. Wait for the content to be imported and appear under the **My Channels** heading.



Note: If the local drive is not detected, try re-inserting the storage device (USB key or external hard disk) and pressing the button **Refresh**.

Export from Kolibri to Local Drive

1. Click **Export** button in **My Channels** pane.
2. Select the local drive where you wish to export **Kolibri** content.
3. Click **Export** button.



1.2.7 Get support

If you want to contact **Learning Equality** Support team to report an issue, or share your experience about using Kolibri, please register at our [Community Forums](#).

Once you register on our forums, please read the the first two pinned topics (*Welcome to LE's Support Community* and *How do I post to this forum?*)

You can add the new topic with the + **New Topic** button on the right. Make sure to select the **Kolibri** category in the **Create a New Topic** window so it's easier to classify and respond to.

https://community.learningequality.org

LEARNING EQUALITY

To make launching your new site easier, you are in bootstrap mode. All new users will be granted trust level 1 and have daily email digest updates enabled. This will be automatically turned off when total user count exceeds 50 users.

all categories ▸ Latest New Unread Top Categories + New Topic

Topic Category Users Replies Views Activity

Welcome to Learning Equality's Support Community

Need support? You've come to the right place. This forum is for the Learning Equality community to ask and answer questions on topics relating to: * KA Lite general support, such as installation and set-up. Im... [read more](#)

How do I post to this forum?

If you have a story to share, we cannot wait to hear it! The more details the better. If you have an inquiry, construct your post with the following structure:

Create a new Topic

What is this discussion about in one brief sentence?

Type here. Use Markdown, BBCode, or HTML to format. Drag or paste images.

choose optional tags for this topic

+ Create Topic cancel

languages can be found here:

- Implementations × 1
All information related to deploying KA Lite or Kolibri in your community. Pedagogical models, funding sources, best practices, etc.
- Site Feedback × 0
Discussion about this site, its organization, how it works, and how we can improve it.
- Kolibri × 0
All discussions related to the development and deployment of the Kolibri platform.

Kolibri

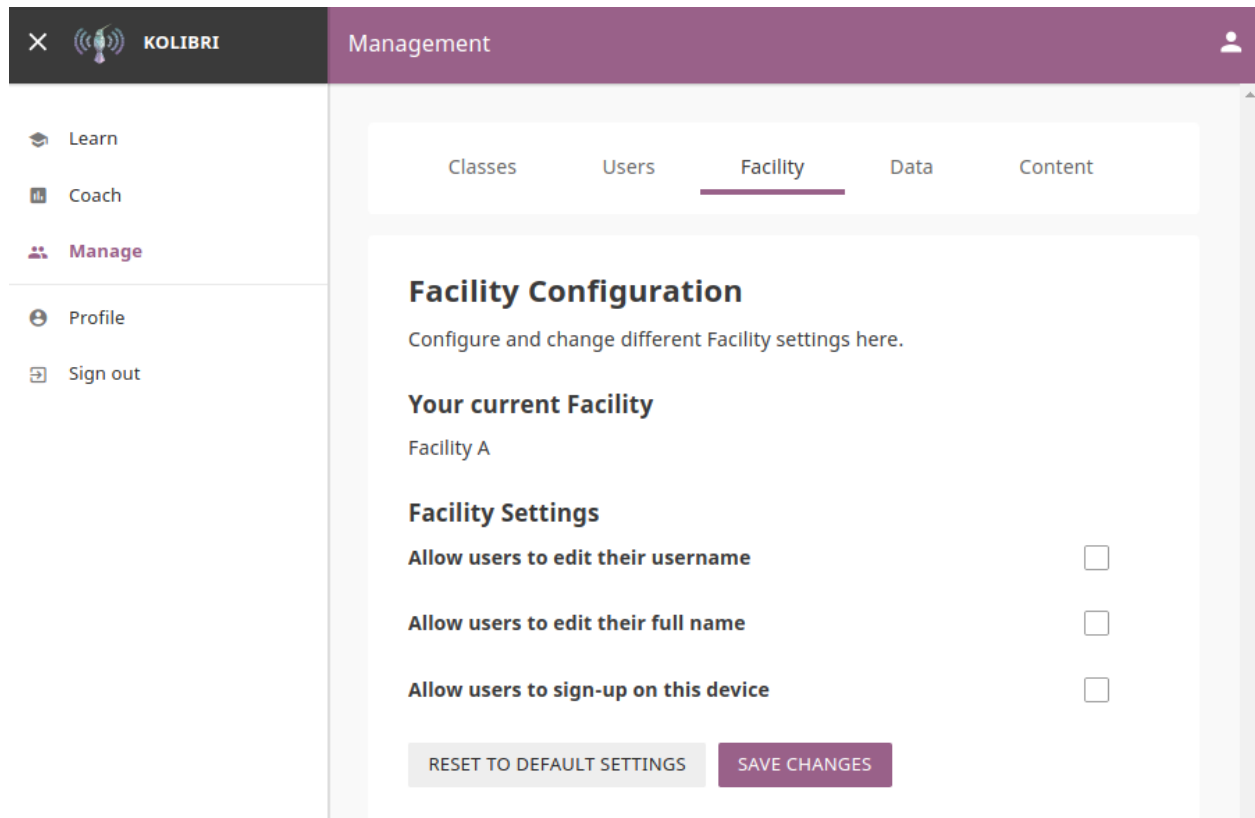
« hide preview

1.3 Admin

1.3.1 Manage Facility

You can edit facility configuration settings in Kolibri from the **Facility** tab in your **Manage** dashboard.

1. Activate the options you want to make available for the users of your facility.
2. Click **Save changes** to apply and finish.



Note: To manage facility configuration settings you must be logged-in as **Device Owner** or **Admin**.

1.3.2 Manage Users

You can search for, filter, add, and edit user accounts in Kolibri from the **Users** tab in your **Manage** dashboard.

The screenshot shows the Kolibri Management interface. The top bar indicates the user is signed in as a Device Owner. The sidebar on the left contains navigation links: Learn, Coach, Manage (highlighted), Profile, and Sign out. The main content area has tabs for Classes, Users (selected), Facility, Data, and Content. Under the Users tab, there is a section titled 'All Users (7)'. It includes a dropdown menu set to 'All Users', a search bar, and an 'ADD NEW' button. Below this is a table of users:

Username	Full Name	Edit
alice	Alice Ally	
george	George Georger	
joe	Joe Doe	
john	John Smith	
sally	Sally Salan	
sarah	Sarah Serao	
tom	Tom Tomer	

Roles are indicated by badges: 'Coach' for george and tom, and 'Admin' for sally.

Note: To manage Kolibri users you must be logged-in as **Device Owner** or **Admin**.

Kolibri User Roles

Kolibri users can have different roles with respective access to features:

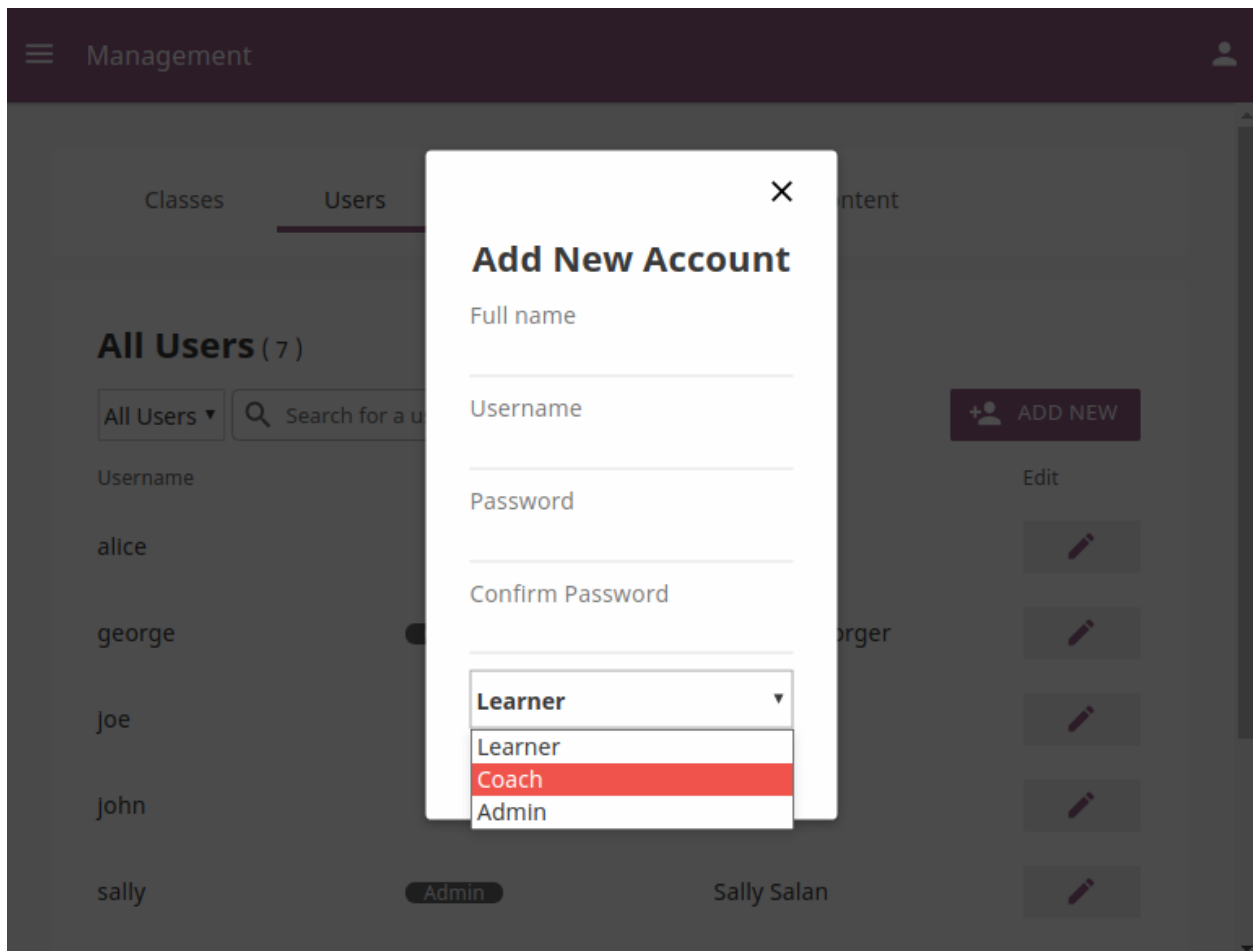
- **Learners** can:
 - View content and have their progress tracked
- **Coaches** can:
 - View content and have their progress tracked
 - View *Coach Reports* to track progress of other users and usage stats for individual exercises
 - Create/Edit/Delete *Groups* in *Classes* and add users to them
 - Create/Edit/Delete *Exams* and assign them to users
- **Admins** can:
 - View content and have their progress tracked
 - View *Coach Reports* to track progress of other users and usage stats for individual exercises

- Create/Edit/Delete other **Admins**, **Coaches**, and **Learners**
- Create/Edit/Delete *Groups* in *Classes* and add users to them
- Create/Edit/Delete *Classes* and enroll users in them
- View/Edit *Facility configuration* settings
- Export *Detail* and *Summary* logs usage data
- **Device Owners** can:
 - View content
 - Create/Edit/Delete **Admins**, **Coaches**, and **Learners**
 - View/Edit *Facility configuration* settings
 - Export *Detail* and *Summary* logs usage data
 - Import/Export content

Create a New User Account

To create a new user account, follow these steps.

1. Click **Add New** button.
2. Fill in the required information (name, username, password).
3. Select user profile (*Admin*, *Coach* or *Learner*).
4. Click **Create Account** to add the new user.



Select Users by Type

1. Click **All Users** selector to display user types.
2. Toggle between options to filter the user roster according to type, or leave it as **All Users** to display all.

Users

Data




Content

All Users (7)

All Users ▾
All Users
Admins
Coaches
Learners

+ Add New

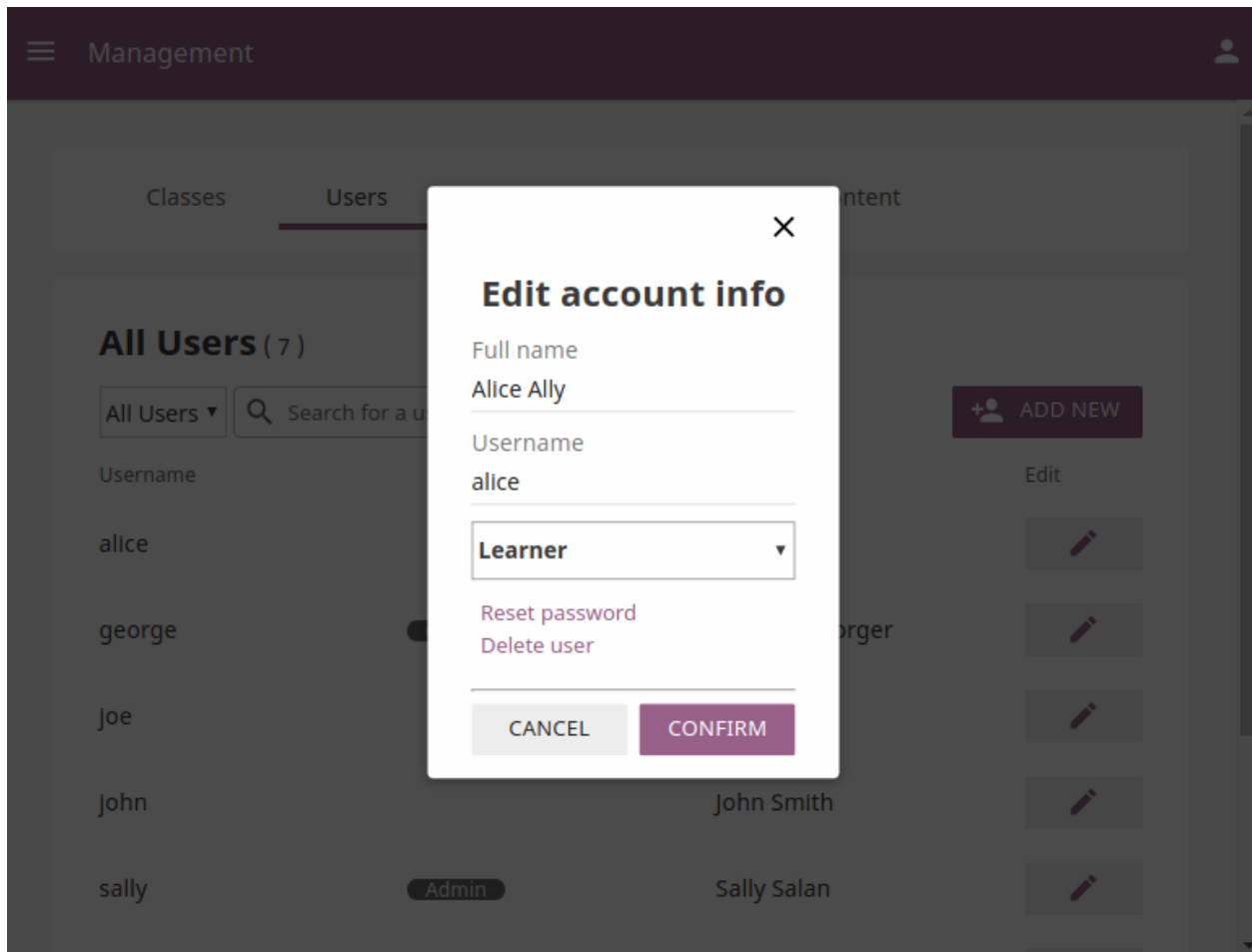
Edit

george		
sally	Coach	
tom	Admin	

Edit User's Account

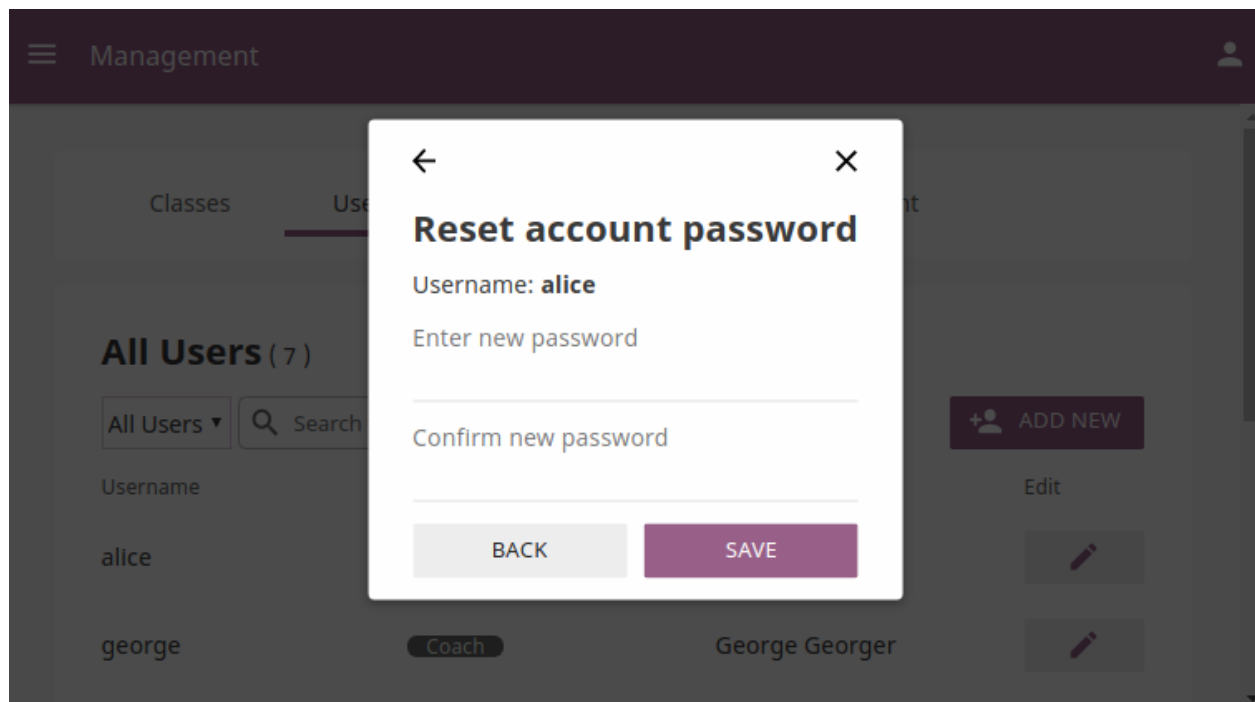
To edit username or the full name account, follow these steps.

1. Click on the **Edit** button (pencil icon) next to the user's name.
2. Edit **Full name** or **Username** in the **Edit account info** window.
3. Click **Confirm** to update the edited information, or **Cancel** to exit without saving.



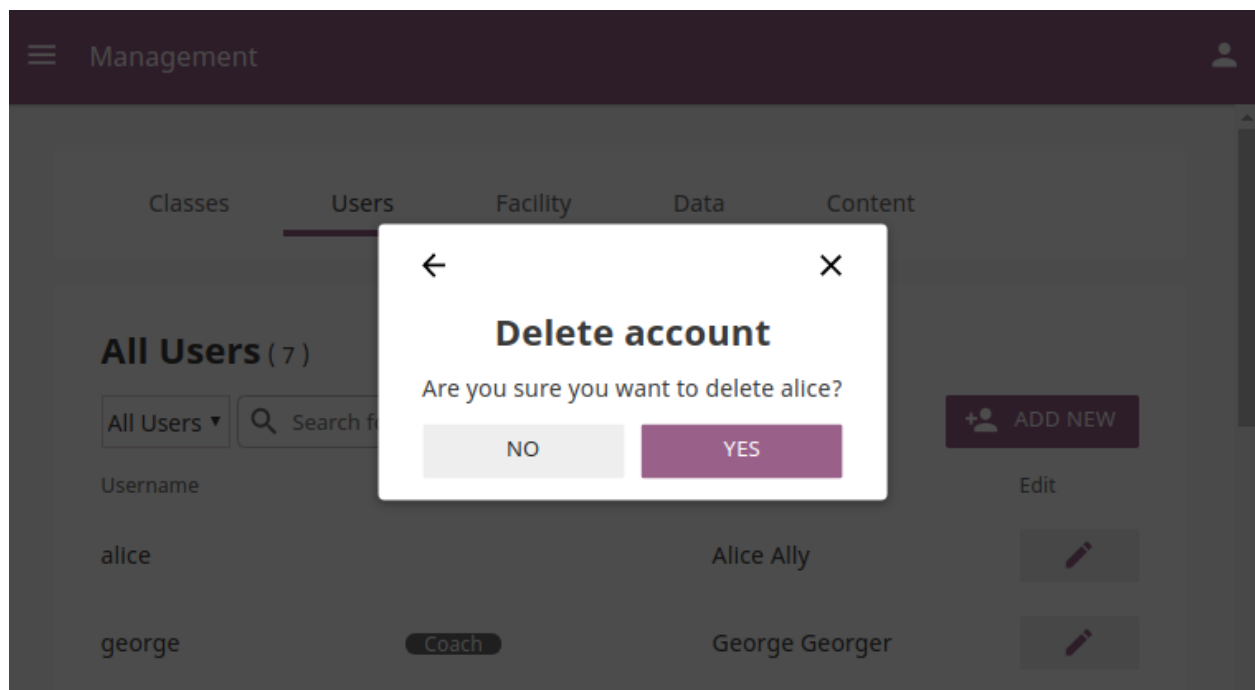
Reset User's Password

1. Click **Reset password** in the **Edit account info** window.
2. Enter the new password in both fields.
3. Click **Save** to confirm, or **Back** to exit without changing the password.



Delete User's Account

1. Click **Delete user** in the **Edit account info** window.
2. Click **Yes** to confirm, or **No** to exit without deleting the account.



1.3.3 Manage Classes

You can view, create and delete classes, as well as search, filter and enroll Kolibri users in them, using the **Classes** tab in your **Manage** dashboard. Default view displays the list of all classes in your facility, with the number of enrolled users for each class.

Class Name	Members	Actions
Advanced Math	12	Delete Class
Calculus AB	7	Delete Class
Geometry	3	Delete Class
History	20	Delete Class

Note: To manage Kolibri classes and groups you must be logged-in as **Device Owner** or **Admin**.

Add New Class

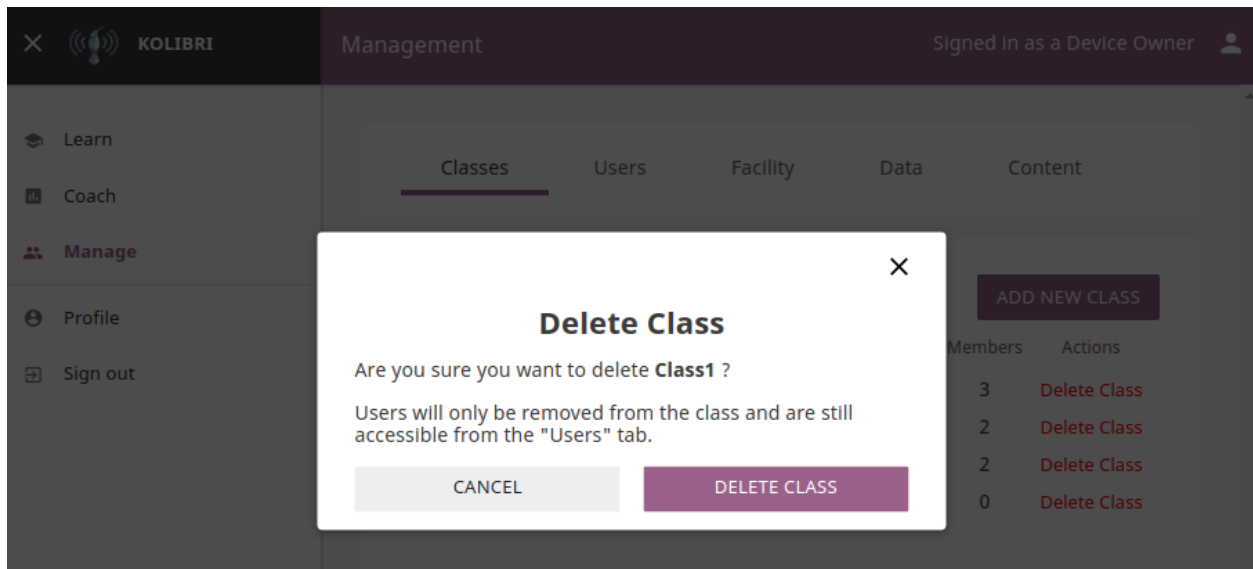
To add a new class, follow these steps.

1. Click **Add new class** button.
2. Fill in the class name.
3. Click **Create** to add the new class, or **Cancel** to exit.

Delete Class

To delete class, follow these steps.

1. Click **Delete class** button for the chosen class from the list.
2. Click **Delete class** in the confirmation window to proceed, or **Cancel** to exit without deleting the class.



Note: Users enrolled in the class you are deleting will not be removed from the database.

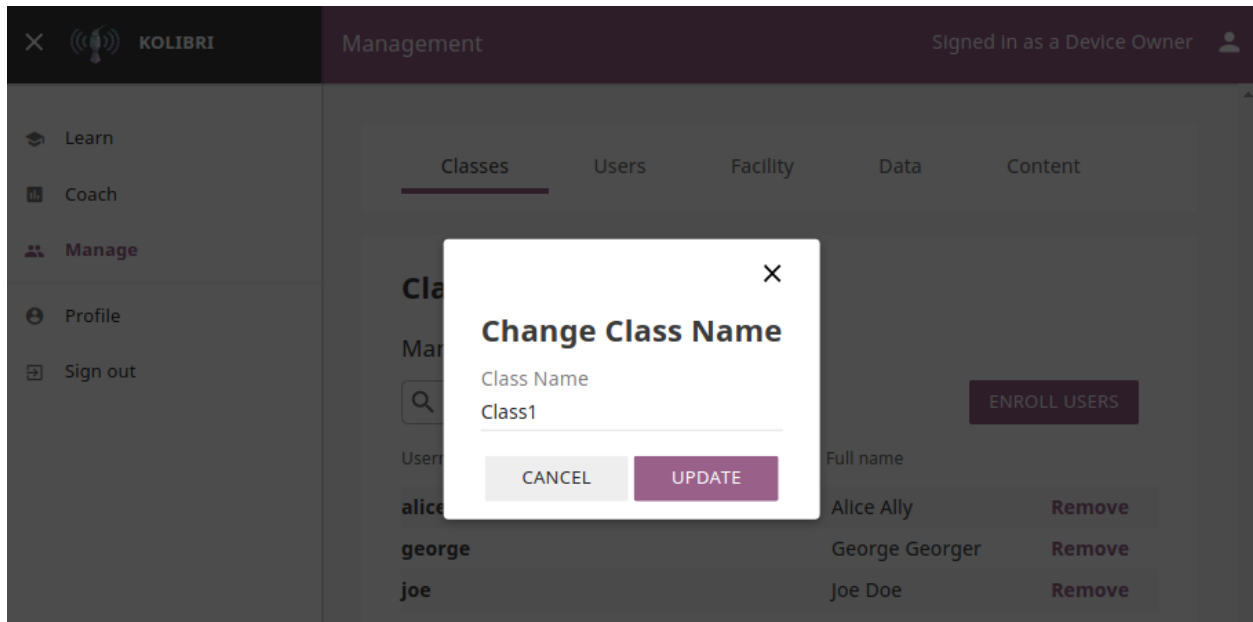
Edit Class and Enroll Users

To edit a class select it from the default view in the **Classes** tab. In the following **Class** view you can change class name, remove currently enrolled users from the class and enroll new ones.

Change Class Name

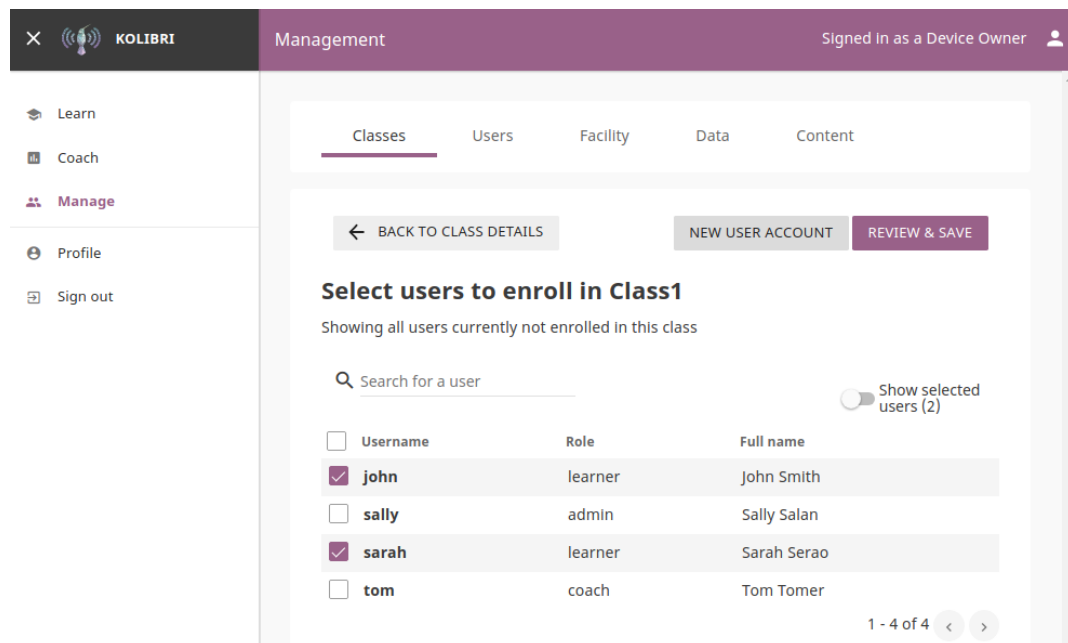
To edit class name, follow these steps.

1. Click on the **Edit** button (pencil icon) next to the class' name.
2. Write the new name in the **Class name** field.
3. Click **Update** to confirm the edited information, or **Cancel** to exit without saving.

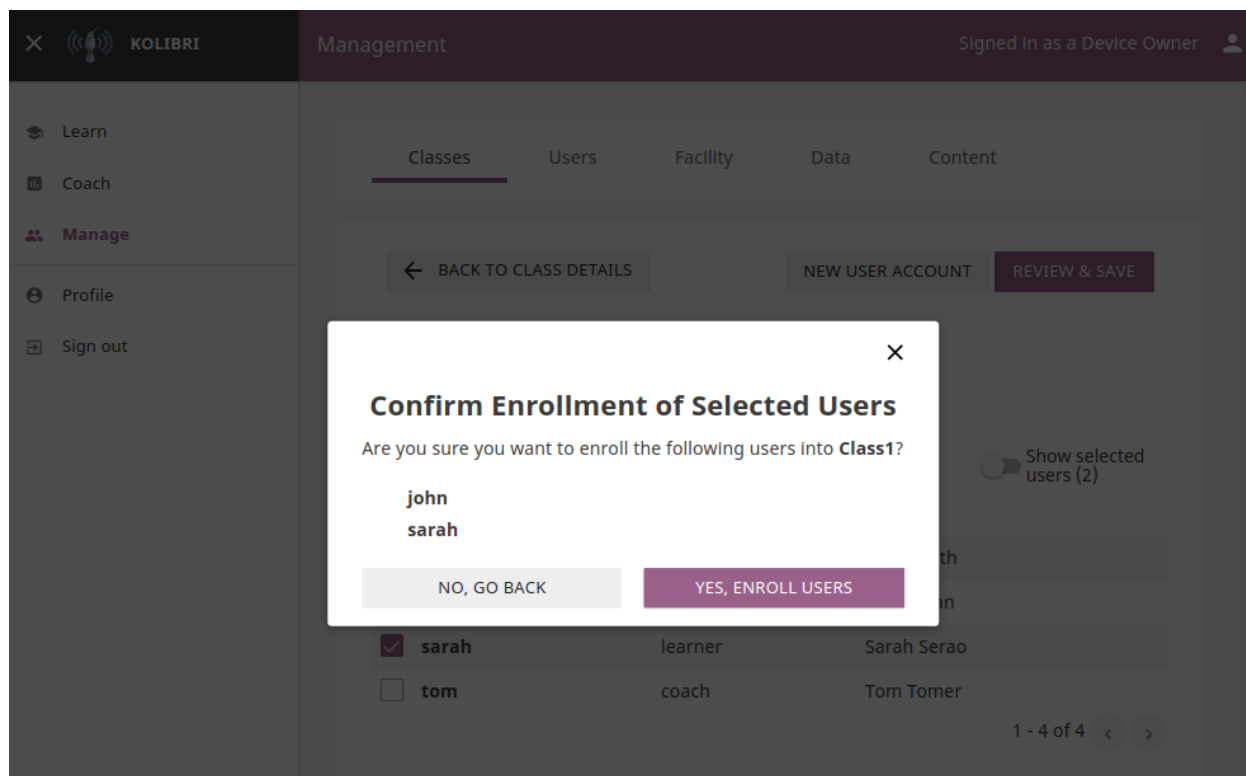


Enroll users to class

1. Click **Enroll users** button.

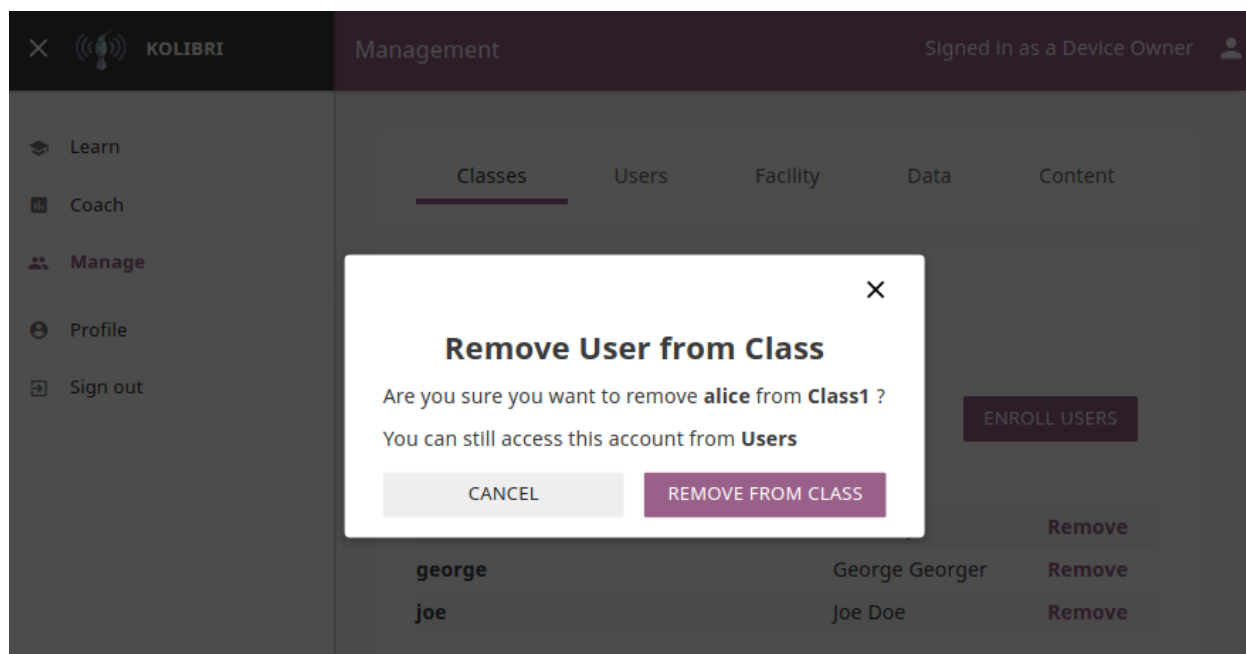


- List in this view contains all the users currently **NOT** enrolled for the selected class.
 - You can search for a specific user by name.
2. Use checkboxes to select all the user in the list, or specific users you want to enroll to class. You can also use the **New user account** button to create a new user AND enroll them at the same time.
 3. Click **Review & save** button.
 4. Click **Yes, enroll users** to confirm, or **No, go back** to exit without enrolling the selected users.



Remove users from class

1. Click **Remove** button for the chosen user.
2. Click **Remove from class** to confirm, or **Cancel** to exit without removing the user.



Note: Users removed from the class will not be deleted from the database, and you can still access their account from the **Users** tab in the **Manage** dashboard.

1.3.4 Manage Data

Note: To manage Kolibri usage data you must be logged-in as **Device Owner** or **Admin**.

You can download Kolibri *Detail* and *Summary* logs usage data and export in the CSV format from the **Data** tab in your **Manage** dashboard.

Users

Data

Export Usage Data

Download CSV (comma-separated value) files containing information about users and their interactions with the content on this device.

Detail Logs

Individual visits to each piece of content.

Download

Note: When a user views a piece of content, we record how long they spend and the progress they make. Each row in this file records a single visit a user made to a specific piece of content. This includes anonymous usage, when no user is logged in.

Summary Logs

Total time/progress for each piece of content.

Download

Note: A user may visit the same piece of content multiple times. This file records the total time and progress each user has achieved for each piece of content, summarized across possibly more than one visit. Anonymous usage is not included.

1.3.5 Get support

If you want to contact **Learning Equality** Support team to report an issue, or share your experience about using Kolibri, please register at our [Community Forums](#).

Once you register on our forums, please read the the first two pinned topics (*Welcome to LE's Support Community* and *How do I post to this forum?*)

You can add the new topic with the **+ New Topic** button on the right. Make sure to select the **Kolibri** category in the **Create a New Topic** window so it's easier to classify and respond to.

← → ↻ <https://community.learningequality.org> ★

LEARNING EQUALITY

To make launching your new site easier, you are in bootstrap mode. All new users will be granted trust level 1 and have daily email digest updates enabled. This will be automatically turned off when total user count exceeds 50 users.

all categories ▾ Latest New Unread Top Categories + New Topic

Topic Category Users Replies Views Activity

Welcome to Learning Equality's Support Community
Need support? You've come to the right place. This forum is for the Learning Equality community to ask and answer questions on topics relating to: * KA Lite general support, such as installation and set-up. Im... [read more](#)

How do I post to this forum?
If you have a story to share, we cannot wait to hear it! The more details the better. If you have an inquiry, construct your post with the following...

Create a new Topic

What is this discussion about in one brief sentence?

Type here. Use Markdown, BBCode, or HTML to format. Drag or paste images.

choose optional tags for this topic

+ Create Topic cancel

languages can be found here:


- Implementations** × 1
All information related to deploying KA Lite or Kolibri in your community. Pedagogical models, funding sources, best practices, etc.
- Site Feedback** × 0
Discussion about this site, its organization, how it works, and how we can improve it.
- Kolibri** × 0
All discussions related to the development and deployment of the Kolibri platform.

Kolibri


« hide preview


1.3.6 Coach View


As an Admin you have access to the same Kolibri **Coach** view as the *Coach* in your facility.


 KOLIBRI


Coach




 Learn

 **Coach**

 Manage

 Profile

 Sign Out

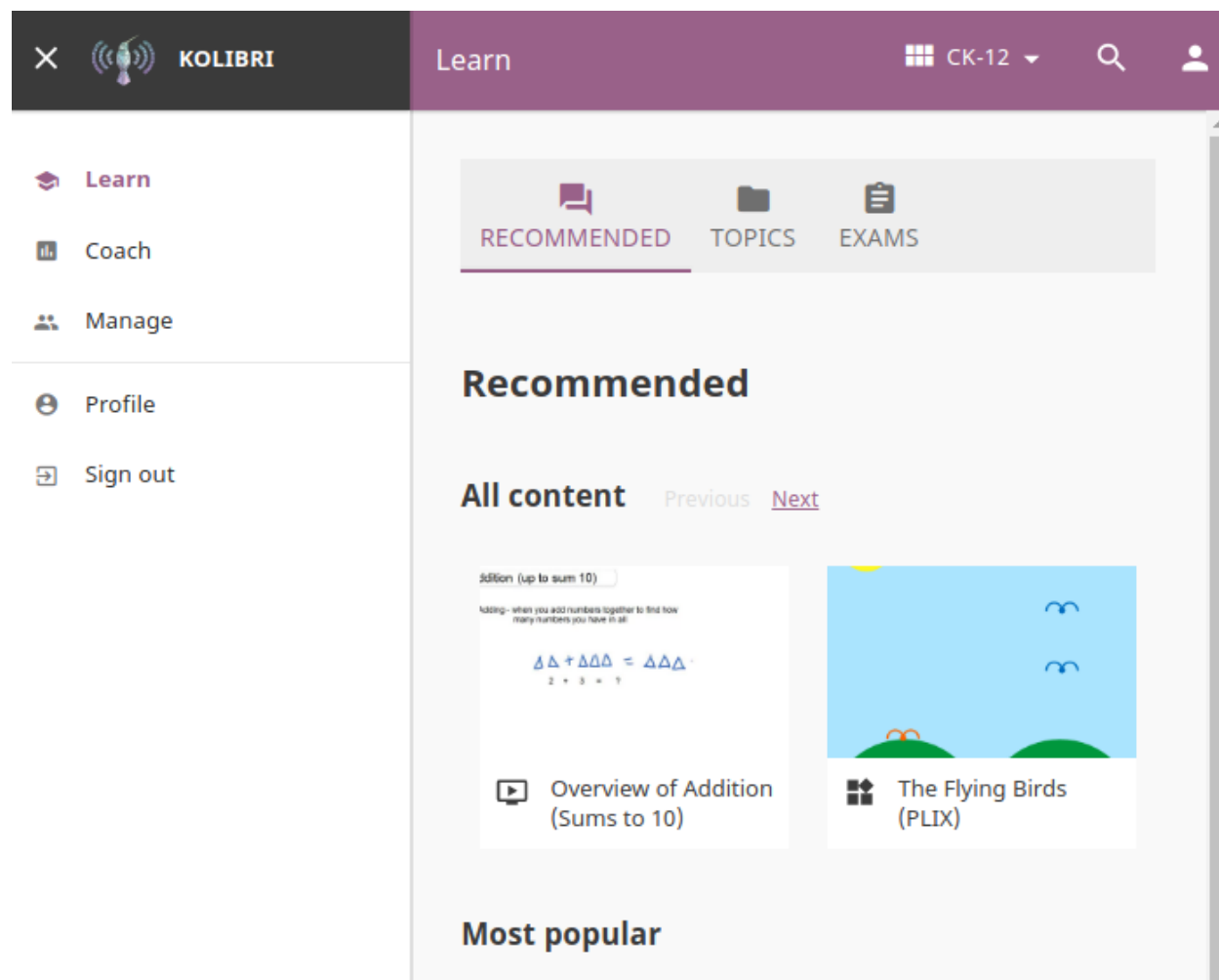
My Classes

View your Learners' progress and performance

Class Name	Learners	Groups
Math 2A	25	2
Math 2B	22	2
Science 1A	17	2

1.3.7 Learn View

As an Admin you have access to the same Kolibri **Learn** view as the *Learner* in your facility.




1.4 Coach






Warning: Coach dashboard is still under development. We strive to keep the documentation updated, but depending on the release version you have installed, screenshots of certain features may not correspond exactly.


You can track progress of the **Learner** users, create and assign **Exams** to classes or learner groups from the **Coach** dashboard. The default view of the **Coach** dashboard presents the list of **Classes** with number of learners enrolled to each class.

Select a class from the list to access the progress-tracking features and create exams.

 KOLIBRI

Coach


 Learn
 **Coach**
 Manage
 Profile

 Sign Out

My Classes

View your Learners' progress and performance

Class Name	Learners	Groups
Math 2A	25	2
Math 2B	22	2
Science 1A	17	2

1.4.1 Recent Activity View

This is the default view when you select a class from the **Coach** dashboard. It displays the list of channels and items (exercises and resources - videos, reading material, etc.) accessed during the last 7 days by learners of the selected class.

Learn

Coach

Manage

Profile

Sign Out

Class Math 2A

Math 2A

Class coaches: Khang Mach, Betty White

Recent

Topics

Exams

Learners

Groups

Recent class activity

Showing Recent Activity in past 7 days

Channels / Nalanda Maths

Name	Progress
★ Adding Fractions	<div></div> 8/12 Mastered
📺 Introduction to Linear Algebra	<div></div> 3/21 Watched
🎵 Matrices	<div></div> 19/21 Listened
★ Do the Math	<div></div> 9/21 Mastered
★ Subtracting Fractions	<div></div> 14/21 Mastered
📺 Why Should We Learn Math?	<div></div> 19/21 Watched

If the class learners have access to more than one channel, you will first see the list of channels which you can navigate by topics and subtopics until you arrive to a specific item. In this view you can see the progress of each class learner for that specific item.

Learn

Coach

Manage

Profile

Sign Out

Class Math 2A

Math 2A

Class coaches: Khang Mach, Betty White

Recent

Topics

Exams

Learners

Groups

Recent class activity

Showing Recent Activity in past 7 days

Channels / Nalanda Maths / Adding Fractions

★ Adding Fractions

★ Mastery Requirement: 4 of 5 questions in a row correct

Name ↓	8/12 Mastered	Group	Last Active
Aaron Andrews	★ Mastered	Discoverers	Today
Evelyn	○ Not Started	Discoverers	-
Francis	⌘ In Progress	Explorers	1 week ago
Gustavo	★ Mastered	Discoverers	1 week ago

1.4.2 Topic Activity View

Use this view to access the full report of activity progress for the selected class. You can navigate channels by topics and subtopics until you see the progress of each class learner for one specific item.

KOLIBRI

Class

Math 2A

Learn

Coach

Manage

Profile

Sign Out

Math 2A

Class coaches: Khang Mach, Betty White

Recent

Topics

Exams

Learners

Groups

Math Fundamentals

33 Exercises • 24 Resources

Channels / Nalanda Maths / Math Fundamentals

Name ↓	Exercise Progress	Resource Progress	Last Active
Addition & Subtraction	<div><div></div></div> 45%	<div><div></div></div> 60%	Today
Algebra 1	<div><div></div></div> 42%	<div><div></div></div> 47%	1 month ago
Algebra Level 2	<div><div></div></div> 84%	<div><div></div></div> 95%	2 weeks ago
Geometry	<div><div></div></div> 0%	<div><div></div></div> 0%	-
Adding Fractions	<div><div></div></div> 8/12 Mastered		-

1.4.3 Manage Exams

You can view, create and delete exams, as well as assign them to learners, using the **Exams** tab in your **Coach** dashboard. Default view displays the list of all exams in a selected class, with a series of options to set the visibility, (de)activate when required, and view report of students who took them.

Math 2A

Class coaches: Khang Mach, Betty White



Recent



Topics



Exams



Learners











Groups

Exams

Show: ☒ All ☐ Active ☐ Inactive

+ New Exam

	Title	Visible to	Action
	Recent Exam	Entire Class Change	Activate 
	Recent Exam	Entire Class Change	Deactivate 
	Recent Exam	Entire Class Change	Deactivate 
	Exam	Entire Class Change	Activate 

Note: To manage **Exams** in Kolibri classes and groups you must be logged-in as **Coach** or **Admin**.

Create New Exam

To create a new exam, follow these steps.

1. Click + **New exam** button.

2. Select the content channel from which you wish to select questions for the exam.
3. Click **Create exam** to confirm, or **Cancel** to exit the confirmation window.
4. Fill in the field for exam title.
5. Fill in the field for number of questions you want exam to contain.
6. Navigate through the topic tree and select checkboxes of those exercises you want to include in the exam.

× **Create Exam**

Exam title
 Title your assignment ×


Number of questions
 Enter a number ×


Add content to your exam
🔍 Search for content! ×


Nalanda / Mathematics / Pre-Algebra / Pre-Algebra Basics 1


Title ↓

☐ **Select all**

☐  Pre Algebra Fundamentals

☐  Pre Algebra Basics

☐  Exercise 1

☐  Exercise 2

As you keep adding the exercises you will see confirmation messages at the bottom.

7. Click **Preview** button to view the result in overlay window.

Preview exam

10 questions

RANDOMIZE QUESTIONS

Completing the Whole - Add Together Practice

QUESTION 1

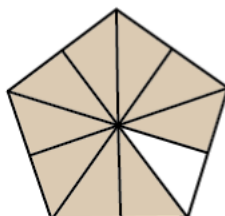
QUESTION 2

QUESTION 3

QUESTION 4

QUESTION 5

Which of the shapes completes the whole?



Fraction of a Whole Practice



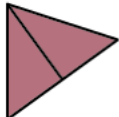
QUESTION 6

QUESTION 7

QUESTION 8

QUESTION 9

QUESTION 10

<input type="radio"/>	
<input type="radio"/>	
<input type="radio"/>	

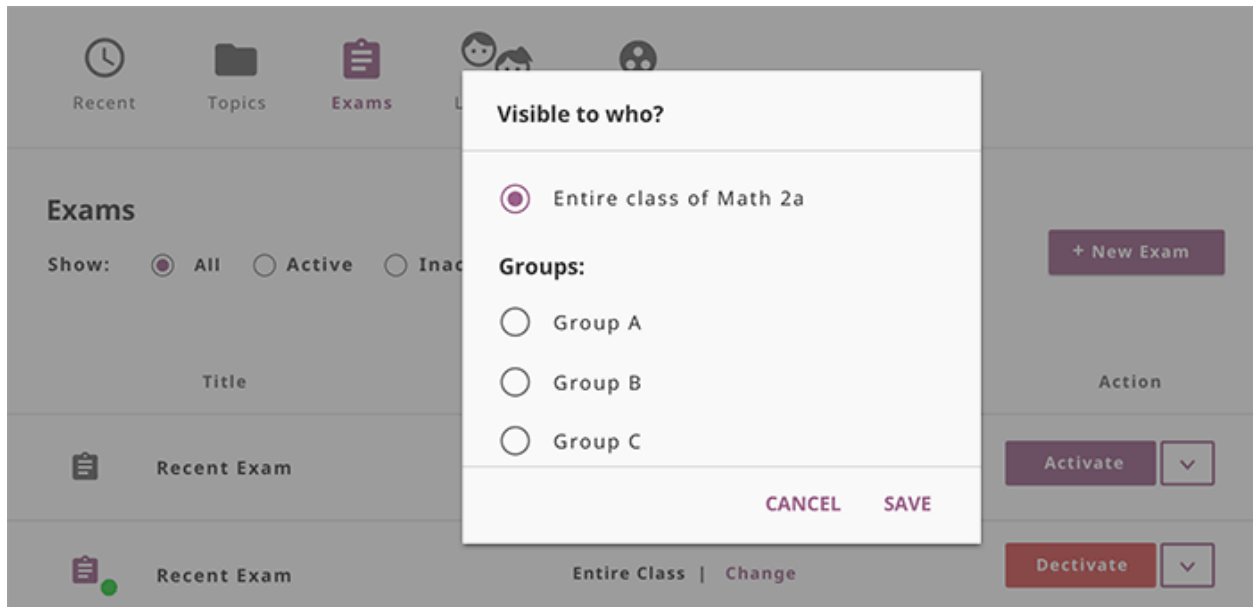
- Click **Randomize questions** button to present them in the different order from those in the topic origin.
- Click the **Close (X)** button in the upper right corner to return to the exam home page.

8. Click **Finish** button to save the result.

Change Exam Visibility

Newly created exam will be visible to entire class. To change exam visibility, meaning to assign it only to one group of learners instead of the whole class, follow these steps.

1. Click **Change** button under the **Visible to** column in the list of exams.
2. Select the group(s) of learners to whom you wish to assign the exam.
3. Click **Update** to confirm, or **Cancel** to exit the confirmation window.



Activate/Deactivate Exam









Once you set the visibility of exam to the chosen group(s) of learners, you need to **Activate** it in order for it to appear in the **Learn** view of the learners to whom you assigned it.

- Click **Activate** button under the **Action** column in the list of exams.
- When the exam period concludes, click the **Deactivate** button.







View Exam Report

To view the report on learners who have taken the exam, follow these steps.

1. Click down arrow near the **Activate** button for the desired exam from the list.
2. Select **View report** in the drop-down menu.

	Recent Exam	Entire Class Change	Deactivate 
	Recent Exam	Entire Class Change	<div> Preview Exam View Report Rename Delete </div>
	Exam	Entire Class Change	
	Exam 2	2 Groups Change	Activate 
	Exam 3	2 Groups Change	<div> Preview Exam View Report Rename Delete </div>
	Exam 4	4 Groups Change	


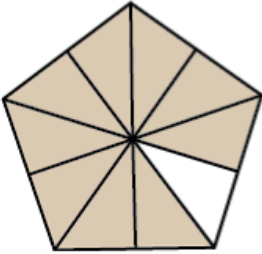










3. Click the name of the learner to view the detailed report with preview of results for each question.

Unit Exam #4 Report			Exit Report	
	Exam taken by: 40 Learners			
	Average Score: 80%			
	↓ Name	Status	Score	Group
	Aaron A.	Completed	92%	Group A
	Learner Name	Completed	91%	Group A
	Learner Name	Completed	88%	Group A
	Learner Name	Incomplete	-	Group A
	Learner Name	Completed	82%	Group A

4. Click each of the questions to preview it and understand better which question learners answered correctly in the exam and those they struggled with.

[← Back to Summative Exam Report](#)**Aaron Andrews - Exam Performance**Overall Score: **72%**Questions Correct: **39 of 50 correct** **Completed**

on 18 Nov 2016

from Exercise 1	Preview exam
 Question 1	<p>Which of the shapes completes the whole?</p> 
 Question 3	
 Question 4	
 Question 6	
from Exercise 2	
 Question 1	<input type="radio"/> 
 Question 4	
 Question 7	<input type="radio"/> 
 Question 10	
 Question 11	

Delete Exam

To delete exam, follow these steps.

1. Click down arrow near the **Activate** button for the desired exam from the list.
2. Select **Delete** in the drop-down menu.
3. Click **Delete** button in the confirmation window to proceed, or **Cancel** to exit without deleting the exam.

Warning: All data from the exam you are deleting will be lost.


Rename Exam






To rename exam, follow these steps.


1. Click down arrow near the **Activate** button for the desired exam from the list.
2. Select **Rename** in the drop-down menu.
3. Change the exam title in the confirmation window.
4. Click **Rename** button to proceed, or **Cancel** to exit without renaming the exam.

1.4.4 Manage Groups

You can create and delete groups, as well as assign learners to them from the **Groups** tab in your **Coach** dashboard. Default view displays the list of all groups for the selected class, with the list of assigned learners for each group.

 KOLIBRI







 Learn
 **Coach**
 Manage
 Profile

 Sign Out

Class
Math 2A ▾

Math 2A

Class coaches: Khang Mach, Betty White

 Recent Topics Exams Learners **Groups**

Class Groups

+ NEW GROUP

Discoverers 3 Learners 0 Selected

MOVE LEARNERS ▾

<input type="checkbox"/>	Name	Username
<input type="checkbox"/>	James Howard	jameshoward
<input type="checkbox"/>	Bobby Schultz	bobbyschultz
<input type="checkbox"/>	Steve Harvey	steveharvey

Rename Group

Delete Group

Explorers 2 Learners 0 Selected

MOVE LEARNERS ▾

<input type="checkbox"/>	Name	Username
<input type="checkbox"/>	Madison Carter	madisoncarter
<input type="checkbox"/>	Betty White	bwhite

Adventurers 0 Learners 0 Selected

MOVE LEARNERS ▾

<input type="checkbox"/>	Name	Username
--------------------------	------	----------

No Learners in this Group

Note: To manage groups of learners in Kolibri class you must be logged-in as **Coach** or **Admin**.

Create a New Group

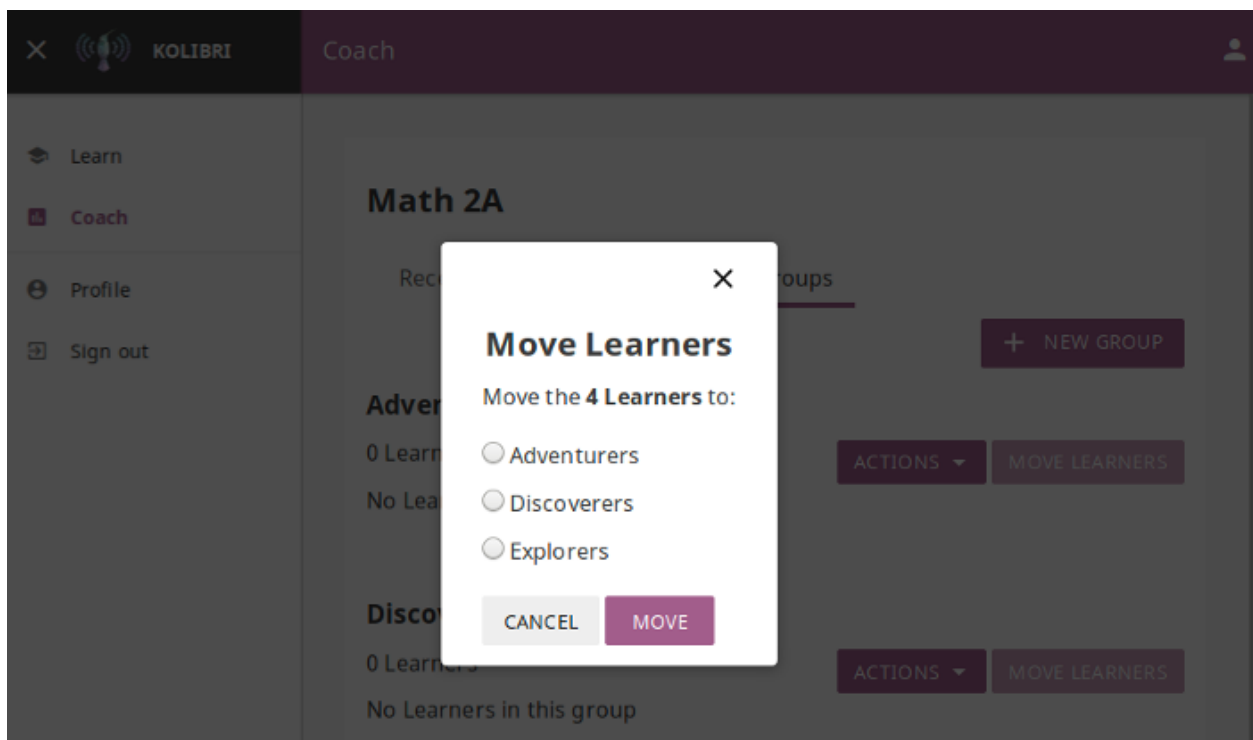
To create a new learner group, follow these steps.

1. Click **+ New group** button.
2. Give group a desired name.
3. Click **Save** to confirm, or **Cancel** to exit without creating a group.

Assign Learners to Group

Below existing groups there is a list with all learners currently **NOT** assigned to any groups.

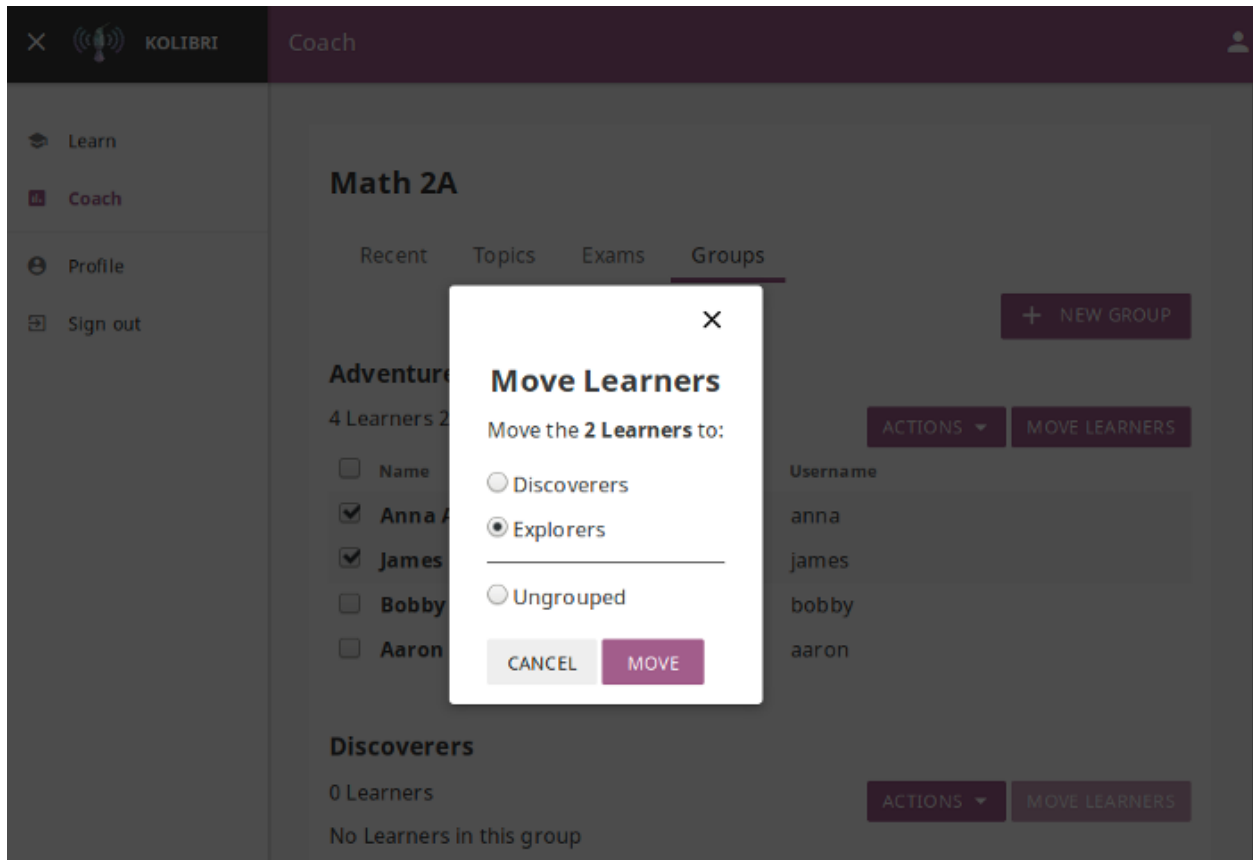
1. Use checkboxes to select all the learners in the list, or specific ones you want to assign to the group.
2. Click **Move learners** button on the right side of the list.
3. Select the group to which you want to assign the selected learners in the confirmation window.
4. Click **Move** to proceed, or **Cancel** to exit without assigning.



Move learners between groups

1. Use checkboxes to select all the user in one group, or specific users you want to assign to another group.
2. Click **Move learners** button on the right side of the origin group.

3. Select the group to which you want to move the selected learners, or the **Ungrouped** option if you want to remove them from the origin group without assigning to a new one.
4. Click **Move** to proceed, or **Cancel** to exit without moving.



Rename Group

To rename group, follow these steps.

1. Click the down arrow icon on the right edge of the desired group from the list.
2. Select the **Rename group** from the drop-down menu.
3. Input the new name for the group in the confirmation window.
4. Click **Save changes** button to proceed, or **Cancel** to exit without renaming the group.

Delete Group

To delete a group, follow these steps.

1. Click the down arrow icon on the right edge of the desired group from the list.
2. Select the **Delete group** from the drop-down menu.
3. Click **Delete group** button in the confirmation window to proceed, or **Cancel** to exit without deleting the group.

Note: Learners currently assigned to group will become ungrouped.

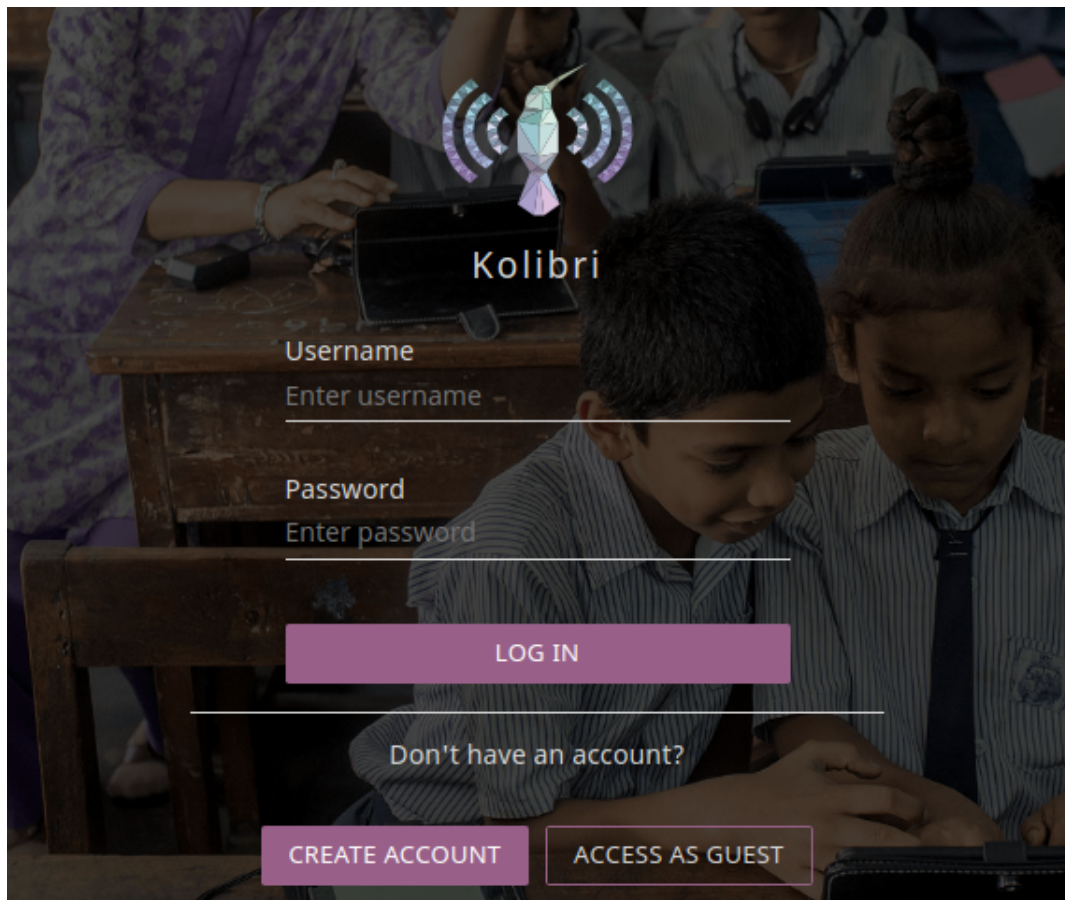
1.5 Learner

1.5.1 Accessing Kolibri

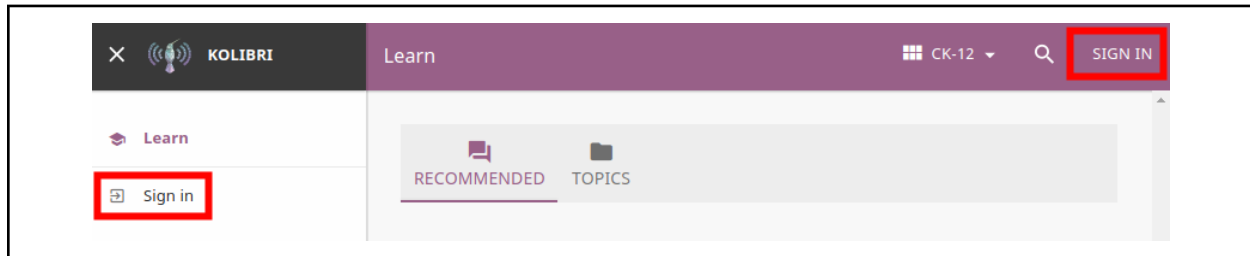
Note: If you use Kolibri in your school, education center or facility, your coach or administrator will provide the instructions how to open the sign-in page, and username and password if necessary.

To sign in to **Kolibri** and start learning follow these steps:

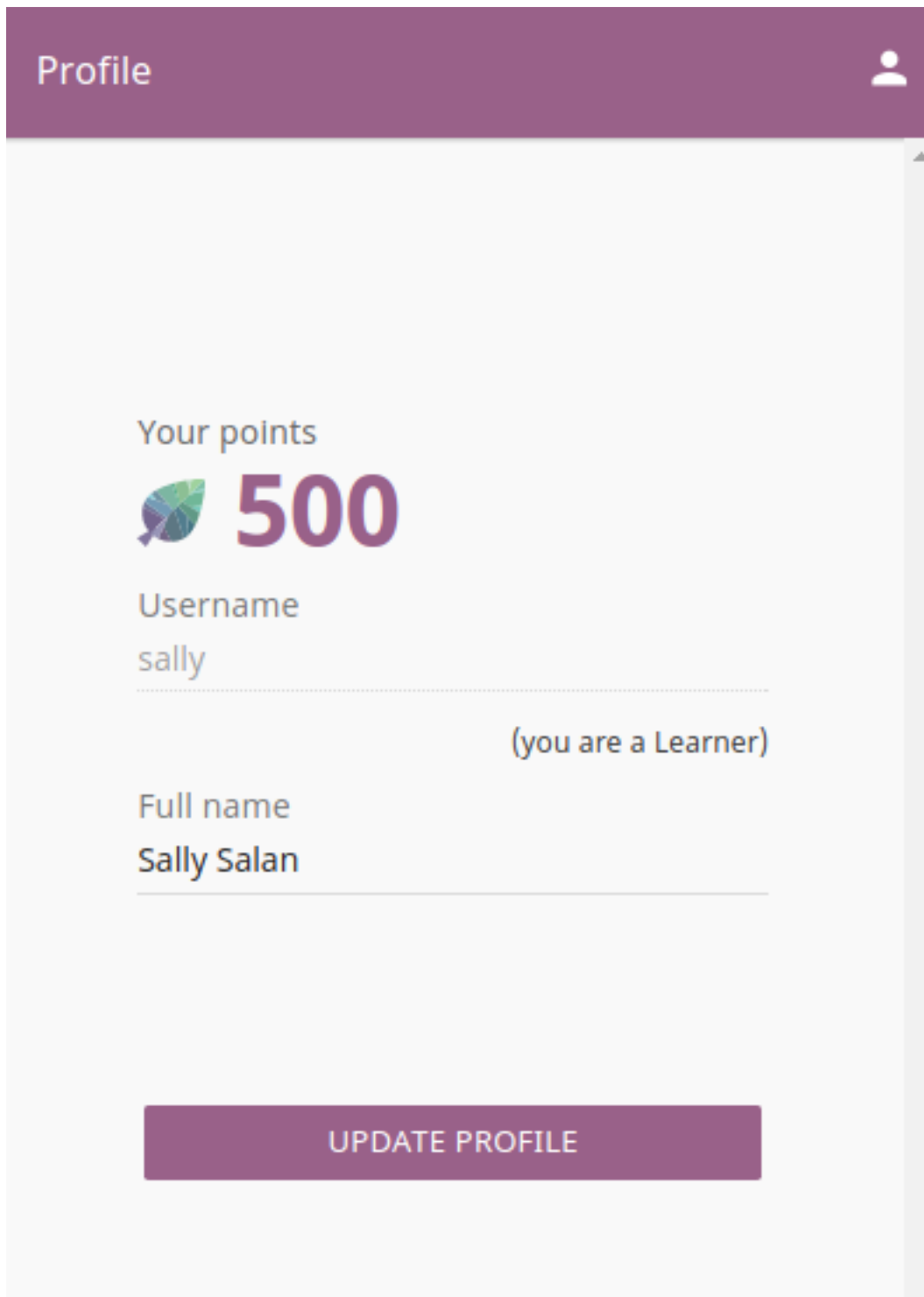
1. Type your username and password.
2. Click the **LOG IN** button.



Warning: If you start browsing Kolibri as a guest, you need to click the **Sign in** icon either in the upper right corner, or in the main menu (left or bottom) to open the sign-in page.



Once you have logged in into Kolibri, you can see and edit your user data from the **Profile** option in the main menu (below **Learn**).



To logout from Kolibri you can either:

- Click the user icon in the upper right corner and select **Sign out** option.

OR

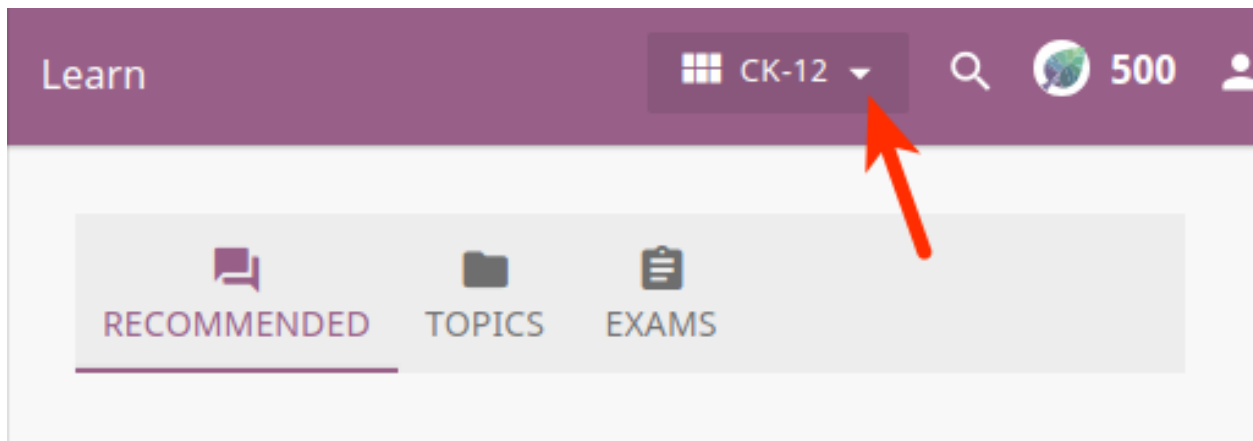
- Select **Sign out** option in the main menu.

1.5.2 Using Kolibri

Channels

In **Kolibri** you can find content from different sources grouped in **Channels**. Depending on how your school and teachers or coaches decided to organize the content, you may have one or more **Channels** available. Follow the indications by your teachers or coaches on how to use the content from each available **Channel**.

To switch between channels available to you, use the **Channel selector**.



Learn

Each time you login into **Kolibri**, the first thing you will see is the **Learn** page. Here you will find learning topics and materials related to what you were doing the last time you used Kolibri, or those recommended by your teachers and coaches (not visible if you are browsing as a guest).

The content you see in the **Learn** page will be different for each channel. Change the channel to explore the rest of content you can use in Kolibri.

Learn | KHAN ACADEMY TESTING | 500

RECOMMENDED | TOPICS | EXAMS

Recommended

All content [Previous](#) [Next](#)

- Counting with small numbers
- Count with small numbers
- Counting in order

Most popular

- Missing numbers between 0 and 120
- Counting with small numbers
- Simple adding

[SHOW MORE](#)

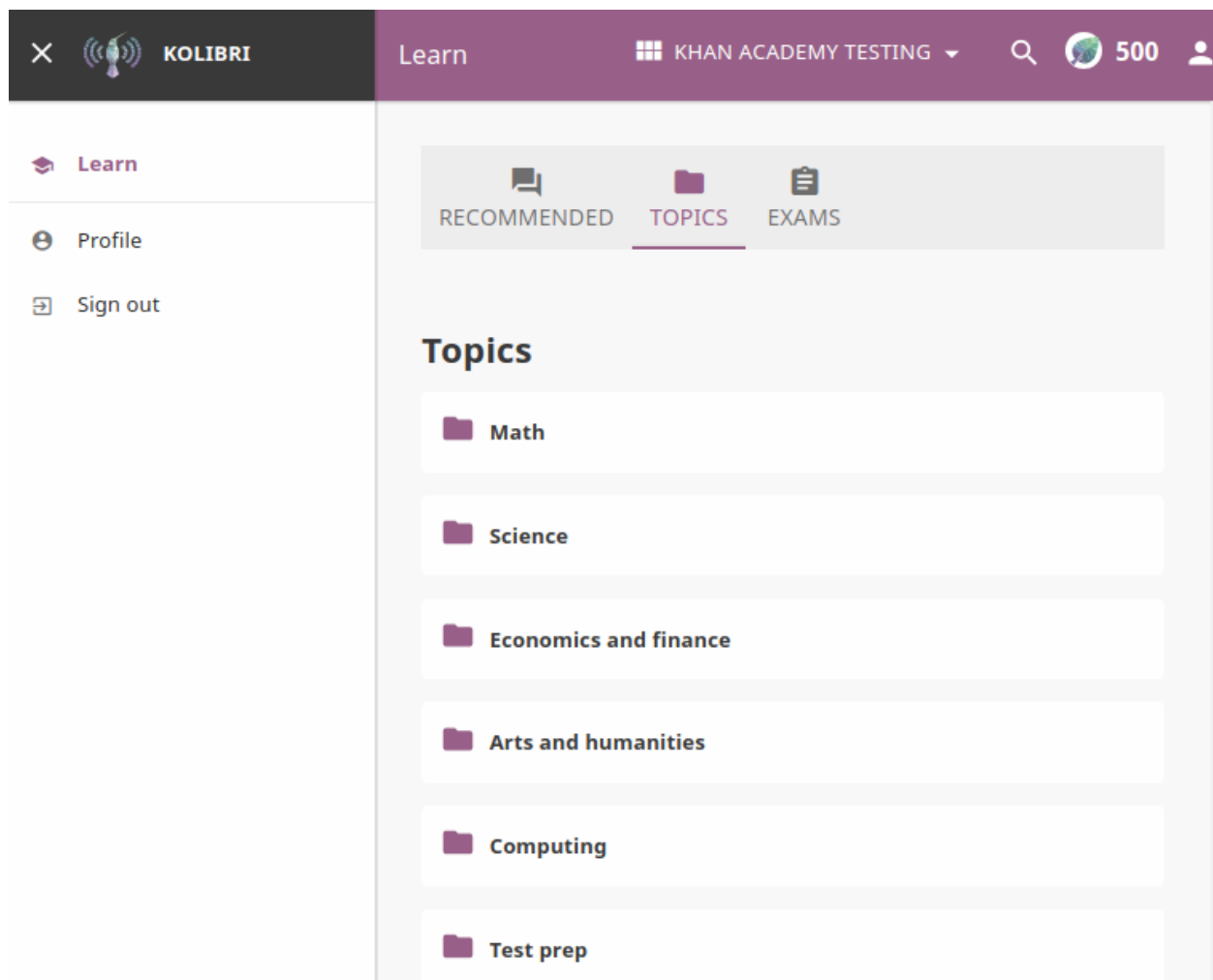
Recommended

In the **Recommended** tab you can see two sections:

- **All content** section where you can browse through all learning materials in the currently selected channel by using the **Previous** and **Next** links.
- **Most popular** section which displays the most frequently used materials in your school or facility.


Topics


Topics tab offers you the option to navigate through the complete set of learning topics and materials available in Kolibri, for the currently selected channel. Use it as you wish, or according to indications from your teachers and coaches.




Navigate Kolibri topics with breadcrumb links

When you are browsing a topic in Kolibri, the **Breadcrumb** links indicate previously visited, more general topics. Current topic is at the last position, and you can click any of the previous links in the breadcrumb to go back to a specific broader topic.

 RECOMMENDED

 TOPICS

 EXAMS

Topics > Math > Early Math > Counting > Counting small numbers > Counting small numbers


 Counting with small numbers
 Count with small numbers
 Counting in order

When you are viewing a video or doing an exercise, the **Breadcrumb** links are not visible anymore. To go back to the previous topic you visited, click the left arrow button above the video or exercise title.

Exams

If your teacher/coach scheduled an exam for you or your class, it will be available through the **Exams** tab.

Search

If you are looking for a specific subject, topic, or term, use the **Search** feature:

1. Click the magnifying glass icon in the upper right corner.
2. Type the word or combination of words you are looking for in the search field.
3. Press **Enter** to display search results below the field.

Search
count

Showing results for "count"
9 results

Topics


Counting


Counting small numbers


Counting small numbers

Counting objects

Content


Counting objects 1


Counting with small numbers


Count with small numbers


Exercises


Kolibri **Exercises** can require you to do different things: fill in a missing number, write a formula, choose one of the available options, etc. Each correct answer gets you a checkmark, and majority of exercises require 5 correct answers in a row to be completed. Some exercises can offer one or more hints, to help you solve the problem.

Independent of the required action (writing an answer yourself or choosing one of the options), these are the steps to follow.

1. **Read the question carefully.**
2. Write the answer or choose one of the provided options.
3. When you are ready to submit, click the **Check answer** button.
 - If the answer is correct and a checkmark appears, click the **Next question** button to proceed.
 - If the answer is incorrect, click the **Get a hint** button, read the suggestions, and try to answer again.
4. Once you have achieved the required number of correct answers in a row, click the **Next item** button, to continue learning with the rest of the material in that topic.

5. If you are unable to solve some questions, try reviewing the videos in the **Recommended** section below the exercise, or seek help from your peers or teacher/coach.


 **RECOMMENDED**

 **TOPICS**

←

★ Count with small numbers ⌚

Put 6 mice in the box.



Try to get 5 check marks to show up

✓

CHECK ANSWER

GET A HINT

Practice counting up to 10 objects.

Video Player options

To play videos in Kolibri you have several available control buttons at the bottom of the video player screen. Move the cursor or tap on the video player screen to make appear the control buttons while playing the video.

The screenshot shows the Kolibri video player interface. At the top, there are three tabs: 'RECOMMENDED', 'TOPICS' (which is selected), and 'EXAMS'. Below the tabs is a back arrow. The video title is 'Adding by Counting (Sums to 10)' and it has 500 views. The video content shows a math problem: 'How many apples are there altogether?' with two groups of apples (3 and 5) and a number line from 3 to 8. Below the number line is the equation $3 + 5 = \underline{\quad}$. The video player controls at the bottom include a play/pause button, a progress bar showing 0:00 / 1:09, a volume icon, a 1x speed selector, and a fullscreen button. Below the video player, there is a 'DOWNLOAD CONTENT' button and a dropdown menu showing 'Low Resolution (1.47 ...)'. To the right of these buttons, the author 'Elizabeth Vu', license '1', and copyright holder '-' are listed.

(controls at the bottom of video player)

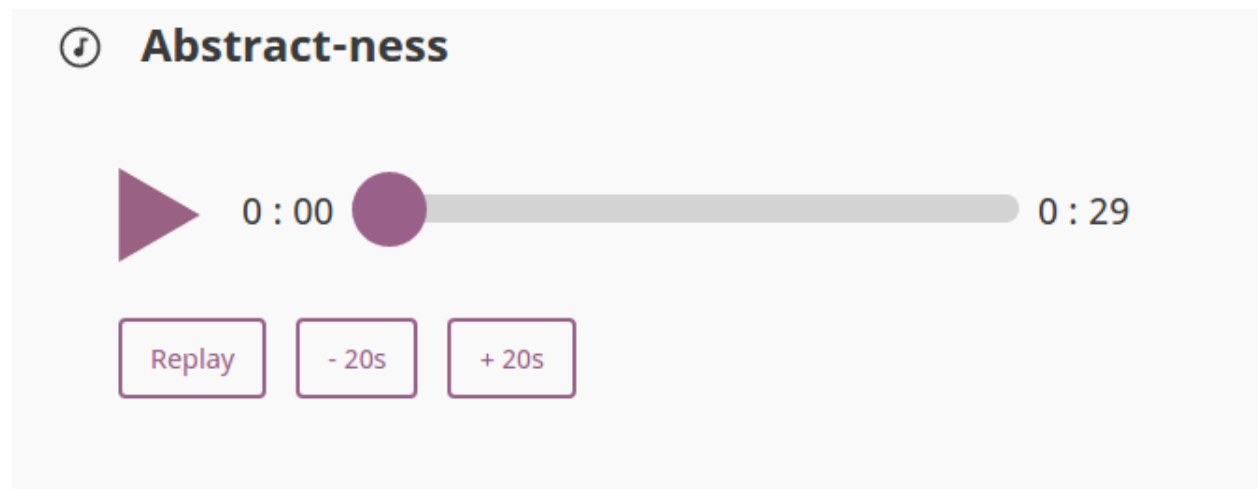
- Play/Pause buttons
- Rewind/Fast forward buttons by +/- 10 seconds
- Time tracker indicator with progress bar
- Video duration indicator
- Volume scrollbar
- Playback speed selector
- Fullscreen button

Use the **Download content** button below the video player to download the video and thumbnail files to your computer. Some videos will provide multiple resolution options.

Audio Player options

To play audio files in **Kolibri** you have available several control buttons:

- Play/Pause
- Time tracker indicator with progress bar
- Audio duration indicator
- Replay button
- Rewind/Fast forward buttons by +/- 20 seconds



Use the **Download content** button below the audio player to download the audio and thumbnail files to your computer.

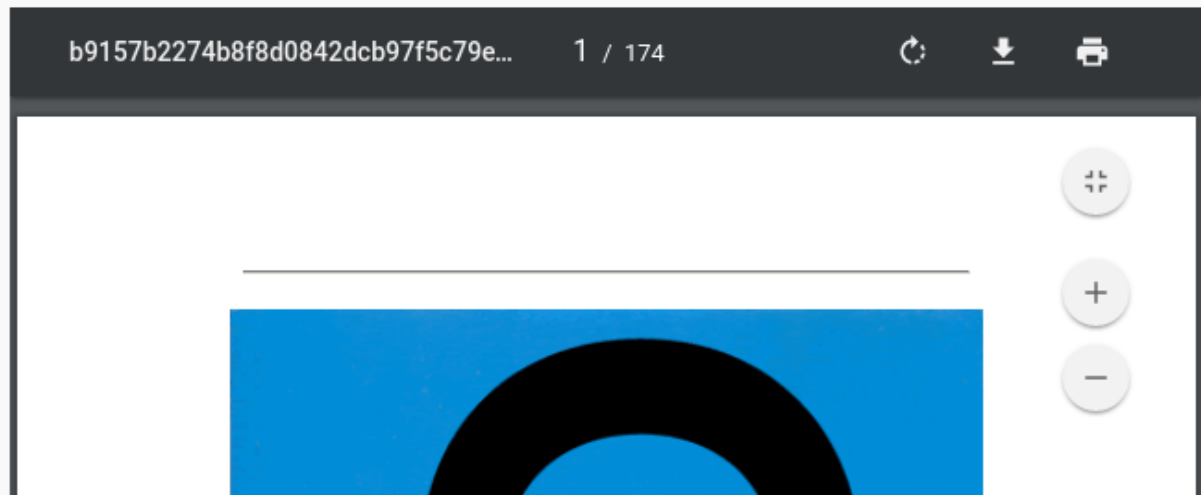
PDF Viewer options

Note: Options for viewing PDF files will depend on the browser and operating system you are using to view Kolibri.

- Use the **Toggle Fullscreen** button to open the PDF file in fullscreen view.
- Use the **Esc** button to close the fullscreen view and return.

Origins of algebra

Toggle Fullscreen



Where did the word "Algebra" and its underlying ideas come from?

Use the **Download content** button below the PDF viewer to download the PDF file to your computer.

2.1 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!

2.1.1 Setting up Kolibri for development

Most of the steps below require entering commands into your Terminal (Linux, Mac) or command prompt (`cmd.exe` on Windows) that you will learn how to use and become more comfortable with.

Tip: In case you run into any problems during these steps, searching online is usually the fastest way out: whatever error you are seeing, chances are good that somebody already had it in the past and posted a solution somewhere... ;)

Git & 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](#).

Install Environment Dependencies

1. Install [Python](#) if you are on Windows, on Linux and OSX Python is preinstalled (recommended versions 2.7+ or 3.4+).
2. Install [pip](#) package installer.

3. Install [Node](#) (recommended version 4+).
4. Install Yarn according to the [instructions](#) specific for your OS.

Note:

- On Ubuntu install Node.js via [nvm](#) to avoid build issues.
 - On a Mac, you may want to consider using the [Homebrew](#) package manager.
-

Ready for the fun part in the Terminal? Here we go!

Checking out the code

1. Make sure you [registered your SSH keys on GitHub](#).
2. **Clone** your Kolibri fork to your local computer. In the following commands replace `$USERNAME` with your own GitHub username:

```
# using SSH
git clone git@github.com:$USERNAME/kolibri.git
# using HTTPS
git clone https://github.com/$USERNAME/kolibri.git
```

3. Enable syncing your local repository with **upstream**, which refers to the Kolibri source from where you cloned your fork. That way you can keep it updated with the changes from the rest of Kolibri team contributors:

```
cd kolibri # Change into the newly cloned directory
git remote add upstream git@github.com:learningequality/kolibri.git # Add
↪the upstream
git fetch upstream # Check if there are changes upstream
git checkout develop
```

Warning: `develop` is the active development branch - do not target the `master` branch.

Virtual environment

It is best practice to use [Python virtual environment](#) to isolate the dependencies of your Python projects from each other. This also allows you to avoid using `sudo` with `pip`, which is not recommended.

You can learn more about using [virtualenv](#), or follow these basic instructions:

Initial setup, performed once:

```
$ sudo pip install virtualenv # install virtualenv globally
$ mkdir ~/.venvs             # create a common directory for multiple virtual
↪environments
$ virtualenv ~/.venvs/kolibri # create a new virtualenv for Kolibri dependencies
```

Note: We create the `virtualenv` *outside* of the Kolibri project folder. You can choose another location than `~/.venvs/kolibri` if desired.

To activate the `virtualenv` in a standard Bash shell:

```
$ source ~/.venvs/kolibri/bin/activate # activate the venv
```

Now, any commands run with `pip` will target your virtualenv rather than the global Python installation.

To deactivate the virtualenv, run the command below. Note, you'll want to leave it activated for the remainder of project setup!

```
$ deactivate
```

Tip:

- Users of Windows and other shells such as Fish should read the [guide](#) for instructions on activating.
 - If you set the `PIP_REQUIRE_VIRTUALENV` environment variable to `true`, `pip` will only install packages when a virtualenv is active. This can help prevent mistakes.
 - Bash users might also consider using [virtualenvwrapper](#), which simplifies the process somewhat.
-

Install Project Dependencies

Note: Make sure your virtualenv is active!

To install Kolibri project-specific dependencies make sure you're in the `kolibri` directory and run:

```
# Python requirements
(kolibri)$ pip install -r requirements.txt
(kolibri)$ pip install -r requirements/dev.txt

# Kolibri Python package in 'editable' mode, so your installation points to
↳ your git checkout:
(kolibri)$ pip install -e .

# Javascript dependencies
(kolibri)$ yarn install
```

Tip:

- We've adopted this concatenated version with added cleanup: `make clean && pip install -r requirements.txt --upgrade && pip install -e . && yarn install.`
 - In case you get webpack compilation error with Node modules build failures, add the flag `--force` at the end, to ensure binaries get installed.
-

2.1.2 Running Kolibri server

Development server

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

```
(kolibri)$ kolibri --debug manage devserver --webpack
```

Wait for the build process to complete. This takes a while the first time, will complete faster as you make edits and the assets are automatically re-built.

Now you should be able to access the server at `http://127.0.0.1:8000/`.

Tip: If you need to make the development server available through the LAN, you must leave out the `--webpack` flag, and use the following command:

```
(kolibri)$ yarn run build
(kolibri)$ kolibri --debug manage devserver -- 0.0.0.0:8000
```

Now you can simply use your server's IP from another device in the local network through the port 8000, for example `http://192.168.1.38:8000/`.

More advanced examples of the `devserver` command:

```
# runs the dev server and rebuild client assets when files change
kolibri --debug manage devserver --webpack

# runs the dev server and re-run client-side tests when files changes
kolibri --debug manage devserver --karma

# runs all of the above
kolibri --debug manage devserver --webpack --karma
```

Running the Production Server

In production, content is served through CherryPy. Static assets must be pre-built:

```
yarn run build
kolibri start
```

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

2.1.3 Contributing code to Kolibri

- Once you've toyed around with things, read through the rest of the *Developer Guide*, especially topics in *Architecture* and *Themes* to understand more about the Kolibri structure.
- When you're up to speed with that, you're probably itching to make some contributions! Head over to the [issues page on GitHub](#) and take a look at the current project priorities. Try filtering by milestone. If you find a bug in your testing, please [submit your own issue](#)
- Once you've identified an issue and you're ready to start hacking on a solution, get ready to *Submit Pull Requests!*

Branching and Release Process

The `develop` branch is reserved for active development. When we get close to releasing a new stable version/release of Kolibri, we generally fork the `develop` branch into a new branch (like `release-0.1.x`). If you're working on an issue tagged for example with the `release-0.1.x` milestone, then you should target changes to that branch. Changes to those branches will later be pulled into `develop` again. If you're not sure which branch to target, ask the dev team!

Note: At a high level, we follow the ‘Gitflow’ model. Some helpful references: * <http://nvie.com/posts/a-successful-git-branching-model/> * <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow/>

Submit Pull Requests

The most common situation is working off of `develop` branch so we’ll take it as an example:

```
$ git checkout upstream/develop
$ git checkout -b name-of-your-bugfix-or-feature
```

After making changes to the code, commit and push them to a branch on your fork:

```
$ git add -A # Add all changed and new files to the commit
$ git commit -m "Write here the commit message"
$ git push origin name-of-your-bugfix-or-feature
```

Go to [Kolibri GitHub page](#), and if you are logged-in you will see the link to compare your branch and and create the new pull request. **Please fill in all the applicable sections in the PR template and DELETE unnecessary 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.

2.1.4 Additional Recommended Setup

If you’re planning on contributing code to the project, there are a few additional steps you should consider taking.

Editor Config

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.

Front-end Dev Tools

If you’re working with front-end Vue.js and use Google Chrome Dev Tools, you may find the [Vue.js devtools](#) helpful

DB Setup

You can initialize the server using:

```
kolibri manage migrate
```

Pre-Commit Install

We use [pre-commit](#) to help ensure consistent, clean code. The pip package should already be installed from a prior setup step, but you need to install the git hooks using this command.

```
pre-commit install
```

2.1.5 Development

Linting

To improve build times, and facilitate rapid development, Javascript linting is turned off by default when you run the dev server. However, all frontend assets that are bundled will be linted by our Travis CI builds. It is a good idea, therefore, to test your linting before submitting code for PR. To run the devserver in this mode you can run the following command.

```
kolibri --debug manage devserver --webpack --lint
```

Code Testing

Kolibri comes with a Python test suite based on `py.test`. To run tests in your current environment:

```
pytest # alternatively, "make test" does the same
```

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, lint and documentation tests, use simply `tox`. This simulates what our CI also does.

To run Python linting tests (pep8 and static code analysis), use `tox -e lint` or `make lint`.

Note that `tox`, by default, 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.

We strive for 100% code coverage in Kolibri. When you open a Pull Request, code coverage (and your impact on coverage) will be reported. To test code coverage locally, so that you can work to improve it, you can run the following:

```
tox -e py3.4  
coverage html
```

Then, open the generated `./htmlcov/index.html` file in your browser.

Kolibri comes with a Javascript test suite based on `mocha`. To run all tests:

```
yarn test
```

This includes tests of the bundling functions that are used in creating front end assets. To do continuous unit testing for code, and jshint running:

```
yarn run test-karma:watch
```

Alternatively, this can be run as a subprocess in the development server with the following flag:

```
kolibri --debug manage devserver --karma
```

You can also run tests through Django's test management command, accessed through the `kolibri` command:

```
kolibri manage test
```


To run specific tests only, you can add `--`, followed by a label (consisting of the import path to the test(s) you want to run, possibly ending in some subset of a filename, classname, and method name). 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`:

```
kolibri manage test -- kolibri.auth.test.test_permissions.  
↳ MembershipPermissionsTestCase.test_admin_can_delete_membership
```

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
```

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 Manual Testing & QA

2.2.1 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](#).

Here are a few tools that we use in testing for accessibility:

- WAVE Evaluation Tool - [Firefox Add-on](#) and [Chrome extension](#).
- [totally](#) accessibility visualization toolkit - bookmarklet for Firefox and Chrome.
- [Accessibility Developer Tools](#) - Chrome extension.
- aXe Accessibility Engine - [Firefox Add-on](#) and [Chrome extension](#).

There is a much longer list on our [Kolibri Accessibility Tools Wiki page](#) if you want to go deeper, but these four should be enough to help you avoid the most important accessibility pitfalls.

2.2.2 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 9+ on Windows XP and up
- 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. [This tutorial](#) was written for KA Lite, but much of it still applies to Kolibri.

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.

2.2.3 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.

2.2.4 Slow Network Connection Speeds

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).

2.2.5 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.

2.2.6 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 the each currently existing channel on the system. Use the *-help* flag for options.

2.3 Architecture

2.3.1 Tests

Running the test suite

A prerequisite for testing is to have the test environment installed. As a developer, simply fetch the “dev” requirements:

```
$ pip install -r requirements/dev.txt
```

Running all the tests in your local environment:

```
$ py.test
```

Running tests with a specific Python version:

```
$ tox -e py2.7 # or... $ tox -e py3.4 # or... $ tox -e py3.5
```

To run a subset of tests:

```
$ py.test test/test_kolibri.py
$ # ...or with a tox environment
$ tox -e 3.5 test/test_kolibri.py
```

JS unit tests

Note: TODO! This will be written by one of the JS devs :)

Testing philosophy

Warning: This section an unfinished draft. We should carefully import stuff from our Dev Bible.

We want to achieve a >90% test coverage! To do that, it's best to do TDD, meaning to write tests that express what you want to achieve or fix and then implement that feature or fix the bug.

At all costs? No, definitely not. We don't want too many slow integration tests that depend on virtual browsers (Selenium) and the like.

Unreliable tests? Are not welcome at all.

Writing tests

Kolibri has a test discoverer (`py.test`) which automatically detects Python modules with test cases.

Inside the module that your tests target, place a package `test` and files called `test_my_module.py`:

```
mymodule/
  __init__.py
  test/
    __init__.py
    test_feature.py
```

2.3.2 Tech Stack

Kolibri is a web application built primarily using [Python](#) on the server-side and [JavaScript](#) on the client-side.

We use many run-time, development, and build-related technologies and tools, as outlined below.

Server

The server is a [Django 1.9](#) application, and contains only pure-Python (2.7+) libraries dependencies at run-time.

The server is responsible for:

- Interfacing with the database ([PostgreSQL](#)) containing user, content, and language pack data
- Authentication and permission middleware
- Routing and handling of API calls, using the [Django REST Framework](#)
- Basic 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

TODO - how does Morango fit into this picture? Logging?

Client

The front-end user interface is built using HTML, the [Stylus](#) CSS-preprocessing language, and the [ES2015 preset features of ES6](#) JavaScript.

The frontend targets IE9 and up, with an emphasis on tablet-size screens. We strive to use accessible, semantic HTML with support for screen readers, keyboard interaction, and right-to-left language support.

The client is responsible for:

- Compositing and rendering the UI using [Vue.js](#) components to build nested views
- Managing client-side state using [Vuex](#)
- Interacting with the server through HTTP requests

Additionally, [loglevel](#) is used for logging and [normalize.css](#) is included for browser style normalization.

Internationalization

We leverage the [ICU Message](#) syntax for formatting all user-facing text.

On the client-side, these strings are rendered using [Format.js](#) and integrated with Vue.js using [vue-intl](#).

TODO: server-side, message extraction, translation

Developer Docs

Documentation for Kolibri developers are formatted using [reStructuredText](#) and the output is generated using [Sphinx](#). Most of the content is in the `/docs` directory, but some content is also extracted from Python source code and from files in the root directory. We use [Read the Docs](#) to host a public version of our documentation.

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

Build Infrastructure

Client-side Resources

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
- Stylus 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
- Linting to enforce code styles

Server Setup

The standard Django `manage.py` commands are used under-the-hood for database migration and user set-up.

TODO: is this accurate?

Installers and Packages

TODO: introduce stack (sdist, PyPi, Debian, Windows, etc)

Continuous Integration

TODO: introduce stack (GitHub, CodeCov, Travis, commit hooks)

Tests and Linting

We use a number of mechanisms to help encourage code quality and consistency. These checks enforce a subset of our [Project Conventions](#).

- `pre-commit` is run locally on `git commit` and enforces some Python conventions
- We use [EditorConfig](#) to help developers set their editor preferences
- `flake8` is also used to enforce Python conventions
- `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, [Stylint](#) for Stylus, and [HTMLHint](#) for HTML and Vue.js components.
- Client-side code is tested using a stack of tools including [Karma](#), [Mocha](#), [PhantomJS](#), [Sinon](#), and [rewire](#). *TODO: Explain what each of these do*
- [codecov](#) reports on the test coverage for Python and Node.js code. *TODO - also client-side?*

Helper Scripts

TODO: introduce stack (kolibri command, setup.py, makefiles, yarn commands, sphinx auto-build, etc)

2.3.3 Plugins

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` command. See `./user/cli`.

Other stuff you can do with plugins

Plugins can implement Javascript code as a Kolibri module that can be used in the frontend as a plugin to the core Kolibri Javascript code. Each of these Javascript plugins are defined in the `kolibri_plugin.py` file by subclassing the `KolibriFrontEndPluginBase` class to define each frontend Kolibri module. This defines the base Javascript file that defines the Kolibri module. In addition, this Plugin object within the app will automatically add these Kolibri modules to an internal frontend asset registry for loading in the front end. For more information on developing frontend code for Kolibri please see [Front-end Architecture](#).

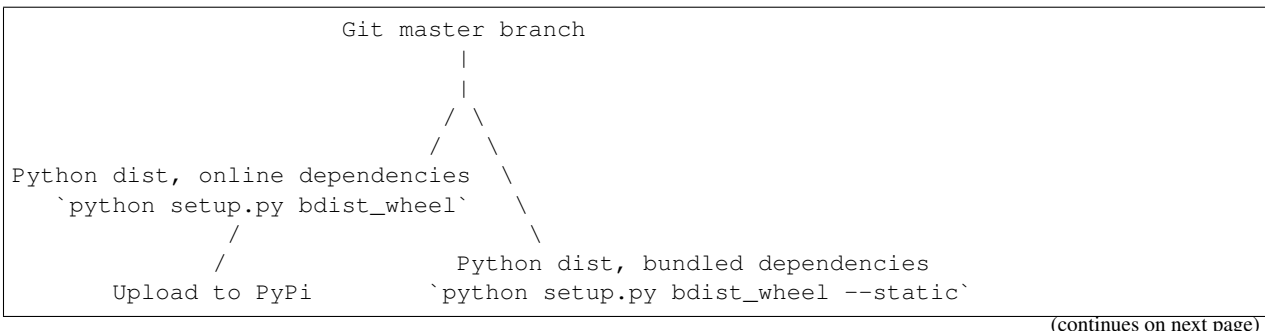
Plugins can be standalone Django apps in their own right, meaning they can define templates, models, new urls, and views just like any other app. However the API for all of this hasn't yet been determined... Coming soon!

Core plugin example

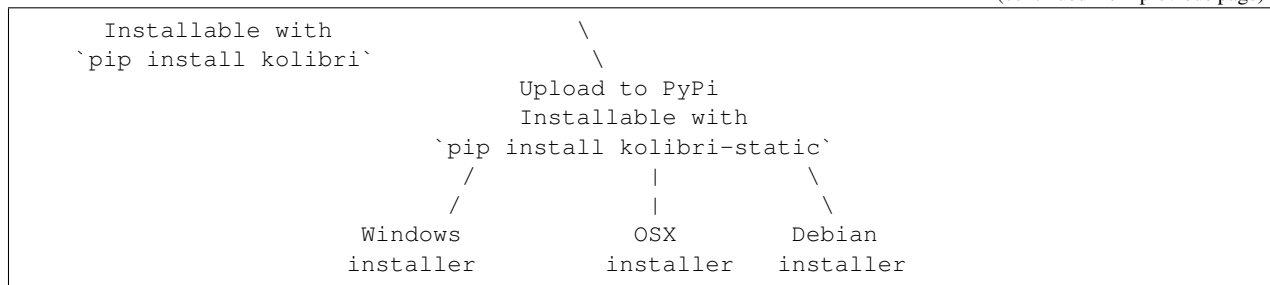
View the source to learn more!

2.3.4 Distribution and installers

The Kolibri Package build pipeline looks like this:



(continued from previous page)



Make targets

To build both the slim Kolibri and the one with bundled dependencies, simply run *make dist*. The *.whl* files will now be available in *dist/*whl* and you can install them with *pip install dist/filename.whl*.

Automated CI tests

If you add *[setup]* to your commit message, our CI will automatically test that builds work.

Otherwise, changes to certain files like *requirements/** and *setup.py* will automatically prompt test builds to fire.

2.3.5 Project Conventions

TODO

Documentation

reStructuredText, docstrings, requirements for PRs to master...

Git Workflow

stable master, develop, feature branches, tags, releases, hot fixes, internal vs external repos...

Python Code

PEP8, additional conventions and best practices...

Vue.js Components

Note that the top-level tags of **Vue.js components** are `<template>`, `<script>`, and `<style>`.

- Whitespace
 - an indent is 2 spaces
 - two blank lines between top-level tags
 - one blank line of padding within a top-level tag
 - one level of indent for the contents of all top-level tags

- Keep most child-components stateless. In practice, this means using `props` but not `data`.
- Avoid using Vue.js' camelCase-to-kebab-case mapping. Instead, use square brackets and strings to reference names.
- Use `scoped` styles where ever possible
- Name custom tags using kebab-case
- Components are placed in the `vue` directory. The root component file is called `vue/index.vue`, and is mounted on a tag called `<rootvue>`.
- Components are defined either as a file with a `.vue` extension (`my-component.vue`) or as a directory with an `index.vue` file (`my-component/index.vue`). Both forms can be used with `require('my-component')`.
- Put child components inside the directory of a parent component if they are *only* used by the parent. Otherwise, put shared child components in the `vue` director.
- Any user visisble interface text should be rendered translatable, see [i18n](#) for details.

JavaScript Code

- We use the [AirBnB Javascript Style guide](#) for client-side ES6 code in Vue components.
- `use strict` is automatically inserted.
- Use ES6 `import/export` statements, not CommonJS-style `require` and `module.exports` statements.
- For logging statements we use a thin wrapper around the `log-level` JS library, that prefixes the log statements with information about the logging level and current file. To access the logger, simply include the following code snippet:

```
import logger from 'kolibri.lib.logging';
const logging = logger.getLogger(__filename);
```

Stylus and CSS

- clear out unused styles
- avoid using classes as JS identifiers, and prefix with `js-` if necessary

HTML

attribute lists, semantic structure, accessibility...

2.3.6 Front-end Architecture

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 [Stylus](#)), and logic which is also scoped to that component (all code, including that in Vue components should be written using [Bubl  compatible ES2015 JavaScript](#)). Non-scoped styles can also be added, but these should be carefully namespaced.

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 [Project Conventions](#) for some important consistency tips on writing new components.

Layout of Frontend Code

Front-end 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/
        core-base.vue                 # global base template, used by apps
        core-modal.vue                # example of another shared component
        core-global.styl              # globally defined styles, included in head
        core-theme.styl               # style variable values
        font-NotoSans.css             # embedded font
      test/
        ...                           # tests for core assets
  plugins/
    learn                               # learn plugin
      assets/
        src/
          vue/
            index.vue                  # root view
            some-page.vue              # top-level client-side page
            another-page/              # 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/
              vue/user-page.vue        # nested-view
              vue/index.vue            # root view
              app.js                   # instantiate mgmt app on client-side
            test/
              app.js
```

In the example above, the *vue/another-page/index.vue* file in *learn* can use other assets in the same directory (such as *child.vue*), components in *vue* (such as *shared.vue*), and assets in *core* (such as variables in *core-theme.styl*). 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 front-end 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 [Front-end Asset Loading](#)), but it can take some time to understand how it all hangs together.

SVG Icons

SVGs can be inlined into Vue components using a special syntax:

```
<svg src="icon.svg"></svg>
```

Then, if there is a file called `icon.svg` in the same directory, that file will be inserted directly into the outputted HTML. This allows aspects of the icon (e.g. fill) to be styled using CSS.

Attributes (such as vue directives like `v-if` and SVG attributes like `viewbox`) can also be added to the `svg` tag.

Single-page Apps

The Kolibri front-end 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’, as shown in the example above. 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 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 `kolibriGlobal`. 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 [Front-end Asset Loading](#) 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.

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 accepts a json file defining the content types that it renders:

```
.. automodule:: kolibri.content.hooks
```

members

noindex

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 `defaultFile`, `files`, `supplementaryFiles`, and `thumbnailFiles` props, defining the files associated with the piece of content.

```
{
  props: [
    'defaultFile',
    'files',
  ]
};
```

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 two additional props will be passed: `itemId` and `answerState`. `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). 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.

```
{
  props: [
    'defaultFile',
    'files',
```

(continues on next page)

(continued from previous page)

```

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

```

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

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

- vue - the Vue.js object
- vuex - the Vuex object
- logging - our wrapper around the [loglevel logging module](#)
- core-base - a shared base Vue.js component (*core-base.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. If the module in question has the ‘requireName’ attribute set on the core specification, then it can be used in code with a standard CommonJS-style require statement - e.g.:

```

const vue = require('kolibri.lib.vue');
const coreBase = require('kolibri.coreVue.components.coreBase');

```

Adding additional globally-available objects is relatively straightforward due to the [plugin and webpack build system](#).

To expose something on the core app, add a key to the object in *apiSpec.js* which maps to an object with the following keys:

```

modulePath: {
  module: require('module-name'),
}

```

This module would now be available for import anywhere with the following statement:

```

const MODULE = require('kolibri.modulePath');

```

For better organisation of the Core API specification, modules can also be attached at arbitrarily nested paths:

```
modulePath: {
  nestedPath: {
    module: require('module-name'),
  }
}
```

This module would now be available for import anywhere with the following statement:

```
const MODULE = require('kolibri.modulePath.nestedPath');
```

For convenience (and to prevent accidental imports), 3rd party (NPM) modules installed in `node_modules` can be required by their usual name also:

```
const vue = require('vue');
```

Bootstrapped Data

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

For example, we currently embellish the `kolibriGlobal` 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.

Styling

For shared styles, two mechanisms are provided:

- The `core-theme.styl` file provides values for some globally-relevant Stylus variables. These variables can be used in any component's `<style>` block by adding the line `@require '~core-theme.styl'`.
- The `core-global.styl` file is always inserted into the `<head>` after `normalize.css` and provides some basic styling to global elements

Additional Functionality

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

```
kolibriGlobal.register_kolibri_module_async  // Register a Kolibri module for_
↪asynchronous loading.
kolibriGlobal.register_kolibri_module_sync   // Register a Kolibri module once it_
↪has loaded.
kolibriGlobal.stopListening                  // Unbind an event/callback pair from_
↪triggering.
kolibriGlobal.emit                           // Emit an event, with optional args.
```

Unit Testing

Unit testing is carried out using [Mocha](#). 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.

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.js* for plugin specific webpack configuration (loaders, plugins, etc.). These options will be merged with the base options using *webpack-merge*.

2.3.7 i18n

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.

Backend Translation

For any strings in Django, we are using the standard Django i18n machinery (*gettext* and associated functions) to provide translations. See the [Django i18n documentation](#) for more information.

Frontend Translation

For any strings in the frontend, we are using [Vue-Intl](#) an in house port of [React-intl](#).

Within Kolibri, messages are defined on the body of the Vue component:

```
- ``$trs``, an object of the form::

  {
    msgId: 'Message text',
  }

- ``$trNamespace``, a string that namespaces the messages.
```

User visible strings should be rendered directly in the template with `{{ $tr('msgId') }}`. These strings are collected during the build process, and bundled into exported JSON files. These files are then uploaded to Crowdin for translation.

An example Vue component would then look like this:

```
<template>

  <div>
    <p>{{ $tr('exampleMessage') }}</p>
  </div>

</template>

<script>

  module.exports = {
    $trNamespace: 'example',
    $trs: {
      exampleMessage: 'This message is just an example',
    },
  };

</script>

<style lang="stylus" scoped></style>
```

Downloaded files from crowdin need to be copied into the relevant plugins from which they have come using the *distributefrontendmessages* make command.

These messages will then 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 is based off the current Django language for the request.

2.4 Themes

2.4.1 Content

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

Concepts and Definitions

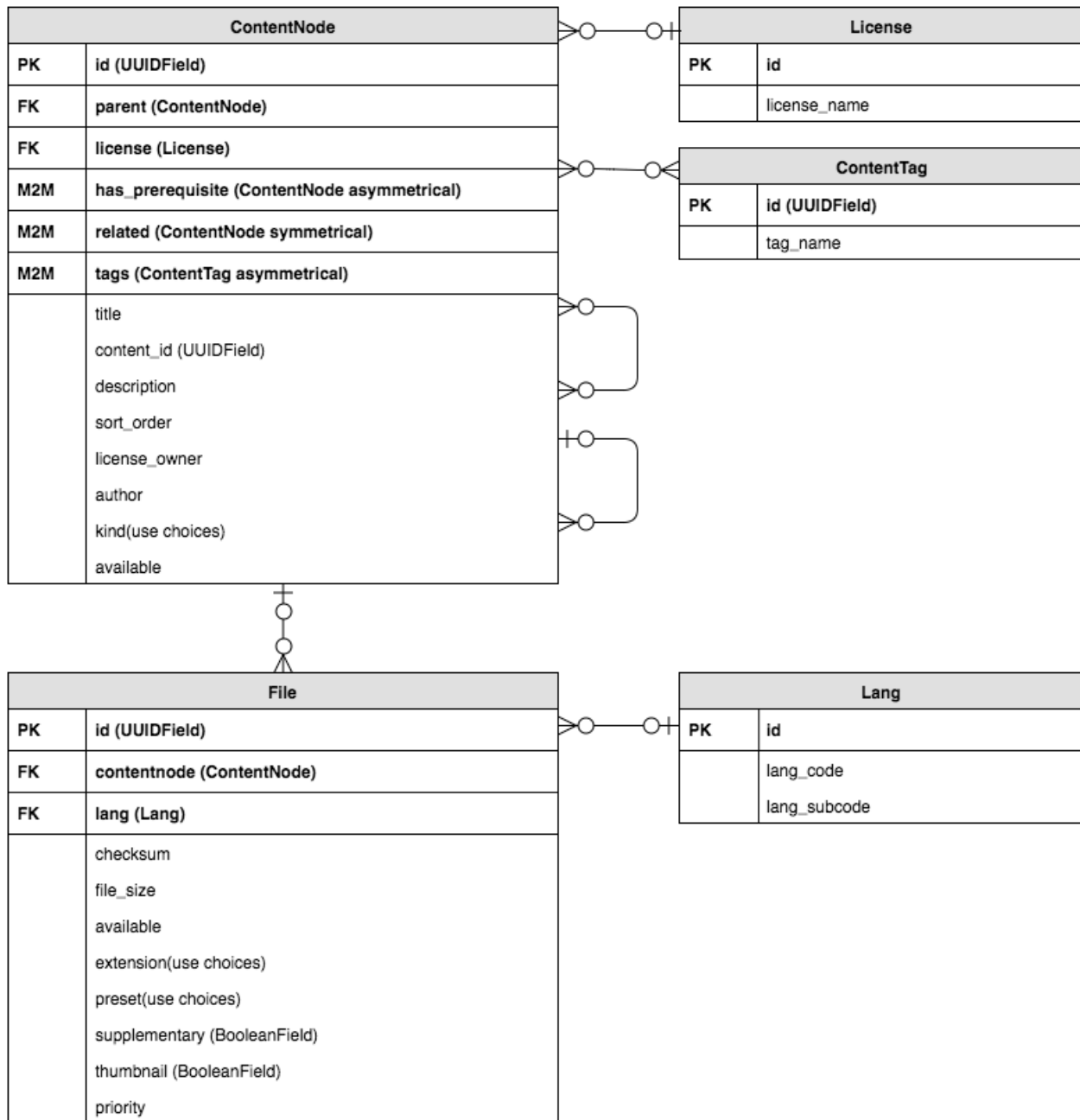
ContentNode

High level abstraction for representing different content kinds, such as Topic, Video, Audio, Exercise, 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

A Django model that is used to store details about the source file, such as what language it supports, how big is the size, which format the file is and where to find the source file.

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

A Django 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 being 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](#), which 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:

```
/home/user/.kolibri/content/storage/9/8/9808fa7c560b9801acccf0f6cf74c3ea.
mp4
```

As you can see, it is fine to store your content files outside of the kolibri project folder as long as you set the `settings.CONTENT_STORAGE_DIR` accordingly.

The front-end will then use the `extension` field to decide which content player should be used. When the `supplementary` field's value is `True`, that means this File object isn't necessary and can display the content without it. For example, we will mark caption (subtitle) file as supplementary.

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 we currently designed to handle content UI rendering and content playback rendering

- Content UI Rendering

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 Playback 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

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
```

Models

2.4.2 Server-Client Communication

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).

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.

Finally, in the `api_urls.py` file, the ViewSets are given a name (through the `base_name` 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('content', ChannelMetadataCacheViewSet, base_name="channel")

content_router = routers.SimpleRouter()
content_router.register(r'contentnode', ContentNodeViewSet, base_name='contentnode')
content_router.register(r'file', FileViewSet, base_name='file')

urlpatterns = [
    url(r'^$', include(router.urls)),
    url(r'^content/(?P<channel_id>[^\./]+)/', include(content_router.urls)),
]
```

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`

```
const Resource = require('kolibri.lib.apiResource').Resource;

class ChannelResource extends Resource {
```

(continues on next page)

(continued from previous page)

```

static resourceName() {
  return 'channel';
}
}

module.exports = ChannelResource;

```

Here, the `resourceName` static method must return `'channel'` in order to match the `base_name` assigned to the `/content` endpoint in `api_urls.py`.

However, in the case of a more complex endpoint, where arguments are required to form the URL itself (such as in the `contentnode` endpoints above) - we can add additional required arguments with the `resourceIdentifiers` static method return value

Listing 3: `contentNode.js`

```

const Resource = require('kolibri.lib.apiResource').Resource;

class ContentNodeResource extends Resource {
  static resourceName() {
    return 'contentnode';
  }
  static idKey() {
    return 'pk';
  }
  static resourceIdentifiers() {
    // because ContentNode resources are accessed via
    // /api/content/<channel_id>/contentnode/<pk>
    return [
      'channel_id',
    ];
  }
}

module.exports = ContentNodeResource;

```

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 instantiated as needed, such as in the coach reports module

```

const ContentSummaryResourceConstructor = require('./apiResources/contentSummary');
const ContentSummaryResource = new ContentSummaryResourceConstructor(coreApp);

```

First the constructor is imported from the require file, and then an instance is created - with a reference to the Kolibri core app module passed as the only argument.

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 `getModel`

```

// corresponds to resource address /api/content/<channelId>/contentnode/<id>
const contentModel = ContentNodeResource.getModel(id, { channel_id: channelId });

```

The first argument is the database id (primary key) for the model, while the second argument defines any additional required `resourceIdentifiers` that we need to build up the URL.

We now have a reference for a representation of the data on the server. To ensure that it has data from the server, we can call `.fetch` on it which will resolve to an object representing the data

```
contentModel.fetch().then((data) => {  
  logging.info('This is the model data: ', data);  
});
```

The `fetch` 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 you want to pass additional GET parameters to the REST API (to only return a limited set of fields, for example), then you can pass GET parameters in the first argument

```
contentModel.fetch({ title: true }).then((data) => {  
  logging.info('This is the model data: ', data);  
});
```

If it is important to get data that has not been cached, you can call the `fetch` method with a `force` parameter

```
contentModel.fetch({}, 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 cached.

The first argument defines any additional required `resourceIdentifiers` that we need to build up the URL, while the second argument 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 a representation of this data on the server. To ensure that it has data from the server, we can call `fetch` on it, this will resolve to an array of the returned data objects.

```
contentCollection.fetch().then((dataArray) => {  
  logging.info('This is the model data: ', dataArray);  
});
```

The `fetch` 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 you want to pass additional GET parameters to the REST API (to only return a limited set of fields, for example), then you can pass GET parameters in the first argument

```
// GET /api/content/<channelId>/contentnode/?popular=1&title=true  
contentCollection.fetch({ title: true }).then((dataArray) => {  
  logging.info('This is the model data: ', dataArray);  
});
```

If it is important to get data that has not been cached, you can call the `fetch` method with a `force` parameter


```
contentCollection.fetch({}, true).then((dataArray) => {
  logging.info('This is the model data: ', dataArray);
});
```

Data Flow Diagram

2.4.3 Users, Authentication, and Permissions

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

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

There are two kinds of users: `FacilityUser` and `DeviceOwner`. A `FacilityUser` is associated with a particular `Facility`, and the user’s account and data may be synchronized across multiple devices. A `DeviceOwner` account is not associated with a particular `Facility`, but is specific to one device, and is never synchronized across multiple devices. A `DeviceOwner` is like a superuser, and has permissions to modify any data on her own device, whereas 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` (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. 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, she is also implicitly 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 also stores the “kind” of the role (currently, one of “admin” or “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 Group Q, we can say that “Bob is a coach for Alice”.

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, but we can delegate most of the work to the database engine (and leverage efficient hierarchy lookups afforded by `MPTT`). The following algorithms and explanations will refer to the naming in the following diagram:

In pseudocode, the query for “What Roles does Source User have in relation to Target User?” would be implemented in the following way:

```
Fetch all Roles with:
    User: Source User
    Collection: Ancestor Collection
For which there is a Membership with:
    User: Target User
    Collection: Descendant Collection
And where:
    Ancestor Collection is an ancestor of (or equal to) Descendant Collection
```

At the database level, this can be written in the following way, as a single multi-table SQL query:

```

SELECT DISTINCT
    source_role.kind
FROM
    collection_table AS ancestor_coll,
    collection_table AS descendant_coll,
    role_table,
    membership_table
WHERE
    role_table.user_id = {source_user_id} AND
    role_table.collection_id = ancestor_coll.id AND
    membership_table.user_id = {target_user_id}
    membership_table.collection_id = descendant_coll.id AND
    descendant_coll.lft BETWEEN ancestor_coll.lft AND ancestor_coll.rght AND
    descendant_coll.tree_id = ancestor_coll.tree_id;

```

Similarly, performing a queryset filter like “give me all ContentLogs associated with FacilityUsers for which Source User has an admin role” can be written as:

```

SELECT
    contentlog_table.*
FROM
    contentlog_table
WHERE EXISTS
    (SELECT
        *
    FROM
        collection_table AS ancestor_coll,
        collection_table AS descendant_coll,
        role_table,
        membership_table
    WHERE
        role_table.user_id = {source_user_id} AND
        role_table.collection_id = ancestor_coll.id AND
        membership_table.user_id = contentlog_table.user_id
        membership_table.collection_id = descendant_coll.id AND
        descendant_coll.lft BETWEEN ancestor_coll.lft AND ancestor_coll.rght AND
        descendant_coll.tree_id = ancestor_coll.tree_id
    )

```

Note the `membership_table.user_id = contentlog_table.user_id` condition, which links the role-membership-collection hierarchy subquery into the main query. We refer to this condition as the “anchor”.

To facilitate making queries that leverage the role-membership-collection hierarchy, without needing to write custom SQL each time, we have implemented a `HierarchyRelationsFilter` helper class. The class is instantiated by passing in a queryset, and then exposes a `filter_by_hierarchy` method that allows various parts of the role-membership-collection hierarchy to be constrained, and anchored back into the queryset’s main table. It then returns a filtered queryset (with appropriate conditions applied) upon which further filters or other queryset operations can be applied.

The signature for `filter_by_hierarchy` is:

```

def filter_by_hierarchy(self,
    source_user=None,
    role_kind=None,
    ancestor_collection=None,
    descendant_collection=None,
    target_user=None):

```

With the exception of `role_kind` (which is either a string or list of strings, of role kinds), these parameters accept either:

- A model instance (either a `FacilityUser` or a `Collection` subclass, as appropriate) or its ID
- An `F` expression that anchors some part of the hierarchy back into the base queryset model (the simplest usage is just to put the name of a field from the base model in the `F` function, but you can also indirectly reference fields of related models, e.g. `F("collection__parent")`)

For example, the `ContentLog` query described above (“give me all `ContentLogs` associated with `FacilityUsers` for which Source User has an admin role”) can be implemented as:

```
contentlogs = HierarchyRelationsFilter(ContentLog.objects.all()).filter_by_hierarchy(
    source_user=my_source_user, # specify the specific user to be the source user
    role_kind=role_kinds.ADMIN, # make sure the Role is an admin role
    target_user=F("user"), # anchor the target user to the "user" field of the
    ↪ContentLog model
)
```

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` (for `DeviceOwner`, this always returns `False`)
- 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 queryset and returns a queryset filtered down 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:

```
from kolibri.auth.api import KolibriAuthPermissions, KolibriAuthPermissionsFilter

class FacilityViewSet(viewsets.ModelViewSet):
    permission_classes = (KolibriAuthPermissions,)
    filter_backends = (KolibriAuthPermissionsFilter,)
    queryset = ContentLog.objects.all()
    serializer_class = ContentLogSerializer
```

Models

2.4.4 User Management

For now, this is a high-level spec that identifies the major components of a work-in-progress part of Kolibri. It is a mixture of a descriptive specification for an app, as well as how it interacts with the `kolibri.auth` back-end layer below it. Eventually, it could serve as a user manual.

The User Management allows a user with sufficient permissions to do a number of things related to managing accounts and roles. It’s divided into two distinct sections

Learner Management

Learner Management provides an interface for:

1. Viewing Classrooms and Learner Groups and a list of the learners they contain.
2. Creating new Classrooms.
3. Creating new Learner Groups.
4. Creating new user accounts and assigning them to Classrooms and Learner Groups.
5. Assigning existing accounts individually or in batches to Classrooms and Learner Groups.
6. Editing a learner’s details, including resetting their password.

The main interface of the Learner Management app is currently described in the mailing list thread “Learner Management app in Kolibri”. We assume the session user (the user who is visiting Learner Management) has write permissions for any object represented in the Learner Management interface. For example, only classrooms and learner groups for which the user has write permissions will be displayed in the Classroom and Group Selectors. In practice this could mean that when the page loads a list of classrooms for which the session user is either a coach or admin will be fetched. At the time of this writing, the only source to determine which users enjoy which permissions is the `kolibri.auth` test suite.

Note: Roadmap: Moving forward, we are making digital prototypes for Learner Management. The aim is to get quality feedback from likely users to inform the design. IMO it is premature to consider a design as stable prior to such feedback. Role Management should be given a similar treatment – quickly create digital prototypes and get quality feedback.

Learner Management has several conceptual parts. These may not reflect how they're divided as Vue components, so I try to reference the current implementation below. The application corresponds to `management/assets/app-root.vue` and has several subcomponents.

Learner Roster

Displays a list of learners determined by the current selectors and filters. Will update automatically based on user interaction with the selectors and filters. Each item in this list corresponds to a learner and has:

- A checkbox for bulk-selecting learners. Selecting multiple learners enables some actions described below.
- A `last name, first name` clickable link. Doing so summons a detail view modal for the learner.
- A `manage` button which summons a class and group management modal for that student.

Note: The roster described here corresponds to `user-page.vue`.

The detail view modal displays learner account data and provides a mechanism to reset a learner's password.

The class and group management modal displays a list classrooms to which the student belongs. Each classroom has a dropdown menu for assigning that learner to a specific group within that classroom. Additionally each classroom has a checkbox for bulk-selection. Bulk-selecting permits the session user to remove the learner from the selected classrooms. Clicking “add” reveals classrooms to which the user doesn't already belong. The learner may be added to those classrooms by selecting them with the corresponding checkboxes, and simultaneously select a group through the associated group dropdown.

Note: UI simplification: There are a number of simplifying assumptions made here. For one, kolibri.auth permits a learner to belong to multiple groups within a classroom. Here we only allow a learner to belong to one group per classroom in order to simplify the UI.

Note: UI simplification: Secondly, kolibri.auth has no notion of being “ungrouped”. The kolibri.auth module defines a Membership model that associates users to Learner Group and Classroom models. For the purposes of this app, when a learner is assigned to a group, then a Membership object to the underlying Learner Group object is created. Membership in a Learner Group implies the user is a member of the containing Classroom as well. When a learner is assigned to the “Ungrouped” group of a Classroom, it corresponds to creating a Membership object associated with the Classroom. In all cases re-assigning a user to a different group should both destroy the existing Membership object and create a new one.

Warning: Roadmap: I consider the detail view and class and group management modals to be somewhat unsettled prior to getting quality user feedback.

Selectors and filters

The UI allows the list of learner in the roster to be filtered. This includes:

- A classroom selector. This is populated by a list of classrooms for which the session user has write permissions, and the special option “All classrooms”. The list of learners is filtered to only show learners who are members of the selected classroom, or all learners if “All classrooms” is selected.
- A group selector. This is disabled if “All classrooms” is selected. Otherwise it is populated with the list of Learner Groups in the classroom with the special option “All groups”. This filters the list of learner analogously to the classroom selector.
- Potentially other filters, for example listing learners in alphabetical or reverse-alphabetical order.

Miscellaneous widgets

Next to the classroom and group selectors are “add” and “remove” buttons. Clicking “add” summons a modal form for creating a new classroom and a new learner group within the currently selected classroom, respectively. The “add” button for groups is disabled if “All classrooms” is selected. Clicking “remove” deletes the currently selected classroom or learner group, respectively. The corresponding “delete” button is disabled if “All classrooms” or “All groups” is selected.

Space is reserved next to the roster for an information panel to display elaborating information based on the current selection. Right now it includes only the total # of students which match the criteria determined by the selectors and filters.

Facility Management

Facility Management (previously referred to as Role Management) will provide an interface for managing user Roles in a Facility. Users may multiply possess Coach and Admin roles for a Facility or Classrooms within a Facility.

Kolibri user data is fundamentally divided into Facilities – a user who belongs to one Facility can never see or interact with user account data from another Facility. However Kolibri provides another user type, called a Device Owner. Device Owners differ from Facility Users in the following ways:

- Device Owners are not syncable from device to device – this account type belongs to one physical machine only. In contrast, Facility Users account and their associated data are syncable.
- Device Owners enjoy every permission. They can be considered Admins for every Facility on the device.
- Device Owners may see and edit *all* Facilities on their Device, including choosing *which* Facility data sets are present on a physical device.

The purview of the Facility Management app is to allow users to give and revoke the Coach and Admin roles for the Facility they belong to and the various Classrooms in that Facility. Moreover the functionality of the app is slightly different if the session user is a device owner:

- If the session user is a Device Owner, the user may select which Facility to manage. Facility Users may only manage their own Facility.
- A Device Owner may edit or delete a Facility. Editing a Facility can change its details like name, description, etc. Deleting a Facility does not destroy it – it is just removed from that device, so that Facility Users tied to that Facility may no longer log in.

Note: Roadmap: Jessica has begun designing this. See [the invision prototype](#).

2.4.5 Front-end Asset Loading

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 also facilitates building of that code into compiled assets with Webpack) and a Kolibri Module (a subclass of `KolibriModule`) on the JavaScript side (see [Front-end Architecture](#)).

Kolibri has a system for synchronously and asynchronously loading these bundled JavaScript modules which is mediated by a small core JavaScript app, `kolibriGlobal`. Kolibri Modules define to which events they subscribe, and asynchronously registered Kolibri Modules are loaded by `kolibriGlobal` 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 `kolibriGlobal` 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 tags defined in `kolibri/core/webpack/templatetags/webpack_tags.py`.

The first way is simply by using the `webpack_asset` template tag.

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 `webpack_base_assets` tag.

This relies on the following function to collect all registered Kolibri Modules and load them synchronously: `kolibri.core.webpack.utils.webpack_asset_render`

Asynchronous loading can also, analogously, be done in two ways. Asynchronous loading registers a Kolibri Module against `kolibriGlobal` 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 `kolibriGlobal` 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`.

2.4.6 User Logs

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

Concepts and Definitions

Content Interaction Log

This Model provides a record of an interaction with a content item. As such, it should encode the channel that the content was in, and the id of the content. Further, it may be required to encode arbitrary data in a JSON blob that is specific to the particular content type.

As a typical use case, a `ContentInteractionLog` object might be used to record an interaction with one instance of an exercise (i.e. one question, but possibly multiple attempts within the same session), or a single session of viewing a video.

Finally, these Logs will use MorangoDB to synchronize their data across devices.

Content Summary Log

This Model provides a summary of all interactions of a user with a content item. As such, it should encode the channel that the content was in, and the id of the content. Further, it may be required to encode arbitrary data in a JSON blob that is specific to the particular content type.

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 InteractionLog is saved, the associated SummaryLog is updated at the same time. This means that the SummaryLog acts as an aggregation layer for the current state of progress for a particular piece of content.

To implement this, a content viewer app would define the aggregation function that summarizes interaction 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.

Finally, 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.

Content Rating Log

This Model provides a record of user feedback on content.

As a typical use case, a ContentRatingLog object might be used to record user feedback data about any content.

Finally, these Logs will use MorangoDB to synchronize their data across devices.

User Session Log

This Model provides a record of an user session in Kolibri. As such, it should encode the channels interacted with, the length of time engaged, and the pages visited.

As a typical use case, a UserSessionLog object might be used to record which pages a user visits, and how long the user is logged on for.

Finally, these Logs will use MorangoDB to synchronize their data across devices.

Implementation Details

Permissions

See *Encoding Permission Rules*.

Models

2.5 References

2.5.1 Files and Directories

`.cache/...` Testing-related, and ignored by git. *TODO - what does it contain?*

`.eggs/...` Packaging-related, and ignored by git. *TODO - what does it contain?*

.github/... These are files used by GitHub to generate templates for things like new pull requests and issues.

.tox/... Tox is a tool for testing software in a range of environments - for example using different versions of Python and Node.

This directory is ignored by git.

TODO - what does it contain?

dist-packages-cache Packaging-related, and ignored by git. *TODO - what does it contain?*

dist-packages-temp Packaging-related, and ignored by git. *TODO - what does it contain?*

docs/... reStructuredText-based documentation, along with Sphinx-based build code

frontend_build/... Code for integrating Kolibri's plugin system with webpack instrumentation for bundling client-side dependencies.

karma_config/... Configuration for Karma, our client-side unit test framework

kolibri/... main code-base, a Django application

requirements/... Python dependency files for PIP

test/... helper files for running tests in Travis CI *TODO - is this correct?*

.editorconfig general editor configuration file

.eslintrc.js configuration file for ESLint, our client-side javascript linter

.gitignore standard .gitignore file

.htmlhintrc configuration for our HTML linter, HTMLHint

.pre-commit-config.yaml configuration for our pre-commit hooks

.stylinttrc configuration for our Stylus linter, Stylint

.travis.yml configuration for Travis

AUTHORS.rst, CHANGELOG.rst, CONTRIBUTING.rst reStructuredText-formatted files. Also imported by the generated */docs*

LICENSE plain-text license files

Makefile wrapper for some scripts, including building packages and docs

MANIFEST.in list of non-python files to include in the Python package

package.json javascript dependencies, helper scripts, and configuration

pytest.ini configuration file for pytest

pytest_runner-2.7.1-py2.7.egg ?

README.rst reStructuredText-formatted file readme

requirements.txt Python PIP dependency requirements, simply redirects to *requirements/base.txt*

setup.cfg ?

setup.py configuration for Python package related to setuptools

tox.ini configuration for our Tox test environments

2.5.2 Glossary

Words with special meanings in the Kolibri ecosystem.

App A Kolibri ‘app’ is a special sort of plugin which provides a top-level URL and a self-contained single-page javascript application. Each app attaches a single root view component to the HTML returned by a skeleton Django template.

Examples of apps are the Learn, Admin, and Coach Reports apps.

Bundle ‘Bundle’ is a webpack term, referring to a collection of client-side assets. See [Front-end Asset Loading](#) for more info.

Component A ‘view component’ is a composable UI element on the client-side, defined using Vue.js components.

Components are defined using using HTML, other components, styling, internationalized text, internal logic, and – if necessary – internal state. Every component has an interface defined by its input parameters, events, and slots that can take arbitrary HTML to render in itself.

See [Front-end Architecture](#) for more info.

Hook A ‘hook’ is the server-side mechanism by which plugins interact with each other and with the core app. Hooks allow behaviors and interactions to be defined abstractly by Kolibri core and then implemented concretely by plugins.

See [Plugins](#) for more info.

Plugin Kolibri ‘plugins’ define both functionality client- and server-side functionality. They can be enabled and disabled on a per-installation basis. Plugins are decoupled from each other, but are dependent on the core Kolibri application.

In theory, any plugin can be disabled and the application should still function without error, albeit limited functionality.

Examples of plugins include the Learn application, and a content renderer for vector video.

See [Plugins](#) for more info.

Changes are ordered reverse-chronologically.

3.1 0.4.9

- User experience improvements for session timeout

3.2 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

3.3 0.4.7

- Fix bug that made updating existing Django models from the frontend impossible

3.4 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

3.5 0.4.5

- Frontend and backend changes to increase performance of the Kolibri application under heavy load
- Fix bug in frontend simplified login code

3.6 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

3.7 0.4.3

- Speed improvements for content recommendation #1798

3.8 0.4.2

- Fixes for morango database migrations

3.9 0.4.1

- Makes usernames for login case insensitive #1733
- Fixes various issues with exercise rendering #1757
- Removes wrong CLI usage instructions #1742

3.10 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

3.11 0.3.3

- Turns video captions on by default

3.12 0.3.2

- Updated translations for Portuguese and Kiswahili in exercises.
- Updated Spanish translations

3.13 0.3.2

- Portuguese and Kaswihili updates
- Windows fixes (mimetypes and modified time)
- VF sidebar translations

3.14 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

3.15 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 Bubl  instead of Babel
- Webpack build process compatible with plugins outside the kolibri directory
- Vue2 refactor
- HTML5 app renderer

3.16 0.1.1

- SVG inlining
- Exercise completion visualization
- Perseus exercise renderer
- Coach reports

3.17 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

3.18 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

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given. You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at <https://github.com/learningequality/kolibri/issues>.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

4.1.4 Write Documentation

Kolibri could always use more documentation, whether as part of the official Kolibri docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at <https://github.com/learningequality/kolibri/issues>.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *kolibri* for local development.

1. [Fork](#) the *kolibri* repo on GitHub.
2. Clone your fork locally:

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

3. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

4. When you're done making changes, check that your changes pass style and unit tests, including testing other Python versions with tox, and testing any Javascript changes with npm:

```
$ tox
$ npm test
```

To get tox, just pip install it. To get node (and hence npm) install it from <https://nodejs.org>.

5. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. For your pull request description on Github, consider using our template, see [Pull Request Template](#).
2. Remember to add yourself to `AUTHORS.rst` and fill in `CHANGELOG.rst` with your feature or bug fix.

3. The pull request should include tests.
4. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
5. The pull request should work for Python 2.7, 3.4, and 3.5, and for PyPy. PRs will be automatically tested, we recommend running the `tox` command locally before submitting your PR for review.

4.3.1 Pull Request Template

Copy-paste the following to your Pull Request description on Github:

```
## Summary

*Short description*

## TODO

- [ ] Have tests been written for the new code?
- [ ] Has documentation been written/updated?
- [ ] New dependencies (if any) added to requirements file
- [ ] Add an entry to CHANGELOG.rst
- [ ] Add yourself it AUTHORS.rst if you don't appear there

## Reviewer guidance

*If you PR has a significant size, give the reviewer some helpful remarks*

## Issues addressed

List the issues solved or partly solved by the PR

## Documentation

If the PR has documentation, link the file here (either .rst in your repo or if built_
↳on Read The Docs)

## Screenshots (if appropriate)

They're nice. :)
```

4.4 Tips

To run a subset of tests:

```
$ py.test test/test_kolibri.py
```


5.1 Development Lead and Copyright Holder

- Learning Equality – info@learningequality.org

5.2 Community

Please feel free to add your name on this list if you do a PR!

- Benjamin Bach (benjaoming)
- Michael Gallaspy (MCGallaspy)
- Richard Tibbles (rtibbles)
- Jamie Alexandre (jamalex)
- David Cañas (dxcanas)
- Eli Dai (66eli77)
- Devon Rueckner (indirectlylit)
- Rafael Aguayo (ralphiee22)
- Christian Memije (christianmemije)
- Radina Matic (radinamatic)

CHAPTER 6

Kolibri

CHAPTER 7

What is Kolibri?

Kolibri is a Learning Management System / Learning App designed to run on low-power devices, targeting the needs of learners and teachers in contexts with limited infrastructure. A user can install Kolibri and serve the app on a local network, without an internet connection. Kolibri installations can be linked to one another, so that user data and content can be shared. Users can create content for Kolibri and share it when there is network access to another Kolibri installation or the internet.

At its core, Kolibri is about serving educational content. A typical user (called a Learner) will log in to Kolibri to consume educational content (videos, documents, other multimedia) and test their understanding of the content by completing exercises and quizzes, with immediate feedback. A user's activity will be tracked to offer individualized insight (like “next lesson” recommendations), and to allow user data to be synced across different installations – thus a Kolibri learner can use his or her credentials on any linked Kolibri installation, for instance on different devices at a school.

See <https://learningequality.org/kolibri/> for more info.

CHAPTER 8

How can I use it?

Kolibri is under active development and is not yet ready to be used. In the meantime, take a look at [KA-Lite](#), Kolibri's predecessor which is already deployed around the world.

CHAPTER 9

How can I contribute?

Warning: We welcome new contributors but since **Kolibri** is still in development, the API is not yet completely ready to integrate external plugins. Please start by:

- Reading our [Developer Documentation](#) available online, and in the `docs/` directory.
- Contacting us on the Mailing list: [Google groups](#).
- or via IRC: #kolibri on Freenode.