

Section 1: About You

Goals and Objectives

Why do you want to do a GSoC project with Oppia?	<p>There are two main reasons why I want to do a GSoC project with Oppia. Firstly, my recent experience with the LaCE team has been invaluable. I've been working with the LaCE team for quite a while now. When I started, the codebase felt very overwhelming, but after spending some time playing with it, I really liked the way the project is organized. I also got to learn so much from the code reviews and other team members, which helped me to think about other things, not just code. Secondly, I deeply resonate with Oppia's mission and would love to be part of it and give back to the community. These two reasons are why I want to do a GSoC project with Oppia.</p>	
What do you hope to learn/achieve during GSoC?	<p>I definitely hope to improve my technical skills, like creating frontend with Angular and writing backend in Python. Apart from this, I'm really excited to work on the testing part, as I have never done much testing at my last internship, and I often skip the testing part when working on my personal projects. So, I'm excited to work on testing as this project also involves writing a lot of e2e and acceptance tests.</p> <p>And I believe GSoC will not only increase my technical knowledge but also provide an opportunity to learn other skills that will help me in my future software engineering journey, like writing maintainable code, debugging, and improving my communication skills.</p>	
Which Oppia teams have you collaborated with, and what have you done on those teams?	LaCE	<p>#19786 - Identified a duplicate issue related to the lesson button and closed it, avoiding duplicates.</p> <p>I have been active in the community for a long time, answering questions in the discussion tab, and I'm also part of the welfare team.</p>
	Angular Migration	Helped team members access gated pages (here)
Contact information	<p>Email/Google Chat: ak341668@gmail.com Linkedin: Afzal Khan</p>	
Preferred method of communication	<ul style="list-style-type: none">• Google Chat and Email: For sharing updates.• Google Meet: For weekly meetings. <p>And open to other methods as well.</p>	
Which timezone will you primarily be in during the summer?	Indian Standard Time, UTC + 5:30	

(If you are a student) When are your school holidays?	I'm in my 6th semester, and I will have my summer vacation from May 23 to June 15. This is the time where I'll be dedicating most of my time to the project. And during normal college days, with my classes going on, I can easily dedicate around 25-30 hours per week to the project as the college gives NOC (No Objection Certificate) to interns so I won't need to attend all classes..
What other obligations might you need to work around during the summer?	The only obligation I have during the GSoC period is my end term exams, which is between July 16 to July 22. As I'm in my 6th semester (end of 3rd year), there are no obligations related to internship and placement and no plans for travel. Most subjects are CS related, so I won't need much time for preparation; I'll probably take one day off for the statistics exam.
Planned time commitment	May 27 - June 15: 30+ hours/week (Summer vacations) June 16 - Aug 19: 25-30 hours/week (OK since our college allows internships and doesn't have a strict attendance policy) – I will be taking one day of holiday each week.

Section 2: About Your Project


Project Details

Project title <i>(should match the one on the Project Ideas list)</i>	Infrastructure and navigation for multiple classrooms
Project size <i>(should match the options on the Project Ideas list)</i>	Medium (~175 hours)
Why did you choose this project?	I picked this project because I understand what it's about, and I think I'll enjoy working on it and after going through the codebase related to this project, I feel very confident regarding the technical implementation part, as I have experience working on similar issues. And I believe I can take ownership of this project and complete it within the deadline.

Required Skills

WEB	
I can write Python code with unit tests.¹	#19777 - Fixed logged out user unable to visit exploration editor page #19619 - Added functionality in the admin page to retrieve the set of

¹ To develop this skill: Some [LaCE](#) and [Contributor Dashboard](#) issues have a backend component, and many [Dev Workflow](#) issues do too. Focus on issues that aren't marked as "backlog".

	interactions used in an exploration. #19274 - Making correctness feedback the default setting for all lessons.
I can write TS + Angular code with unit tests.²	#19706 - Same unit types should be accepted in Number with units interaction. #19826 - Migrated the Learner Group Creator page to Lazy-Loaded Angular Module. #19625 - Show a banner if the editor visits their unpublished exploration in the player.
I have good UI judgment and attention to detail.³	#19194 - Updated the image upload prerequisite notice box. #19208 - Align creator dashboard stats elements properly. #18971 - Fixed the alignment of voice artists text in exploration editor.
I can debug and fix CI failures / flakes.⁴	N/A
I can write or modify e2e or acceptance tests.⁵	#20195 - Acceptance test for logged-in learner
I can communicate effectively using debugging docs.⁶	 Blog page refreshes when attempting to go to a specific page
I can write or modify Beam jobs.⁷	N/A
I have participated in QA testing.⁸	release-3.3.5 release-3.3.6 release-3.3.7

Project Timeframe

Note: Oppia will only be offering a single GSoC coding period timeframe this year, starting on **May 27**. All work for Milestone 1 must be completed and submitted by **Jun 28** for internal feedback (with any revisions due by **Jul 8**), and all work for Milestone 2 must be completed and submitted by **Aug 12** for internal feedback (with any revisions due by **Aug 19**). We will not be able to extend these deadlines.

Coding period	<ul style="list-style-type: none"> I will adhere to the above deadlines.
----------------------	---

² To develop this skill: Most [LaCE](#) and [Contributor Dashboard](#) issues have a frontend component. Focus on issues that aren't marked as "backlog".

³ To develop this skill: Tackle a non-backlog responsiveness issue from the [LaCE team](#).

⁴ To develop this skill: See issues labelled "[CI breakage](#)" on GitHub, or look at [CI failures in develop](#) (or cross-signs [here](#)) and figure out how to fix them.

⁵ To develop this skill: Solve part of issue [#17712](#) by covering the CUJs for one or more sets of users.

⁶ To develop this skill: Tackle an issue that requires creating a debugging doc that you share with the broader team. Fixing CI failures counts.

⁷ To develop this skill: See [this doc](#) for some examples of issues to work on.

⁸ To develop this skill: Join the release testers mailing list at oppia-release-testers@googlegroups.com, and look out for calls to help with release testing.

Communication Channels

Note: The Oppia team places a high emphasis on communication, and we have found that daily contact between contributors and mentors is important for helping keep projects on track. This is why we ask that contributors send short daily updates to their mentors explaining what they have done, where they are stuck, and what they plan to do next.

I can commit to sending daily updates to my mentor by email, each day I work during the GSoC period.	<ul style="list-style-type: none">• Yes
In addition to the above: how often, and through which channel(s), do you plan on communicating with your mentor?	<ul style="list-style-type: none">• I will maintain a doc to track the progress and daily update mentors via email or google chat.• And I plan to have at least one meeting every week to discuss issues and plan for the next week.

Section 3: Proposal Details

Problem Statement

Oppia currently has a classroom page for Math, which is featured prominently and accessible via the home page, navigation bar, community library, and learner dashboard. Oppia is also planning to create classrooms for other subjects. The objective of this project is to facilitate the creation of new classrooms directly from the Oppia website and make them available to learners from the community library, the new learn page, a navigation link, and from the learner dashboard.


Target Audience	<ol style="list-style-type: none">1. Learners2. Teachers and parents3. Curriculum Admins
Core User Need	<p>As a learner, I want to be able to find officially Oppia-curated content easily and be able to select and learn new topics in a structured manner (e.g. History, Science, etc.).</p> <p>As a teacher/parent, I want to be able to find curated content easily to educate my kids/students.</p> <p>As a curriculum admin, I would like to create new classrooms directly from Oppia and also manage them completely.</p>
What goals do we want the solution to achieve?	<ol style="list-style-type: none">1. Ability to create and manage new classrooms directly from Oppia without hardcoding anything.2. Ability to access other Oppia curated classrooms.

Section 3.1: WHAT

Key User Stories and Tasks that will be implemented

#	Title	User Story Description (role, goal, motivation) <i>"As a ..., I need ..., so that"</i>	List of tasks needed to achieve the goal (this is the "User Journey")	Links to mocks / prototypes, and/or PRD sections that spec out additional requirements.
1	Access classrooms from the Splash page (oppia.org)	As a learner, I want to browse lessons from the splash page, so that I can start my learning journey immediately.	Upon reaching the home page of Oppia.org, I immediately see the option to browse Oppia lessons ("Browse our lessons")	N/A
			After clicking on "Browse our lessons", I am brought to the Classrooms page, which shows me all of the classrooms on Oppia.	N/A New classroom page at /learn: → If we have a single classroom, redirect to it. → Otherwise, show all classrooms.
2	Access classrooms from navigation bar > learn	As a learner, I need to see the available learning options on Oppia from the navigation bar, so that I can quickly access learning content from any page.	When clicking on "learn" on the navigation bar, a drop-down box appears showing me the options to learn: Oppia Classrooms (Math and Science), Community Library, and to try the Android app	Updated Navigation dropdown → Show a set of learning options in the Learn tab of the navigation bar.
			I can see a brief description of each learning option to understand more.	Same as above
			There are links to the Math classroom and some key science lessons, which will bring me to the respective pages when clicked.	Same as above
			There are links to the Science classroom and some key science lessons, which will bring me to the respective pages when clicked.	Same as above
			There is a "Community Library" link which will bring me to the Community Library	Same as above

			page when clicked.	
			There is a “Try our Android app” link which will bring me to the Android app intro page when clicked.	Same as above
3	Access from Community Library (oppia.org/community-library)	As a learner, I need to be presented with various learning options in the Community Library, so that I can pick the most suitable lesson to start learning.	When I access the Community Library page, I see a section introducing the Oppia classroom.	Updated Community Library page
			I can get a brief introduction of each Oppia-curated subject (classroom) in the Oppia classrooms section.	Same as above
4	Access from Learner Dashboard (oppia.org/learner-dashboard)	As a learner, I need to see the progress of my lessons on my dashboard, so that I can easily track and return to incomplete lessons.	Upon logging in, I am directed to the learner dashboard.	N/A → Existing flow
			[Learner Dashboard - Continue where you left off] On the learner dashboard, I can see a section containing the lessons I’ve started on but not yet completed, along with the progress I have made on each of them. The lessons are grouped based on topics: Math, Science, Community Library.	Learner Dashboard → The mocks are based on the new learner dashboard, which is not in the scope of this project. → If the learner dashboard redesign is completed before this project, then the new classroom feature will be implemented on the new learner dashboard; otherwise, on the old design.
			[Learner Dashboard - Learn something new] On the learner dashboard, I can see a section containing lessons that I have not started learning yet. There should be Math and Science topics listed in this section.	Same as above
			[Learner Dashboard - Progress page] On the revamped “Progress” page found in the Learner Dashboard, I can track the progress of incomplete lessons in the sections for Math Classroom, Science Classroom, and Community	Same as above

			Library.	
			[All] On each incomplete lesson card, there should be a topic label, allowing me to clearly see which topic this lesson falls under.	Same as above
5	Classrooms page (new URL: oppia.org/learn)	As a learner, I need to view all available Oppia classrooms in one page, so that I can choose a topic I wish to learn for the day.	I am presented with an introduction of Oppia classrooms on the Classrooms page, along with the Math and Science classrooms here.	New Classrooms Page → When a learner visits this page, we will show a list of classrooms in card form with their thumbnail, title, and description. → If we have only one classroom, we will directly redirect the learner to that classroom.
			The available classrooms (Math and Science) are clickable and direct me to the relevant classroom pages when clicked.	Same as above
			Classrooms that are not yet available will have an “under construction” banner appearing, and are not clickable. Learners will not be able to click on those classrooms.	Classroom under construction card mock PRD reference:  Oppia PRD - Multiple Cla...
			Under the title for each classroom, I can see a short summary of the content that will be covered in each classroom.	Same as above
6	Per Subject Classroom page (URL: oppia.org/learn/{{classroomName}})	As a learner, I need to navigate back to the classrooms page, so that I can see all available classrooms again.	On each classroom page, I see a link to navigate back to the main Classrooms page.	Generalized Classroom Page
			On each classroom page, I am presented with all lessons related to that topic.	N/A → Existing feature
			On each classroom page, I see a "TAKE A TEST" link which will open the diagnostic test player for the	The Diagnostic Player already exists, but we will need to modify it to work for any classroom, not just math.

			respective classroom, and after completing it, suggested topics from that classroom should be provided.	
7	Subject indicator for each topic	As a learner, I need to know which subject each topic relates to, so that I can plan my learning accordingly for each subject.	On each topic page, I can see a breadcrumb which identifies the subject I'm learning	Topic Viewer Page (Home) Topic Viewer Page (Playing a chapter) → We will show breadcrumbs if the topic belongs to any classrooms; otherwise, we will just show the topic name.
			The breadcrumb is a clickable link that brings me to the subject classroom page (eg.math, science, etc)	
		As a topic editor/curriculum admin, while editing a topic, I need to know which classroom the topic is assigned so I can plan better.	In the details section of the topic editor page, I can see a classroom field which tells me the assigned classroom name. If the topic is not assigned to any classroom, it says "No classroom assigned". And I can also see a note saying, "To change the assigned classroom, please contact one of the curriculum admins: {{username1}}, {{username2}}."	Topic Editor Page
8	Creating and updating a classroom (URL: /classroom-admin)	As a curriculum admin, I need to be able to create a new classroom and manage each and every property of it.	On the classroom admin page, when creating a new classroom, a modal is opened where I can fill in all the necessary details needed to create a classroom. I can add a thumbnail, banner, title, description, topic description, and add some topics as well.	Create classroom modal
			When adding topics, I will be presented with a suggestive dropdown with a list of unassigned topics. I can then enter the topic name in the input field and select the desired topic.	Topic selection dropdown
			After adding all the required details, I should be able to	Editing a classroom

			edit the classroom and also change the publication status of the classroom.	→ We will be able to change the classroom publication status using the select dropdown.
--	--	--	---	---

Technical Requirements

Additions/Changes to Web Server Endpoint Contracts

#	Endpoint URL	Request type (GET, POST, etc.)	New / Existing	Description of the request/response contract (and, if applicable, how it's different from the previous one)
1.	/can_access_classroom_page?classroom_url_fragment (frontend) (backend)	GET	Existing	<p>This is an access handler for the classroom page, using @acl_decorators.open_access, granting access to everyone. We need to restrict this access, as there will be unpublished classrooms, and access should only be given to curriculum admins and super admins.</p> <p>Everything will remain the same. a new acl decorator, can_access_classroom_page will be introduced. It grants access to everyone if the classroom is published, but if it's not, access will only be given to curriculum admins and super admins.</p>
2.	/classroom_data_handler/<classroom_url_fragment> (frontend) (backend)	GET	Existing	<p>The request contains classroom_url_fragment. And the response will contain.</p> <ul style="list-style-type: none"> • course_details (string) • name (string) • topic_list_intro (string) • topic_summary_dicts (List[ClassroomTopicSummaryDict]) • is_published (boolean) → NEW <p>This will be protected by the newly introduced acl decorator can_access_classroom_page to restrict access if the classroom is not published. And a new field will be introduced to tell the publication status, which will be used to display an "Under Construction" banner to curriculum admins.</p>
3.	/classroom/<classroom_id> (frontend) (backend)	PUT	Existing	<p>The request contains classroom_dict with following properties:</p> <ul style="list-style-type: none"> • classroom_id (string) • name (string) • url_fragment (string) • course_details (string) • topic_list_intro (string) • topic_id_to_prerequisite_topic_ids (Dict[str, List[str]]) • is_published (boolean)

				<ul style="list-style-type: none"> • banner: ImageData • thumbnail: ImageData <p>ImageData: { filename: string image_blob: Blob null background_color: string }</p> <p>thumbnail_image and banner_image are optional as we are updating the classroom.</p> <p>All other fields are compulsory.</p>
4.	/classroom/<classroom_id> (frontend) (backend)	GET	Existing	<p>The response contains classroom_dict with following properties:</p> <ul style="list-style-type: none"> • classroom_id (string) • name (string) • url_fragment (string) • course_details (string) • topic_list_intro (string) • topic_name_to_prerequisite_topic_names (Dict[str, List[str]]) • is_published (boolean) • thumbnail_filename (string) • banner_filename (string) <p>Changes are the same as above, but here they are retrieved in the response.</p>
5.	/topics_to_classrooms_relation	GET	New	<p>The response will contain an array of TopicClassroomInfoDict which contains following properties.</p> <p>topics_to_classrooms_relation: { topic_id (string) topic_name (string) classroom_name (string null) url_fragment (string null) }</p> <p>This will be used to fetch all the topics and the classroom name and url_fragment to which they are assigned.</p> <p>→ This data will be used to display a list of available topics on the curriculum admin page when creating or updating a classroom.</p>
6.	/all_classrooms_summary	GET	New	<p>The response contains a list of classroomSummaryDicts, each containing a subset of classroom data.</p> <p>ACL Decorator: open_access</p>

				<p>Response : classroomSummaryDict[]</p> <pre>classroomSummaryDict : { name (string) url_fragment (string) teaser_text (string) is_published (boolean) thumbnail_filename (string) thumbnail_bg_color (string) }</pre> <p>This endpoint's response will be used to display classroom cards in the new classrooms and the community library page.</p>
7.	/classroom/create_new	POST	New	<p>This endpoint will be used to create a new classroom. The request will contain the following properties:</p> <ul style="list-style-type: none"> • name (string) • url_fragment (string)

Calls to Web Server Endpoints

#	Endpoint URL	Request type (GET, POST, etc.)	Description of why the new call is needed, or why the changes to an existing call is needed
1.	/can_access_classroom_page?classroom_url_fragment	GET	<p>The current access handler provides access to everyone. As we will have hidden classrooms as well, we want only curriculum admins to access hidden classrooms and display a banner at the top about the same.</p> <p>To achieve this, the open_access decorator will be replaced by the new can_access_classroom_page decorator, which will give access to public/visible classrooms for everyone and grant access to hidden classrooms to curriculum admins only.</p>
2.	/classroom_data_handler/<classroom_url_fragment>	GET	<p>It will be used to fetch learner-facing classroom data, which will be displayed on the classroom page.</p> <p>Most parts will remain the same. Only the following changes will be made:</p> <ul style="list-style-type: none"> • The <code>acl</code> decorator will be updated from does_classroom_exist to can_access_classroom_page to prevent access to hidden classrooms. If a curriculum admin visits a hidden classroom page, a banner will be shown at the top. If anyone else visits the page, a 404 error will be thrown. • The newly created field, is_published (boolean), will be returned with the response. This field will be used to display a banner to curriculum admins.
3.	/classroom_id_	GET	This request is needed to fetch all classroom IDs and names. This

	to_name_handler		call is made in the curriculum admin page to display all the created classrooms.
4.	/classroom/<classroom_id>	DELETE	This request is needed to delete the classroom with the given classroom_id.
5.	/classroom/<classroom_id>	PUT	<p>This request is needed when creating a new classroom or updating an existing one.</p> <p>New thumbnail_filename, banner_filename, and is_published fields will also be sent in the request, enabling curriculum admins to update each classroom directly from the website.</p>
6.	/classroom/create_new	POST	This request is needed to create a new classroom.
7.	/classroom/<classroom_id>	GET	The request is needed while editing a classroom – it is used to fetch all the classroom properties. Newly added fields (banner_filename, thumbnail_filename, is_published) will also be received in this process.
8.	/topics_to_classrooms_relation	GET	<p>The request is needed to determine which classroom is assigned to which topic, as we cannot add one topic to more than one classroom.</p> <p>The response contains an array of all topics and the classrooms to which they are assigned. This information will be used to display unassigned topics when adding a topic to a classroom. And we will also use this data to translate topic_id to topic name.</p>
9.	/classroom_url_fragment_handler/<classroom_url_fragment>	GET	The request is used to check whether the URL fragment is valid or not, i.e., to verify whether a classroom already exists with this URL fragment.
10.	/all_classroom_summary	GET	<p>The request is made from the new /learn page and community library. It will be used to fetch all the classrooms and a subset of their properties to display classroom cards to learners.</p> <p>The classrooms page needs name, url_fragment, is_published, teaser_text and thumbnail_filename. The community library page also needs all these properties except teaser_text.</p>

UI Screens/Components

#	ID	Description of new UI component	i18n required?	Mock/spec links	A11y requirements
---	----	---------------------------------	----------------	-----------------	-------------------

1.	Classrooms Page Component	The Classrooms Page will display all the available classroom cards in a grid view.	YES	New Classrooms Page	Yes, we will use proper semantic HTML elements, add alt text for the classroom thumbnail and banner, and include ARIA labels.
2.	Classroom Card Component	<p>The classroom card will be used on the /learn page to display the following classroom details:</p> <ul style="list-style-type: none"> • name • course details • thumbnail <p>→ Classrooms that are still not published will be shown with a "Classroom Under Construction" banner and click will be disabled.</p>	<p>YES</p> <p>–We will translate the classroom card's name and course details using the same method we use for the classroom page (explained here).</p> <p>And for translating "Classroom Under Construction" and "More subjects coming soon!" We will add I18N keys for them.</p>	Classroom Card	<p>Yes – Same as above</p> <p>For classrooms that are still unpublished, we will add the 'aria-disabled' attribute. This will enable users who uses screenreader to tab to them and know that these classrooms will be available soon.</p>

Data Handling and Privacy

Not required, as no user data is being collected.

Other Requirements

N/A

Section 3.2: HOW

Existing Status Quo

- To provide a better learning experience for learners, Oppia has a single Math classroom that contains multiple math-related topics. It can be accessed easily from the home page (navigation bar and "browse our lessons" button), community library, and learner dashboard.
- As the classroom contains a set of curated topics, it provides learners with a structured plan to learn new topics.
- We want to create more Oppia-curated content for learners. Currently, although we can create new classrooms, they are not accessible at all to learners, and curriculum admins cannot fully customize new classrooms directly from the /classroom-admin page.
- The conclusion is that the current state is not optimal for multiple classrooms, and it's important to improve the accessibility of multiple classrooms and provide curriculum admins with full control over how classrooms are displayed in the Oppia-Web application.

Solution Overview

The solution can be broken down into two parts: creating and updating new classrooms (curriculum admin-facing) and accessing all the classrooms (learner-facing).

1. Curriculum Admin Facing

The curriculum admin-facing solution part can be further broken down into two sub-parts: **(a) Creating and updating classrooms**, and **(b) Managing topics**.

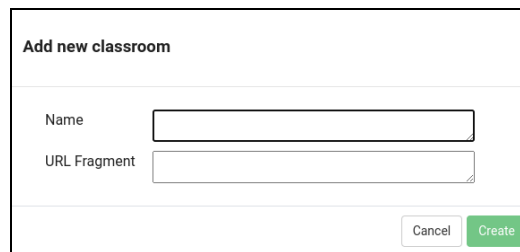
(a) Creating and updating classrooms

The ability to create and manage classrooms directly from Oppia web is a crucial part of this project. To achieve this, there will be some changes to the existing classroom creation flow, and new properties will be introduced in order to fully customize a classroom.

Creating a new classroom

The curriculum admin can create a new classroom from the classroom admin page by clicking on the "Add new classroom" button. Then, a modal will pop up where they will be able to enter the following classroom properties:

- name
- URL Fragment

A modal window titled "Add new classroom" with a light gray background. It contains two text input fields: "Name" and "URL Fragment". Below the inputs are two buttons: "Cancel" (light gray) and "Create" (green).

Add new classroom	
Name	<input type="text"/>
URL Fragment	<input type="text"/>
<div>Cancel Create</div>	

The curriculum admin can save the classroom in any state, as it will not be shown to learners until it is published. However, when they want to publish it, they will need to add all the required properties.

While creating a classroom, the **default publish status will be set to hidden**, as there might be some changes required or the need to add more topics later. After entering all the minimum details, the curriculum admin can create a classroom by clicking on the create button.

Updating existing classrooms

– After creating a classroom admin will be able to view it on the classroom admin page and **edit its publication status** and other properties.

- name
- teaser_text (small introduction of the classroom)
- URL fragment (unique classroom URL fragment, e.g., /learn/math)
- Course details (description of classroom)
- Topic list intro (information about the topics)
- Topics and Prerequisite topics

- Earlier, the curriculum admin had to enter topicID to add a new topic, which involves a lot of back and forth.
- It would be better to add topics by their name. This will also prevent making an extra GET call to translate topic_id to topic_name.
- When clicking on the Topic Name input, a dropdown with all the **unassigned topics** will appear, and the user can add any one by searching for the topic name.
- Classroom thumbnail (the logo for the classroom)
- Classroom banner (It will be shown in background to learners on the /learn/{{classroom}} page)
 - The banner will be different in all classrooms.
 - [Science classroom](#)
 - [Math classroom](#)
- Until the classroom is visible, it will be displayed as "Classroom under construction" to learners, and the card will be disabled.
- If curriculum administrators visit the classroom page, they will see a banner that says, "The classroom is not yet published."

Classroom details

math

Name

math

URL Fragment

math

Teaser Text

Topic List Intro

Course Details

Thumbnail

?

Add image

Banner

?

Add image

Topics and their prerequisites

The prerequisite topics below are used to generate the diagnostic test.

No topics are currently added in the classroom.

Add Topic

+ ADD NEW CLASSROOM

Cancel

Publish

Save

Other features, such as viewing the topic graph and the drag-and-drop topic order, will remain the same.

(b) Managing topics

A classroom is essentially a collection of topics. Topic managers need to know which classroom their topic is in. Therefore, it is essential to update the topics-and-skills dashboard to support filtering based on classrooms. Topic managers and curriculum admins will be able to do the following things from this dashboard:

- Filter topics by classroom name.
- Know which topic is assigned to which classroom.

The mockup shows a dashboard titled "Topic and skill dashboard". On the left is a "Filter" sidebar with a "Reset" button. The sidebar has four sections: "Status" with a "Status Options" dropdown set to "All"; "Classroom" with a search bar and a dropdown menu showing "Math" and "science"; "Sort" with a dropdown menu; and "Keywords" with a search bar. On the right, there are two topic cards. The first card is titled "test topic" and shows "No Classroom Assigned." in a box. The second card is titled "Dummy Topic 1" and shows "Science" in a box, with "description" below it.

Sorting topics based on classroom

The topics can be sorted by the classroom name, and in the topic and skill dashboard, we already display which classroom the topic is assigned to. All the sorting options work, **but the default sorting order is based on the creation date, with newly created topics displayed first. So, we will change it to "Most Recently updated" when selecting the classroom option in the filter, since that is typically what the user will have been working on most recently.**

Assumptions made:

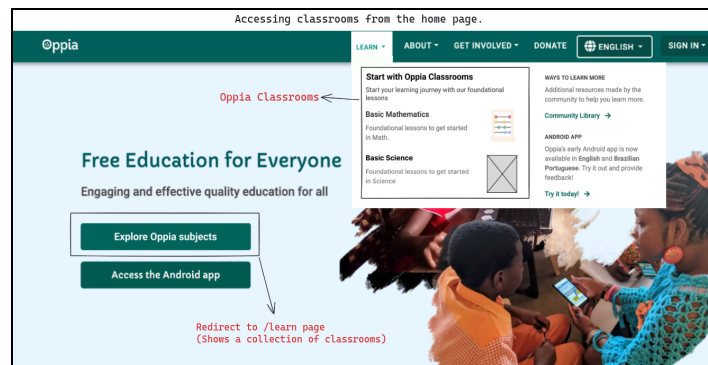
- A topic cannot be assigned to multiple classrooms. (See "Technical hints" section [here](#))
- We are not adding pagination, filters, or any other features to the /classroom-admin page, as in the near future, we only plan to add around 4 classrooms. Hence, these features are unnecessary for now (discussed [here](#)).

– We will also show the classroom name in the topic editor if the topic is assigned to any classroom. Otherwise, we will display the text "No classroom assigned" in the topic details section. ([Mock](#)).

2. Learner Facing

The classrooms will be easily accessible to learners via the Home page, community library, learner dashboard, and the new classrooms page (available at /learn).

(a) Home Page and Navigation Bar

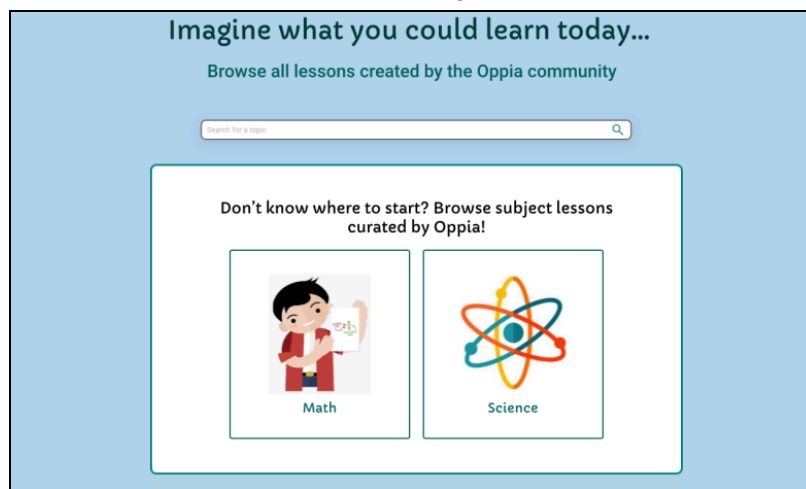


The learner will be able to access classrooms quickly from the home page by clicking on "Explore Oppia subjects" or from the Learn tab of the navigation bar.

- Clicking on the "Explore Oppia subjects" button will redirect the user to the new /learn page, where they will be presented with all classroom cards in a grid view.
- And when clicking on an individual classroom card in the navigation bar, it will redirect the learner to the respective classroom page.

(b) Community Library

- The classrooms will be shown in a carousel on the community library page. The classroom card will only contain the classroom name and the thumbnail image.





(c) New classrooms page (at /learn)



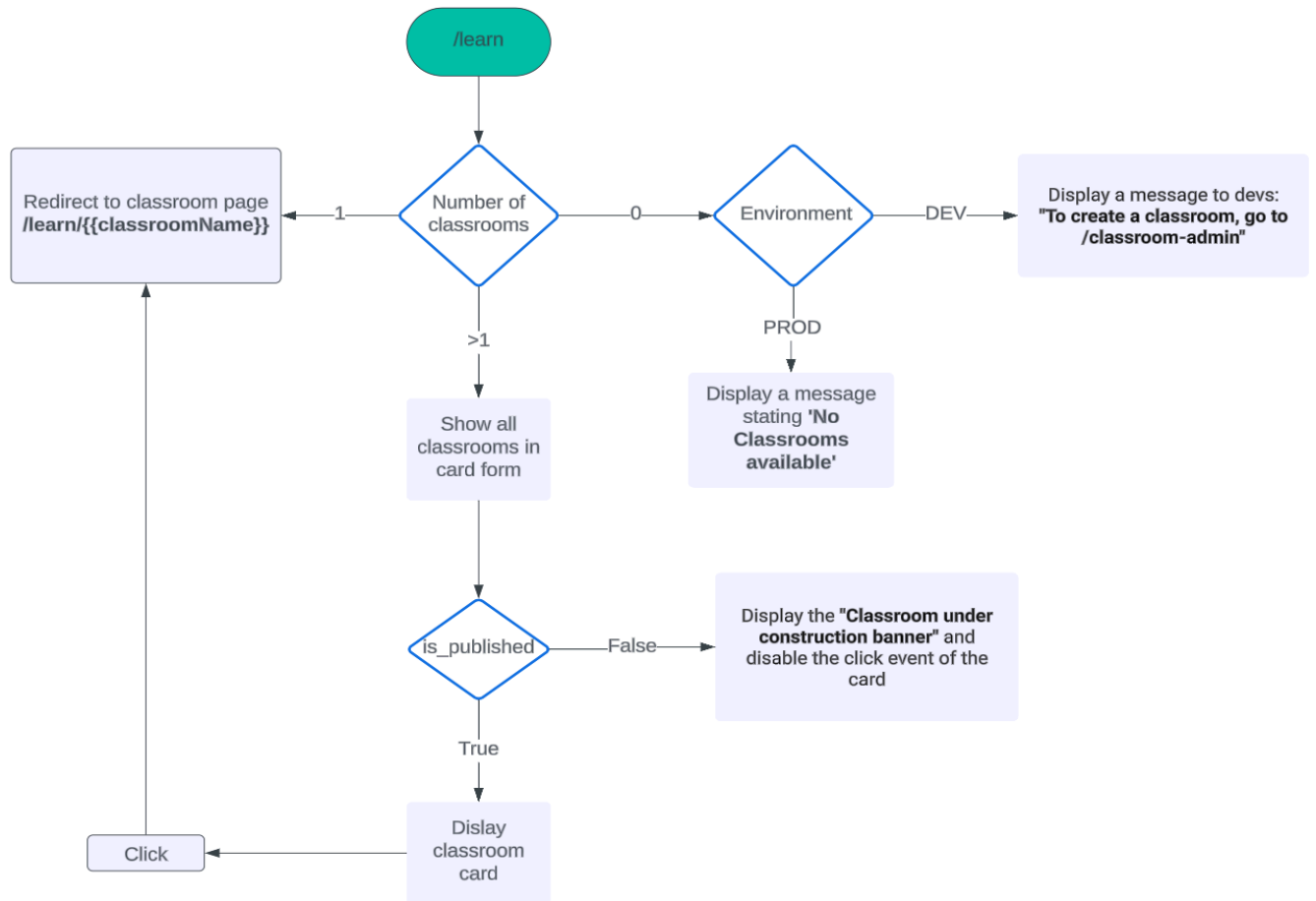
The new classroom page (located at /learn) will contain all Oppia classrooms in a card form and the card will display the classroom thumbnail, title, and description. Classrooms with hidden publication status will be shown as "Classroom under construction," and the card click action will be disabled.

→ When clicking on any classroom card, the learner will be redirected to the respective classroom page.

<p>Classroom Card (Public)</p> <p>In this card, we will show:</p> <ul style="list-style-type: none"> - Classroom Name - Course Description - Thumbnail <p>onClick() → open the classroom page /learn/{{classroomName}}</p>	 <p>Math</p> <p>Learn math through fun stories!</p>
<p>Classroom Card (Private/Hidden)</p> <p>In this card, we will show:</p> <ul style="list-style-type: none"> - Classroom Name - Course Details - Thumbnail – "Classroom Under Construction" banner <p>onClick() will be disabled for this card.</p>	 <p>English</p> <p>We are working on more classrooms just for you. Check back soon!</p>

Routing for the new /learn page and individual classroom pages.

Currently, if a user goes to the /learn page, they are redirected to the /learn/math page by default. A new ClassroomHomePage will be implemented. The routing will be handled in the following way:



The routing for individual classroom pages located at /learn/:classroom_url_fragment is already implemented. ([here](#)).

```
},
{
  (property) "ROUTE": "learn/:classroom_url_fragment"
  path: AppConstants.PAGES_REGISTERED_WITH_FRONTEND.CLASSROOM.ROUTE,
  pathMatch: 'full',
  loadChildren: () =>
    import('pages/classroom-page/classroom-page.module').then(
      m => m.ClassroomPageModule
    ),
},
},
```

The routing for the /learn route already exists, but it redirects learners to the default classroom page (math). Instead, we will render the new learn page and handle the routing in the frontend as illustrated in the above flow chart.

```
class DefaultClassroomRedirectPage(
    base.BaseHandler[Dict[str, str], Dict[str, str]]
):
    """Redirects to the default classroom page."""

    URL_PATH_ARGS_SCHEMAS: Dict[str, str] = {}
    HANDLER_ARGS_SCHEMAS: Dict[str, Dict[str, str]] = {'GET': {}}

    @acl_decorators.open_access
    def get(self) -> None:
        """Redirects to default classroom page."""
        self.redirect('/learn/%s' % constants.DEFAULT_CLASSROOM_URL_FRAGMENT)
```

(d) A new generalized classroom page

A new generalized classroom page with the same design as the current math classroom page will be implemented, and the page will be made generalizable so that all other classrooms can use it. All the pages will be fully internationalized with the help of **hacky translations** technique.

– To achieve this, we will first generalize the [getClassroomTranslationKey\(\)](#) function, which will take the following arguments:

- classroomName: string
- **property: string** - This is a new parameter

```
getClassroomTranslationKey(classroomName: string, property: string): string {
    return `I18N_CLASSROOM_${classroomName.toUpperCase()}_${property.toUpperCase()}`;
}
```

– We are adding the new **property** parameter because currently, the function is hardcoded to create only the classroom title I18N key. Now, we aim to internationalize other classroom properties as well.

We will call this function to obtain the I18n key for all the classroom data such as name, course details, and topic list intro.

`getClassroomTranslationKey("math", "name")` \longrightarrow `I18N_CLASSROOM_MATHS_NAME`

– We will store I18n keys for all properties in a JS object called "**classroomI18nKeys**" with keys representing the property and values being the I18n keys for that classroom.

```
classroomI18nKeys = {
    name: 'I18N_CLASSROOM_MATH_NAME',
    course_details: 'I18N_CLASSROOM_MATH_COURSE_DETAILS',
    topic_list_intro: 'I18N_CLASSROOM_MATH_TOPIC_LIST_INTRO'
}
```

– And we will update the [isHackyClassroomTranslationDisplayed\(\)](#) function to check whether the I18N key is present for the given property by introducing an additional property parameter, as earlier it only checked for the classroom name.

```
isHackyClassroomTranslationDisplayed(property: string): boolean {
    return (
        this.i18nLanguageCodeService.isHackyTranslationAvailable(
            this.classroomI18nKeys[property]
        ) && !this.i18nLanguageCodeService.isCurrentLanguageEnglish()
    );
}
```

– Now we can use this function to check for other properties as well. We don't need to create separate functions for each and every property.

```
<span *ngIf="!isHackyClassroomTranslationDisplayed('name')">
  {{ classroomDisplayName }}
</span>
<span *ngIf="isHackyClassroomTranslationDisplayed('name')">
  {{ classroomI18Nkeys.name | translate }}
</span>
```

(e) Learner Dashboard (The services for all the required data already exist)



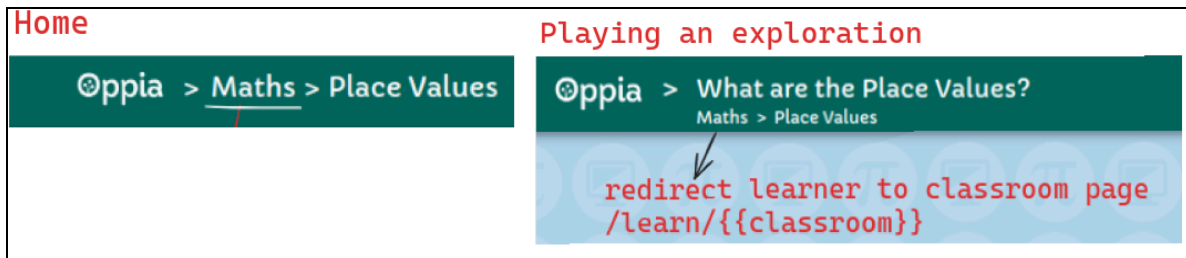
– In the case of the Learner Dashboard:

- We will show progress for all the classrooms in the In “Progress” tab for all classrooms the learner has started.
- The learner will be able to continue from where they left off.

We do not need to introduce a new service or handler for these changes – we already have those. The only change we will need to make is in the data handlers.

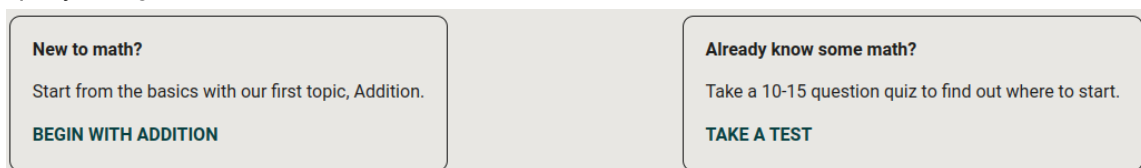
(f) Topic Viewer Page

Learners will be able to see the classroom name in the topic viewer page via the breadcrumbs, and can navigate to the classroom page by clicking on it.



(f) Diagnostic test for multiple classrooms

The diagnostic test feature is used to suggest topics to learners for a classroom. Currently, we have hardcoded it for the math classroom only. To make it work for other classrooms, we will need to know for which classroom we have to fetch the questions. This can be easily done if we pass the classroom name as a query string when learners click on the "TAKE A TEST" button from a classroom.



Updating the 'TAKE A TEST' link to indicate for which classroom we need to run the test.

```
<a [routerLink]="/diagnostic-test-player" [queryParams]="{cuf: classroomUrlFragment}"
  class="e2e-test-take-diagnostic-test">
  <span class="oppia-classroom-page-tile-button-text">
    <strong>{{ 'I18N_CLASSROOM_PAGE_TAKE_A_TEST_BUTTON' | translate }}</strong>
  </span>
</a>
```

Currently, it's hardcoded

```
classroomUrlFragment: string = 'math';
```



Getting the classroomUrlFragment from URL

```
this.route.queryParams.subscribe(params => {
  this.classroomUrlFragment = params['classroom'];
});
```

Getting the classroom ID for the given classroom URL fragment.

This call is made in `ngOnInit()`, which is used to obtain the `classroomId` corresponding to the `classroomUrlFragment`.

```
this.getProgressText();
this.classroomBackendApiService
  .getClassroomIdAsync(this.classroomUrlFragment)
  .then(classroomId => {
    this.classroomId = classroomId;
  });
```

– We don't need to change anything in the other services related to diagnostic tests; they work on the topic provided when starting the diagnostic test.

```

startDiagnosticTest(): void {
  this.classroomBackendApiService
    .getClassroomDataAsync(this.classroomId)
    .then(response => {
      this.diagnosticTestTopicTrackerModel =
        new DiagnosticTestTopicTrackerModel(
          response.classroomDict.topicIdToPrerequisiteTopicIds
        );
      this.diagnosticTestIsStarted = true;
    });
}

```

The `startDiagnosticTest()` function is called when the user clicks on the 'Start Test' button. The response data, which is `topicIdToPrerequisiteTopicIds`, is used to track topic IDs and suggest the next question according to the user's answer.

Third-Party Libraries

No new third-party libraries are required.

"Service" Dependencies

There are no new dependencies on other services.

Impact on Other Oppia Teams

The LaCE team will be responsible for maintaining this feature and will work on further improving it. Additionally, the Android Team will need to replicate the same feature for Oppia android users. The marketing team will play a crucial role in promoting the new classrooms.

Key High-Level and Architectural Decisions

Most parts of the project will be built more or less on top of the existing flow, and there won't be much implementation of new flows or changes to the existing one, except for finding the relation between classrooms and topics. Additionally, topics will be added to classrooms using topic names instead of topic IDs.

Decision 1: Finding the relation between the classroom and the topic

We need to find the relationship between classrooms and topics for the following purposes:

- (a) In the topic and skill dashboard to display which topics are assigned to which classroom, and
- (b) In the classroom admin page to show a list of topics available to assign to a classroom (topics which are not yet assigned to any classroom).
- (c) Showing the classroom name in the topic editor.
- (d) Showing the classroom name in the topic viewer.

To find the relation between the topic and the classroom, i.e., to know which topic is assigned to which classroom, we have considered the following alternatives:

Option 1: Updating the existing [TopicModel](#).

Introduce a new field named **classroom_id (string)** in TopicModel, which will store the classroom id to which the current topic is assigned; otherwise, it will be null.

In the [TopicsAndSkillsDashboardPageDataHandler](#), we can avoid calling the `get_all_classrooms()` function, which returns all the classroom models. Thus, we wouldn't need to loop over the classroom and `topic_name_to_prerequisite_topic_names` fields. This makes it more optimal for a large number of classrooms. As we will only query a single `get()` to retrieve the classroom details of the classroom with the given `classroom_id`.

Option 2: Using the [ClassroomModel's](#) `topic_id_to_prerequisite_topic_ids` field.

The current implementation already covers this functionality, so there's no need to make changes to the storage model or create new endpoints. However, we can optimize the current implementation of finding the relationship between classrooms and topics.

- Currently, we have a dict called `topic_classroom_dict` ([Code](#)), In which we will store some topic and classroom data.
- We iterate over each classroom's `topic_id_to_prerequisite_topic_ids` field and store them in `topic_classroom_dict`. ([Code](#))
- Then, we iterate over `topic_rights_dicts` ([code](#)) to check whether the topic is assigned to any classroom or not using the `topic_classroom_dict`. This method is not very optimal because we are iterating first over classrooms and their `topic_id_to_prerequisite_topic_ids` and then through all the topic summary list.

```
classrooms = classroom_config_services.get_all_classrooms()
all_classroom_names = [
    classroom.name for classroom in classrooms]

topic_classroom_dict = {}
for classroom in classrooms:
    for topic_id in classroom.get_topic_ids():
        topic_classroom_dict[topic_id] = classroom.name

for topic_summary_dict in topic_summary_dicts:
    topic_summary_dict['classroom'] = topic_classroom_dict.get(
        topic_summary_dict['id'], None)
```

Option 3: Introducing a new **TopicClassroomLinkModel**.

To find the relation between classroom and topic, we can create a new model that will store key topic and classroom data. We will only store those topics which are assigned to a classroom. The model will look something like this:

```
TopicClassroomLinkModel : {
    topic_id (str)
    classroom_id (str)
}
```

This model will be updated when adding a topic to a classroom or when removing a topic from a classroom. The data from this model can be fetched using `topic_id` or `classroom_id`.

Among these, we believe that **Option 2** is the best approach, because:

- Fewer datastore calls are made for all cases a, b, c, d, and e.
- And especially for purposes (b) and (c), option 2 is more optimal, as those pages will be visited more frequently by learners and curriculum admins. This reduces the number of queries because in option 2 we are only making an extra `get_all()` query.
- Although we need to iterate over the classroom's topic list to find the relation, the number of classrooms will only be 4 in the near future, so the total number of iterations is negligible.

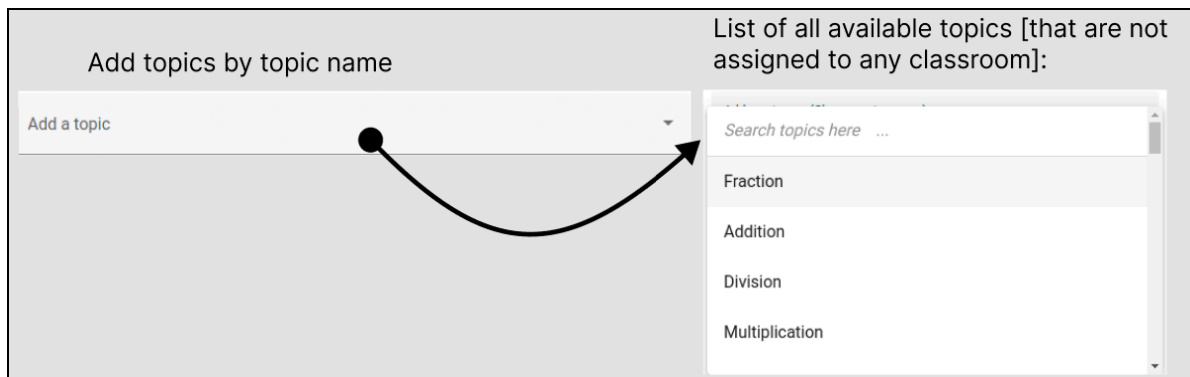
The above approaches are contrasted in detail in the following table:

Factor		Option 1: Updating the existing TopicModel	Option 2: Using the ClassroomModel's <code>topic_id_to_prerequisite_topic_ids</code> field	Option3: New TopicClassroomLinkModel
Additional fields to be included		<code>classroom_id</code> : (string)	No new fields are required.	This is a new model with 2 fields (<code>topic_id</code> and <code>classroom_id</code>).
Implementation complexity		Medium	Low	Medium
Datastore calls for finding the assigned classroom.	(a) Topic and skill dashboard (TopicsAndSkillsDashboardPageDataHandler)	One <code>get_multi()</code> query to ClassroomModel to retrieve classroom data for each topic, as we are storing only the id and not the classroom data itself.	One <code>get_all()</code> query to ClassroomModel to retrieve classroom data and iterate to find the relation.	One <code>get_multi()</code> call to TopicClassroomLinkModel to retrieve classroom id for the given topics. + One <code>get_multi()</code> call to ClassroomModel to get the classroom data.
	(b) Topic Editor Page (EditableTopicDataHandler)	One additional <code>get_by_id()</code> query to ClassroomModel is needed.	Same as above.	One additional <code>get_by_id()</code> query to TopicClassroomLinkModel to get the <code>classroom_id</code> corresponding to the given <code>topic_id</code> . + One <code>get_by_id()</code> query to ClassroomModel to get classroom name and url fragment.
	(c) Topic Viewer	Same as above.	Same as above.	Same as above.

	(TopicPageDataHandler)			
(d) Datastore calls for adding a new topic to the classroom.	2 put() calls: (1) for updating the TopicModel and (2) updating the classroom model.	1 put() call just for updating the classroom model.	2 put() calls: (1) for updating the TopicClassroomLinkModel and (2) updating the classroom model.	
(e) Datastore calls for adding/removing a topic from a classroom.	Same as above.	Same as above.	Same as above.	
Time complexity (Only considering additional computation) for all cases (a, b,c and d)	No computation is required.	Let n be the total number of classrooms, m be the average number of topics in each classroom, and k be the total number of topics. $O(N*M + K) \rightarrow O(N*M)$	No computation is required.	

Decision 2: Adding topics by name instead of ID.

Currently, topics are added to classrooms by pasting the topicID, which involves a lot of back and forth and does not provide the best experience. Hence, to improve this, we have decided to implement a feature to allow curriculum admins to add topics by topic name, giving the admins a list of available published topics that are not assigned to any classroom in a searchable dropdown.



– We will be making a GET call to the newly created endpoint [/all_published_topics_classroom_info_handler](#) Which will return a list of all published topics and their assigned classroom details (classroom_name and url_fragment) in the form of following dict.

- topic_name (string)
- topic_id (string)
- classroom_name (string)
- url_fragment (string)

- This data will also be used to translate the topic_id of topic_id_to_prerequisite_topic_ids field to the topic name.
- The topic_name dropdown will be used to show a list of topics.
- When the admin clicks on any topic, its corresponding topic_id will be added to the classroom's topic_id_to_prerequisite_topic_ids.

Risks and mitigations

Potential Risk	Mitigation
Dependency on the new learner dashboard.	To mitigate this, we can implement all other features except the learner dashboard part first. If the learner dashboard is not ready by the end of the project, we can incorporate the multiple classroom features into the existing learner dashboard. Later on, we can change it when the new page is ready.

Implementation Approach

Storage Model Layer Changes

We will be updating the existing ClassroomModel and adding a new TopicClassroomLinkModel model.

- **[ClassroomModel](#)** As curriculum admins will now be able to update the classroom's banner, thumbnail, and publication status, these fields need to be added.
 - teaser_text (StringProperty, required=True)
 - thumbnail_filename (StringProperty)
 - thumbnail_bg_color (StringProperty)
 - banner_filename (StringProperty)
 - banner_bg_color (StringProperty)
 - is_published (BooleanProperty) (default: False)

Storage Model Migrations

N/A

Domain Objects

Backend

- **[Classroom](#) and [ClassroomDict](#)** (classroom_config_domain.py)
The following new fields will be added:
 - teaser_text (str)
 - thumbnail_filename (str)
 - thumbnail_bg_color (str)
 - banner_filename (str)
 - banner_bg_color (str)
 - is_published (bool)

- For [CreatorTopicSummary](#) and [FrontendTopicSummaryDict](#) We will rename 'classroom' to 'classroom_name' as we are not storing the whole classroom data and change the type from string | undefined to string | null and update the getter functions type too. We will add the following new field also.
 - classroom_url_fragment (string | null)

Frontend

- [ClassroomData](#) (classroom-data.model.ts)
The following new fields will be added, along with getter functions for them:
 - _thumbnailFilename (string)
 - _thumbnailBgColor (string)
 - _bannerFilename (string)
 - _bannerBgColor (string)
 - _isPublished (string)
 - _teaserText(string)

New Functions to be added:

Function Signature	Need
getThumbnailFilename(): string	To retrieve the thumbnail filename of the classroom.
getBannerFilename(): string	To retrieve the banner filename of the classroom.
isPublished(): boolean	To check whether the current classroom is published or not.

- [ExistingClassroomData](#) and [NewClassroomData](#) are used for updating and creating a new classroom.

ExistingClassroomData class will have the following additional fields:

- _thumbnail_data (ImageData)
- _banner_data (ImageData)
- _is_published (string)
- _course_details (string)
- _topic_list_intro (string)
- _teaser_text (string)
- New Functions to be added:

Function Signature	Need
changePublicationStatus(status: boolean): void	This will be used to update the publication status of the classroom when editing it.
isPublished(): boolean	To check whether the current classroom is published or not.
getClassroomDetailsValidationErrors(): string	This will be used to validate the classroom's course_details property and show an error message to the user.

getClassroomTopicListIntroValidationErrors(): string	Same as above but for validating topic_list_intro
getClassroomTeaserTextValidationErrors(): string	Same as above but for validating teaser_text
getClassroomThumbnailValidationErrors(): string	Same as above but for validating thumbnail_filename
getClassroomBannerValidationErrors(): string	Same as above but for validating banner_filename
setThumbnailData(thumbnailData: ImageData): void	Updates _thumbnail_data property
setBannerData(thumbnailData: ImageData): void	Updates _banner_data property

- [ClassroomBackendDict](#), [ClassroomDict](#) and [ClassroomDataBackendDict](#)

(classroom-backend-api.service.ts)

- thumbnail_data: ImageData
- banner_data: ImageData
- is_published (string)
- teaser_text (string)

- [CreatorTopicSummary](#) (creator-topic-summary.model.ts)

The following new fields will be added, along with getter functions for it:

- classroom_name (string | null)
- classroom_url_fragment (string | null)
- New Functions to be added:

Function Signature	Need
getClassroomName(): string null	To retrieve the name of the classroom to which the topic is assigned, if not assigned it will return null.
getClassroomUrlFragment(): string null	To retrieve the url_fragment of the classroom to which the topic is assigned, if not assigned it will return null.

- TopicClassroomInfoDict

- topic_name (string)
- topic_id (string)
- classroom_name (string | null)
- classroom_url_fragment (string | null)

User Flows (Controllers and Services)

New Controllers

- **AllClassroomsSummaryHandler** (*classroom.py*)
 - It will be used to retrieve a summary of all classrooms, including their title,url_fragment, teaser text, thumbnail filename, and publication status. This data will be used to display classroom cards on the new classrooms page.
 - Method: GET
 - Endpoint: /all_classroom_summary
 - Args: classroom_url_fragment (string)
 - It will return a list of classroomSummaryDict, which contains the following properties:
 - name (string)
 - url_fragment (string)
 - teaser_text (string)
 - thumbnail_filename (string)
 - thumbnail_bg_color (string)
 - is_published (boolean)
- **TopicsToClassroomsRelationHandler**(*classroom.py*)
 - It will be used to retrieve a list of all the published topics and their assigned classroom data.
 - Method: GET
 - Endpoint: /topics_to_classrooms_relation
 - It will return a list of all topics and their assigned classrooms data in the form of [TopicClassroomInfoDict](#)
- **NewClassroomHandler** (*classroom.py*)
 - It will be used to create a new classroom.
 - Body:
 - name (string)
 - url_fragment (string)
 - Method: POST
 - Endpoint: /classroom-admin/create_new

ACL Decorators

- **can_access_classroom_page:** As we will have hidden classrooms, we're introducing a new can_access_classroom_page decorator. It grants access to everyone if the classroom is published. However, if it's not, access will only be granted to curriculum admins and super admins.

Existing Controllers

- [ClassroomAccessValidationHandler](#)
 - This handler is used to validate the request to the classroom page. (eg - /learn/math)
 - Method: GET
 - Args: classroom_url_fragment (string)

- acl decorator: **can_access_classroom_page**
- ClassroomHandler (PUT)
 - Add following checks:
 - A topic can only be assigned to one classroom

NOTE: I have listed all endpoints and the respective handler code references [here](#). In this "Existing Handlers" section, I will discuss the changes necessary to accommodate classroom creation/updation and the changes needed to establish relationships between topics and classrooms.

1. Finding the assigned classroom of a topic.

(b) Topic Editor

The GET call is handled by the get() method of [EditableTopicDataHandler](#). To show the classroom name on the topic editor page, we will use the method given [here](#).

(c) Topic Viewer Page

Handler Used: [TopicPageDataHandler](#)

– To get the classroom name and url_fragment we will use the same method used in the topic editor page.

(a) Topic and Skill dashboard

A GET call is made from the [fetchDashboardDataAsync\(\)](#) to [TopicsAndSkillsDashboardPageDataHandler](#)

→ This handler's GET method is used for fetching the topic data in the topic and skill dashboard. It returns a list of all the topics in the form of topic_summary_dicts. Each member of this dict is of type FrontendTopicSummaryDict, which includes a classroom field. To get the classroom name we will use the same method used in the topic editor page.

1. Handling classroom creation and updation

ClassroomHandler – It handles classroom creation and updating. It takes classroom_dict as an argument and transfers control to *classroom_config_services.py* for the creation and updating of the classroom. If a classroom with the given url_fragment exists, it will call the create_new_classroom(classroom) function; otherwise, it will call the update_classroom(classroom) function.

- The update_classroom() function currently takes two arguments: (1) classroom data and (2) instance of ClassroomModel. This is not preferred, so we will update it to only take the updated classroom data.

(a) Handling image uploads

– To handle image uploads, we will be using the updated **oppia-image-uploader** component. When creating and updating a classroom, we will take image blob data from the client and upload it directly from the server.

The file path for the saved images looks like this:

assetsdevhandler/<entity-type>/<entity-id>/assets/image/<filename>

Currently, we don't handle the image upload for classrooms, so we need to define 'entity-type' and 'entity-id' in order to access and save images to the server.

EntityType	classroom
EntityId	classroom_id

For pages like topic-editor and exploration player, we can obtain the entityType and entityId from the URL as the topicId/explorationId is part of the URL. But in the case of the classroom, we don't have a separate editor page so we will use the setCustomEntityContext() function to set the entityType and entityId whenever the user clicks on the classroom card in the curriculum admin.

(1) When creating a new classroom

When creating a new classroom, we will use the oppia-image-uploader component which will return the cropped image blob data. Afterward, we'll send the image blob data to the server, which will take care of saving the images.

```
<oppia-thumbnail-uploader [aspectRatio]='4:3'
                          [disabled]='false'
                          [useLocalStorage]='true'
                          [allowedBgColors]='allowedBgColors'
                          previewDescriptionBgColor="#2F6687">
</oppia-thumbnail-uploader>
```

Saving image

```
entity_id = classroom_id
filename_prefix = 'thumbnail'

image_is_compressible = (
    file_format in feconf.COMPRESSIBLE_IMAGE_FORMATS)
fs_services.save_original_and_compressed_versions_of_image(
    thumbnail_filename, feconf.ENTITY_TYPE_CLASSROOM, entity_id, raw_image,
    filename_prefix, image_is_compressible)
```

The filename is generated by the [generateImageFilename\(\)](#) function.

(2) When updating a classroom

When updating a classroom, we need to display the images that have already been uploaded to admins. For image preview, the oppia-image-uploader component calls [getTrustedResourceUrlForThumbnailFilename\(\)](#), which takes three parameters: filename, entityType, and entityId. We can get the filename from the classroom data and the method to get the entityType and entityId is defined [here](#).

All details of the updated oppia-image-uploader component are given [here](#).

Services

The topic-related service will not require changes. We only need to append the classroom_name and classroom_url_fragment properties to frontend dicts and make some frontend adjustments to display the breadcrumb, show classroom names, and allow learners to redirect to them.

And as the major changes are required for classroom creation, I have discussed those, while other services will remain more or less the same – we will just need to extend them slightly to accommodate new properties.

Classroom-related services

classroom-backend-api.service.ts

This service is used for fetching the classroom data. The existing methods returning dictionaries will be updated to accommodate the changes, and the following new methods will be created:

- **getTopicsToClassroomsMapping(): TopicClassroomInfoDict[]**
 - This function will be used to retrieve the list of all published topics in the form of a list of [TopicClassroomInfoDicts](#).
 - The result will be used to display the available topic list to admins in a searchable dropdown, allowing them to select and add topics to classrooms.
 - Sorting Order: lexicographical order (Dictionary order) by topic_name field.
- **createNewClassroomAsync(classroomId: string, classroomDict: ClassroomBackendDict, thumbnail_image: blob, banner_image: blob):**
 - This function will be the same as the **updateClassroomDataAsync**, with the only difference being that when creating a classroom, we will call this function with image blob data. When updating the classroom, we will use **updateClassroomDataAsync**, as we don't need to provide blob data; we will only provide filenames and update them in the backend.

Other classroom properties will be updated similarly to how the topic_id_to_prerequisite_ids, course description, topic list intro, and title are currently handled. This will be done using the **createNewClassroomAsync()** and **updateClassroomDataAsync()** functions.

classroom-admin-data.service.ts

This service handles the validation for classroom fields. Currently, it only handles the validation for the classroom name, URL fragment, and topics relation. We will be adding the following validations to it:

- New variables: **thumbnailValidationError**, **bannerValidationError**, **courseDetailsValidationError**, **topicListIntroValidationError**, **topicValidationError** and **teaserTextValidationError**.
- These variables will store an error message (if any); otherwise, they will be null.

Function	Validation
courseDetailsValidationError, topicListIntroValidationError and teaserTextValidationError	We will set a limit for the maximum number of characters allowed, and we will throw the corresponding error message if these limits are not met. <ul style="list-style-type: none">● Course details – 720 characters● Topic list intro – 240 characters● Teaser text - 68 characters
thumbnailValidationError and bannerValidationError	The thumbnail_filename and banner_filename should not be empty.

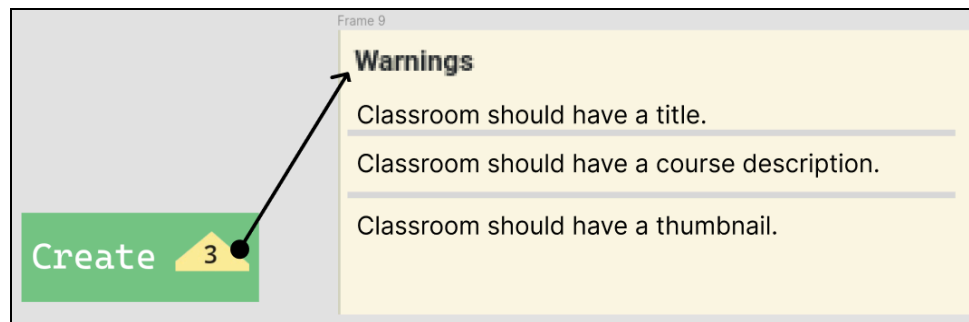
topicValidationError	A classroom should have at least one topic assigned, and we can assign 20 topics at most to it.
----------------------	---

- These validation flags will be handled by the **validateClassroom()** function. All error messages will be shown in a consolidated view, and the save/create button will be disabled if the length of the issues array is greater than 0.

Web frontend changes

When updating a classroom we will add two oppia-image-uploader components: one for the classroom thumbnail and the other for the banner. Additionally, we will implement a searchable dropdown (mat-select component) to display the topic list for adding to the classroom. The dropdown will follow the same design as other pages, such as the exploration editor, for selecting categories.

The save button will be disabled until all errors are fixed, and these errors will be shown in a consolidated view when the user hovers over the 'caution' icon on the save/create button.



oppia-image-uploader [Fix issue [#20303](#)]

Currently, we have two main components: oppia-thumbnail-uploader and oppia-image-uploader. We will extend the functionality of the oppia-image-uploader component so that it can be used in all cases by providing the necessary props. This will allow us to remove the oppia-thumbnail-uploader component from the codebase.

The component will be divided in 3 parts:

- **oppia-image-receiver:** This is the component which handles taking images from the user and listens for drag events.
- **oppia-image-uploader-modal:** This component shows the uploaded image, handles cropping, and shows the preview card after cropping (in the case of a topic, story, or subtopic).
- **oppia-image-uploader:** This is the highest level component which takes all the props and shows the preview, and handles all the events.

Instead of uploading the image from the component, the (3) component will return the final cropped image data, which we will pass when creating and updating. This will save us from making unnecessary calls.

New Classrooms Page (at /learn)

The page will display all the classrooms in card form, each containing the classroom title, course description, and thumbnail. For this, we will create a **<classroom-summary-tile>** component that will take the following props:

- classroomSummary: {
 - classroom_id (string)
 - name (string)
 - url_fragment (string)
 - teaser_text (string)
 - thumbnail_filename (string)
 - thumbnail_bg_color (string)}

– Upon clicking on the card, the user will be redirected to the respective classroom page using the passed url_fragment prop.

Documentation changes

No changes are required.

Metrics Plan

Event (see PRD)	Event parameters (see PRD)	Do we already record the event + parameters? <ul style="list-style-type: none">• If so, please link to the corresponding code on GitHub.• If not, describe the changes needed to do so.
1(a): Learner clicks on "Science Classroom" link in the Oppia Classrooms page	N/A	<p>We will need to create three new functions for these events</p> <ul style="list-style-type: none">(a) registerClickClassroomsPageClassroomButtonEvent()(b) registerClickCommunityLibraryClassroomButtonEvent()(c) registerClickNavigationBarClassroomButtonEvent <p>This function will take an argument classroomName (string)</p> <p>The functions eventCategory will be (a) 'CommunityLibraryClassroomButton' (b) CommunityLibraryClassroomButton and for (c) NavigationBarClassroomButton</p> <p>The event will trigger a 'click' event for all three, with the event_label being the classroom name. The event_category will be based on the source: the Navigation bar, community library, or classrooms page.</p> <p><i>Example code:</i></p> <pre>registerClickCommunityLibraryClassroomButtonEvent(classroom Name: string): void { const eventParameters = { event_category: 'CommunityLibraryClassroomButton', event_label: classroomName }; this._sendEventToGoogleAnalytics('click', eventParameters);</pre>
1(b): Learner clicks on "Science Classroom" from navigation bar		
1(c): Learner clicks on "Science Classroom" from Community library		

		}
2. Learner lands on the Science classroom page	User's analytics ID (to determine whether the visit is unique)	This event (unique page visit) is automatically handled by the Google Analytics service.
3. Learner clicks on an in-progress science lesson card present in the progress tab of learner dashboard and gets directed to the science lesson content	N/A	We will need to create a new function similar to registerClassroomLessonEngagedWithEvent(args) . We will call this function <code>registerInProgressClassroomLessonEngagedWithEvent(args)</code> , and it will be invoked when the learner clicks on any lesson card from the "In progress" tab of the learner dashboard.
4. Learner clicks on a new (not yet started) science lesson card present in the home page of learner dashboard and gets directed to the science lesson content	N/A	Yes we already record this event registerClassroomLessonEngagedWithEvent(args) We will call this function when the learner starts the lesson from home tab of learner dashboard (new lesson)

Testing Plan

Acceptance tests for core user flows

Note: The acceptance test will be written for both desktop and mobile views.

Dummy Topic: A dummy topic will have 2 skills, each with 3 questions, and it will consist of 3 chapters, each containing 1 exploration with 3 questions/statements.

Dummy Classroom: A dummy classroom will have 2 dummy topics and other properties such as name, URL fragment, course details, topic list intro, banner, and thumbnail.

1. Curriculum Admin User Journey

As a curriculum admin user, I should be able to create a classroom, add topics to it, and change the publication status of the classroom.

#	Test name	Initial setup steps	Steps	Expectations
1.	Creating and updating a new	1. Sign up a new user 'testLearner' with email	1. As classroomCreator, go	1. The number of classrooms should be

	classroom	<p>'test_user@example.com'</p> <p>2. Create a user 'classroomCreator', with email 'classroomcreator@example.com' who has curriculum admin permissions.</p> <p>3. Create a Topic 'Test topic' with 'Test subtopic', description as 'Test Description'. Add sub topic and skill and publish it</p> <p>4. Copy the topic id from URL</p>	to the /classroom-admin page.	zero.
			<p>2. Click on "Create new classroom," and enter classroom details:</p> <p>Title: 'Test Classroom'</p> <p>Url fragment: 'testclassroom'</p> <p>Course description: 'Test description'</p> <p>Topic list intro: 'Test topic list intro'</p> <p>Upload classroom thumbnail and banner</p> <p>Click on the add topic search bar and select 'Test topic'.</p>	<p>2. The classroom creation modal should open.</p> <p>In the topic dropdown, 'Test topic' should be present.</p>
			3. Click on the "Create" button.	<p>3. The classroom creation modal should close after clicking on the "Create" button.</p> <p>In the curriculum admin page, the number of classrooms should be one.</p>
			4. Click on the 'Test Classroom' tile.	<p>4. The classroom details should be correct, and the default publication status should be hidden. There should be a test topic present in the classroom.</p>
			5. As testLearner visit /learn/testclassroom	5. A 404 error page is seen.

			6. As the person who created the classroom, click on the 'Test Classroom' tile. Change the publication status from 'hidden' to 'public'. Then click on "Save", and afterward, click on the 'Test Classroom' tile again.	6. The value of the visibility field should be set to public.
			7. As the testLearner, go to '/learn/testclassroom' again.	7. The classroom page should load successfully with all the classroom details.

2. Logged-out Learner User Journey

As a logged-out learner, I should be able to access classrooms from all available sources such as the splash page, community library, and the classrooms page (at /learn).

#	Test name	Initial setup steps	Steps	Expectations
1.	Classroom Index Page Tests	<p>Create a user 'classroomCreator', with email 'classroomcreator@example.com' who has curriculum admin permissions.</p> <p>Create another user, testLearner, with the email learner@testLearner.com.</p>	(0 classrooms) 1. As the test learner, go to the /learn page.	1. A message should be visible saying "No classrooms are available". NOTE: This message might change; it is here just as a placeholder.
			(1 classroom) 2. As the classroomCreator, create a dummy math classroom and visit the /learn page as the testLearner.	2. Learners should be redirected to the math classroom page, i.e., /learn/math in this case.
			(2 classrooms) 3. As the classroomCreator, create a dummy science classroom and visit the /learn page as the testLearner.	3. Learners should stay on the learn page and be able to see the created math and science classroom cards. Verify that the math and science classroom cards are displayed with the correct title and description.

2.	Tests for classroom navigation (a) Community library page (b) Splash page (c) Navigation bar	1. Create a user classroomCreator, with email classroomcreator@example.com who has curriculum admin permissions. 2. Create another user, testLearner, with the email learner@testLearner.com. <u>Populating datastore</u> – Login as classroomCreator. – Create dummy math classrooms.	(a) Community library page	1. Learners should be able to see math classroom card.
			1. As testLearner, go to the community library page. 2. As classroomCreator create a science classroom and as testLearner refresh the community library page.	Desktop: 2. Learners should be able to see both math and science classroom cards. Mobile: 2. Learners should see a single math classroom card and carousel buttons and be able to change to the science classroom card by clicking on the next carousel button.
			3. As classroomCreator create a history classroom and as testLearner refresh the community library page.	Desktop: 3. Learners should be able to see math and science classroom cards and be able to view the history classroom card by clicking on the next carousel button. Mobile: Learners should see a single math classroom card and carousel buttons and be able to change to the science and history classroom cards by clicking on the next carousel button.
			4. Click on the science classroom card.	5. The science classroom page should be loaded with the classroom name, course details, topic list introduction, and all the topic cards.

			5. Return to the community library page.	5. Learners should be able to see math and science classrooms in a carousel component.
			6. Click on the math classroom card.	6. The math classroom page should be loaded with the classroom name, course details, topic list introduction, and all the topic cards.
			(b) Splash page 7. Go to the splash page as the test learner and open the navigation tab. Click on the "Classrooms" link.	7. Learners should be able to see math and science classrooms in a carousel component.
			(c) Navigation bar 6. Hover on the "Learn" button in the navigation bar.	6. I should see Math and science classroom cards.
			7. Click on the math classroom card in the navbar's learn dropdown.	7. Math classroom page should open.

3. Logged-in Learner User Journey

As a logged-in learner, we should be able to access classrooms and their topics from the learner dashboard. And, we should be able to track the progress within the topics and return to the classroom by clicking on the classroom name present in the breadcrumb.

#	Test name	Initial setup steps	Steps	Expectations
1.	Learner dashboard tests	1. Create a user 'classroomCreator', with email 'classroomcreator@example.com' who has curriculum admin permissions. 2. Create another user, testLearner, with the email learner@testLearner.co	1. Go to the learner dashboard.	1. Learners should be able to see the math and science classroom topics in the Home tab of the learner dashboard.
			2. Go to the "Progress" tab.	2. The "Progress" tab should be empty.
			3. Click on the topic with the title 'Test Topic' under the math	3. The topic player page should open and a 'Math' classroom link should be

		m. 3. Create dummy math and science classrooms.	classroom section.	present in the breadcrumb. Mobile: Instead of Math we should see "Back to Math" link.
			4. Click on the 'math' classroom link present in the breadcrumb.	4. The Math classroom page should open.
			5. Go to the learner dashboard and click on the topic with the title 'Test Topic' under the math classroom section and play it, completing some checkpoints.	5. The topic player should open, and the learner should be able to play the topic (complete chapter 1).
			6. Return to the learner dashboard and open the "Progress" tab.	6. Learners' progress should be saved i.e. the "Test Topic" card should be present in the "Progress" tab and it should be 33% completed.
			7. Resume the 'Test Topic' and complete chapter 2. Then go to the "Progress" tab again.	7. The progress of "Test Topic" should be increased to 66%.

Implementation Plan

Adding feature flags for the new classrooms page (/learn) and the generalized classroom page.

We will have a single feature flag with the name "**ENABLE_MULTIPLE_CLASSROOMS**," which will be used to hide the new classrooms page and the carousel present in the community library page. It makes sense to show the carousel and the classrooms page only if we have more than one classroom.

Launch Process

As the existing math classroom models don't have the "teaser_text," "banner_filename", "thumbnail_filename", and "is_published" fields, we will show the default images we currently use to avoid breaking the UI on the classroom page and the classroom card. And, we will add "teaser_text" in English in en.json for the math classroom only.

Since the math classroom is published but a new is_published field is introduced, which will be null for the math classroom, we will need to return the following value in the is_published field for the math classroom so the flow doesn't break.

- **[Classroom admin page]:** We will add custom logic to return false for the math classroom as we need to change the publication status from hidden to true. I.e., we need to return true in the is_published field from client to server.
- **[All other pages]:** We will return true in the is_published field as the math classroom is complete and published, and we do want to show it to learners.
- **3rd Week of August:** The server admin will follow the given steps to update the existing math classroom. [Instruction for server admin](#)
 - As curriculum admin go to /classroom-admin page
 - Click on the math classroom tile
 - Change the publication status from 'hidden' to 'public'
 - Upload the math classroom thumbnail image and banner image
 - Click "Save"
- Right after the release, we will make a cleanup PR to remove the custom logic previously added for the math classroom, as the data is now populated.

Milestone Table (include both PRs and other actions that need to be taken prior to launch)

Community Bonding Period

No.	Description of PR / action	Prereq PR numbers	Target date for PR creation	Target date for PR to be merged
1	Extend the functionality of the oppia-image-uploader component and remove oppia-thumbnail-uploader #20303 .	–	22-05-24	31-05-24

Milestone 1

Key objective for this milestone: The objective for this milestone will be to implement the classroom creation feature, implement the classroom-topic relation logic, create a new classrooms page (at /learn), generalize the classroom page, and finally migrate the math classroom to use it.

- Curriculum admins will be able to create and update classrooms directly from the website, including all their properties. Additionally, they will be able to add topics by name instead of by ID.
- Curriculum admins will be able to update the publication status (hidden/public) of a classroom from the classroom-admin page.
- Topic editors and curriculum admins will be able to see which classroom is assigned to each topic in the topic and skill dashboard and topic editor pages. They will also be able to filter topics based on classrooms and the default sorting order will be "Most Recently Updated".

- A new classrooms page will be implemented at /learn, and the routing will be updated to remove any direct reference to /learn/math.
- The classroom page will be generalized so that all classrooms can use it dynamically without any hardcoding, and the page will be fully internationalized.
- Migrate the math page to use the updated classroom page, and remove the hardcoding for /learn/math.

No.	Description of PR / action	Prereq PR numbers	Target date for PR creation	Target date for PR to be merged
1	Add ENABLE_MULTIPLE_CLASSROOMS feature flag.	–	28-05-24	30-05-24
2	Update the backend handler and the storage model for ClassroomModel. and frontend services and modify the UI for classroom creation to allow image uploads and add other new fields.	CBP 1	06-06-24	16-06-24
3	Write acceptance test CUJ 1.1	2	24-06-24	27-06-24
4	Modify topic handlers (topic & skill dashboard and topic-editor) to include assigned classroom info on respective pages, and update classroom filters.	–	05-06-24	12-06-24
5	[Gated by ENABLE_MULTIPLE_CLASSROOMS] Create the new classrooms page and update the routing accordingly.	1	20-06-24	23-06-24
6	Write acceptance test CUJ 2.1	5	24-06-24	27-06-24
7	Update and generalize the classroom page.	–	21-06-24	23-06-24
–	Meeting with PM: Full demo for M1	1, 2, 3, 4, 5, 6, 7	24-06-24	–

Before proceeding to milestone two, I'd like to schedule a meeting with the PM to discuss any required changes up to this point. If necessary, I will incorporate those changes first before starting the milestone.

Milestone 2

Key objective for this milestone: The main objective of this milestone will be to make the multiple classroom feature available to learners and to launch the science classroom.

- Update the splash page and navigation bar to point to the new classroom index page
- Update the community library and learner dashboard to point to the new classroom pages
- In the topic viewer page, display the classroom to which the topic is assigned in a breadcrumb, and learners will be able to navigate to that classroom page.

- Implement the necessary analytics events for classroom usage

– **Prerequisite:** Working with the PM and launching the Science classroom.

No.	Description of PR / action	Prereq PR numbers	Target date for PR creation	Target date for PR to be merged
1	[Gated by ENABLE_MULTIPLE_CLASSROOMS] Update the community library page to point to new classrooms by implementing a carousel and creating cards for classrooms.	–	12-07-24	15-07-24
2	Update the topic viewer page data handler and the topic viewer page to show to which classroom the topic is assigned to.	–	17-07-24	19-07-24
3	Update the navigation links to support multiple classrooms. If multiple classrooms exist, the navbar will show these classrooms. If only one classroom exists, the user experience will be the same as it currently is.	–	22-07-24	24-07-24
–	Update the learner dashboard if any features are missing.	–	22-07-24	26-07-24
4	Write the acceptance test for the learner dashboard CUJ 3.1 and navigation links CUJ 2.2 .	–	28-07-24	31-07-24
5	Update the diagnostic player page to suggest lessons for multiple classrooms (currently it only works for math classrooms).		04-08-24	07-08-24
–	Meeting with the PM – M2 Demo	1, 2, 3, 4	08-08-24	–
6	Implement the necessary analytics events.	1, 2, 3	09-08-24	11-08-24
7	<p>Work with the server admin to deploy the code to production. First, it will be tested on the test server, and once everything works in the testing, then it will be made public.</p> <p>Create a cleanup PR to remove the code that returns default values for the newly introduced fields in the math classroom model.</p> <p>– If we don't encounter any issues from users, we will create a final PR to remove the feature flag.</p>	1, 2, 3, 4, 5, 6	12-08-24	–

Future Work

Note: *This section is mainly for reference (since items listed here won't be part of the GSoC project). Proposals will primarily be evaluated based on the implementation plan above.*

I believe the classroom feature will be a great resource for learning new concepts in a particular subject. There are plenty of features we can implement in the future, such as segregating topics based on difficulty or success rate and displaying these categorized topics to learners. Additionally, we can add the option to subscribe to a classroom so that users will be notified whenever a new topic is added ([issue](#)).