Section 1: About You

Goals and Objectives

Why do you want to do a GSoC project with Oppia?	I started contributing to Oppia in September 2023. I've found it to be an incredibly rewarding experience. Oppia has not only allowed me to enhance my Android development skills but also provided a welcoming and supportive community environment. Additionally, Oppia's mission of making quality education accessible to all resonates deeply with me, and I believe that participating in a GSoC project with Oppia will allow me to contribute meaningfully to this noble cause while furthering my own growth as a developer.	
What do you hope to learn/achieve during GSoC?	I am eager to delve into deeper concepts of Android development, focusing on clean code principles and acquiring knowledge of standard coding practices. Additionally, I aspire to enhance my communication skills, recognizing their crucial role in effective collaboration and conveying ideas within the development community.	
Which Oppia teams have you collaborated with, and what have you done on those teams?	**Android** **CLaM**	Here are some ***non-coding*** contributions that I would like to highlight: • Reviewed PRs. [Link] [Link] • Helped fellow contributors in reproducing issues. [Link] • Guided new contributors. [Link] • Helped team members with investigation of issues. [Link] • Provided team members with insights related to Bazel setup on Windows. [Link] • Helped in identifying duplicate issues. [Link]
Contact information	**Email**	saptakmanna100@gmail.com
	GitHub	github.com/theMr17
	LinkedIn	linkedin.com/in/saptak-manna
Preferred method of communication	Google chat or Email for regular updates and general queries.	
Which time zone will you primarily be in during the summer?	Indian Standard Time (+5:30 GMT).	
(If you are a student) When are your school holidays?	My summer holidays will be starting from 22nd June to 18th August this year. Last year, it was from 19th June to 7th August.	
What other obligations might you need to work	During the summer, I have no other obligations that I need to work around.	

around during the summer?	
Planned time commitment	My planned time commitment will be a. 7-8 hours/day when on holiday. b. 6 hours/day when in college without exams. c. 4 hours/day when in college with exams (This shall not be faced as my exams would be from 9th September onwards. Last year, it was from 11th September.) Following advice, I won't be working 7 days a week. Instead, I'll take a suitable leave each week, depending on the project's status. So, going by that let's consider I will be working for 6 days a week. For Milestone 1 (May 27 - June 28), I will be working for 6 hours/day for the first 3 weeks. and then for the last week I will be working for 8 hours/day (Please refer to summer holiday dates and milestone dates). That's a total of **156 hours** (*[6 hrs/day * 6 days/week * 3 weeks] + [8 hrs/day * 6 days/week * 1 week]*). Moving ahead to Milestone 2 (July 8 - August 12), I will be working for 7 hrs/day for all the 5 weeks as it will be in my summer holidays. That's a total of **210 hours** (7 *hrs/day * 6 days/week * 5 weeks)*.

Section 2: About Your Project

Project Details

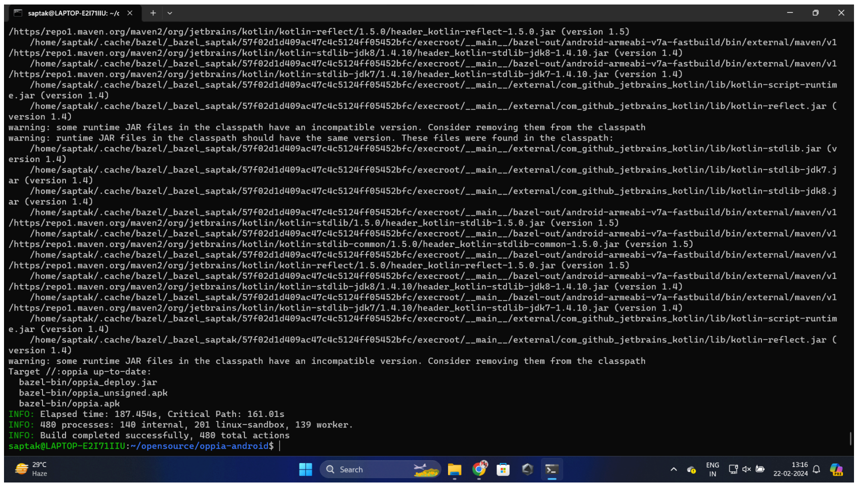
Project title *(should match the one on the Project Ideas list)*	Multiple Classrooms Support.
Project size *(should match the options on the Project Ideas list)*	Large (~350 hours).
Why did you choose this project?	I chose this project because it aligns perfectly with my interests and aspirations. The opportunity to contribute to the expansion of an educational app, moving beyond basic numeracy lessons to encompass a broader range of subjects like financial literacy and science, greatly excites me. Additionally, the structured approach outlined in the project, with clear milestones, deliverables, and technical guidance, appeals to my preference for organized and methodical project management. By working on this project, I anticipate gaining valuable experience in Android

	development, especially in implementing features at a larger scale, while also refining my skills in adhering to industry-standard coding practices. Overall, this project presents a unique chance for me to make a meaningful impact in the field of education technology, which is why I'm eager to be a part of it.

Required Skills

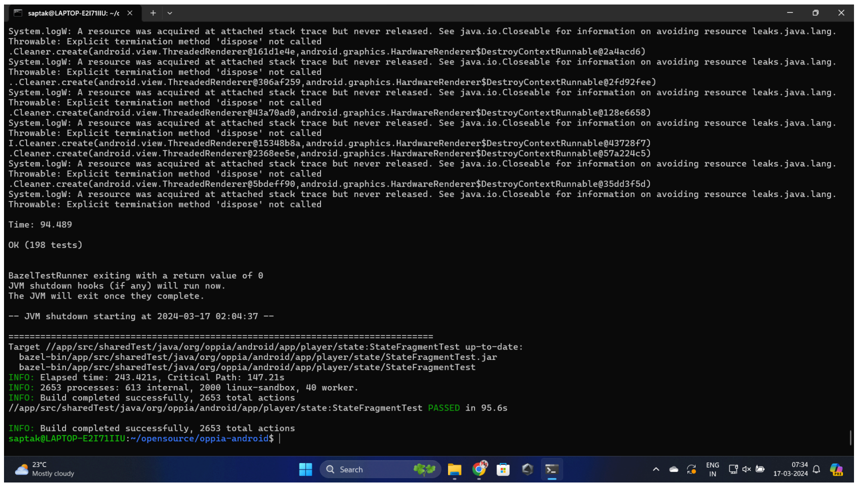
ANDROID	
I can write code and tests in Kotlin.	• Fix part of #5070: Display empty answer message in ratio input interaction • Fix part of #5070: Display empty answer message in number input interaction • Fix #5136: Modifies all caps buttons to sentence case
I can build the app and run tests using Bazel.	***Build the app using Bazel*** (screenshot below) ***Run tests using Bazel*** (screenshot below)

Build the app using Bazel



```
/https/repo1.maven.org/maven2/org/jetbrains/kotlin/kotlin-reflect/1.5.0/header_kotlin-reflect-1.5.0.jar (version 1.5)
    /home/saptak/.cache/bazel/_bazel_saptak/57f02d1d409ac47c4c5124ff05452bfc/execroot/__main__/bazel-out/android-armeabi-v7a-fastbuild/bin/external/maven/v1
/https/repo1.maven.org/maven2/org/jetbrains/kotlin/kotlin-stdlib-jdk8/1.4.10/header_kotlin-stdlib-jdk8-1.4.10.jar (version 1.4)
    /home/saptak/.cache/bazel/_bazel_saptak/57f02d1d409ac47c4c5124ff05452bfc/execroot/__main__/bazel-out/android-armeabi-v7a-fastbuild/bin/external/maven/v1
/https/repo1.maven.org/maven2/org/jetbrains/kotlin/kotlin-stdlib-jdk7/1.4.10/header_kotlin-stdlib-jdk7-1.4.10.jar (version 1.4)
    /home/saptak/.cache/bazel/_bazel_saptak/57f02d1d409ac47c4c5124ff05452bfc/execroot/__main__/external/com_github_jetbrains_kotlin/lib/kotlin-script-runtim
e.jar (version 1.4)
    /home/saptak/.cache/bazel/_bazel_saptak/57f02d1d409ac47c4c5124ff05452bfc/execroot/__main__/external/com_github_jetbrains_kotlin/lib/kotlin-reflect.jar (
version 1.4)
warning: some runtime JAR files in the classpath have an incompatible version. Consider removing them from the classpath
warning: runtime JAR files in the classpath should have the same version. These files were found in the classpath:
    /home/saptak/.cache/bazel/_bazel_saptak/57f02d1d409ac47c4c5124ff05452bfc/execroot/__main__/external/com_github_jetbrains_kotlin/lib/kotlin-stdlib.jar (v
ersion 1.4)
    /home/saptak/.cache/bazel/_bazel_saptak/57f02d1d409ac47c4c5124ff05452bfc/execroot/__main__/external/com_github_jetbrains_kotlin/lib/kotlin-stdlib-jdk7.j
ar (version 1.4)
    /home/saptak/.cache/bazel/_bazel_saptak/57f02d1d409ac47c4c5124ff05452bfc/execroot/__main__/external/com_github_jetbrains_kotlin/lib/kotlin-stdlib-jdk8.j
ar (version 1.4)
    /home/saptak/.cache/bazel/_bazel_saptak/57f02d1d409ac47c4c5124ff05452bfc/execroot/__main__/bazel-out/android-armeabi-v7a-fastbuild/bin/external/maven/v1
/https/repo1.maven.org/maven2/org/jetbrains/kotlin/kotlin-stdlib/1.5.0/header_kotlin-stdlib-1.5.0.jar (version 1.5)
    /home/saptak/.cache/bazel/_bazel_saptak/57f02d1d409ac47c4c5124ff05452bfc/execroot/__main__/bazel-out/android-armeabi-v7a-fastbuild/bin/external/maven/v1
/https/repo1.maven.org/maven2/org/jetbrains/kotlin/kotlin-stdlib-common/1.5.0/header_kotlin-stdlib-common-1.5.0.jar (version 1.5)
    /home/saptak/.cache/bazel/_bazel_saptak/57f02d1d409ac47c4c5124ff05452bfc/execroot/__main__/bazel-out/android-armeabi-v7a-fastbuild/bin/external/maven/v1
/https/repo1.maven.org/maven2/org/jetbrains/kotlin/kotlin-reflect/1.5.0/header_kotlin-reflect-1.5.0.jar (version 1.5)
    /home/saptak/.cache/bazel/_bazel_saptak/57f02d1d409ac47c4c5124ff05452bfc/execroot/__main__/bazel-out/android-armeabi-v7a-fastbuild/bin/external/maven/v1
/https/repo1.maven.org/maven2/org/jetbrains/kotlin/kotlin-stdlib-jdk8/1.4.10/header_kotlin-stdlib-jdk8-1.4.10.jar (version 1.4)
    /home/saptak/.cache/bazel/_bazel_saptak/57f02d1d409ac47c4c5124ff05452bfc/execroot/__main__/bazel-out/android-armeabi-v7a-fastbuild/bin/external/maven/v1
/https/repo1.maven.org/maven2/org/jetbrains/kotlin/kotlin-stdlib-jdk7/1.4.10/header_kotlin-stdlib-jdk7-1.4.10.jar (version 1.4)
    /home/saptak/.cache/bazel/_bazel_saptak/57f02d1d409ac47c4c5124ff05452bfc/execroot/__main__/external/com_github_jetbrains_kotlin/lib/kotlin-script-runtim
e.jar (version 1.4)
    /home/saptak/.cache/bazel/_bazel_saptak/57f02d1d409ac47c4c5124ff05452bfc/execroot/__main__/external/com_github_jetbrains_kotlin/lib/kotlin-reflect.jar (
version 1.4)
warning: some runtime JAR files in the classpath have an incompatible version. Consider removing them from the classpath
Target //:oppia up-to-date:
  bazel-bin/oppia_deploy.jar
  bazel-bin/oppia_unsigned.apk
  bazel-bin/oppia.apk
INFO: Elapsed time: 187.454s, Critical Path: 161.01s
INFO: 480 processes: 140 internal, 201 linux-sandbox, 139 worker.
INFO: Build completed successfully, 480 total actions
saptak@LAPTOP-E2I71IIU:~/opensource/oppia-android$ |
```

Run tests using Bazel



```
System.logW: A resource was acquired at attached stack trace but never released. See java.io.Closeable for information on avoiding resource leaks.java.lang.
Throwable: Explicit termination method 'dispose' not called
.Cleaner.create(android.view.ThreadedRenderer@161d1e4e,android.graphics.HardwareRenderer$DestroyContextRunnable@2a4acd6)
System.logW: A resource was acquired at attached stack trace but never released. See java.io.Closeable for information on avoiding resource leaks.java.lang.
Throwable: Explicit termination method 'dispose' not called
..Cleaner.create(android.view.ThreadedRenderer@306af259,android.graphics.HardwareRenderer$DestroyContextRunnable@2fd92fee)
System.logW: A resource was acquired at attached stack trace but never released. See java.io.Closeable for information on avoiding resource leaks.java.lang.
Throwable: Explicit termination method 'dispose' not called
.Cleaner.create(android.view.ThreadedRenderer@43a70ad0,android.graphics.HardwareRenderer$DestroyContextRunnable@128e6658)
System.logW: A resource was acquired at attached stack trace but never released. See java.io.Closeable for information on avoiding resource leaks.java.lang.
Throwable: Explicit termination method 'dispose' not called
I.Cleaner.create(android.view.ThreadedRenderer@15348b8a,android.graphics.HardwareRenderer$DestroyContextRunnable@43728f7)
.Cleaner.create(android.view.ThreadedRenderer@2368ee5e,android.graphics.HardwareRenderer$DestroyContextRunnable@57a224c5)
System.logW: A resource was acquired at attached stack trace but never released. See java.io.Closeable for information on avoiding resource leaks.java.lang.
Throwable: Explicit termination method 'dispose' not called
.Cleaner.create(android.view.ThreadedRenderer@5bdeff90,android.graphics.HardwareRenderer$DestroyContextRunnable@35dd3f5d)
System.logW: A resource was acquired at attached stack trace but never released. See java.io.Closeable for information on avoiding resource leaks.java.lang.
Throwable: Explicit termination method 'dispose' not called

Time: 94.489

OK (198 tests)


BazelTestRunner exiting with a return value of 0
JVM shutdown hooks (if any) will run now.
The JVM will exit once they complete.

-- JVM shutdown starting at 2024-03-17 02:04:37 --

================================================================================
Target //app/src/sharedTest/java/org/oppia/android/app/player/state:StateFragmentTest up-to-date:
  bazel-bin/app/src/sharedTest/java/org/oppia/android/app/player/state/StateFragmentTest.jar
  bazel-bin/app/src/sharedTest/java/org/oppia/android/app/player/state/StateFragmentTest
INFO: Elapsed time: 243.421s, Critical Path: 147.21s
INFO: 2653 processes: 613 internal, 2000 linux-sandbox, 40 worker.
INFO: Build completed successfully, 2653 total actions
//app/src/sharedTest/java/org/oppia/android/app/player/state:StateFragmentTest PASSED in 95.6s

INFO: Build completed successfully, 2653 total actions
saptak@LAPTOP-E2I71IIU:~/opensource/oppia-android$ |
```

I can make changes to UIs in Android, write tests for them, and manually verify that they work.	● Fix part of #2480: Fixes 2 Audio Tests ● Fix part of #5070: Display empty answer message in number input interaction ● Fix #3596: Adds Audio Loading UI
I can make changes that involve DataProviders (either by creating them, modifying them, or updating code that depends on them).	● Fix #4042: Implement success criteria metrics for lesson checkpointing *DataProvider* serves as crucial gateways for receiving asynchronous results from operations in a safe and efficient manner. They are designed to support notifications for new data availability, encourage the use of suspend functions to facilitate coroutine usage, and offer utilities for streamlined implementation. *DataProviders* class offers various functions for transforming, combining, and converting data providers. *LiveData*, a lifecycle-aware stateful concurrency primitive within Android Jetpack, is preferred for transferring data to the UI due to its ability to handle background thread communication, lifecycle awareness, and seamless integration with Android databinding. The conversion of DataProviders to LiveData via the *toLiveData* extension function simplifies the process of data presentation in the UI. *[Reference: DataProvider & LiveData Wiki]*
I can make changes to proto files and/or work with generated proto code.	● Fix #4042: Implement success criteria metrics for lesson checkpointing ● *(Reviewed)* Fix part of #5070: Display empty answer message in Math expressions input interaction
I can make changes to production code and/or tests that involve platform parameters.	● *(Reviewed)* Fix Part of #4938: Introduce New App Language Selection Screen Here is an overview of how we use platform parameters & feature flags in Oppia: ● To create a parameter, we start by defining a Qualifier Annotation, String name, and default value in a constants file. ● To provide the parameter involves using Dagger in our *PlatformParameterModule*, where we specify the data type with the *@Provides* annotation. ● To consume the parameter, we inject the specific *PlatformParameterValue<T>* instance with the corresponding Qualifier Annotation. Here is an overview of how we use them for testing code: ● We test the app for different values of parameters. Same parameter can have different values because of the difference between its default and remote value. ● We force specific values to the parameters via *TestPlatformParameterModule*. *[Reference: Platform Parameters & Feature Flags Wiki]*
I can add new dependencies to the app's	● Fix part of #5070: Display empty answer message in ratio input interaction

Dagger graph and I understand how they are used in tests.	● *(Reviewed)* Fix part of #5070: Display empty answer message in Math expressions input interaction ● Fix #4042: Implement success criteria metrics for lesson checkpointing

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	I will adhere to the above deadlines.

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.	Yes.
In addition to the above: how often, and through which channel(s), do you plan on communicating with your mentor?	My plan is to offer daily updates to my mentor for effective communication and a streamlined project flow. I prefer using Google Chat or Email for routine discussions and addressing minor queries. Additionally, I propose two weekly meetings with my mentor, providing flexibility in scheduling and using platforms like Google Meet or any preferred alternative.

Section 3: Proposal Details.

Problem Statement

Target Audience	This feature will primarily impact learners (who will be able to learn different subjects) and teachers (who will utilize the classrooms to teach one or more subjects each). To a lesser degree, admins and other support roles will be impacted.
Core User Need	● As a learner, I want to learn additional subjects on top of the math lessons I

	already enjoy in the app. ● As a teacher, I want to leverage Oppia's proven capabilities to teach new subjects.
What goals do we want the solution to achieve?	Success will be evaluated via engagement and user sentiment. On the engagement side, we'll be looking at ● Usage of additional subjects (MAUs, lessons started, lessons completed) (note that this is tied to content quality as well) ● Counter-metric: Drop in existing users / usage of existing lessons, decreased completion rate for lessons. For sentiment, ● We'll continue to track CSAT, especially when it comes to using classroom features. ● Of particular importance is ensuring, post-launch, that the core concept and UX did not make the app more complicated. To that end, we'll look at onboarding user churn compared to baseline (pre-classrooms launch).

Section 3.1: WHAT

Key User Stories and Tasks that will be implemented

#	Title	User Story Description (role, goal, motivation) *"As a …, I need …, so that …."*	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	Select classroom	As a learner, I need to select the appropriate classroom so that I can work through the associated lessons.	View list of available classrooms. Identify the classroom associated with the lessons I want to learn next. Select the classroom.	Figma UI ● Classroom Carousel Figma Flow ● Recently played below classroom carousel ● Recently played above classroom carousel Reference ● PRD: Select classroom
2	Switch between classrooms	As a learner, after finishing a lesson, I want to learn about a different subject, so that I don't get bored.	Identify the classroom I'm currently in. Open classroom selection affordance.	Figma UI ● Classroom Carousel Figma Flow ● Recently played below classroom carousel

			Select a new classroom.	● Recently played above classroom carousel Reference ● PRD: Switch between classrooms
3	Save progress per subject	As a learner, I need my progress in a given subject and topic to be saved independently of my progress in other subjects/topics, so that I can learn multiple subjects concurrently.	Start a story in a classroom.	Reference ● PRD: Save progress per subject
			Go back to the home screen.	
			Start a story in a different classroom.	
			Go back to the home screen, select the first story.	
4	ID current classroom	As a learner, I need to be able to easily identify which classroom I'm currently in, so that if I return to my screen after a break, I can quickly orient myself and continue.	Find & read classroom ID affordance	Reference ● PRD: ID current classroom
5	View progress across subjects	As a learner, I want to be able to see my progress across multiple classrooms at a glance, so that I can see how I'm progressing.	Go to the overview page.	Figma UI ● Screen - Recently played below classroom carousel ● Screen - Recently played above classroom carousel Figma Flow ● Recently played below classroom carousel ● Recently played above classroom carousel Reference ● PRD: View progress across topics
			View my progress.	

Technical Requirements

Additions/Changes to Web Server Endpoint Contracts

No additions/changes to Web Server Endpoint Contracts is required.

Calls to Web Server Endpoints

The project depends on the asset download script to fetch the multiple classroom data and related assets from the relevant endpoints to provide them pre-bundled into the app builds.

UI Screens/Components

#	ID	Description of new UI component	i18n required?	Mock links	A11y requirements
1.	Classroom Selection Screen	A greeting message will be displayed at the top of the new classroom selection screen. Following that, there will be a carousel featuring different classrooms. Finally, a list of topics pertaining to the selected classroom will be presented. A carousel of recommended stories will be shown either above or below the classroom carousel. (In case it is below the classroom carousel, the recommended stories will be filtered by the selected classroom.)	Yes. All user facing strings need to be translated.	Mocks	When Talkback is enabled, a user will be able to read and respond using their pre-configured accessibility settings.

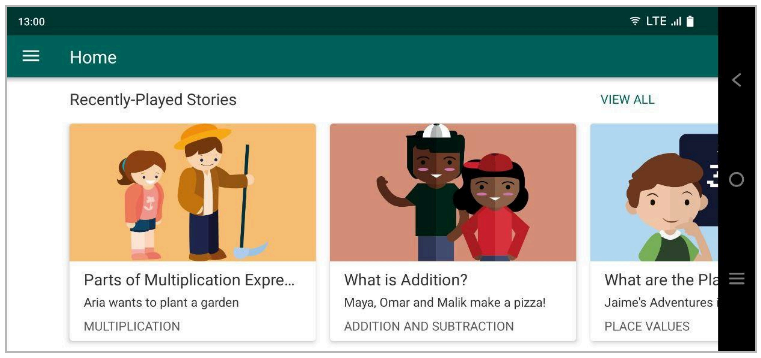
Data Handling and Privacy

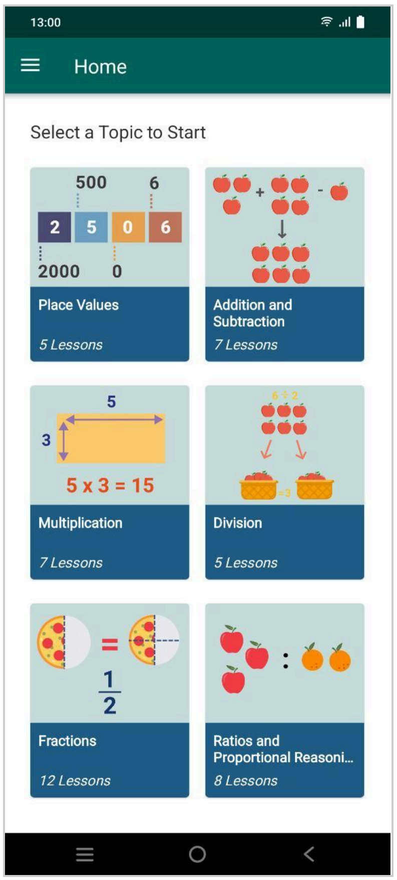
The project does not retain any sensitive user data. While it will gather various metrics pertaining solely to classroom activities, users need not worry about their privacy in this regard because the data we will collect will not contain any user PII.

Section 3.2: HOW

Existing Status Quo

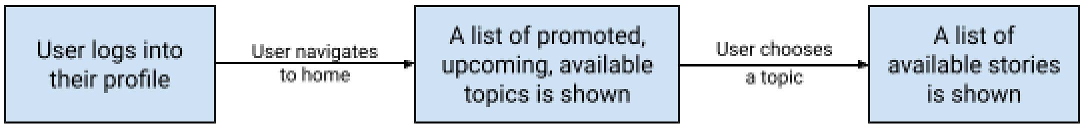
The Android app's home screen currently features a prominent carousel showcasing promoted stories, predominantly composed of recently accessed content (refer image below). Following this carousel is a comprehensive list of available mathematics topics for user exploration (refer image aside). Notably, users are restricted to the mathematics classroom without the option to switch to other subjects.





The *TopicListController* is responsible for populating the topics displayed on the home screen, devoid of any reliance on a *classroomId* for segregation. The existing topic models lack any indication of belonging to a specific classroom, rendering them incapable of self-identifying their classroom association. Similarly, there is a lack of definition of any classroom model to filter and assign topics to particular classrooms.

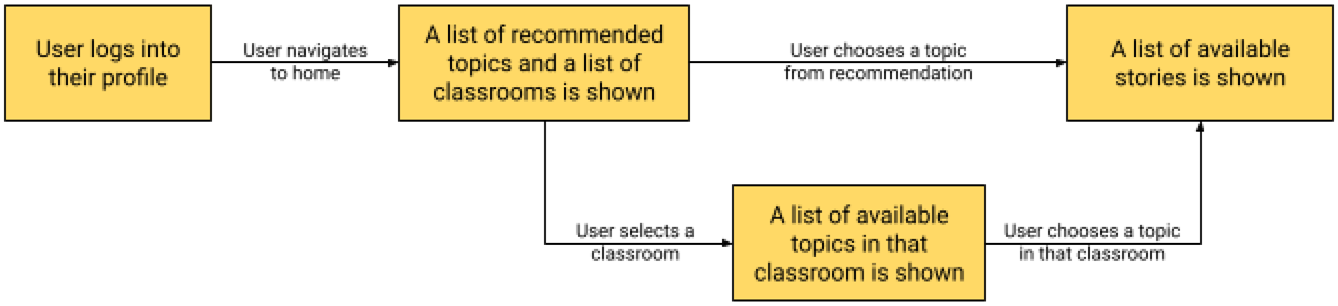
The following flowchart helps us to understand the current user flow on the home screen. The user can select only from a list of mathematics topics.


User logs into their profile
User navigates to home
A list of promoted, upcoming, available topics is shown
User chooses a topic
A list of available stories is shown

Solution Overview

The primary goals of this solution include enabling learners to seamlessly navigate between different subjects and save progress independently for each subject. The solution also aims at updating the asset download script so that we can download classroom assets for a production release.

The following flowchart helps us to understand what we will achieve by this project. The user will have a list of classrooms to choose from. Each classroom will have a range of topics which they can learn as per their requirement.


User logs into their profile
User navigates to home
A list of recommended topics and a list of classrooms is shown
User chooses a topic from recommendation
A list of available stories is shown
User selects a classroom
A list of available topics in that classroom is shown
User chooses a topic in that classroom

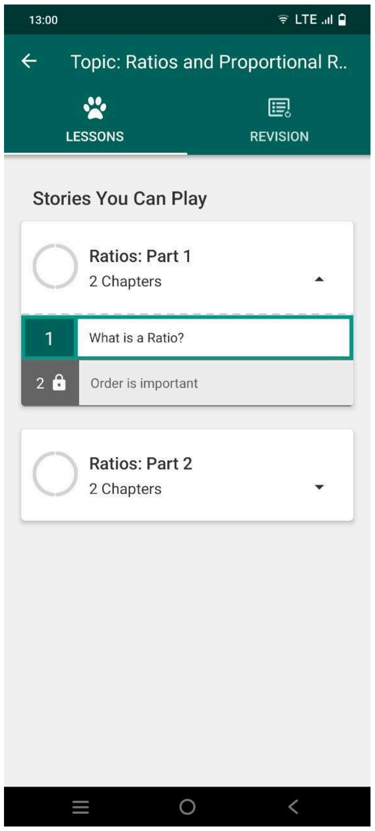
There is no need for any modifications or enhancements to the existing lightweight checkpointing system, as it is fully equipped to independently save the progress of each topic within any classroom or subject. Its functionality ensures that each topic's progress is recorded and managed separately, without the necessity for further adjustments or improvements.

To facilitate clear identification of the current classroom, several options were investigated.

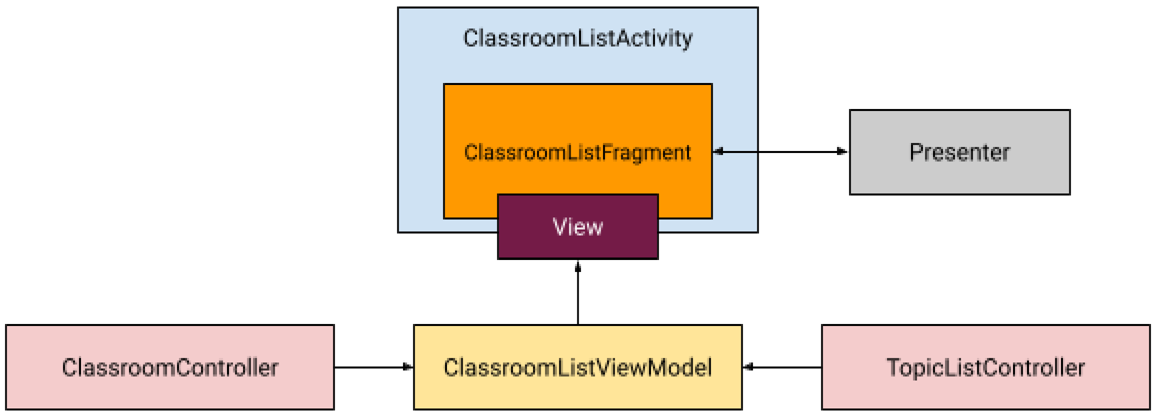
- The possibility of adding a subtitle in the toolbar to denote the current classroom was explored. However, it was found that this might clutter the interface since there are already tabs with icons present.
- Another consideration was placing a label next to the chapter number or replacing the chapter number with the classroom name. However, in scenarios where there are multiple stories listed, this information might be needlessly repeated and not significant enough to warrant such prominence.
- Since classroom information is hierarchically more important than the story, it can be placed at a level similar to the story heading.

Given these points, while the classroom information is indeed important, it was decided against disrupting the current layout as it is not important enough. Therefore, it was suggested not to include it altogether.

[Reference: Figma: Comment]

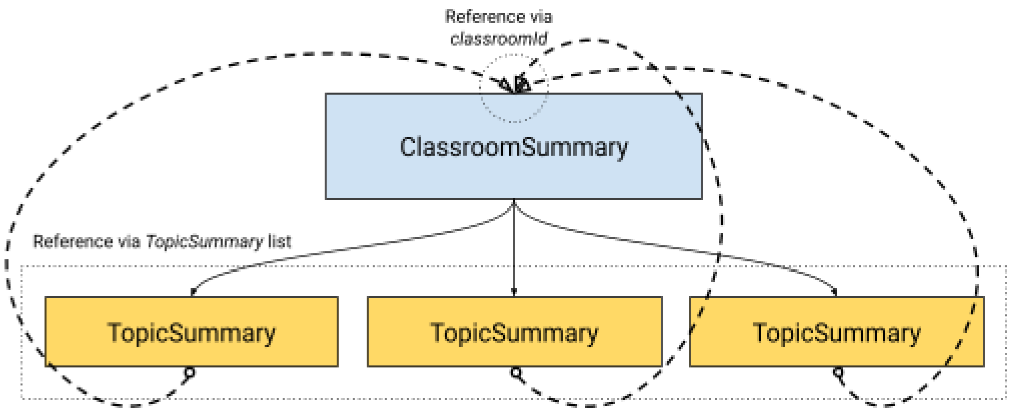


The subsequent sequence diagram illustrates the interaction between *ClassroomListActivity*, *ClassroomListFragment*, *ClassroomListViewModel*, *ClassroomController*, and *TopicListController*. This comprehensive depiction explains the interactions among these components within the system.



Here is a brief overview of how the project will be approached:

- Introduce *ENABLE_MULTIPLE_CLASSROOMS* feature flag to gate the visibility of multiple classrooms feature and related screens to the user.
- Introduce a new proto to define *ClassroomSummary* and *ClassroomIdList*. The *ClassroomSummary* model will contain a list of *TopicSummary* of the topics which will belong to the classroom. The existing *Topic, TopicSummary* and other related topic structures will be updated to include the *classroomId* of the classroom, signifying the classroom to which each topic belongs. This bidirectional reference will enable straightforward access to topics from a classroom and identification of the classroom associated with a specific topic.



- Introduce a *ClassroomController* which will be responsible for surfacing the *getClassroomList* and *getTopicList functions*. Update *TopicListController* to no longer surface the *getTopicList* function. Update *ProfileManagementController* to store the last selected classroom.
- Introduce activities, fragments, presenters, views, tests for classroom selection landing page (replacement for the existing home page).
- Implement new recommendations logic & UI support for the classroom selection page. Update recommendation cards to include classroom information as well.
- Update topic cards to include classroom & topic progress information.
- Hook up the new classroom selection page to replace the existing home activity upon profile login, gated by the feature flag.
- Implement success criteria metrics for the new multiple classroom feature. The project proposes logging events when a lesson is either started or completed. Each of the events will have a

timestamp, *classroomId* & *topicId* parameter. The ***Classroom usage rate*** will be calculated by taking all the above three events into consideration. The ***Lesson completion rate per classroom*** and ***Lesson completion speed per classroom*** will be calculated by considering the lesson started and completed events. Further details regarding the computed outcomes are outlined in PRD - Multiple Classrooms (Android) .

Third-Party Libraries

#	Third-party Library		Link to Third-Party Library	Why is it needed?	License	Min/Target/Max SDK Versions that the library supports
	Name	**Version**				
1	androidx.compose.runtime:runtime	1.2.0-rc02	Compose Runtime	Provides the fundamental building blocks for Compose applications.	Official Android Library	**Min SDK:** 21 **Target SDK:** 33 **Max SDK:** 33
2	androidx.compose.ui:ui	1.2.0-rc02	Compose UI	Contains core UI elements for building Compose applications.		
3	androidx.compose.foundation:foundation	1.2.0-rc02	Compose Foundation	Provides foundational Compose components like layout primitives and basic UI elements		
4	androidx.compose.foundation:foundation-layout	1.2.0-rc02	Compose Foundation Layout	Offers layout components and utilities for arranging UI elements in Compose application		
5	androidx.compose.material:material	1.2.0-rc02	Compose Material	Implements Material Design components for Compose applications		
6	androidx.compose.runtime:runtime-livedata	1.2.0-rc02	Compose Runtime LiveData	Facilitates the integration of LiveData with Compose, allowing reactive programming in Compose apps		
7	androidx.compose.ui:ui-tooling	1.2.0-rc02	Compose UI Tooling	Provides tools for debugging and inspecting Compose UI elements during development		
8	androidx.compose.ui:ui-test-junit4	1.2.0-rc02	Compose UI Test JUnit4	Supports JUnit4-based testing for Compose UI components		
9	androidx.compose.ui:ui-test-manifest	1.2.0-rc02	Compose UI Test	Provides manifest configuration for testing		

			Manifest	Compose UI components		
10	androidx.appcompat:appcompat	Update 1.0.2 to 1.3.0	Appcompat	The support for *ViewTreeLifecycleOwner* was added to *Appcompat* from v1.3.0-alpha01 (Release Notes).		
11	com.android.tools.build:gradle	Update 3.6.1 to 4.2.0	Android Gradle Plugin	AGP 4.0.0 introduces a new method for controlling the activation and deactivation of build features, a prerequisite for enabling Compose (Release Notes). AGP 4.2.0 is chosen as lower versions are incompatible with compose 1.2.0-rc02.		

"Service" Dependencies

The project does not require any Service Dependencies.

Impact on Other Oppia Teams

The project is self contained and does not impact other Oppia Teams.

Key High-Level and Architectural Decisions

Decision 1: How to migrate *getTopicList* from *TopicListController* to *ClassroomController*?

I have considered the following alternatives:

Alternative 1: Migrate only the *getTopicList* function to *ClassroomController* and interact with *TopicListController* for its supporting functions.
Alternative 2: Migrate *getTopicList* function along with its supporting functions from *TopicListController* to *ClassroomController.*

Among these, I believe that complete migration (Alternative 2) is the best approach, because:

- **Single Responsibility Principle:** Alternative 2 aligns better with SRP, as it ensures that *ClassroomController* is responsible for all operations related to managing classrooms and topics within them. Each controller should have a clear and singular purpose, and consolidating related functionalities within *ClassroomController* adheres to this principle.
- **Reduced Coupling (Dependency Inversion Principle):** By migrating both the *getTopicList* function and its supporting functions to *ClassroomController*, Alternative 2 reduces the coupling between controllers. This adheres to the Dependency Inversion Principle, which states that high-level modules should not depend on low-level modules, but rather both should depend on abstractions.

In this case, *ClassroomController* becomes more self-contained and does not rely on the specifics of *TopicListController*.

- **Open/Closed Principle:** Alternative 2 facilitates extension and modification without altering existing code. If there are future changes or enhancements to the functionality related to topic lists in classrooms, having all related functions within *ClassroomController* makes it easier to extend or modify the behavior without modifying *TopicListController*. This conforms to the OCP, which encourages software entities to be open for extension but closed for modification.

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

Criteria	**Alternative 1** (Partial Migration)	**Alternative 2** (Complete Migration)
Single Responsibility Principle	Partially adheres as the *getTopicList* function is moved, but supporting functions remain in *TopicListController.*	Better adherence as all related functionalities are grouped within *ClassroomController*.
Dependency Inversion Principle	Higher coupling as *ClassroomController* still depends on *TopicListController* for supporting functions.	Reduced coupling as *ClassroomController* becomes more self-contained.
Open/Closed Principle	Potential violation as modifications to the functionality may require changes in both controllers.	Adheres better as related functionalities are encapsulated within *ClassroomController*.

Decision 2: How to structure the new Classroom Protocol Buffer?

I have considered the following alternatives:

Alternative 1: Introduce classroom related protos directly inside *topic.proto*.
Alternative 2: Introduce a separate file, *classroom.proto*, dedicated for classroom related protos.

Among these, I believe that dedicating a separate file for classroom related protos will not be necessary So, (Alternative 1) is the best approach, because:

- **Scalability:** Although having a dedicated file could potentially aid in scalability, the current scope of classroom-related features may not warrant a separate file. By keeping all protocol buffer definitions within *topic.proto*, scalability concerns can still be addressed effectively through proper structuring within the file.
- **Clarity and Readability:** While a separate file may enhance clarity in terms of isolating classroom-related protos, it could also introduce additional complexity, especially for developers unfamiliar with the project structure. Embedding classroom-related definitions within *topic.proto* maintains a cohesive structure and promotes readability by keeping related concepts together.
- **Maintenance:** Keeping classroom protos in the same files simplifies maintenance as it eliminates the need for a separate proto library definition in the *BUILD.bazel* file. Conversely, maintaining a separate file for classroom protos complicates maintenance as it requires defining a separate proto library.

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

Criteria	**Alternative 1** (Inline classroom protos)	**Alternative 2** (Separate classroom file)
Scalability & Maintenance	Scalability is not currently a concern since the classroom-related protos are only a few lines of code each. Maintenance will be simpler by keeping classroom protos in the same files, because it will not be necessary to define a separate proto library in *model/BUILD.bazel*.	The amount of code changes is similar to keeping it in the *topic.proto* file, but with additional work to hook up the new file with Bazel by creating a separate proto library in *model/BUILD.bazel*.
Clarity & Readability	The *topic.proto* file currently contains all the structure related to the topic. So, including the classroom related structures in the same file will emphasize the relation between topics and classrooms by keeping related concepts together, enhancing readability.	The *topic.proto* file contains all the structures (other than those related to topics, such as *Subtopic*, *Story*, *Chapter*, etc.) which are hierarchically below *Topic*. In the case of *Classroom*, it is hierarchically above *Topic*. This might create confusion regarding the relationship between the classroom and topic concepts.

Decision 3: How to implement the sticky classroom carousel?

I have considered the following alternatives:

Alternative 1: Implement sticky header using *RecyclerView.ItemDecoration*.
Alternative 2: Introduce a layout container to render the sticky header when the classroom carousel is scrolled out of the screen.
Alternative 3: Introduce Jetpack Compose, to use the sticky header component of the *LazyColumn*.

Among these, I believe that introducing Jetpack Compose to use the sticky header component of the *LazyColumn* (Alternative 3) is the best approach, because:

- **Interactions:** The sticky header, which will be the classroom carousel, should be interactable. The carousel should be able to be scrolled horizontally, and the carousel items should be clickable to switch classrooms. Jetpack Compose offers full interactivity with built-in support for sticky headers. Components are inherently composable, making it easy to integrate horizontally scrollable carousels and clickable items.
- **Implementation Complexity:** Jetpack Compose simplifies the implementation with a declarative UI approach, reducing complexity and enhancing maintainability. Using Compose's *LazyColumn*, built-in modifiers like *stickyHeader* can be leveraged, which makes the development process more straightforward compared to the other alternatives that require additional handling for touch events and synchronization logic.
- **Performance:** Compose’s *LazyColumn* is optimized for performance, handling large lists and interactive components, including sticky headers, efficiently. This ensures a smooth user experience even with complex UI elements like a horizontally scrollable, clickable carousel.

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

Criteria	**Alternative 1** (Using *RecyclerView*'s *ItemDecoration*)	**Alternative 2** (Using Layout Container)	**Alternative 3** (Using Jetpack Compose's Stick Header)
Interactions	The sticky header will not be clickable or horizontally scrollable because it will be drawn as an item decoration of the recycler view.	The sticky header will be clickable and horizontally scrollable as the view itself will be rendered in the layout container.	Offers full interactivity with built-in support for sticky headers. Components are inherently composable, making it easy to integrate horizontally scrollable carousels and clickable items.
Implementation Complexity	Complex to implement interactive elements, requiring additional handling for touch events and updates.	Medium complexity, with additional logic needed to sync the layout container with the RecyclerView scroll.	Simplifies implementation with declarative UI, reducing complexity and enhancing maintainability.
Integration Complexity	Integration is simpler and does not require introduction of any dependencies.	Integration is simpler and does not require introduction of any dependencies.	Setting up Jetpack Compose with both Gradle and Bazel requires significant effort. This includes updating existing dependency versions to ensure compatibility with the new Jetpack Compose dependencies, as well as updating the Kotlin version and minimum SDK of the project.
Performance	Can be performant but lacks flexibility for interactive components.	Can introduce performance overhead due to synchronization requirements.	Compose's *LazyColumn* is optimized for performance, handling large lists and interactive components (including sticky headers) efficiently.

Risks and mitigations

Potential Risk	**Mitigation**
Additional content related to this feature may increase file size.	● Track file sizes without implementing active mitigation measures during this launch, using observations to inform future decisions. ● Consider user-controlled content management, allowing users to download or remove topics at their leisure. ● Develop an automated cleanup feature nudging users to remove unused content based on

	usage patterns. *[Reference: PRD: Risks/concerns and mitigations]*
Inadequate user guidance for classroom selection	● Implement clear and intuitive UI elements for selecting and switching classrooms. ● Provide tooltips or instructional overlays (Spotlight) to guide users through the process.
Incompatibility of Jetpack Compose with existing dependency versions.	● Ensure all dependencies, especially those related to Jetpack Compose, are using compatible versions. ● Upgrade to the minimum required Gradle & Kotlin version.

Implementation Approach

Domain Objects

The ***topic.proto*** file will be updated to include the following two structures:

- ***ClassroomSummary***
  - ***classroom_id*:** This field acts as the unique identifier for each classroom instance. It plays a crucial role in distinguishing one classroom from another within the system. Utilizing a unique identifier ensures accuracy and reliability in referencing and retrieving specific classroom data.
  - ***classroom_title:*** A mandatory field responsible for storing the title of the classroom. The use of the 'SubtitledHtml' type facilitates the accommodation of diverse languages, catering to the linguistic needs of a diverse user base. By supporting multiple languages, this field enhances accessibility and inclusivity, enabling users to engage with classroom content in their preferred language.
  - ***topic_summary:*** This is a *repeated* field to store the list of topic summaries which come under this classroom.
  - ***topic_prerequisites:*** This is a mapping of *topicId* to a list of prerequisite topics in this classroom.
  - ***written_translation_context:*** This field is required to provide context for translation of the *classroom_title*.

```
// A summary of a classroom which contains a list of topic summaries.
message ClassroomSummary {
  // The ID of the classroom.
  string classroom_id = 1;

  // The title of the classroom.
  SubtitledHtml classroom_title = 2;

  // A list of topic summaries contained within this classroom.
  repeated TopicSummary topic_summary = 3;
  // A map of topic ID to a list of prerequisite topics.
  map<string, TopicList> topic_prerequisites = 4;

  // The translation context that should be used for this classroom.
  WrittenTranslationContext written_translation_context = 5;
}
```

- ***ClassroomIdList:*** This structure corresponds to a local file which contains the list of classroom IDs available.
  - ***classroom_ids:*** This field is required to store the list of IDs corresponding to each classroom which are available.

```
// Corresponds to a local file cataloging all classrooms available to load.
message ClassroomIdList {
  // The list of IDs corresponding to classrooms available on the local filesystem.
  repeated string classroom_ids = 1;
}
```

In addition to the above changes, it's necessary to update the existing ***topic.proto*** file to include references to the corresponding classroom for each topic. This entails modifying the structures such as ***Topic***, ***EphemeralTopic***, ***TopicSummary***, ***EphemeralTopicSummary***, ***TopicRecord***, and ***PromotedStory*** to incorporate a field indicating the specific ***classroomId*** & ***classroomTitle*** to which the respective topic belongs.

The ***Topic***, ***TopicSummary***, and ***TopicRecord*** will be updated to include a list of prerequisite topics, similar to the structure in Oppia Web (here). Instead of directly modifying the mentioned protos, we can adopt the existing format from the web version and incorporate a mapping of *topicId* to *TopicList* in the ***ClassroomSummary***. This map will store the prerequisites for each topic.

To accommodate the introduction of the new *ClassroomListActivity*, an additional entry will be included within the ***ScreenName*** enum defined in the ***screens.proto*** file. This enumeration comprises the complete set of names corresponding to all user interface screens supported by the application.

A new string field ***last_selected_classroom_id*** will be introduced in the ***Profile*** object of ***profile.proto*** file. This field will store the ID of the classroom that the user selected during their last session of the app.

For the new success criteria metrics, the events which are required are already being recorded in the app except the lesson start event. The events also include an additional parameter ***classroomId***, which will be added as a field to the ***ExplorationContext*** structure in ***oppia_logger.proto*** file. Further details regarding this are discussed here.

User Flows (Controllers and Services)

In the forthcoming updates to the domain module, the primary modifications will encompass the following aspects:

- Introduction of *ClassroomController* to surface *getClassroomList*
- Migration of *getTopicList* from *TopicListController* to *ClassroomController*

- Modification of *ProfileManagementController* to track last selected classroom
- Modification of *PlatformParameterModule*(s) to provide the feature flag

Introduction of *ClassroomController* to surface *getClassroomList*

The ***getClassroomList*** function, as shown below, serves as the entry point for fetching classroom information. This function will operate based on the provided ***profileId***, leveraging the *TranslationController* to ascertain the appropriate written translation content locale associated with the user's profile. By utilizing this locale, it will be ensured that classroom data is presented in a linguistically suitable manner, **enhancing accessibility for users** across different language preferences.

```kotlin
/**
 * Retrieves a data provider of list of classrooms based on the locale for
 * translation content associated with the specified profile ID.
 *
 * @param profileId of the profile to determine the translation content locale.
 * @return A [DataProvider] that asynchronously provides a list of
 * [ClassroomSummary] objects.
 */
fun getClassroomList(profileId: ProfileId): DataProvider<List<ClassroomSummary>>
```

Load classroomId from assets: The subsequent phase of the data retrieval process will be managed by the ***createClassroomList*** function. This function will be tasked with generating a list of *ClassroomSummary* objects, encapsulating crucial information about each classroom as defined previously. It will make use of a configurable boolean flag, ***loadLessonProtosFromAssets***, to dictate the data source for classroom information retrieval. Depending on the value of this flag, the function will dynamically select between **loading classroom data from *textprotos* or *json* files**.

In scenarios where loading data from textprotos is preferred, *createClassroomList* will orchestrate interactions with an *AssetRepository*. This repository will facilitate the retrieval of classroom IDs, which will then be mapped to corresponding *ClassroomSummary* objects. Conversely, if loading from *textprotos* is not the chosen approach, the function will delegate to *loadClassroomListFromJson*, passing the *contentLocale* for fetching classroom data from *json* files.

```kotlin
/**
 * Creates a list of classroom summaries based on the provided content locale.
 *
 * If [loadLessonProtosFromAssets] is true, it loads classroom IDs from local
 * assets and generates classroom summaries for each ID. Otherwise, it loads the
 * classroom list from JSON files corresponding to the given content locale.
 *
 * @param contentLocale for which to create the classroom summaries.
 * @return A list of [ClassroomSummary] objects.
 */
private fun createClassroomList(
    contentLocale: OppiaLocale.ContentLocale
): List<ClassroomSummary>
```

Within the domain of the ***loadClassroomListFromJson*** function, classroom data will be sourced from *json* files. This function will navigate through the *json* structure to extract relevant classroom IDs, subsequently iterating over each ID to construct the corresponding *ClassroomSummary* objects. Through this process, the systematic retrieval and organization of classroom data will be ensured.

```kotlin
/**
 * Loads a list of classroom summaries from JSON files based on the provided
 * content locale.
 *
 * This function reads JSON data from the "classrooms.json" asset file and extracts
 * the list of classroom IDs. It then creates classroom summaries for each ID using
 * the [createClassroomSummary] function.
 *
 * @param contentLocale The content locale for which to load the classroom
summaries.
 * @return A list of [ClassroomSummary] objects extracted from the JSON data.
 */
private fun loadClassroomListFromJson(
  contentLocale: OppiaLocale.ContentLocale
): List<ClassroomSummary>
```

Load classroom data using *classroomId* from assets: A pivotal aspect of the functionality will lie in the creation of individual *ClassroomSummary* objects, facilitated by the ***createClassroomSummary*** function. This function will operate based on the selected data source, as determined by the *loadLessonProtosFromAssets* flag. If loading from *textprotos*, *createClassroomSummary* will interact with the *AssetRepository* to fetch the classroom summary proto for the specified ID. Conversely, if loading from *json* files, the function will seamlessly delegate to ***loadClassroomFromJson***, passing the relevant classroom ID.

```kotlin
/**
 * Creates a summary for the specified classroom ID based on the content locale.
 *
 * If [loadLessonProtosFromAssets] is true, it loads the classroom summary proto
 * from local assets using the provided classroom ID. Otherwise, it loads the
 * classroom summary from JSON files corresponding to the given content locale and
 * classroom ID.
 *
 * @param classroomId of the classroom for which to create the summary.
 * @param contentLocale for which to create the classroom summaries.
 * @return A list of [ClassroomSummary] objects.
 */
private fun createClassroomSummary(
  classroomId: String,
  contentLocale: OppiaLocale.ContentLocale
): ClassroomSummary
```

The ***loadClassroomFromJson*** function will undertake the task of retrieving detailed classroom data from *json* files. By parsing the *json* content associated with the provided *classroomId*, this function will extract pertinent information such as the classroom title and topic IDs. Leveraging this data, the function will construct a *ClassroomSummary* object encapsulating essential details, poised for presentation within educational applications.

```
/**
 * Loads a classroom summary from JSON files based on the provided classroom ID and
 * content locale.
 *
 * This function reads JSON data corresponding to the specified classroom ID and
 * content locale, and constructs a [ClassroomSummary] object representing the
 * summary of that classroom.
 *
 * @param classroomId The ID of the classroom for which to load the summary.
 * @param contentLocale The content locale for which to load the summary.
 * @return A [ClassroomSummary] object representing the summary of the specified
 * classroom.
 */
private fun loadClassroomFromJson(
  classroomId: String,
  contentLocale: OppiaLocale.ContentLocale
): ClassroomSummary
```

Migration of *getTopicList* from *TopicListController* to *ClassroomController*

At present, *getTopicList* function takes only *profileId* as a parameter and returns all the topics available. This will be updated to take a *classroomId* along with the *profileId*, to return only the topics filtered by the selected classroom.

```
/**
 * Retrieves a data provider of topic list based on the provided classroom ID and
 * the locale for translation content associated with the specified profile ID.
 *
 * @param profileId of the profile to determine the translation content locale.
 * @param classroomId of the selected classroom.
 * @return A [DataProvider] that asynchronously provides a [TopicList].
 */
fun getTopicList(
  profileId: ProfileId,
  classroomId: String
): DataProvider<TopicList>
```

All accompanying functions such as *createTopicList, loadTopicListFromJson, createEphemeralTopicSummary, createTopicSummary* & *createTopicSummaryFromJson* will be migrated from *TopicListController* to *ClassroomController* along with their associated tests. The only change here would be that the topic list will be retrieved from the selected classroom textproto or JSON and not from a universal topic list as done presently.

The *classroomId* will be passed from the ***ClassroomSummaryViewModel*** via the listener, fragment, and its presenter to the controller to fetch the topic list. The state will not be cached in the controller. When the user navigates to the ***TopicActivity*** the *classroomId* will also be passed along with the *topicId* and *storyId*. When returning back to the ***ClassroomListActivity***, the *classroomId* will be passed back and that respective classroom will be selected as default.

Modification of *ProfileManagementController* to track last selected classroom

The following update and retrieve functions will be added to the ***ProfileManagementController*** for the ***lastSelectedClassroomId*** field.

```
/**
 * Updates the last selected classroom ID for the specified profile.
 *
 * @param profileId The ID of the profile to update.
 * @param classroomId The ID of the classroom selected by the user.
 * @return A [DataProvider] representing the asynchronous operation result.
 */
fun updateLastSelectedClassroomId(
    profileId: ProfileId,
    classroomId: String
): DataProvider<Any?>

/**
 * Retrieves the last selected classroom ID for the specified profile.
 *
 * @param profileId The ID of the profile to retrieve the last selected
 * classroom ID for.
 * @return A [DataProvider] containing the last selected classroom ID.
 */
fun retrieveLastSelectedClassroomId(
    profileId: ProfileId
): DataProvider<String>
```

Modification of *PlatformParameterModule*(s) to provide the feature flag

The changes to the ***PlatformParameterModule***, ***PlatformParameterAlphaModule***, and ***PlatformParameterAlphaKenyaModule*** will introduce a new function tailored to accommodate different build variants supported by Oppia. These modules cater to distinct build configurations, where the status of feature flags may vary based on the build variant. For instance, in alpha builds, new features are typically enabled by default, a behavior subject to potential alterations influenced by other factors.

```
@Provides
@EnableMultipleClassrooms
fun provideEnableMultipleClassrooms(
  platformParameterSingleton: PlatformParameterSingleton
): PlatformParameterValue<Boolean> {
  return platformParameterSingleton
    .getBooleanPlatformParameter(ENABLE_MULTIPLE_CLASSROOMS)
    ?: PlatformParameterValue.createDefaultParameter(
      ENABLE_MULTIPLE_CLASSROOMS_DEFAULT_VALUE
    )
}
```

The newly introduced function as shown above, annotated with ***@Provides*** and ***@EnableMultipleClassrooms***, will facilitate the retrieval of the feature flag's status. It will leverage the *PlatformParameterSingleton* to access platform parameters, allowing for dynamic determination of whether multiple classrooms are enabled. In cases where the flag is not explicitly set, a default value, *ENABLE_MULTIPLE_CLASSROOMS_DEFAULT_VALUE*, will be returned to ensure consistent behavior across different build variants.

Additionally, the ***TestPlatformParameterModule*** will be updated to accommodate the testing requirements associated with the new feature flag. This module will be responsible for enforcing specific values for the feature flag during test execution, thereby **ensuring comprehensive test coverage across different scenarios and configurations**.

UI changes

In this section the following points will be discussed:

- Introduction of Classroom Selection Screen.
- Data Flow for User Interactions.
- Update recently played cards to include classroom information.
- Migration of subpackages from home package to classroom package.
- Gate the new *ClassroomListActivity* to replace the existing *HomeActivity*.

Introduction of Classroom Selection Screen

The new classroom selection screen will require us to introduce the following files

- *ClassroomListActivity* & its presenter, view & test files
- *ClassroomListFragment* & its presenter, view & test files
- *ClassroomListViewModel*

The *ClassroomListActivity*, like any other activity in the application, will contain a *Toolbar & FrameLayout* to host *ClassroomListFragment*. Since, this activity will be a replacement for the existing *HomeActivity*, it will also host a navigation drawer.

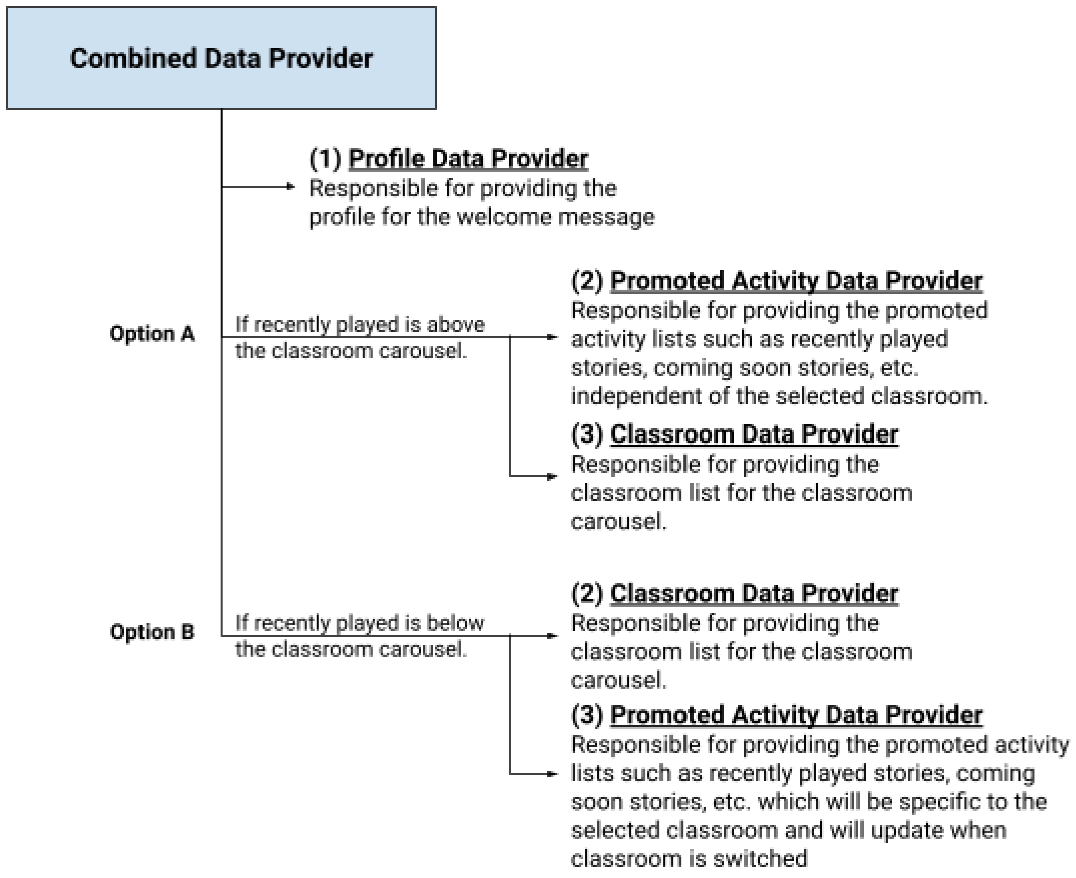
The *ClassroomListFragment*, unlike the *HomeFragment*, will contain a *ComposeView* which will serve as a universal view for displaying recommended stories, classrooms, and topic lists. The *ComposeView* will host a *LazyColumn* (equivalent to *RecyclerView*). This *LazyColumn* will be rendering all the item view models, along with the sticky header for classroom carousel.

The data for the *LazyColumn* will be provided by the *ClassroomListViewModel* in the form of a *LiveData*. The view model will be responsible for combining several *DataProviders*. The combined *DataProviders* will include profile data (from *ProfileManagementController*), recommended stories data (from *TopicListController*), classroom list data (from *ClassroomController*).

The *LazyColumn* of Jetpack Compose has an experimental feature *stickyHeader*, which can host any *@Composable* content as demonstrated below.

```
@Composable
fun ClassroomListScreen(homeItemViewModelList: List<HomeItemViewModel>) {
  LazyColumn {
    homeItemViewModelList.forEach {
      when (it) {
        is WelcomeViewModel -> item { WelcomeText() }
        is PromotedStoryListViewModel ->
          item { PromotedStoryCarouselWithHeading() }
        is ClassroomSummaryViewModel ->
          stickyHeader { ClassroomCarouselWithHeading() }
        is TopicSummaryViewModel -> item { TopicList() }
        else -> {}
      }
    }
  }
}
```

The recently played stories carousel could be placed above the classroom carousel, making it independent of the selected classroom. Alternatively, it could be displayed below the classroom carousel. In this case, the recently played stories would update when the selected classroom is switched, along with the topic list.


Combined Data Provider
(1) Profile Data Provider
Responsible for providing the profile for the welcome message
Option A
If recently played is above the classroom carousel.
(2) Promoted Activity Data Provider
Responsible for providing the promoted activity lists such as recently played stories, coming soon stories, etc. independent of the selected classroom.
(3) Classroom Data Provider
Responsible for providing the classroom list for the classroom carousel.
Option B
If recently played is below the classroom carousel.
(2) Classroom Data Provider
Responsible for providing the classroom list for the classroom carousel.
(3) Promoted Activity Data Provider
Responsible for providing the promoted activity lists such as recently played stories, coming soon stories, etc. which will be specific to the selected classroom and will update when classroom is switched

Data flow for User Interactions

There will be majorly 3 types of data flow:

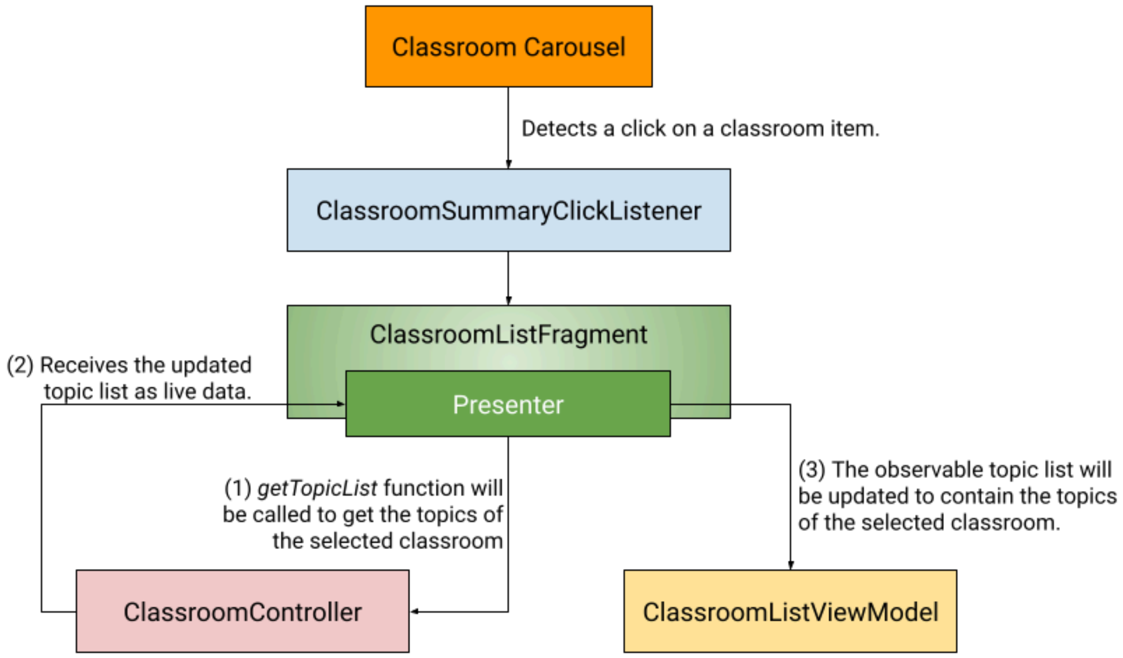
1. Let's discuss how the data flow will occur in this situation when the user logs into their profile and the classroom selection screen is displayed (as a replacement of the existing home screen).

   The classroom selection screen will consist of a single *LazyColumn*. The data of the *LazyColumn* will be a live data (***homeItemViewModelListLiveData***) of the above mentioned ***homeItemViewModelListDataProvider***. Whenever there is a change in the individually combined data providers (for example, ***classroomSummaryListDataProvider***), the live data will be updated and the corresponding UI will re-render accordingly.

   The promoted activity list data will be fetched from the ***TopicListController***. The classroom list and topic list data will be fetched from the ***ClassroomController***. The default selected classroom will be the user's last selected classroom which will be fetched via ***ProfileManagementController***. In case, the fetched classroom ID is empty or invalid, the first classroom in the classroom list (probably Maths classroom) will be selected as default.

2. Coming to the data flow that will occur when the user clicks/selects a classroom from the classroom carousel. A ***ClassroomSummaryClickListener*** will be attached to the fragment. Upon detecting a click on a classroom item, the *onClassroomSummaryClicked* in the above listener will be triggered passing the ***ClassroomSummary*** object from the ***ClassroomSummaryViewModel***.

Once the fragment receives the trigger, it will pass on to its corresponding presenter to handle it. The presenter will be calling the ***getTopicList*** function from the ***ClassroomController***.


Classroom Carousel
Detects a click on a classroom item.
ClassroomSummaryClickListener
ClassroomListFragment
Presenter
(2) Receives the updated topic list as live data.
(1) *getTopicList* function will be called to get the topics of the selected classroom
(3) The observable topic list will be updated to contain the topics of the selected classroom.
ClassroomController
ClassroomListViewModel

3. When a topic from the topic list is clicked, the ***TopicSummaryClickListener*** will be triggered and the user will be redirected to the ***TopicActivity***. Similar to what we already have for the home screen.

Update recently played cards to include classroom information

The *recently_played_story_card* and *promoted_story_card* will be updated to add a Jetpack Compose's *Text* component for the classroom title.

Migration of subpackages from home package to classroom package

The *promotedlist*, *recentlyplayed*, and *topiclist* subpackages will be migrated to the new classroom package. All the item viewmodels used for these views are inherited from the abstract class ***HomeItemViewModel***. To ensure generalization when both the Home screen and Classroom Screens are dependent on this class, we can rename it to ***TopicItemViewModel*** and update any KDoc comments referencing only the home to include both home and classroom screens. However, this generalization might not be necessary as the home screen-related files will be completely removed by the end of this project.

Gate the new *ClassroomListActivity* to replace the existing *HomeActivity*

The *HomeActivity* can be launched from any of the following screens:

- *PinPasswordActivity*
- *ProfileChooserFragment*
- *NavigationDrawerFragment*
- *MyDownloadsActivity*

Each of the above mentioned locations, will be updated to either launch the *ClassroomListActivity* or the *HomeActivity* depending on the ***enableMultipleClassrooms*** feature flag as shown below.

```
activity.startActivity(
  if (enableMultipleClassrooms.value)
    ClassroomListActivity.createClassroomListActivity(activity, profileId)
  else
    HomeActivity.createHomeActivity(activity, profileId)
)
```

Test data changes

To ensure the successful testing of the new multiple classroom support feature, it is essential to incorporate the following test data:

- ***classrooms.json***: This file will contain the list of classroom IDs of all available classrooms.

```
{
  "classroom_id_list": [
    "test_classroom_id_0",
    "test_classroom_id_1",
  ]
}
```

- ***${classroomId}.json***: This file will contain the details of that particular classroom. Here is a sample of the file which might change later as per requirement.

```
{
  "id": "test_classroom_id_0",
  "translatable_title": {
    "html": "Maths",
    "content_id": "title",
  },
  "topic_ids": [
    "GJ2rLXRKD5hw",
    "omzF4oqgeTXd",
  ]
}
```

- ***${topicId}.json***: This file will be updated to contain the reference of the classroom of which it is a part of. Additionally, it is recommended to augment the test data with a few additional topics to ensure a sufficient number of topics per classroom which will enhance the robustness of the testing process.

[**Note:** Textproto files will also be added along with the above mentioned JSON files.]

Testing library changes

The following files will require update/migration of existing test cases, or addition of new tests:

- *ClassroomListFragmentTest*
- *ClassroomControllerTest*
- *TopicListControllerTest*
- *HomeActivityTest*

The *ExplorationContext* will be updated to include *classroomId* parameter. The corresponding *EventLog* subjects to use for testing the context.

Script & CI changes

The project does not require any Script & CI changes apart from the testing portion which shall be covered as a part of writing tests for the new UIs and the *ClassroomController*.

Documentation changes

The project does not require any Documentation changes.

Metrics Plan

Event (see PRD)	**Event parameters (see PRD)**	**Do we already record the event + parameters?** ● If so, please link to the corresponding code on GitHub. ● If not, describe the changes needed to do so.
Lesson is interacted with (can be lesson start, question answered, lesson finished)	Interaction timestamp, classroom ID, lesson state	Yes, we already record the following events: ● Lesson start: We don't record this. Further details are discussed in the following row. ● Answer Submitted: here ● Lesson Finished: here For parameters, the **timestamp** is already recorded. But the **classroom ID** parameter will be added in the *ExplorationContext*. **Lesson state** can be determined by the event which is logged when the calculations are made using Firebase Query.
Lesson is started	Timestamp, lesson ID, classroom ID	A new log will be introduced in *ExplorationProgressController*, when a new exploration is started. The event will be logged here, along with the *ExplorationContext* and additional **classroom ID**.
Lesson is completed	Timestamp, lesson ID, classroom ID	As mentioned and linked above, we record the event here. **Timestamp** and **lesson ID** parameters are already recorded. But the **classroom ID** parameter will be added in the *ExplorationContext*.

Testing Plan

Acceptance tests for core user flows

#	Test name	Initial setup steps	Steps	Expectations
1.	Log in the profile when the feature flag is disabled.	Install the app (with multiple classroom feature flag disabled).	Launch the app.	The onboarding screen of the app is displayed.
			Skip/Complete the onboarding.	The profile selection screen is displayed.
			Log into a profile.	The existing home screen is displayed with a list of mathematics topics and no option to switch classrooms.
2	Log in the profile when the feature flag is enabled.	Install the app (with multiple classroom feature flag enabled).	Launch the app.	The onboarding screen of the app is displayed.
			Skip/Complete the onboarding.	The profile selection screen is displayed.
			Log into a profile.	A new classroom selection screen is displayed with a carousel of classrooms and a list of topics of the selected classroom.
3.	Scroll down the new classroom selection screen.	Install the app (with multiple classroom feature flag enabled).	Launch the app.	The profile selection screen is displayed.
			Log into a profile.	The classroom selection screen is displayed as following: ● A greeting message will be displayed at the top. ● Followed by a carousel of recommended stories. ● Followed by a carousel of classrooms. ● Followed by a list of topics of the selected classroom.
			Scroll down the screen.	The classroom selection screen is updated as following: ● The top greeting message and

				recommended story carousel is scrolled above the screen. ● The classroom carousel sticks to the top of the screen irrespective of how much the screen is scrolled down. ● Followed by a list of topics of the selected classroom.
4.	Switch Classrooms	Install the app (with multiple classroom feature flag enabled).	Launch the app.	The profile selection screen is displayed.
			Log into a profile.	The classroom selection screen is displayed.
			Click on a classroom card.	The list of topics below is changed according to the selected classroom.
5.	Resume the classroom selection screen. (Coming back to the classroom selection screen from another screen)	Install the app (with multiple classroom feature flag enabled).	Launch the app.	The profile selection screen is displayed.
			Log into a profile.	The classroom selection screen is displayed.
			Switch to a different classroom.	The list of topics of the selected classroom is displayed.
			Click on a topic card from the list.	The topic details screen is displayed.
			Return back to the previous screen.	The previously selected classroom is still selected.
6.	Resume the classroom selection screen. (Relaunching the app after closing it)	Install the app (with multiple classroom feature flag enabled).	Launch the app.	The profile selection screen is displayed.
			Log into a profile.	The classroom selection screen is displayed.
			Switch to a different classroom.	The list of topics of the selected classroom is displayed.
			Close and relaunch the app.	The previously selected classroom is selected by default.

7.	**Recently played is per classroom and within the classroom section when a classroom is selected.**			
	Recently played stories section updates with the selected classroom.	Install the app (with multiple classroom feature flag enabled).	Launch the app.	The profile selection screen is displayed.
			Log into a profile.	The classroom selection screen is displayed.
			Play a few stories of a topic of different classrooms.	
			Logout and relogin into the previously logged-in profile.	A recently played stories section is displayed below the classroom carousel which contains only the recently played stories of the selected classroom.
			Switch to a different classroom.	The recently played stories section is updated to display the recently played stories of the switched classroom.
	Recently played is at the top, above the list of classrooms.			
	Recently played stories section does not update with the selected classroom.	Install the app (with multiple classroom feature flag enabled).	Launch the app.	The profile selection screen is displayed.
			Log into a profile.	The classroom selection screen is displayed.
			Play a few stories of a topic of different classrooms.	
			Logout and relogin into the previously logged-in profile.	A recently played stories section is displayed above the classroom carousel which contains all the recently played stories irrespective of the selected classroom.
			Switch to a different classroom.	The recently played stories section does not update.

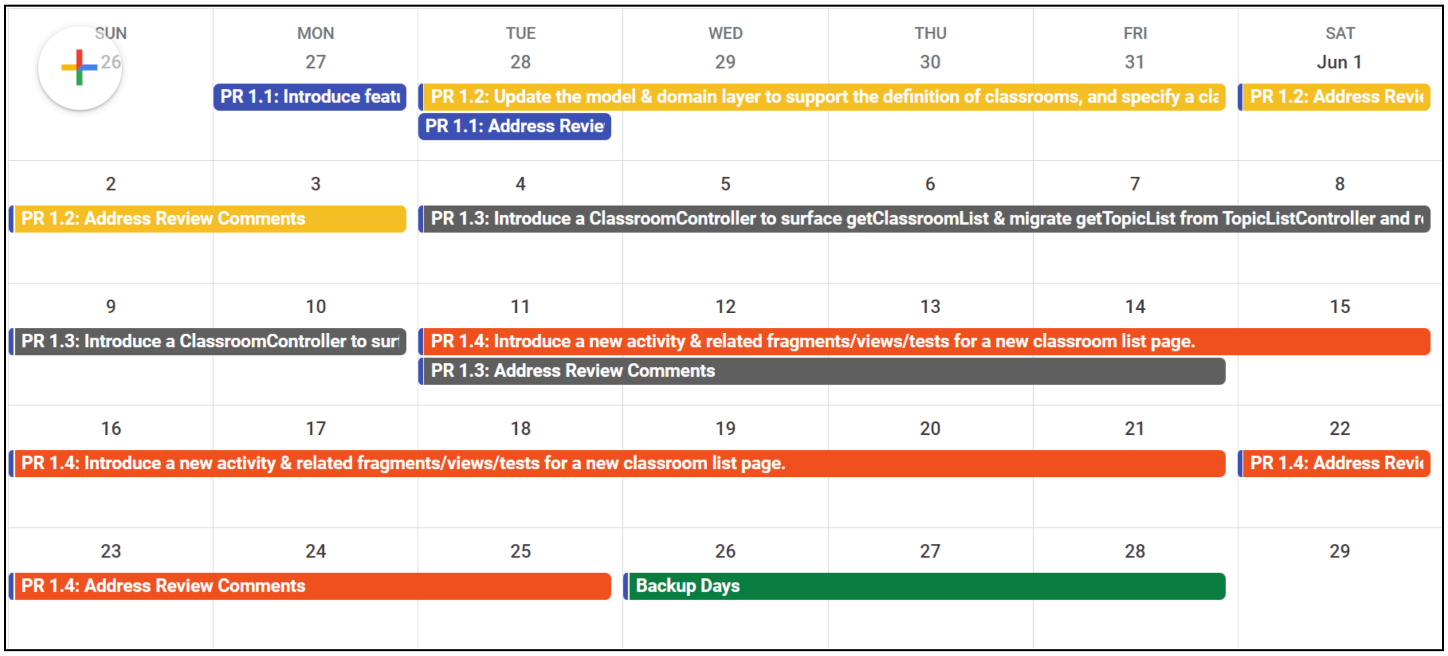
Implementation Plan

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

Milestone 1 (May 27 to June 28)

Key objective: Introduce UI and domain logic for classroom selection.

No.	Description of PR / action	Prereq PR numbers	Target date for starting work on PR	Target date for PR creation	Target date for PR to be merged
1.1	Introduce a new feature flag for the multiple classrooms feature.	None	May 27	May 27	May 28
1.2	Update the model & domain layer to support the definition of classrooms, and specify a classroom per-topic.	None	May 28	May 31	June 3
1.3	Introduce a *ClassroomController* to surface *getClassroomList* & migrate *getTopicList* from *TopicListController* and related tests will be added & migrated respectively.	1.2	June 4	June 10	June 14
1.4	Introduce a new activity & related fragments/views/tests for a new classroom list page and gate the new activity.	1.2, 1.3	June 11	June 21	June 25

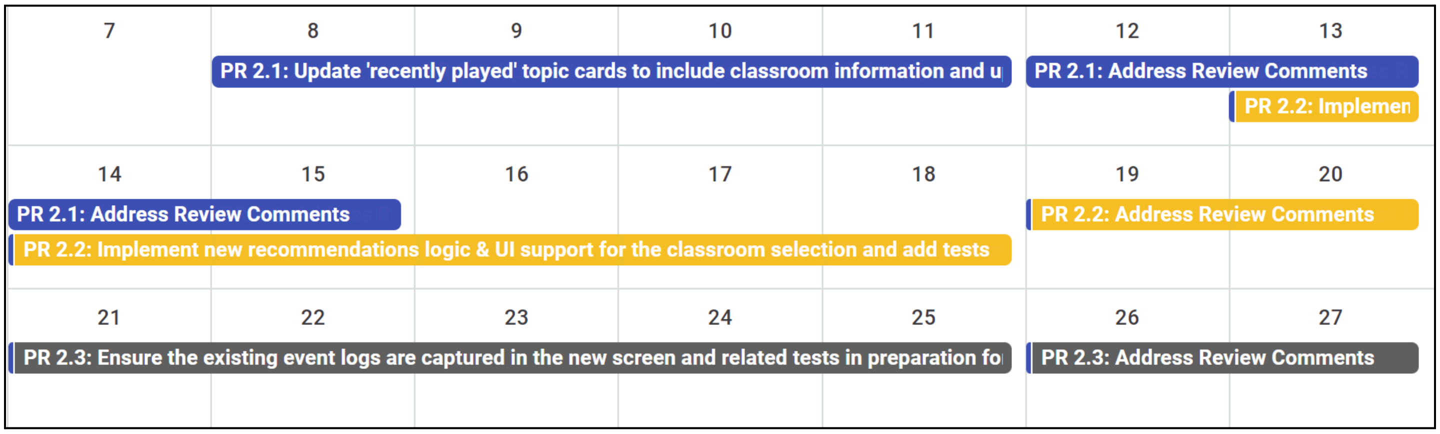


SUN	MON	TUE	WED	THU	FRI	SAT
26	27	28	29	30	31	Jun 1
	PR 1.1: Introduce feat	PR 1.2: Update the model & domain layer to support the definition of classrooms, and specify a cla PR 1.1: Address Revie				PR 1.2: Address Revi
2	3	4	5	6	7	8
PR 1.2: Address Review Comments		PR 1.3: Introduce a ClassroomController to surface getClassroomList & migrate getTopicList from TopicListController and r				
9	10	11	12	13	14	15
PR 1.3: Introduce a ClassroomController to sur		PR 1.4: Introduce a new activity & related fragments/views/tests for a new classroom list page. PR 1.3: Address Review Comments				
16	17	18	19	20	21	22
PR 1.4: Introduce a new activity & related fragments/views/tests for a new classroom list page.						PR 1.4: Address Revi
23	24	25	26	27	28	29
PR 1.4: Address Review Comments			Backup Days			

Milestone 2 (July 8 to August 12)

Key objective: Update topic cards UI, implement new recommendation logic. Complete and launch the feature.

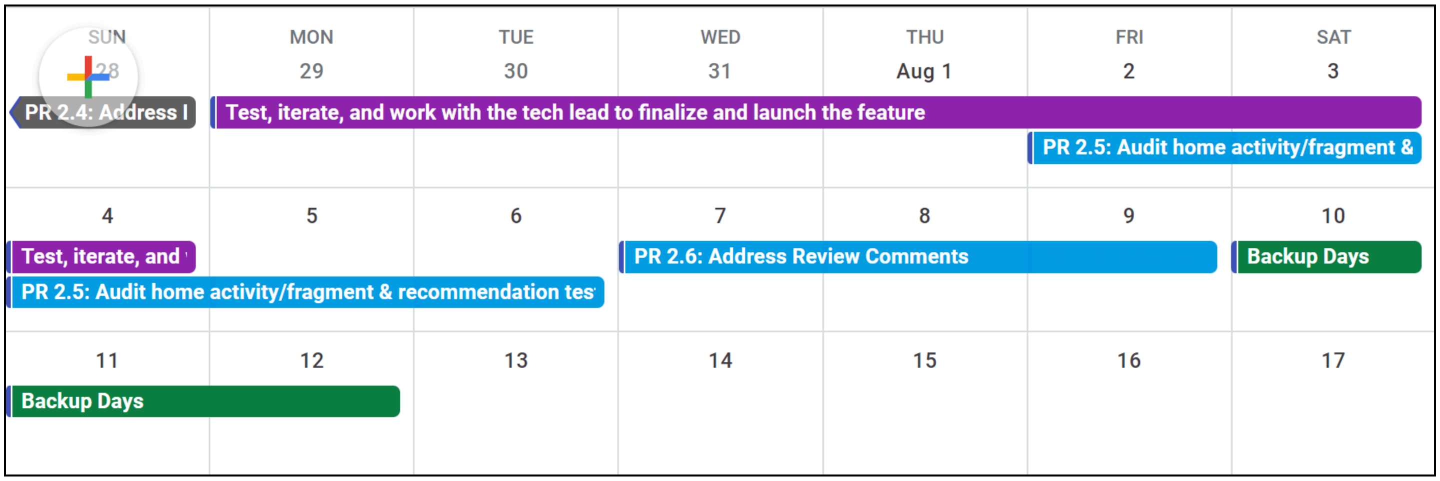
No.	Description of PR / action	Prereq PR numbers	Target date for starting work on PR	Target date for PR creation	Target date for PR to be merged
2.1	Update 'recently played' topic cards to include classroom information and update tests.	1.2, 1.3	July 8	July 11	July 15
2.2	Implement new recommendations logic & UI support for the classroom selection and add tests.	1.3	July 13	July 18	July 20
2.3	Ensure the existing event logs are captured in the new screen and related tests in preparation for removal & implement new event log.	1.4	July 21	July 25	July 28
2.4	Test, iterate, and work with the tech lead to finalize and launch the feature.	1.1 — 1.5, 2.1 — 2.3	July 29 - August 4		
2.5	Audit home activity/fragment & recommendation tests to ensure the new utilities cover the same behaviors. Remove the old home activity/fragment.	1.1 — 1.5, 2.1 — 2.3	August 2	August 6	August 9



7	8	9	10	11	12	13
	PR 2.1: Update 'recently played' topic cards to include classroom information and u				PR 2.1: Address Review Comments	PR 2.2: Implemen

14	15	16	17	18	19	20
PR 2.1: Address Review Comments					PR 2.2: Address Review Comments	
PR 2.2: Implement new recommendations logic & UI support for the classroom selection and add tests						

21	22	23	24	25	26	27
PR 2.3: Ensure the existing event logs are captured in the new screen and related tests in preparation fo					PR 2.3: Address Review Comments	



SUN 28	MON 29	TUE 30	WED 31	THU Aug 1	FRI 2	SAT 3
PR 2.4: Address I	Test, iterate, and work with the tech lead to finalize and launch the feature					
					PR 2.5: Audit home activity/fragment &	

4	5	6	7	8	9	10
Test, iterate, and			PR 2.6: Address Review Comments			Backup Days
PR 2.5: Audit home activity/fragment & recommendation tes						

11	12	13	14	15	16	17
Backup Days						

Future Work

One of the primary future work includes adding spotlighting to the new Classroom List screen to ensure that users are not having trouble understanding the flow of the screens.