

GSoC'22 Proposal: Interactive Onboarding Flow

By **Jishnu Goyal**

Section 1: About You

What project are you applying for?

Project 6.2. - Interactive Onboarding Flow

Why are you interested in working with Oppia, and on your chosen project?

Oppia's mission is to help anyone learn anything they want in an effective and enjoyable way. With this vision in mind, I wish to contribute to oppia to make free, quality education accessible and enjoyable for students all across the world. I also choose Oppia because of the talented and experienced team working here. I believe I will be able to learn a lot by working with them.

Prior experience

My name is Jishnu Goyal and I started Android Development when the lockdown first started in March 2020.

I have been contributing to Oppia since October 2021 and have learnt a lot in these 6 months.

1. The **first app** I ever built is called [Assignments](#) (5000+ installs) [Firebase] which enables teachers and students in sending and receiving assignments and receiving a grade in the 2020 lockdown (available on Google PlayStore)
2. Interned with [UpTodd](#) under supervision of graduates from **MIT, Stanford** and **IITs**, building a highly scalable app in a team of 3. ([Link to the app](#))
3. I believe in sharing what I learn - I run a [YouTube Channel](#) to teach what I learn, and create the content in a way tailored for beginners to grasp easily. I also have an [Instagram Page](#) where I regularly share good coding practices and tips in Android.
4. **First Prize winner** at State Level Science Innovation contest - promoted for **National Level**. An **Arduino** based health tracking device. ([Link](#))

5. Created a **hand-gesture-universal remote** control system using Arduino to control devices and appliances.
6. I love to compose digital music at my home studio and play football in my free time

My Contributions at Oppia:

PRs:

1. <https://github.com/oppia/oppia-android/pull/4081>: Entirely new tested utility "ProfileNameValidator" introduced.
2. <https://github.com/oppia/oppia-android/pull/4204>: Engineered a way to affect platform parameter values before consuming those in tests; Hide general settings options and gate this functionality by introducing a new platform parameter.
3. <https://github.com/oppia/oppia-android/pull/3977>

Issues created:

1. <https://github.com/oppia/oppia-android/issues/3993>

Project size

Large (~350 hours)

Project timeframe

I'll be working during the default GSoC coding period, i.e June 13 - September 12

Contact info and timezone(s)

Name: Jishnu Goyal

University: Maharaja Institute Of Technology, Delhi

Country: India

Email: jishnugoyal007@gmail.com

Github: <https://github.com/JishnuGoyal>

Timezone: Indian Standard Time (IST) (+5:30 GMT)

Preferred method of communication: Hangouts, e-mail and Gitter.

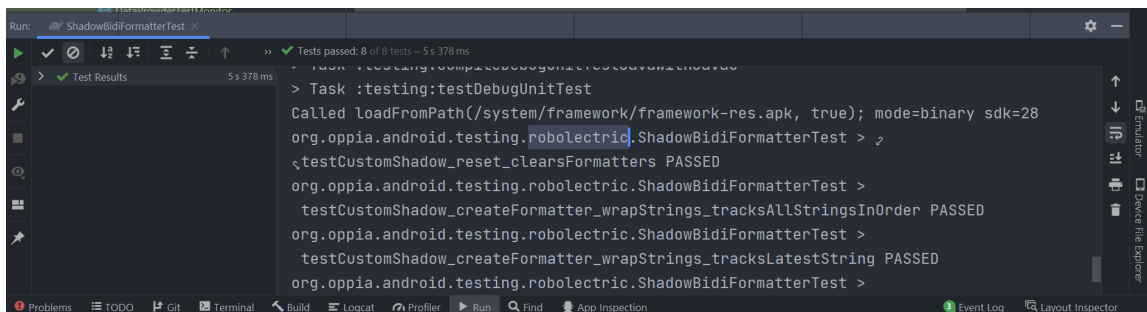
Time commitment

I am committed to spending 5-6 hours a day on this project on weekdays (Monday to Saturday). The time I devote to the project on Sunday will depend upon the work to be completed in the project in that week. In total, I am committed to working at least 35 hours a week.

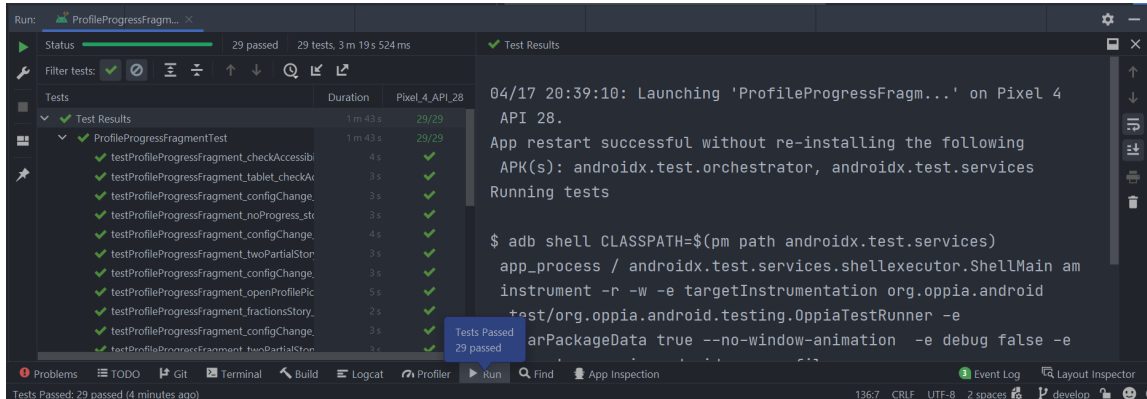
Essential Prerequisites

Answer the following questions (for Oppia Android GSoC contributors):

- I am able to run a single Robolectric test target on my machine via Android Studio. (Show a screenshot of a successful test.)



- I am able to run a single Espresso emulator test target on my machine via Android Studio. (Show a screenshot of a successful test.)



Other summer obligations

During the second and third week of June, I'll be having my End term examinations and will be able to give less time to the project. Apart from that I don't have any other obligations.

Communication channels

I am comfortable with any mode of communication that the mentor chooses, be it email, gitter or gmeet. Mentors can expect a response from me in about an hour.

Section 2: Proposal Details

Problem Statement

Link to PRD (or N/A if there isn't one)	Oppia Android Lightweight Onboarding PRD
Target Audience	Learners using the Oppia android app for the first time
Core User Need	<p>The current onboarding workflow that the Oppia Android app offers is not intuitive enough according to the study conducted mentioned in this PRD.</p> <p>Users get confused or spend more time than necessary figuring out the features and how to navigate in the app due to lack of a better onboarding workflow.</p> <p>Sometimes users do not use a feature available to them that can ease their learning journeys due to lack of information</p>
What goals do we want the solution to achieve?	<p>We want the best possible onboarding experience for a user that experiences the app for the first time. The goal is to create an onboarding experience using spotlights to help users easily navigate through the app and be able to start their learning process with minimum effort.</p> <p>We do this by not only creating a spotlight experience but also by redesigning some parts of the UI which are otherwise confusing, or not as intuitive as they can be. The project includes adding, removing, hiding and redesigning some UI elements.</p> <p>The spotlight experience created should be easy to understand; shouldn't require too many clicks, shouldn't be text heavy and be easily dismissable.</p> <p>The changes we make to the app should be gated behind a platform parameter where possible.</p>

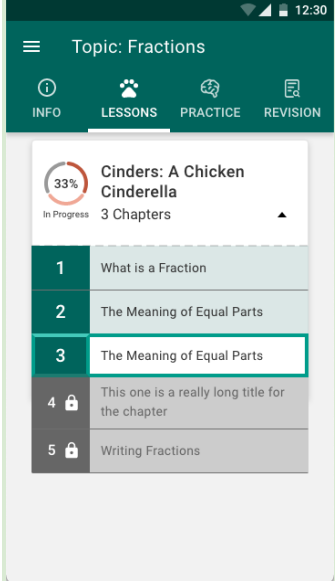
Section 2.1: WHAT

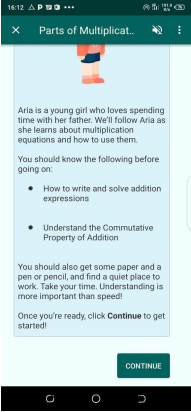
This section enumerates the requirements that the technical solution outlined in "Section 2: HOW" must satisfy.

Key User Stories and Tasks

Please note, the word 'Spotlight' in this context means "Highlighting UI using the Spotlight library"

#	Title	User Story Description (role, goal, motivation) <i>"As a ..., I need ..., so that ..."</i>	Priority ¹	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	Spotlight buttons to navigate through onboarding screens	As a new user, I need to know how to navigate to the next screen or skip the onboarding screen entirely	Must-have	Spotlight 'next' (right arrow) button on the onboarding screen and show a hint for the same	Link
3	Spotlight parts of home screen	As a new user, I need a general idea of how the home screen works	Must-have	When the user opens the app for the first time, show only all topics (and not recommended stories)	Link
				Spotlight a promoted story and tell user about this new section once it appears (after the first visit)	Link
4	Spotlight parts of topic screen	As a new user, I need a general idea of how the topic screen works	Must-have	Remove info and practice tabs. When the user is using the app for the first time, show a spotlight on the lessons tab.. After the user has completed 1 chapter, show a spotlight on the revisions tab. In case the user clicks on the revisions tab before completing the chapter out of curiosity, show a spotlight describing this tab.	Link , Link

				<p>Redesign lessons list.</p> <p>Have the lessons list open by default.</p> <p>Show a lock button that lets the user know which lessons are locked.</p>	
				<p>Spotlight 'lessons' tab.</p> <p>Spotlight a 'lesson' to show how to select a lesson to start learning.</p> <p>When the user has completed at least one lesson, spotlight the revisions tab.</p>	<p>Link</p>
5	Design changes for lessons screen	As a new user, I need to know what all features on the lessons screen are available for me to start learning effectively	Must-have	<p>Change the current speaker icon to be the headphones instead of the current speaker icon.</p>	<p>Link</p>
				<p>Using a spotlight walkthrough to tell the user what this button does. Note that this will not be shown to the user the first time they open the app. It will be shown when the user has already made some measurable progress in the lesson.</p>	<p>Link</p>
				<p>Using a spotlight walkthrough to tell the user how to change the language of the voiceover</p>	<p>Link</p>
				<p>Changing the current hints icon to be animated to attract user attention.</p>	<p>Link</p>

				<p>Implement a "TextInputLayout" where the answer expects a user input. It creates a boundary around the type-able area and also adds a cursor. Moreover there also is a label in a TextInputLayout. This label should be similar to the default placeholders used in the oppia web app.</p> <p>Using the spotlight to highlight that the (X) button should be used in order to exit the lesson. This spotlight also tells the user that their progress shall be saved.</p>	<p>Link</p> <p>Link</p>
		<p>As a learner I find it difficult to navigate through the app at the initial stage. For instance, from the screenshot below, they didn't know they should click on the continue button to proceed.</p> 	<p>should-have</p>	<p>Jiggle the button</p>	<p>Link</p>
6	<p>Design change for revision screen</p>	<p>As a user I should be able to differentiate between lesson and revision tabs</p>	<p>Should-have</p>	<p>Coloring the headers in a different color for revision cards.</p>	<p>Link</p>
		<p>As a user I should be able to to the next revision card</p>	<p>Must-have</p>	<p>Implement next and previous card options as shown in the mock for</p>	<p>Link</p>

		without having to go back to the topic page		quicker navigation	
--	--	---	--	--------------------	--

Technical Requirements

Additions/Changes to Web Server Endpoint Contracts

No new additions/changes are needed to be made to the web server endpoints.

Calls to Web Server Endpoints

No new calls are needed to web server endpoints

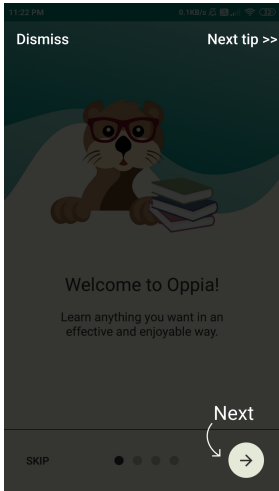
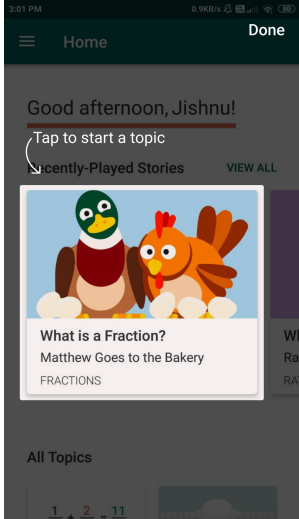
UI Screens

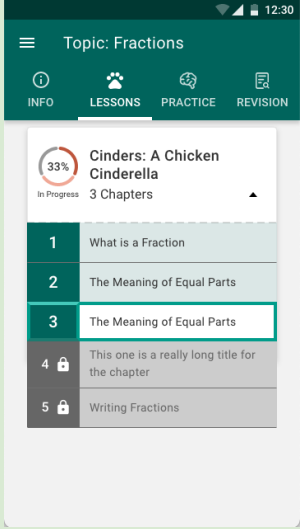
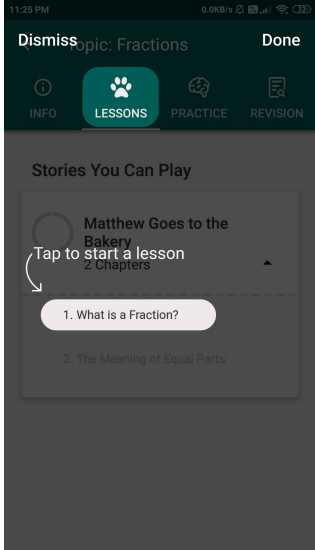
Please find: [Link to all required mocks at a glance](#)

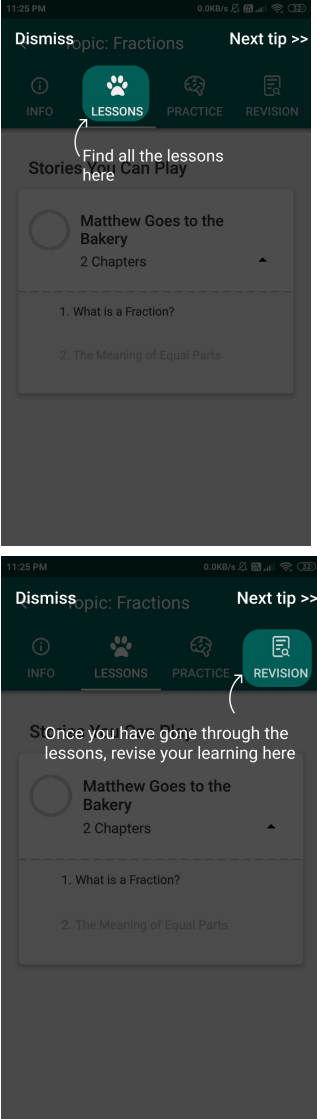
Link to [prototype of entire app flow with spotlight](#)

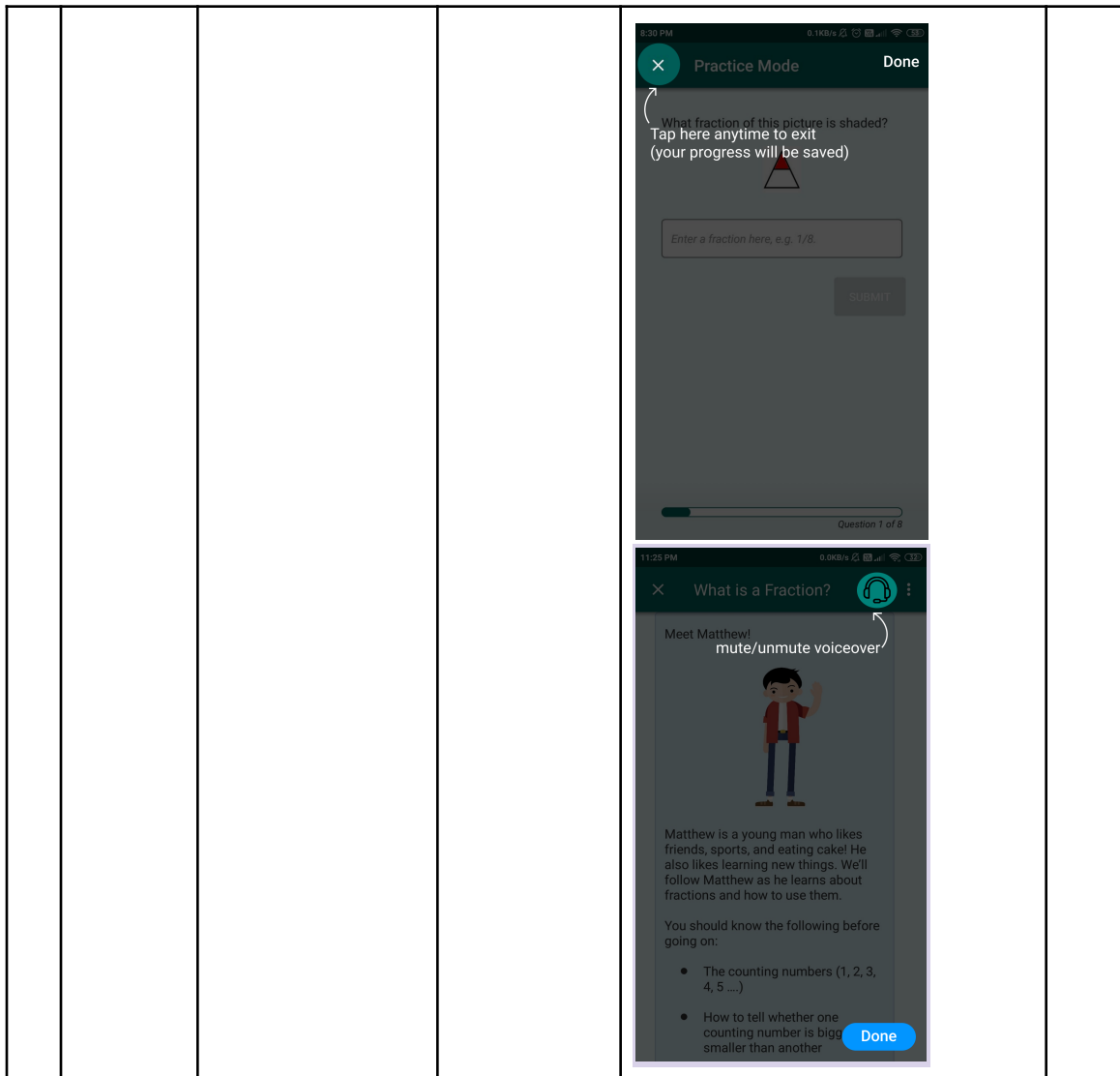
[NOTE: The hint texts shown in the mocks are just for the prototype; The latest hint texts have been added to the spotlight table at the bottom of this doc.]

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

1.	Onboarding screen spotlight	Spotlight screens highlighting the "Next" button	Yes	Mock Prototype 	Yes
3	Home screen spotlight	Spotlight screens highlighting promoted/recently played stories. (This is a quick navigation guiding a new user to view recommended stories when he enters the app for the second time)	Yes	Mock Prototype 	Yes

4	Topic screen spotlight	<ol style="list-style-type: none"> 1) Spotlight screens highlighting tabs such as lessons tab where lesson will be started; revisions tab 2) New design for the chapter list recycler adapter. 	Yes	<p>Mock Prototype</p>  <p>New design for the recycler adapter.</p> 	Yes
---	------------------------	--	-----	--	-----

					
5	Exploration screen spotlight	Spotlight screens highlighting that the user can exit anytime using the (X) button without losing any progress; how audio can be muted/unmuted[this one will be shown when the user has completed at least three cards in the lesson].	Yes	Mock Prototype	Yes



Data Handling and Privacy

#	Type of data	Description	Why do we need to store this data?	Anonymized?	Can the user opt out?	Wipeout policy	Takeout policy
1.	SpotlightView State	The data about which screen's onboarding (walkthrough) has already been seen.	We need this to determine whether to show or not the walkthrough to the user when they open the app. If the user has already seen the walkthrough, do not show it again.	No, this data is specific to the user.	No. This data is required otherwise the app won't know whether to show a walkthrough or not. This data is saved only on the client side, and also there isn't any privacy concern.	Delete when the user deletes their account	N/A, since this data is not saved on the server side.

2.	Login_count	This counts how many times a user has logged into the app.	Visibility of some UI elements is based on how many times the user has entered the app.	No, this data is specific to the user.	No.	Delete when the user deletes their account	N/A, since this data is not saved on the server side.
----	-------------	--	---	--	-----	--	---

Section 2.2: HOW

Existing Status Quo

The current onboarding flow in the Oppia App is not intuitive and sufficient enough for users who are not very familiar with using mobile devices. This leads to users missing out on some features and even getting confused as to how to use the app properly.

The users of the Oppia app will be able to utilize the full potential of the app and would be able to make use of each feature efficiently. They will also get a guided path as to how to start using the app and will become much more familiar with it as a new user.

Solution Overview

This project can be divided into four broad parts:

- 1) Using the spotlight library and creating an onboarding experience for the user – and gating this entire functionality behind a platform parameter.
- 2) Using persistence to record onboarding SpotlightViewStates– This means recording whether or not to show the user the onboarding flow. And also knowing, from ‘where’ to resume the onboarding flow again, in case the user exited the app in-between the onboarding flow. **Essentially, this part is about making a mechanism to save, retrieve and modify SpotlightViewStates.**
- 3) Writing tests that cover the entire onboarding flow. This involves making a new end-to-end test suite that mimics the entire journey a user would take; Unit tests to test the individual onboarding flows.
- 4) Redesigning some elements/app flows to make the user journey smooth such as using gifs/animations to draw user attention at elements that require quick focus (like hints button); redesigning the chapter list recycler adapter and so on, adding elements that guide and ease navigation and accessibility options for the added functionality in the project.

Third-Party Libraries

No.	Third-party library name and version	Link to third-party library	Why it is needed	License ² (if third-party library)	[Android only] Min / target / max SDK version that the library supports
1	Spotlight (v2.0.5)	Link	For the solution to this problem, Spotlight library helps to "highlight" specific parts of the UI to provide a walkthrough which will be used to guide new users	Apache 2.0	Min SDK: 14 Target SDK: 30 Compile SDK: 30

"Service" Dependencies

No new services required.

Impact on Other Oppia Teams

This project has all the changes in the app itself, meaning there are no calls to the server and no data related to this project leaves the app - that being said, the project doesn't impact the other teams in any way.

Key High-Level and Architectural Decisions

Implementing this project will touch the following modules. The module wise implementation of the project is represented as follows.

Added

Modified

1. model module
 - a. `spotlight.proto`
 - i. Addition of protobufs to store the spotlight progress made by the user.
 - b. `profile.proto`
 - i. Addition of an int field 'number_of_logins' to `profile.proto` to store how many times 'this' user has entered the app. We need this information since visibility of some of the spotlights/views depends on whether the user is viewing this screen for the first or second time.
2. domain module
 - a. `SpotlightStateController` **Package: spotlight**

- i. This injectable class is solely responsible for all spotlight related persistence such as recording and retrieving spotlight view states for a given feature and profileId. It is injected inside fragment presenters requiring a spotlight highlight.
 - ii. Addition of new layout files that will be used as [overlays](#).
 - 1. Overlay_top_right.xml → when the anchor position is top right
 - 2. Overlay_top_left.xml
 - 3. Overlay_bottom_right.xml
 - 4. overlay_bottom_left.xml
 - b. ProfileManagementController **Package: profile**
 - i. Add function to increment login count.
 - c. PlatformParameterModule **Package: platformparameter**
 - i. EnableSpotlightUi
 - ii. EnableInfoTabUi
- 3. app module
 - a. SpotlightFragment **Package: spotlight**
 - i. Helper class that automates/encapsulates placement and creation of spotlights and overlay elements. Also automates the domain functionality of recording and retrieving spotlightViewStates.
 - b. TopicFragmentPresenter **Package: topic**
 - i. spotlight lesson and revision tabs
 - ii. hide info tab
 - c. ExplorationActivityPresenter **Package: story**
 - i. change speaker icon to headphone icon in lessons screen (exploration_activity.xml)
 - ii. spotlight 'headphones icon' and back (X) button
 - iii. Add a scaling to the 'continue' button (res/anim/scale.xml Added)
 - d. HomeFragmentPresenter, HomeViewModel **Package: home**
 - i. Hide the promoted stories section when the app is opened for the first time
 - ii. spotlight promoted stories section
 - e. TopicLessonFragmentPresenter **Package: topic.lesson**
 - i. Redesign recycler adapter item (lessons_chapter_view.xml)
 - ii. Add recycler item for locked chapter (lessons_locked_chapter_view.xml)
 - iii. Spotlight first lesson
 - f. RevisionFragmentPresenter **Package: topic.revision**
 - i. Implement next and previous card options as shown in the mock for quicker navigation
 - ii. Color headers in orange color
 - g. ExplorationFragmentPresenter/AudioFragmentPresenter **Package: player**
 - i. Spotlight audio options button (audio_fragment.xml)
 - h. Accessibility

- i. For screens that require a spotlight, we make sure up to date content descriptions are there
- ii. RTL compatibility

Implementation Approach

[Android only] Model

spotlight.proto

ProtoBuffers required to record SpotlightViewStates

In order to record SpotlightViewStates, the following new ProtoBuffers will be added which will hold this information:

The proposed structures use enums to ensure strong type-safe structures to store values and potentially eliminate any chances of typing error.

Each 'feature' that requires a spotlight highlight is added to this protobuf message called Spotlight. Here, for example, onboarding_next_button is a feature. This protobuf is analogous to a sealed class that makes each feature 'one of' a spotlight.

```
// Superclass for all the SpotlightViewStates
message Spotlight {
  oneof feature {
    // feature for onboarding screen next button
    SpotlightViewState onboarding_next_button = 1;

    // feature for topic fragment lessons tab
    SpotlightViewState topic_lesson_tab = 2;

    // feature for topic fragment revision tab
    SpotlightViewState topic_revision_tab = 3;

    // feature for first lesson under lessons tab
    SpotlightViewState first_chapter = 4;

    // feature for promoted stories
    SpotlightViewState promoted_stories = 5;
  }
}
```



```

// feature for hint icon
SpotlightViewState hint_icon = 6;

// feature for back button in exploration screen
SpotlightViewState lessons_back_button = 7;

// feature for voice over icon in exploration screen
SpotlightViewState voiceover_play_icon = 8;

// feature for voice over language options in audio player
SpotlightViewState voiceover_language_icon = 9;
}
}

```

The SpotlightViewState is an enum field. When the user enters the app for the first time, the SpotlightViewState is not seen.

```

// View States for a spotlight
enum SpotlightViewState {
  SPOTLIGHT_VIEW_STATE_UNSPECIFIED = 0;
  SPOTLIGHT_SEEN = 1;
  SPOTLIGHT_NOT_SEEN = 2;
}

```

And each of these will be stored in SpotlightStateDatabase on a per profile basis.

```

// Top level proto for storing SpotlightsViewStates
message SpotlightStateDatabase {
  SpotlightViewState onboarding_next_button = 1;
  SpotlightViewState topic_lesson_tab = 2;
  SpotlightViewState topic_revision_tab = 3;
  ...
}

```

Each feature is listed in this protobuf message.

profile.proto

```
Profile{
  ProfileId id = 1;
  ...
  // add this field
  int number_of_logins = 13;
}
```

Some functionality depends on how many times the user has logged in. We count this using this proto.

[Android only] Domain

SpotlightStateController.kt

The domain layer requires a new SpotlightStateController for spotlights. This injectable class is solely responsible for all spotlight related persistence such as recording and retrieving spotlight view states for a given feature and profileId. It is injected inside fragment presenters requiring a spotlight highlight.

Flow:

- 1) User opens the app
- 2) An activity is started
- 3) If the EnableSpotlightUI feature flag is true, we proceed with the following steps. Otherwise, usual app flow occurs.
- 4) The fragment presenter requests a spotlightViewState from the SpotlightStateController as live data
- 5) The fragment presenter observes the data and initializes the spotlight targets list based on what targets have already been previously shown. Spotlight fragment handles showing of spotlight and notifying the domain layer that the spotlight has been seen.
- 6) When the user clicks 'Next' on any spotlight overlay screen, the SpotlightViewStates is updated and saved by calling spotlightStateController.markSpotlightViewed() inside the spotlight fragment.

```
/** Handles saving and retrieving feature spotlight states. */
class SpotlightStateController @Inject constructor()
```

Function to record SpotlightViewStates

```
/**
 * Marks the [SpotlightViewState] of a spotlit feature for a given profile as seen
 *
 * @param profileId the ID of the profile viewing the spotlight
 * @param feature the spotlight feature who's view state is to be recorded
 * @param viewState SpotlightViewState of this spotlight feature
 */
fun markSpotlightViewed(
    profileId: ProfileId,
    feature: Spotlight.FeatureCase,
)
```

Function to retrieve SpotlightViewStates

```
/**
 * Retrieves the current [SpotlightViewState] of a spotlit feature for a given profile.
 *
 * @param profileId the ID of the profile that will be viewing the spotlight
 * @param the feature to be spotlit
 *
 * @return DataProvider containing the current [SpotlightViewState] corresponding to the specified [feature]
 */
fun retrieveSpotlightViewState(
    profileId: ProfileId,
    feature: Spotlight.FeatureCase,
): DataProvider<SpotlightViewState>
```

ProfileManagementController.kt

In the loginToProfile function, we increment the number_of_logins variable which was defined in the profile.proto file.

PlatformParameterModule.kt

EnableSpotlightUI

```
@Qualifier
annotation class EnableSpotlightUi

/** Default value for the feature flag corresponding to [EnableSpotlightUi]. */
const val ENABLE_SPOTLIGHT_UI_DEFAULT_VALUE = true
```

EnableInfoTab

```
@Qualifier
annotation class EnableExtraTopicTabsUi

/** Default value for the feature flag corresponding to
[EnableExtraTopicTabsUi] */
const val ENABLE_EXTRA_TOPIC_TABS_UI_DEFAULT_VALUE = false
```

[Android only] UI changes

SpotlightFragment.kt

Launching a spotlight to highlight an element

To launch a spotlight, we simply pass the elements that are needed to be spotlighted as a vararg of `SpotlightTargets` to the `SpotlightFragment` and then call the `spotlightFragment`.

```
data class SpotlightTarget(
    val anchor: View,
    val hint: String = "",
    val shape: SpotlightShape = SpotlightShape.RoundedRectangle,
    val feature: Spotlight.FeatureCase
)
```

To pass an `spotlightTargets` to the `SpotlightFragment`, `spotlightFragment.requestSpotlight(spotlightTarget(s))` should be used.

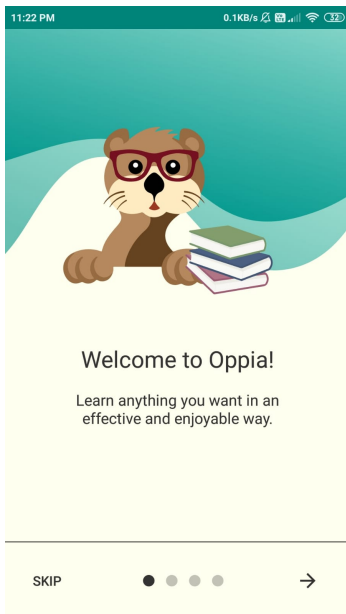
Launching a spotlight (this code should be called from the fragment presenter):

```

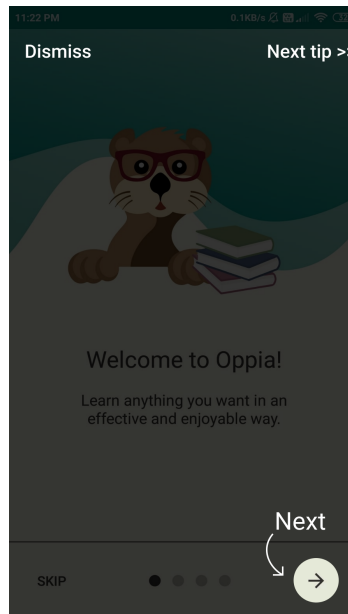
spotlightFragment.requestSpotlight(
    SpotlightTarget(
        binding.onboardingFragmentNextImageView,
        R.string.next,
        SpotlightShape.Circle,
        Spotlight.FeatureCase.ONBOARDING_NEXT_BUTTON
    )
)

activity.supportFragmentManager.beginTransaction()
    .replace(R.id.onboarding_fragment_placeholder, spotlightFragment)
    .commitNow()

```



Main screen



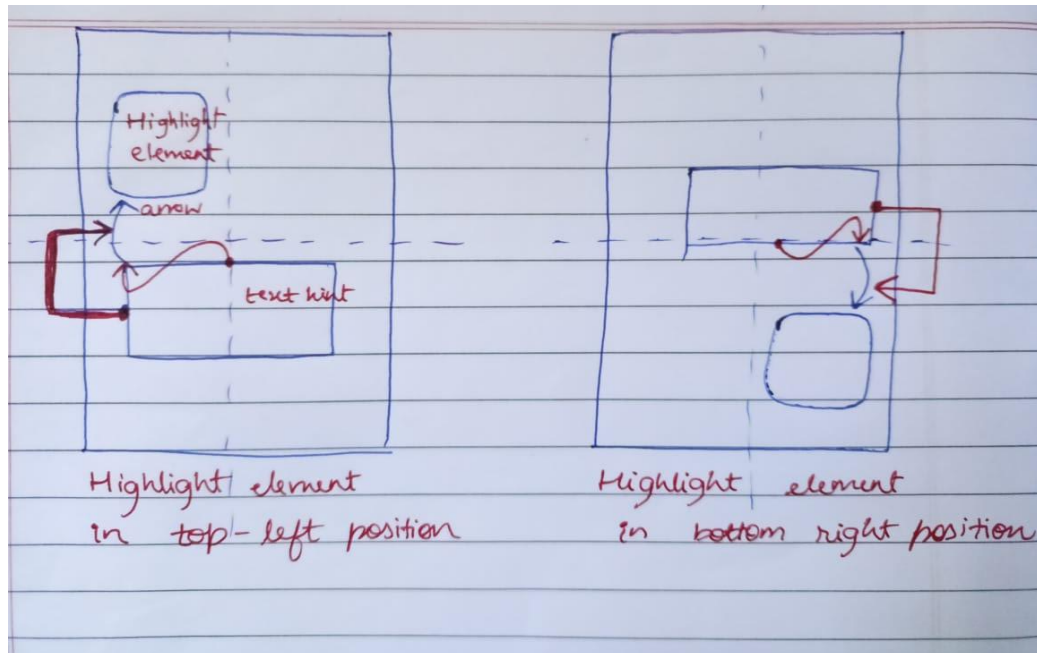
Overlay

The overlay screen and its components:

An overlay screen consists of these components:

- **Done Button:** This button ends the spotlight highlight/switches to the next spotlight in the queue.
- **Highlighted area:** This is the UI element being highlighted.
- **Hint:** This is the extra text on the overlay screen that prompts the user to make a decision.
- **Arrow:** This is a image view containing arrow vector image which points to the highlighted element and from the hint text

The math behind Overlay component placement



Internally, there are 4 overlay files that are selected to inflate based on the position of the anchor (highlightable element).

The diagram above shows two different cases where the anchors could be placed. The other two possible placements of anchor positions are analogous to these.

Now, the red lines show how the text view is constrained to the arrow.

The arrows themselves get their positions by calculating the absolute position of the anchors, taking into account the anchor's width and height.

To summarize:

1. The absolute positions and measurements of the anchors are measured first.
2. Based on that, an overlay fragment is selected.
3. The arrows calculate their positions using math (based on positions measured in step 1).
4. The text views automatically get placed since they are constrained.

The selected overlay fragments are intelligent enough to know the rotation of arrows and the constraint requirements of the text views so they are correctly placed.

TopicFragmentPresenter.kt

To hide Info Tab:

Add an enum field to each topic tab with new positions when there will be only 2 tabs.

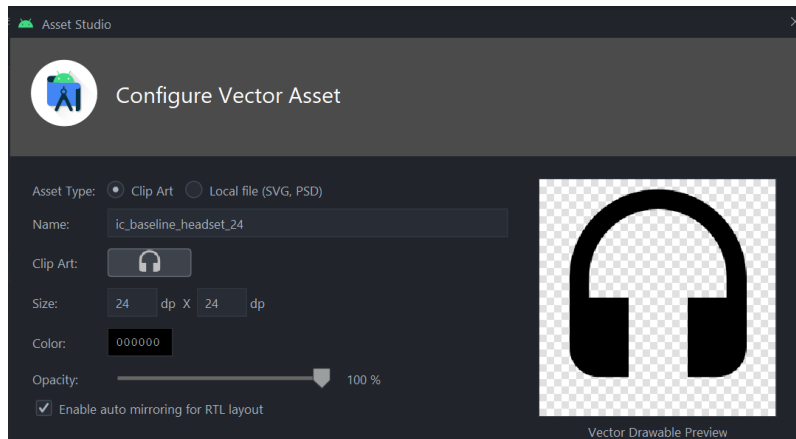
Changes to TopicTab.kt:

The plan is to replace EnablePracticeTabUi with EnableExtraTopicTabsUi which will hide info and practice tabs together.

Now there will be two possible states: 2 tabs and 4 tabs.

ExplorationActivityPresenter.kt

Change the audio_action_player button (imageView) inside exploration_activity.xml from speaker icon to headphones icon. This will require a new vector asset which is already available in android to be used.



We also spotlight this icon.

Scaling the 'Continue' button:

An animation resource file is added (scale_animation.xml)

The plan is to scale the button by first enlarging it then shrinking it back a little. This makes sure that the readability of the button is maintained, even for the slow readers while capturing the attention of the user.

Another idea was to use a jiggle animation, but this might have led to decreased readability for some users.

The button which may be below the fold, will start start animation 30 seconds after it gets visible (it gets visible after the user scrolls till the end of the screen)

For the 30 second timer, the lifecycle-safe timer factory is used.

At the end, we change all the text boxes to textInputLayouts wherever the user is expected to type an input as the answer.

HomeFragmentPresenter.kt

In order to hide the promoted stories section for the first time the user enters the app, we will need to check for the number of logins inside HomeViewModel and populate the homeItemViewModelList according to the number of logins.

This will make sure promoted stories are shown only if the number of logins > 1.

We also Spotlight the promoted stories section if the user is logging in for the second time.

Accessibility

For users requiring Talkback

I plan to completely remove the spotlight flow for the talkback users.

Spotlight is completely a **visual** tool that tells the user “where on the screen a particular element is that should be used now”. And for people using talkback, who can't see what a spotlight is, will only be annoyed/slowed by the spotlight flow, which is totally opposite to what the goal is.

Instead, what I want is the screen reader to focus on some planned UI elements (going in a linear fashion and the linear flow is similar to what we would want a regular user to follow) and start reading the content descriptions of these elements.

Additional concern not included in the table:

Accessibility plan for the hint button: The current plan is to do a 'forced announcement' when the hint button gets visible.

Table to show accessibility plan and side by side comparison with actual spotlight flow

Spotlight in linear flow

The spotlight will be shown based on measured user progress

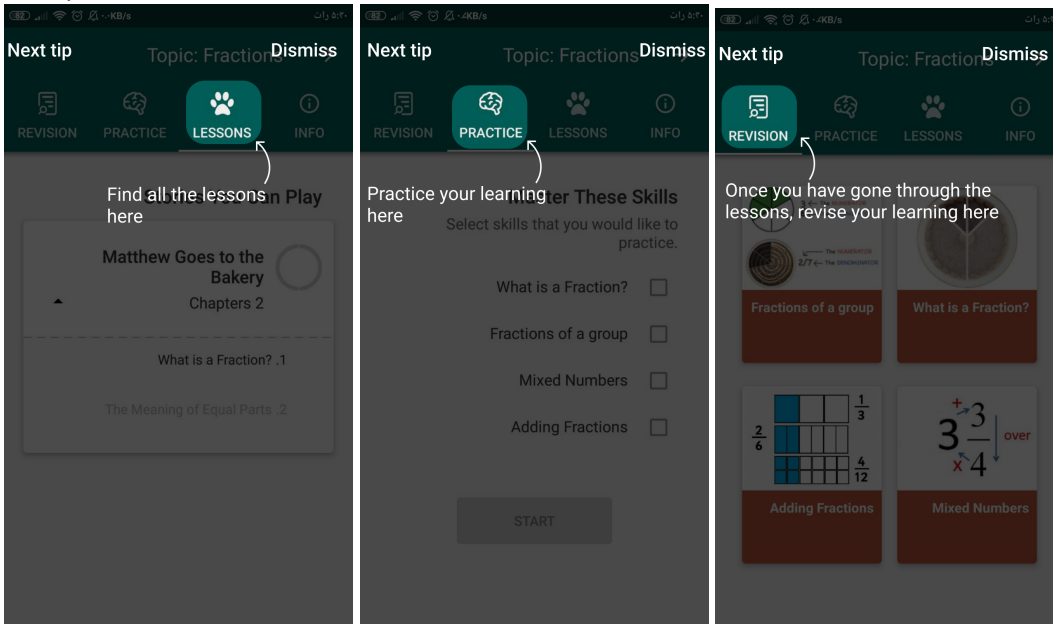
S.no	Spotlight element	Screen name	Accessibility Plan	Hint text	Additional information
1	Next button	Onboarding screen	For talkback users, there wouldn't be any distracting elements on the screen. When they would swipe, they would come down along the 'natural app flow' and find the next and skip buttons. No changes here.	Tap here for the next slide	

2	Promoted story (first card)	Homepage	For the talkback user, we won't be hiding this section. No changes required in context to accessibility.	From now, here you can view stories you might be interested in	This will be shown when the user enters the app for the second time.
3	Lessons tab	Topic screen	Lessons tab content description shall be read first. Added description: "Start learning here under this tab" After that the revision tab description. After that the chapter list. Specialized content description shall be required here to inform them that this is the chapter to start. From the next time onwards, we should make sure that the same content description is not read, as they now know how to start a chapter.	Find all your lessons here	
4	First lesson in the list	Topic screen	Discussed above	Tap to start a chapter	
5	Revision tab	Topic screen	Add content description: "If you a completed a lesson, revise your concepts here in the revision tab"	Revise your concepts here after completing the lesson	Will be shown only when the user enters the topic screen after completing at least one lesson.
6	Exit button (X)	Lesson screen	While going through the natural flow of the app, this button shall be encountered. No changes required. Content description added: "Exit. Your lesson progress will be saved."	Exit anytime using this button. We will save your progress.	

7	Voiceover icon	Lessons screen	Content Description proposed: (updated) “Voice over button. Tap here if you would like Oppia to read the lesson for you.”	Would you like Oppia to read for you? Tap on this button to try!	Will be shown when the user has completed 3 cards in the lesson.
8	Voiceover language options	Lessons screen	Content Description proposed: (updated) “Change voice over language”	Tap here to change voice over language	Will be shown when the user clicks on the voiceover icon.
9	Forced announcement for hint bar added. “Feeling stuck? Try using a hint at the bottom of the screen”				

For RTL users:

RTL topic screens for reference:



Technical requirements for RTL

The plan is to simply use the position, $x_{rtl} = \text{ScreenWidth} - x$ for the arrow (which was previously calculated as x) in order to flip it such that the arrow correctly points to the mirrored element. Note: x is the absolute horizontal position of the element's start on the screen.

The text hint view will automatically follow the arrow as it is constrained to it. Additionally, in case of rtl, we switch from TopRightOverlay to TopLeftOverlay in order to account for the reverse direction of the arrow curvature and reversed constraints of the textView hint.

The requirements mentioned in [this](#) wiki are being taken care of.

[Android only] Test data changes

No new testing data required.

[Android only] Testing library changes

No changes required.

[Android only] Script & CI changes

No changes required.

Documentation changes

Documentation will be added on the wiki for how to create a new spotlight for any UI element.

Testing Plan

E2e testing plan

#	Test name	Initial setup step	Step	Expectation
1.	Walkthrough_screens_spotlight_test	The app is launched for the first time	1.0) The first onboarding screen appears	Spotlight appears highlighting 'Next' button
			1.1) After step (1.0) user clicks on "Done" button	Spotlight finishes. The user may choose to skip or click the next button until the profile screen starts.
			2.0) The user exits the app after spotlight is shown (onboarding not completed)	When the user opens the app again, spotlight is shown again
3	Home_screen_spotlight_test	User logs in For the first time	1.0) The home screen appears	All topics is shown and promoted stories are not shown
			2.0) User opens the app for the second time	Promoted stories are shown. Spotlight highlights the promoted stories.
			3.0) User opens the app for the third time	Spotlight is not shown for the promoted stories

4	Topic_screens_spotlight_test	User taps on any lesson	1.0) topic screen appears	Spotlight highlights the lessons tab.
			1.1)user presses "done"	Spotlight highlights the first chapter
			1.2) user presses done	Spotlight over
			1.2) user completes a chapter and comes back to the lessons tab	Spotlight highlights 'revision tab' and shows a hint
			1.3) "done" button clicked	Spotlight over
5	Exploration_screen_spotlight_test	User taps on any lesson	1.0) Exploration screen starts	Spotlights highlights (X) back button.
			1.1) user opens the app and sees the exploration screen for the third time	Spotlight highlights headphone icon and shows the usage of voiceover feature
			1.2) User clicks on "done" button and then clicks the headphone icon	Spotlight highlights how voice over audio language can be changed
			1.3) "done" button clicked	Spotlight over

Feature testing

Does this feature include non-trivial user-facing changes?

YES

Implementation Plan

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

Milestone 1

No.	Description of PR / action	Prereq PRs	Target date for work start	Target date for PR creation	Target date for PR to be merged
-----	----------------------------	------------	----------------------------	-----------------------------	---------------------------------

1.1	<p>Add a method in profileManagementController to count the number of logins per user</p> <p>Add unit tests</p>		July 15	July 15	July 18
1.2	<p>Lesson Screen:</p> <p>Change speaker icon to headphone icon</p> <p>Add scale animation for continue button</p> <p>Fix placeholders for text based interactions</p> <p>Add unit tests</p>		July 18	July 23	July 26
1.3	<p>Add EnableInfoTabUi platform parameter</p> <p>Modify Topic screen: Hide info and practice tabs; gate this behind a feature flag</p> <p>Add unit tests</p>	1.1	July 23	July 27	July 30
1.4	<p>Modify HomeScreen: Hide promoted story section for the first time user</p> <p>Add CTA for the All topics section</p> <p>Add unit tests</p>	1.1	July 27	Aug 5	Aug 15
1.5	<p>Lessons Tab:</p> <p>Modify the design of recycler adapter list item</p> <p>Create new recycler item view for locked lessons</p> <p>Have lessons list expanded by default</p>		Aug 5	Aug 10	Aug 22

	Add unit tests				
1.6	Revision Screen: Color the revision toolbars in orange; Add “next” and “previous” chapter image views to navigate between revision cards (implementation for mobile devices only) Add unit tests		Aug 10	Aug 20	Sept 2
1.7	Add EnableSpotlightUi Add spotlight.proto Implement SpotlightStateController to record and track states. Add unit tests for this class		Aug 20	Sept 3	Sept 14

Milestone 2

No.	Description of PR / action	Prereq PR numbers	Target date for work start	Target date for PR creation	Target date for PR to be merged
2.1	Implement SpotlightFragment class Implement 4 overlay xml files for different combinations of anchor placements. + Add RTL functionality to all spotlights. Add function to suspend spotlights for talkback	1.8	Sept 20	Sept 26	Oct 5

	users in spotlight fragment Add unit tests				
2.2	Lesson Screen Spotlight 'headphones' icon Spotlight (X) Back button Add unit tests	1.2, 2.1	Oct 1	Oct 3	Oct 7
2.3	Lessons Tab Spotlight first chapter Add unit tests	1.6, 2.1	Oct 3	Oct 5	Oct 8
2.4	Modify HomeScreen Spotlight promoted stories when the user visits the second time Add unit tests		Oct 5	Oct 6	Oct 12
2.5	Modify Topic screen Spotlight lessons and revisions tabs Add unit tests		Oct 8	Oct 7	Oct 18
2.6	Audio Fragment Spotlight voiceover language option Add unit tests		Oct 11	Oct 12	Oct 21
2.7	Add wiki documentation for how to start a new spotlight		Oct 18	Oct 20	Oct 25

2.8	<p>Accessibility:</p> <p>Add content descriptions to all elements requiring spotlight highlights</p> <p>Add forced announcement for hint icon</p> <p>Add unit tests</p>		Oct 20	Oct 23	Oct 31
N/A	<Flexible padding for exams since exact dates aren't known>	N/A		Nov 1-Nov 14 (if exams occur at the end)	

Future Work

- Add spotlight for the “Next” button for onboarding screens and unit test
- Use `textInputLayout` in place of text box in all text based interactions

Appendix

Definitions

ProtocolBuffer (Protobuf) is a free and open-source cross-platform data format used to serialize structured data. It is useful in developing programs to communicate with each other over a network or for storing data.