

Google Summer of Code 2021

Redesigning and Updating the learner dashboard

Krishita Jain

About You

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

“**Education** is the most powerful weapon which you can use to change the world”. I believe that education is a human right and a necessity for modern societies to function. Unfortunately, it is becoming a luxury for the poor, especially in developing and underdeveloped countries. Oppia’s mission to “Provide high-quality education to those who lack access to it” is what attracted me. I wish to contribute to it and give it my all so that students across the world have access to high-quality free lessons.

Oppia’s unique architecture which involves explorations, classrooms and one-on-one tutor/learner conversations make the learners focus and participate actively. It beats the monotony of schools and provides a fun way for the students to learn and explore new things. Apart from this, what I find distinctive about Oppia is that it provides a versatile platform for the creators. With the tools that Oppia provides, they can illustrate their ideas in the form of interactive lessons. Also, It’s been a great learning experience for me so far. I will continue to contribute to Oppia even after the GSoC period ends.

What interests me about this project?

Measuring **progress** to your goal lets you see whether you’ve made a dent, are at a standstill, or have fallen behind. At present, there is no way for a learner to track their progress in different classrooms. This project aims to solve this and also incorporate planning and recommendation of topics, subtopics and stories to the learners.

Prior experience

I am sufficiently familiar with the technologies that Oppia uses. I have been using Python and JavaScript for more than a year now. In particular, I have been using the framework Oppia is built on, Angular, for more than six months now. Some of my open-source projects that showcase my technical skills are [here](#).

I have been participating in competitive coding competitions for more than a year now. My profiles: [CodeChef](#). [CodeForces](#).

Apart from this, I have been actively contributing to Oppia for more than five months now, by creating PRs and filing issues. I am a member of the Learner and Creator Experience team and I have gained sufficient knowledge regarding the parts of the codebase that are relevant to this project.

Some of my best contributions are:

- Rename field `activity_ids` to `exploration_ids` in the `UserSubscriptionsModel` ([#11316](#))
- Remove deprecated field from `UserSubscriptionsModel` ([#11597](#))
- Added "Hint tip" dialogue box ([#11621](#))
- Adding limit to story description([#11732](#))
- Switching order of panes in the history view([#11854](#))

Here is the full [list of PRs](#) and [list of issues](#) I have filed.

Contact info and timezone(s)

Email: jainkrishita15@gmail.com

Github Profile: [krishita30j](#)

I will stay in India throughout the summer. The time zone will be Indian Standard Time (GMT+5:30)

Time commitment

I plan on dedicating 30-35 hours a week to this project.

Due to COVID-19, I am not sure about the dates of my final exams. They will most likely last a week and my work hours would reduce to 2-3 hours a day during that week. But as I plan on starting during the community bonding period, exams won't be an issue.

Essential Prerequisites

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

```
typing-extensions==3.7.4.3 # via apache-beam
typing==3.7.4.3 # via apache-beam, typing-extensions
uritemplate==3.0.1 # via google-api-python-client
urllib3==1.25.11 # via elasticsearch, requests
webapp2==3.0.0b1 # via -r requirements.in
webencodings==0.5.1 # via -r requirements.in, bleach, html5lib
webob==1.8.6 # via webapp2

# The following packages are considered to be unsafe in a requirements file:
# setuptools
All third-party Python libraries are already installed correctly.
Redis-cli is already installed.
ElasticSearch is already installed.
Copying Google Cloud SDK modules to third_party/python_libs...
Checking that all google library modules contain __init__.py files...
Installing buf and protoc binary.
Compiling protobuf files.

yarn install v1.22.10
[1/4] Resolving packages...
[2/4] Fetching packages...
warning Pattern ["@definitelytyped/typescript-versions@latest"] is trying to unpack in the same destination "/Users/krishitajain/Library/Caches/Yarn/v6/npm-@definitelytyped-typescript-versions-0.0.67-04dafe75272abdf57a83a0cdce9261a19ce4e2-integrity/node_modules/@definitelytyped/typescript-versions" as pattern ["@definitelytyped/typescript-versions@0.0.67", "@definitelytyped/typescript-versions@0.0.67"]. This could result in non-deterministic behavior, skipping.
[3/4] Linking dependencies...
warning " > esymetrik/ngx-leaflet@8.1.0" has unmet peer dependency "tslib@2".
warning " > @ng-bootstrap/ng-bootstrap@8.0.4" has unmet peer dependency "@angular/localize@10.0.0".
warning " > bootstrap@4.6.0" has unmet peer dependency "popper.js@1.16.1".
[4/4] Building fresh packages...
+ Done in 4.33s.
Installing pre-commit hook for git
Symlink already exists
Making pre-commit hook file executable ...
pre-commit hook file is now executable!
Installing pre-push hook for git
Symlink already exists
Making pre-push hook file executable ...
pre-push hook file is now executable!

Tasks still running:
-----
core.domain.activity_domain_test (started 20:31:42)
-----
15:01:51 FINISHED core.domain.activity_domain_test: 9.0 secs

SUMMARY OF TESTS
-----
SUCCESS core.domain.activity_domain_test: 6 tests (0.5 secs)

Ran 6 tests in 1 test class.
All tests passed.

Done!
krishitajain@KRISHITAS-MacBook-Air oppia %
```

- I am able to run all the frontend tests at once on my machine. (Show a screenshot of a successful test.)

```
Chrome Headless 89.0.4389.114 (Mac OS 11.2.3): Executed 4462 of 4471 SUCCESS (0 secs / 58.710 secs)
ERROR: 'Error communicating with server. Please try again.'
Chrome Headless 89.0.4389.114 (Mac OS 11.2.3): Executed 4462 of 4471 SUCCESS (0 secs / 58.918 secs)
ERROR: 'Error communicating with server. Please try again.'
Chrome Headless 89.0.4389.114 (Mac OS 11.2.3): Executed 4462 of 4471 SUCCESS (0 secs / 58.918 secs)
LOG: 'Spec: Exploration rights service should use reject when changing exploration right viewability fails has passed'
Chrome Headless 89.0.4389.114 (Mac OS 11.2.3): Executed 4462 of 4471 SUCCESS (0 secs / 58.918 secs)
ERROR: 'Unexpected call to ExplorationDataService for pathname: /context.html'
Chrome Headless 89.0.4389.114 (Mac OS 11.2.3): Executed 4463 of 4471 SUCCESS (0 secs / 58.933 secs)
LOG: 'Spec: Exploration rights service should check user already has roles has passed'
Chrome Headless 89.0.4389.114 (Mac OS 11.2.3): Executed 4463 of 4471 SUCCESS (0 secs / 58.933 secs)
ERROR: 'Unexpected call to ExplorationDataService for pathname: /context.html'
Chrome Headless 89.0.4389.114 (Mac OS 11.2.3): Executed 4464 of 4471 SUCCESS (0 secs / 58.942 secs)
ERROR: 'Error communicating with server. Please try again.'
Chrome Headless 89.0.4389.114 (Mac OS 11.2.3): Executed 4464 of 4471 SUCCESS (0 secs / 58.942 secs)
LOG: 'Spec: Exploration rights service should change exploration right viewability has passed'
Chrome Headless 89.0.4389.114 (Mac OS 11.2.3): Executed 4464 of 4471 SUCCESS (0 secs / 58.942 secs)
ERROR: 'Unexpected call to ExplorationDataService for pathname: /context.html'
Chrome Headless 89.0.4389.114 (Mac OS 11.2.3): Executed 4465 of 4471 SUCCESS (0 secs / 58.963 secs)
LOG: 'Spec: Welcome Translation Modal Controller should initialize $scope properties after controller is initialized has passed'
Chrome Headless 89.0.4389.114 (Mac OS 11.2.3): Executed 4465 of 4471 SUCCESS (0 secs / 58.963 secs)
ERROR: 'Unexpected call to ExplorationDataService for pathname: /context.html'
Chrome Headless 89.0.4389.114 (Mac OS 11.2.3): Executed 4466 of 4471 SUCCESS (0 secs / 58.963 secs)
ERROR: 'Error communicating with server. Please try again.'
Chrome Headless 89.0.4389.114 (Mac OS 11.2.3): Executed 4466 of 4471 SUCCESS (0 secs / 58.963 secs)
LOG: 'Spec: Pie Chart component when chart is defined and $scope data is an array should redraw chart has passed'
Chrome Headless 89.0.4389.114 (Mac OS 11.2.3): Executed 4466 of 4471 SUCCESS (0 secs / 58.963 secs)
ERROR: 'Unexpected call to ExplorationDataService for pathname: /context.html'
Chrome Headless 89.0.4389.114 (Mac OS 11.2.3): Executed 4467 of 4471 SUCCESS (0 secs / 58.978 secs)
ERROR: 'Error communicating with server. Please try again.'
Chrome Headless 89.0.4389.114 (Mac OS 11.2.3): Executed 4467 of 4471 SUCCESS (0 secs / 58.978 secs)
LOG: 'Spec: Pie Chart component when $scope data is not an array should not redraw chart has passed'
Chrome Headless 89.0.4389.114 (Mac OS 11.2.3): Executed 4467 of 4471 SUCCESS (0 secs / 58.978 secs)
ERROR: 'Unexpected call to ExplorationDataService for pathname: /context.html'
Chrome Headless 89.0.4389.114 (Mac OS 11.2.3): Executed 4468 of 4471 SUCCESS (0 secs / 58.988 secs)
ERROR: 'Error communicating with server. Please try again.'
Chrome Headless 89.0.4389.114 (Mac OS 11.2.3): Executed 4468 of 4471 SUCCESS (0 secs / 58.988 secs)
LOG: 'Spec: Exploration Correctness Feedback Service should toggle correctness feedback display has passed'
Chrome Headless 89.0.4389.114 (Mac OS 11.2.3): Executed 4469 of 4471 SUCCESS (0 secs / 58.999 secs)
ERROR: 'Unexpected call to ExplorationDataService for pathname: /exploration/0'
Chrome Headless 89.0.4389.114 (Mac OS 11.2.3): Executed 4470 of 4471 SUCCESS (0 secs / 59.011 secs)
LOG: 'Spec: Exploration data service should throw error when pathname is not valid has passed'
Chrome Headless 89.0.4389.114 (Mac OS 11.2.3): Executed 4470 of 4471 SUCCESS (0 secs / 59.011 secs)
Chrome Headless 89.0.4389.114 (Mac OS 11.2.3): Executed 4471 of 4471 SUCCESS (1 min 8.854 secs / 59.015 secs)
TOTAL: 4471 SUCCESS
TOTAL: 4471 SUCCESS
15 04 2021 19:09:12.478:WARN [launcher]: ChromeHeadless was not killed in 2000 ms, sending SIGKILL.
Done!
krishitajain@KRISHITAS-MacBook-Air oppia %
```

- I am currently unable to run one suite of e2e tests on macOS with M1 chipset. This is being tracked on [this issue](#).

Other summer obligations

I have no other jobs this summer and I plan to spend my time working on this project solely.

Communication channels

I will mostly be active via email, Hangouts, and Gitter.

I plan to communicate with my mentor every other day to update them on the project's progress as well as to ask them doubts. I also plan to have weekly meetings to talk about the tasks I will be doing the following week.

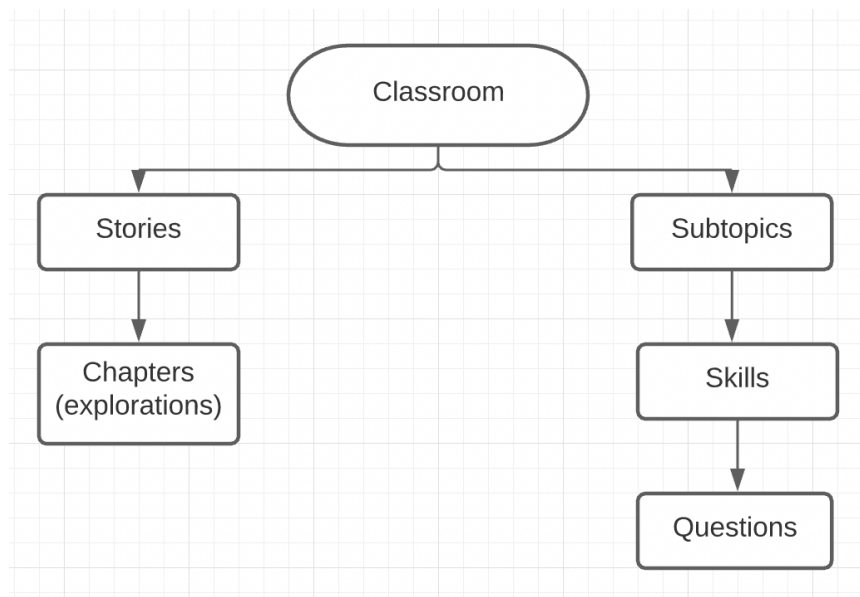
Application to multiple orgs

I am only applying for Oppia.

Project Details

Product Design

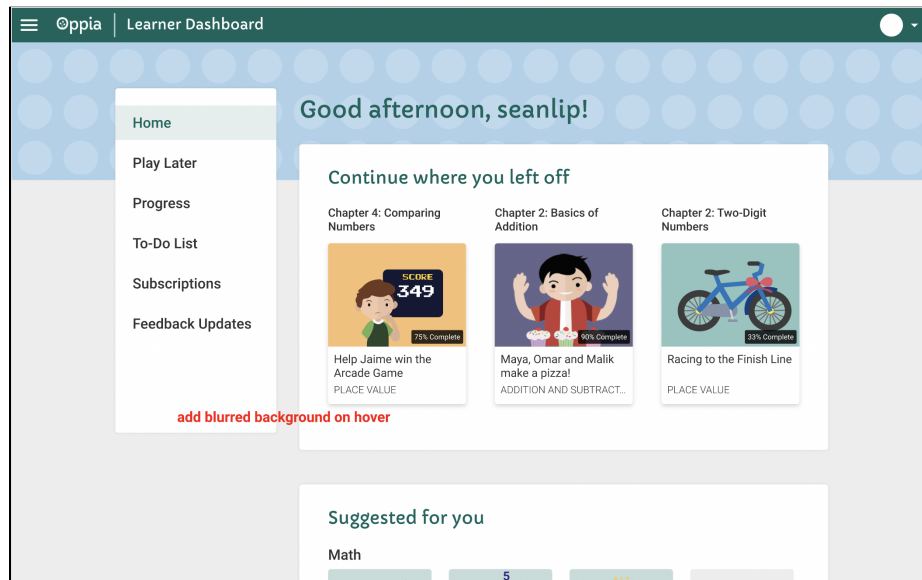
Oppia brings forth a platform to conduct comprehensive courses in the form of classrooms. A classroom contains many different topics in it. Each topic is divided into two parts: Stories and Subtopics. Stories contain questions in the form of chapters(explorations). Subtopics on the other hand contain practice questions to refresh a particular part of the topic. A subtopic is made up of many different skills. Each skill has many questions in it.



Four new tabs are added inside the Learner Dashboard page namely **Home**, **Progress**, **To-Do List** and **Community lessons** which will replace the existing **In Progress** and **Completed** tabs. The features inside each of these tabs are as follows:

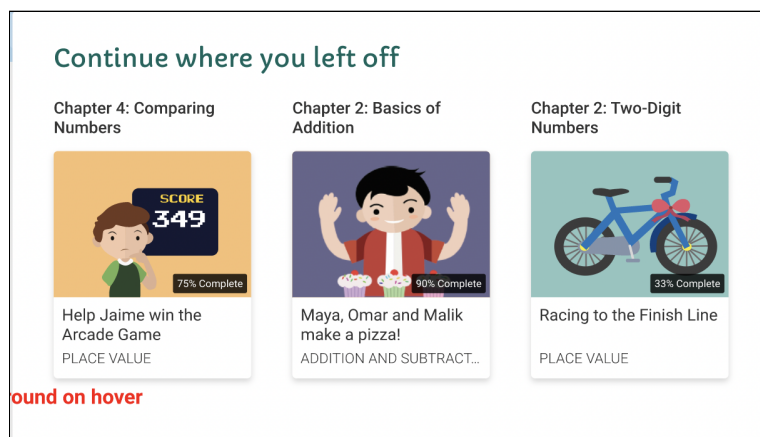
1. **Home tab:**

- a. The **Home** tab is added to the top of the learner dashboard menu. When a learner is directed to the learner dashboard page, the **Home** tab opens up on the screen by default. It starts with a greeting to the learner. The tab is further divided into two sections: **Continue where you left off** and **Suggested for you**.

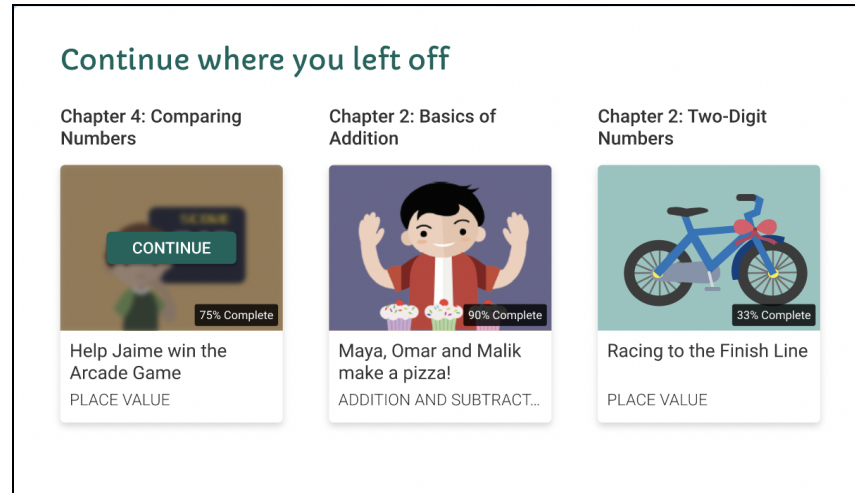


- b. **Continue where you left off:** This section left contains stories of the topics that the user selects in the **To-Do List** tab for easy access. It displays Story cards linking to the next incomplete chapter in the story. The cards inside this section will be displayed in descending order ordered by the percentage of story completed by the user. Each story card displays:

- Chapter name
- Story thumbnail
- Percent of the story completed
- Name of the story
- Name of the Topic.



If there are more than 3 cards, they will be displayed using a carousel. When the learner hovers over a card, a Continue button is displayed in the middle of the card and the card background is blurred. Clicking on the Continue button directs the learner to that chapter in the story. (i.e. directly play the exploration)



- c. In case the user has not selected any topic in the **To-Do List** tab, the **Continue where you left off** section will be empty. Hence instead of topics, it will display a message:

'Hmm, this looks empty. Select a topic in the **ToDoList** to get started!'

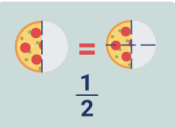
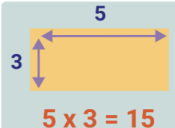
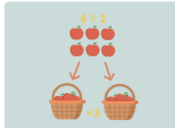
This **ToDoList** will be a link which will direct the learner to the **To-Do List** tab from where they can select any topic of their choice.

- d. **Suggested for you:** This section contains topic suggestions for the learner. It displays three topic cards of the topics inside a classroom. Each row contains four cards. The first three cards are the topic cards that link to that topic. Each topic card displays:
- Topic thumbnail
 - Topic name
 - The number of stories inside that topic.


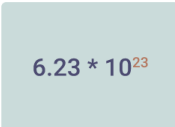

The last card in the row is linked to the classroom page which shows all the topics inside that classroom. Even after the learner has completed all the topics of a classroom, this section will still display three topics.

Suggested for you

Math

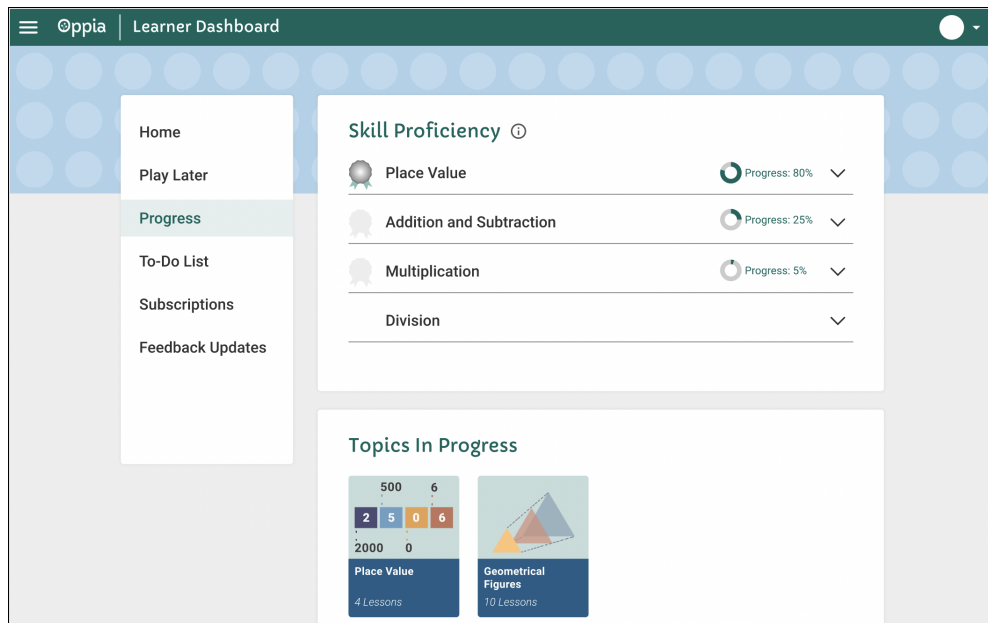
 <p>Fractions</p> <p>12 Lessons</p>	 <p>Multiplication</p> <p>8 Lessons</p>	 <p>Division</p> <p>8 Lessons</p>	<p>View All Math Topics</p>
---	---	--	------------------------------------

Science

 <p>Nutrition</p> <p>3 Lessons</p>	 <p>Scientific Notation</p> <p>1 Lesson</p>	 <p>States Of Matter</p> <p>4 Lessons</p>	<p>View All Science Topics</p>
--	---	--	---------------------------------------

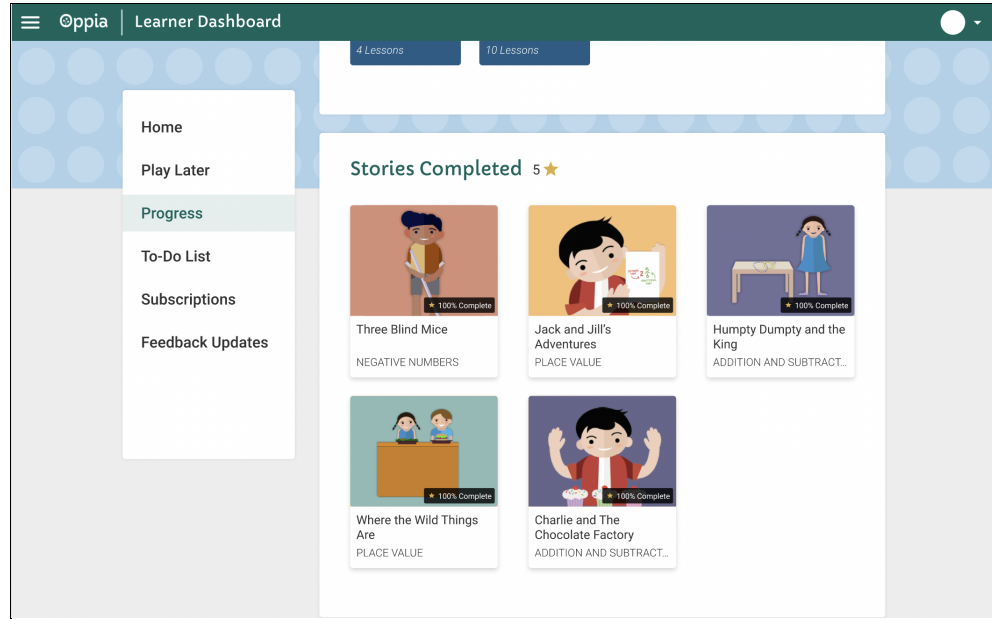
2. [Progress tab](#)

- a. The **Progress** tab tracks the progress made by the learner inside classrooms. It is divided into three sections namely **Skill Proficiency**, **Topics in Progress** and **Stories Completed**.



The screenshot shows the Oppia Learner Dashboard with the 'Progress' tab selected. The dashboard is divided into three main sections:

- Navigation Menu:** Home, Play Later, Progress (selected), To-Do List, Subscriptions, Feedback Updates.
- Skill Proficiency:** A list of skills with progress indicators:
 - Place Value: Progress: 80%
 - Addition and Subtraction: Progress: 25%
 - Multiplication: Progress: 5%
 - Division: Progress: 0%
- Topics In Progress:** Two topic cards are shown:
 - Place Value:** 4 Lessons. The card displays a place value chart with digits 2, 5, 0, 6 in the thousands, hundreds, tens, and ones places respectively.
 - Geometrical Figures:** 10 Lessons. The card displays a diagram of a triangle with a smaller triangle inside it.



b. **Skill Proficiency <i-icon>**: When the learner starts a new topic(i.e. when the user has completed a chapter/node in the story of that topic) or selects a topic from the **To-Do List** tab, the topic along with the subtopics of that topic are autofilled in the **Skill Proficiency** section and are tracked accordingly.

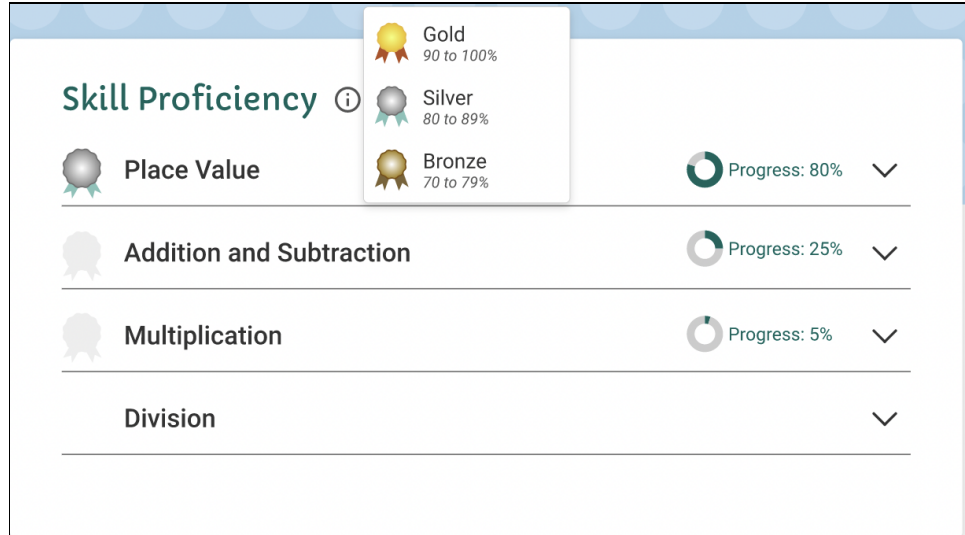
When the user hovers over the **i-icon** beside the **Skill Proficiency** heading, it displays the percent of avg skill mastery of the subtopic required to receive **Bronze, Silver** and **Gold** badges respectively.

- If the percent of avg skill mastery in the subtopic is between **70 to 79%**, the user will receive a **Bronze** Badge for that topic.
- If the percent of avg skill mastery in the subtopic is between **80 to 89%**, the user will receive a **Silver** Badge for that topic.
- If the percent of avg skill mastery in the subtopic is between **90 to 100%**, the user will receive a **Gold** Badge for that topic.

If the avg skill mastery of a subtopic is 0%, the empty badge will not be displayed.

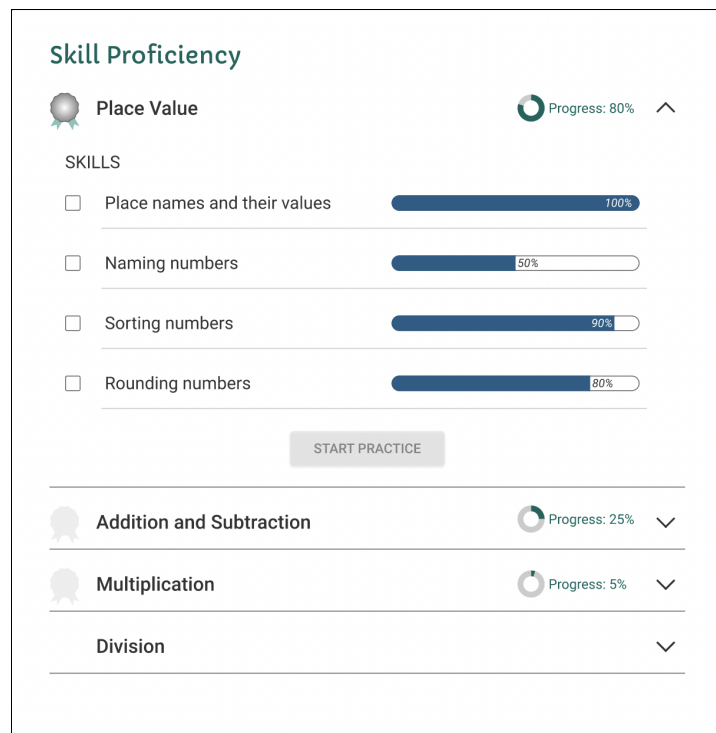
This section contains a list of topics. Each list item has:

- An Empty Badge, which will be replaced by a Bronze, Silver or Gold Badge according to the user's progress
- Name of the topic
- A Pie Chart illustrating the numerical proportion of the progress
- Progress: Percent
- A dropdown icon.

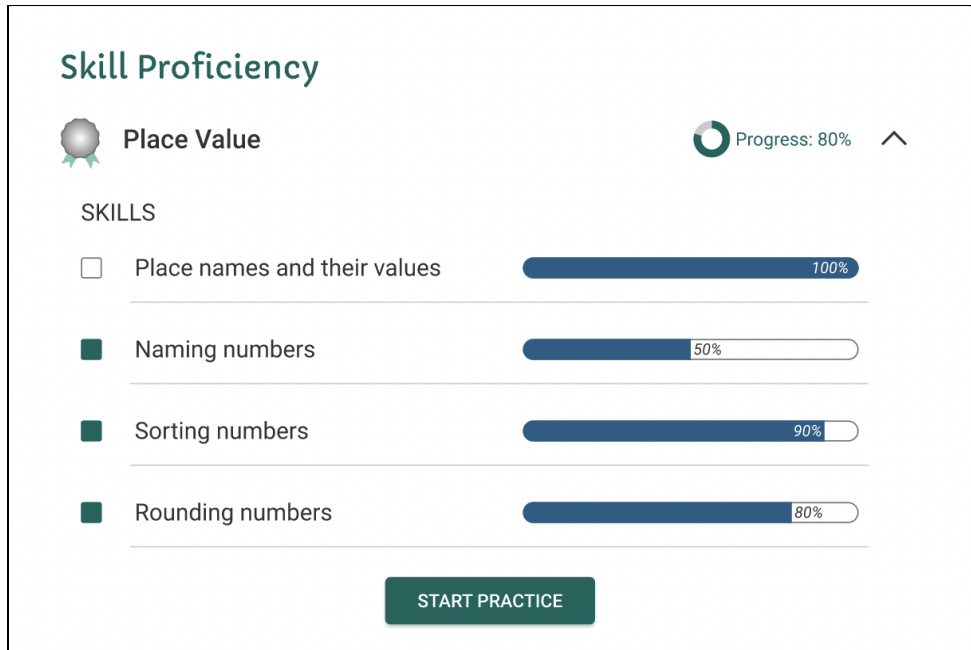


When the user clicks on the dropdown icon, a list of all the subtopics inside the topic is displayed. Each subtopic inside the dropdown has:

- A checkbox
- Name of the subtopic
- A capsule-shaped bar that displays the percent of avg skill mastery in the subtopic, achieved by the user.



When the user selects a subtopic, the Start Practice button will be enabled. The user can select any number of subtopics to practice. If they select more than one subtopics, those subtopics will be played in succession.

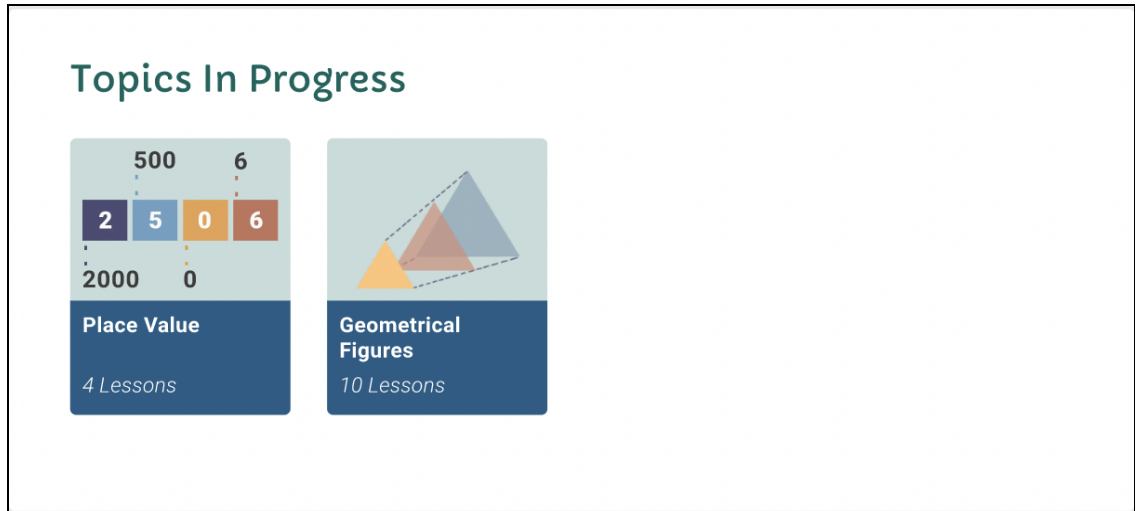


- c. In the case when the learner has not started any topic, the **Skill Proficiency** section will be empty. Hence it will display a message:

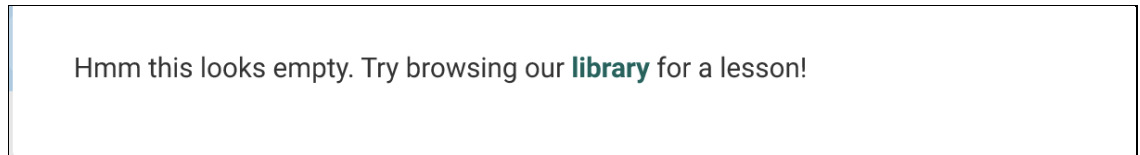
Hmm this looks empty. Try browsing our [library](#) for a lesson!

This **library** link will direct the learner to a classroom page (Math classroom for now) from where they can start any topic of their choice. When the learner starts the topic, the subtopics classified under that topic will be autofilled in the **Skill Proficiency** section.

- d. Topics in Progress:** This section displays the topics which the learner has started (i.e. when the user has completed a chapter/node in the story of that topic) but not yet completed (A topic is considered to be completed when they have finished all stories in it). Each row contains four Topic cards. (The specifications of the topic card are the same as described above for the **Suggested for you** section inside the **Home** tab). If the learner clicks on a topic card, they are directed to that topic.



- e. In the case when the learner has not started any topic, the **Topics in Progress** section will be empty. Hence it will display a message:

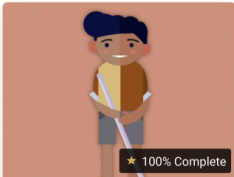


This **library** link will direct the learner to a classroom page (Math classroom for now) from where they can start any topic of their choice. When the learner starts the topic, it will be displayed inside the **Topics in Progress** section.

- f. **Stories Completed** <Number of stories completed> <star icon>: This section contains all the stories that the learner has completed. Each row contains three Story cards. Each story card contains:
- Story thumbnail
 - <star icon> 100% Completed
 - Story name
 - Topic name


If the learner clicks on the story card, they are directed to that story.

Stories Completed 5 ★



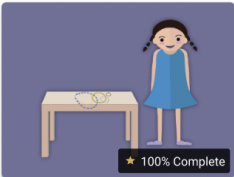
★ 100% Complete

Three Blind Mice
NEGATIVE NUMBERS



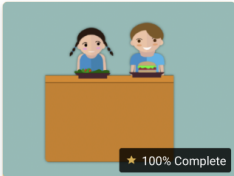
★ 100% Complete

Jack and Jill's Adventures
PLACE VALUE




★ 100% Complete

Humpty Dumpty and the King
ADDITION AND SUBTRACT...



★ 100% Complete

Where the Wild Things Are
PLACE VALUE

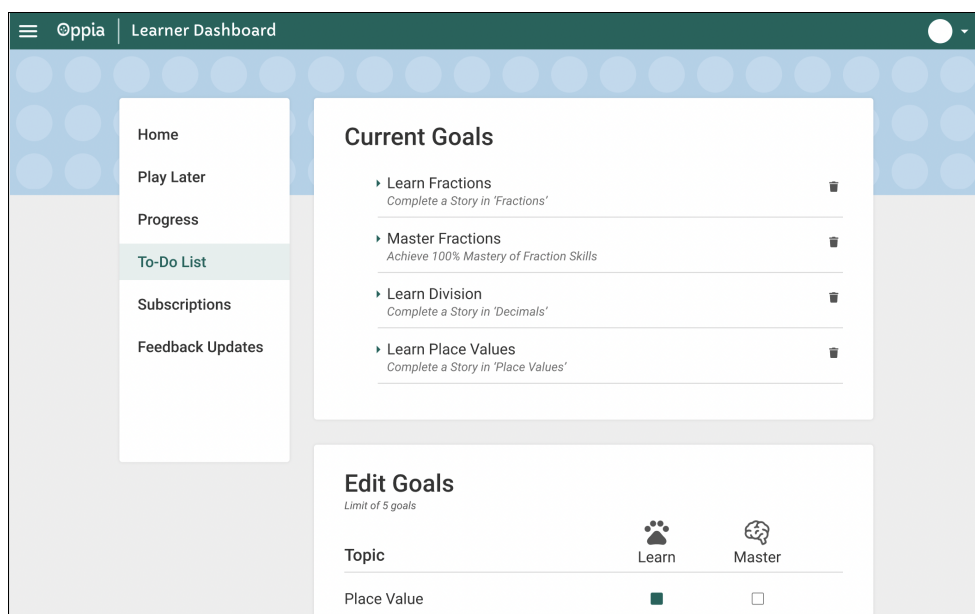


★ 100% Complete

Charlie and The Chocolate Factory
ADDITION AND SUBTRACT...

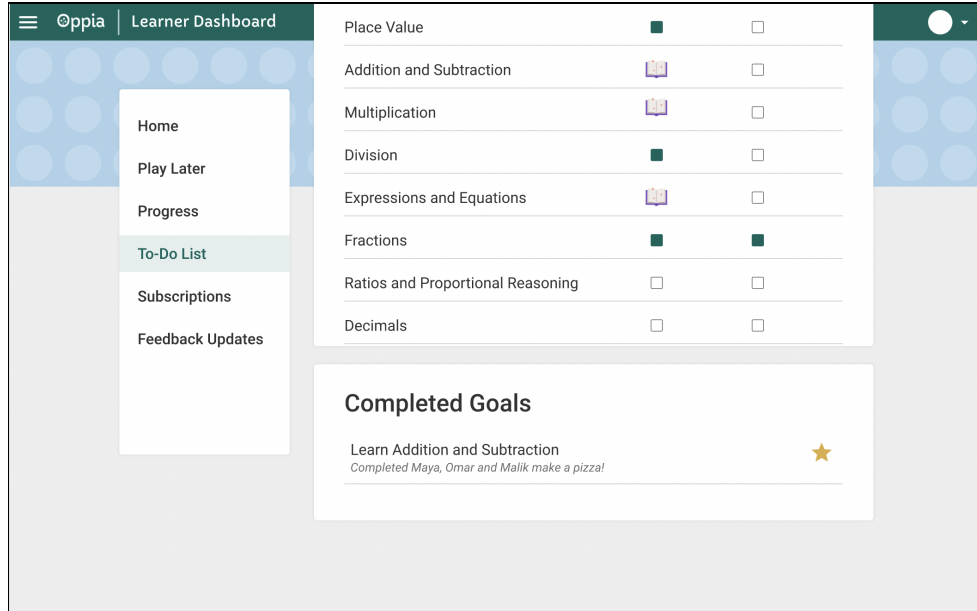
3. [To-Do List tab](#)

- a. The **To-Do List** tab allows the learner to set goals for themselves and keep a track of those goals. This tab is divided into three sections namely **Current Goals**, **Edit Goals** and **Completed Goals**.
(NOTE: This mock shows Mastery for Skills as well but it is not a part of this Project)



The screenshot shows the OPIA Learner Dashboard with the 'To-Do List' tab selected. The dashboard includes a navigation menu on the left with options: Home, Play Later, Progress, To-Do List (selected), Subscriptions, and Feedback Updates. The main content area is divided into two sections:

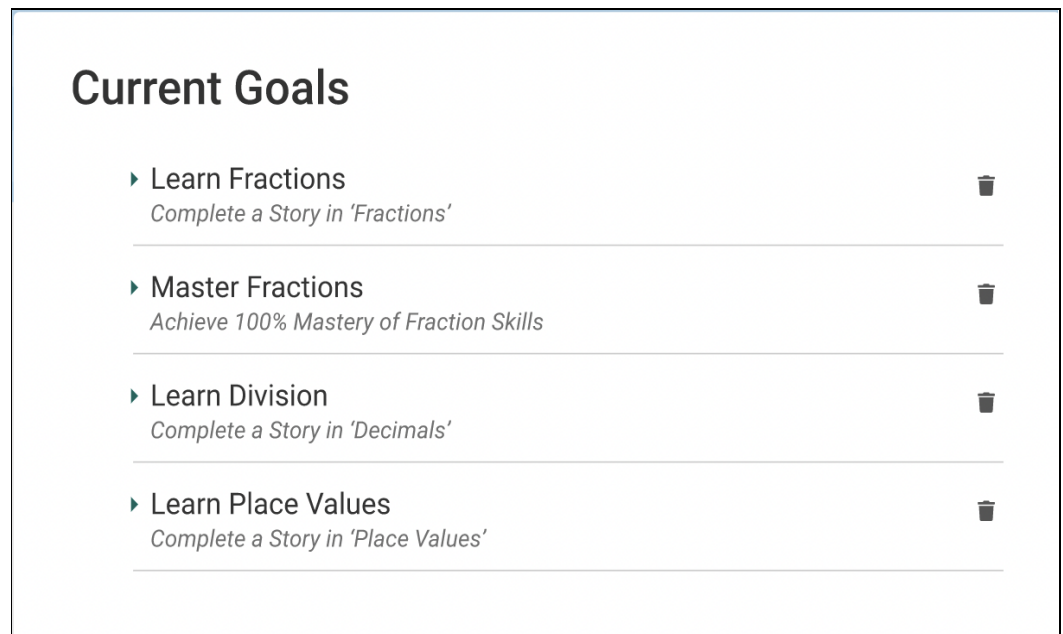
- Current Goals:** A list of four goals, each with a trash icon for deletion:
 - Learn Fractions (Complete a Story in 'Fractions')
 - Master Fractions (Achieve 100% Mastery of Fraction Skills)
 - Learn Division (Complete a Story in 'Decimals')
 - Learn Place Values (Complete a Story in 'Place Values')
- Edit Goals:** A section with a 'Limit of 5 goals' and two options: 'Learn' (with a paw print icon) and 'Master' (with a brain icon). Below this, the 'Place Value' goal is shown with a green square under 'Learn' and an empty square under 'Master'.



b. **Current Goals:** This section displays the topics that the learner selects from the **Edit Goals** section. For example in the above images, The learner selects the 'Place value' topic, 'Division topic' and the 'Fractions topic' from the **Edit Goals** section. These same selected topics are displayed in the **Current Goals** section.

Each list item contains:

- Learn <Topic name>
- Dropdown icon
- Delete icon
- Complete a Story in the '<Topic name>'




(NOTE: This mock shows Mastery of Skills but it is not a part of this project)

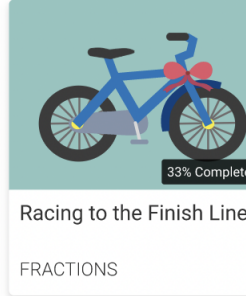
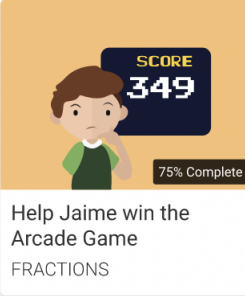
A maximum of 5 goals can be selected as **Current Goals**.

When the learner clicks on the dropdown icon, story cards of all the incomplete stories from that topic are displayed. Each row consists of 3 stories. Each story card contains:

- Story thumbnail
- Percent of Story completed
- Story name
- Topic name


Current Goals


▼ Learn Fractions 
Complete a Story in 'Fractions'




Help Jaime win the Arcade Game
FRACTIONS

Racing to the Finish Line
FRACTIONS

▶ Master Fractions 
Achieve 100% Mastery of Fraction Skills

▶ Learn Division 
Complete a Story in 'Decimals'

▶ Learn Place Values 
Complete a Story in 'Place Values'

When the learner clicks on a story, they are directed to that story page.








If the learner clicks on the delete icon, it will remove the goal from the **Current Goals** section.

- c. **Edit Goals:** This section contains all the topics present in all classrooms. Each list item contains:
- Topic name
 - Checkbox

When the learner selects a topic as their Current Goal, the checkbox is filled with green. The selected topic is automatically updated into the **Current Goals** section. A learner can select a maximum of 5 goals at the same time as their current goal.

If the learner has completed all the stories of a topic from the **Current Goals** section, it will be removed from the **Current Goals** section. The **checkbox** inside the **Edit Goals** section will be replaced by a **Book** icon for that topic and the completed topic will be added to the **Completed Goals** section.

In case the learner has already completed all the stories inside a topic **without/before** placing it inside the **Current Goals** section, the **checkbox** inside the **Edit Goals** section will be replaced by a **Book** icon for that topic and the completed topic will be added to the **Completed Goals** section.


Edit Goals	
<i>Limit of 5 goals</i>	
Topic	 Learn
Place Value	
Addition and Subtraction	
Multiplication	
Division	
Expressions and Equations	
Fractions	
Ratios and Proportional Reasoning	<input type="checkbox"/>
Decimals	<input type="checkbox"/>

- d. **Completed goals:** This section contains all the topics that the learner has completed from the **Current Goals** section. Each list item has:
- Learn <Topic name>
 - Completed <Name of story>
 - A star icon

If there is more than one story in a topic, they will be displayed with bullet points. Also, topics will mostly have only one story.

Completed Goals

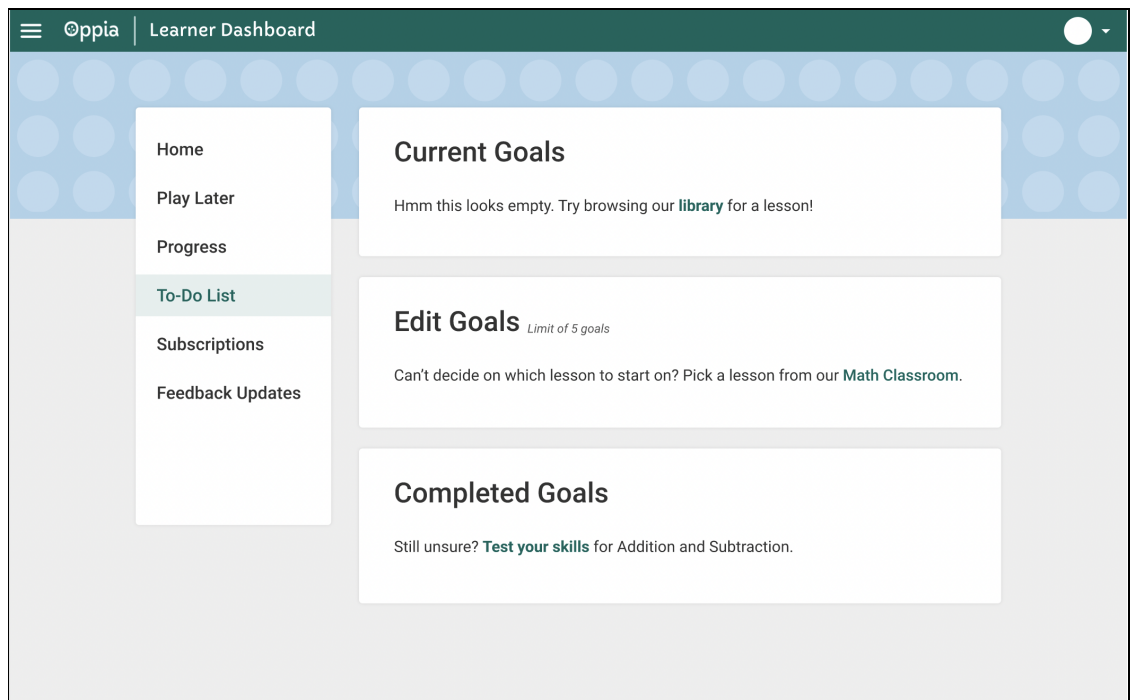
Learn Addition and Subtraction
Completed Maya, Omar and Malik make a pizza!



In case the learner has already completed all the stories in a topic **without/before** placing it inside the **Current Goals** section, it will be placed inside the **Completed Goals** section.

- e. In the case when the learner has not selected any topic from the **Edit Goals** section, the **Current Goals** section will be empty.

If the learner has not completed any goals inside the **Current Goals** section, the **Completed Goals** section will be empty.



The screenshot shows the 'Learner Dashboard' interface. On the left is a navigation menu with options: Home, Play Later, Progress, To-Do List (highlighted), Subscriptions, and Feedback Updates. The main content area is divided into three sections: 1. 'Current Goals' with the text 'Hmm this looks empty. Try browsing our library for a lesson!'. 2. 'Edit Goals' with a sub-header 'Limit of 5 goals' and the text 'Can't decide on which lesson to start on? Pick a lesson from our Math Classroom.'. 3. 'Completed Goals' with the text 'Still unsure? Test your skills for Addition and Subtraction.'.

(NOTE: This mock shows the **Edit Goals** section empty but this section will never be empty as all the available topics will autofill this section when the learner loads the learner dashboard page)

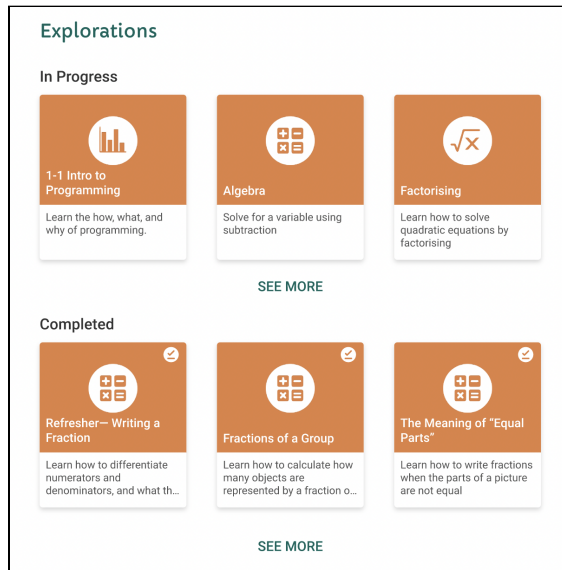
(NOTE: If the **Current Goals** section is empty, the text displayed will be 'Hmm this looks empty. To start, pick a topic from **Edit Goals** section')

4. Community Lessons:

- a. At present the Community Lessons i.e. the Explorations and the Collections are displayed in the '**In Progress**' and '**Completed**' tabs of the learner dashboard page.

In this project, these tabs will be removed and will be replaced by a single tab called **Community Lessons**.

This tab will contain a single section called **Community Lessons** which will display both the exploration and collections together (This is because there are not many collections present). All the explorations and the collections that are in progress and completed by the learner will be displayed. Each row will contain three cards. Each exploration and collection card will have the same specifications as they have now.



(NOTE: This mock displays **Explorations** as the heading but it will be **Community Lessons**)

If the learner clicks on the exploration/collection card, they are directed to that exploration/collection.

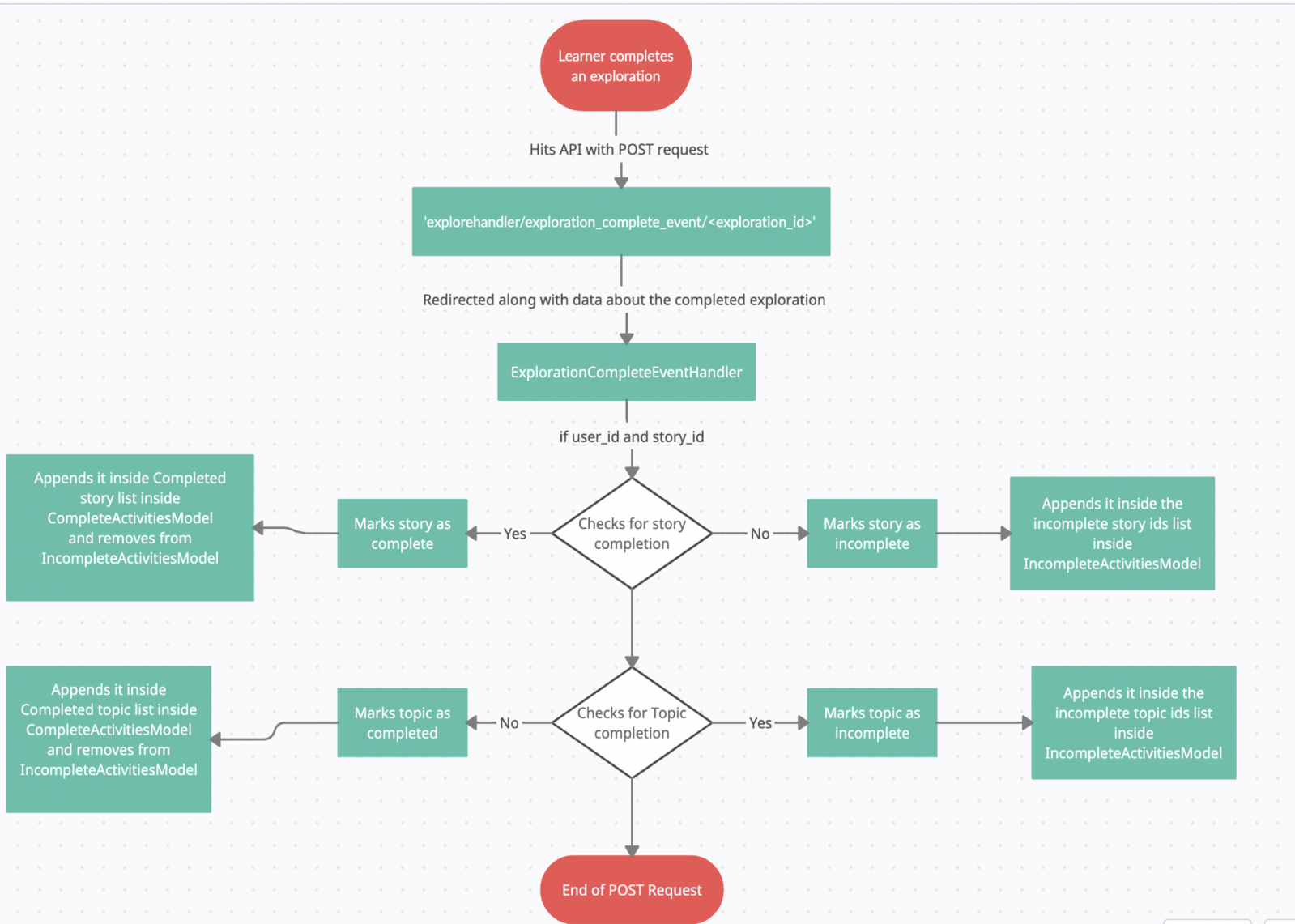
Clicking on See More will display more cards for the same.

Technical Design

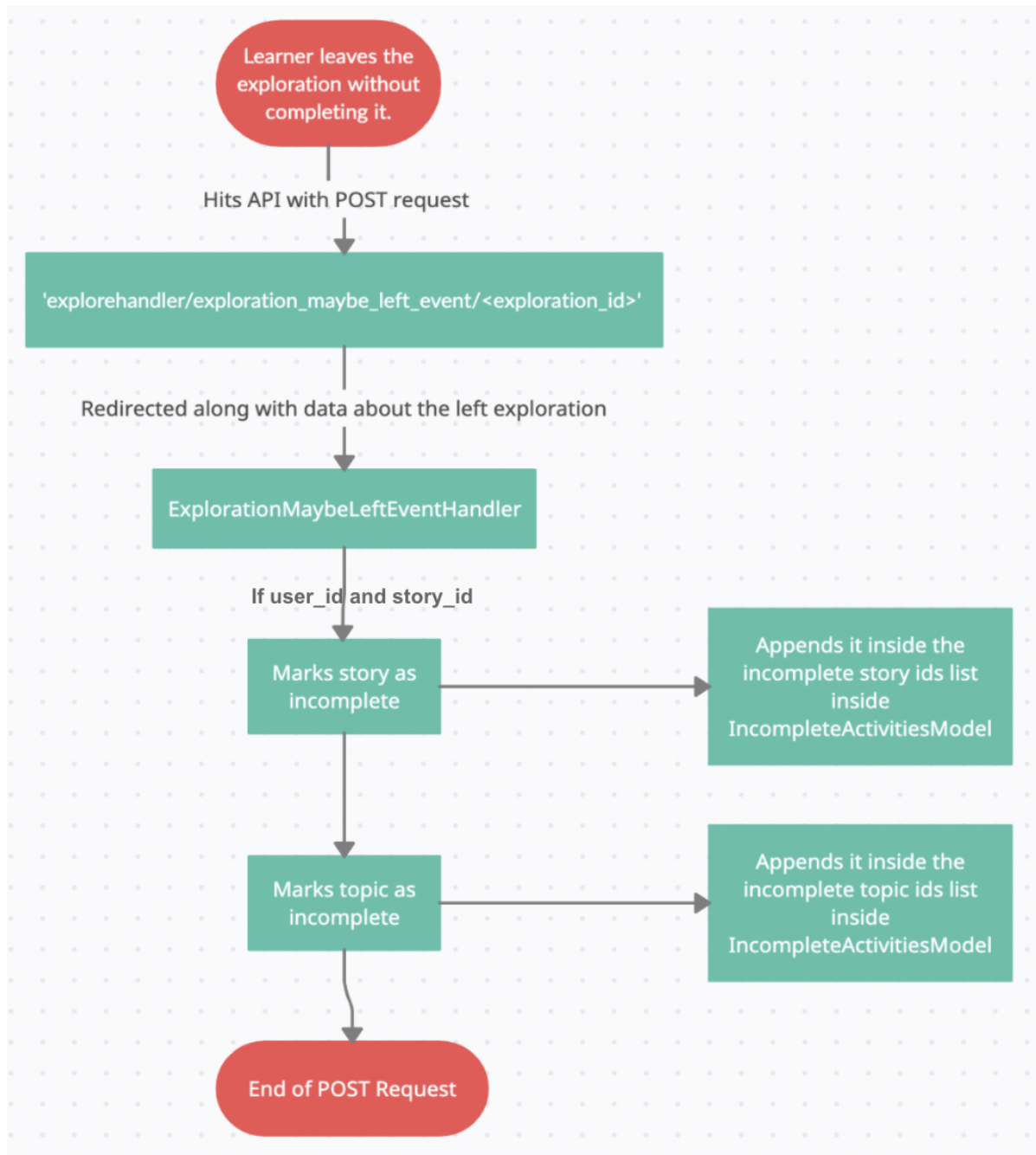
Architectural Overview

Four new folders namely home, progress todo-list and community-lessons will be created inside core/templates/pages/learner-dashboard-page. These folders will contain component files for respective tabs which will be integrated inside the learner-dashboard component file.

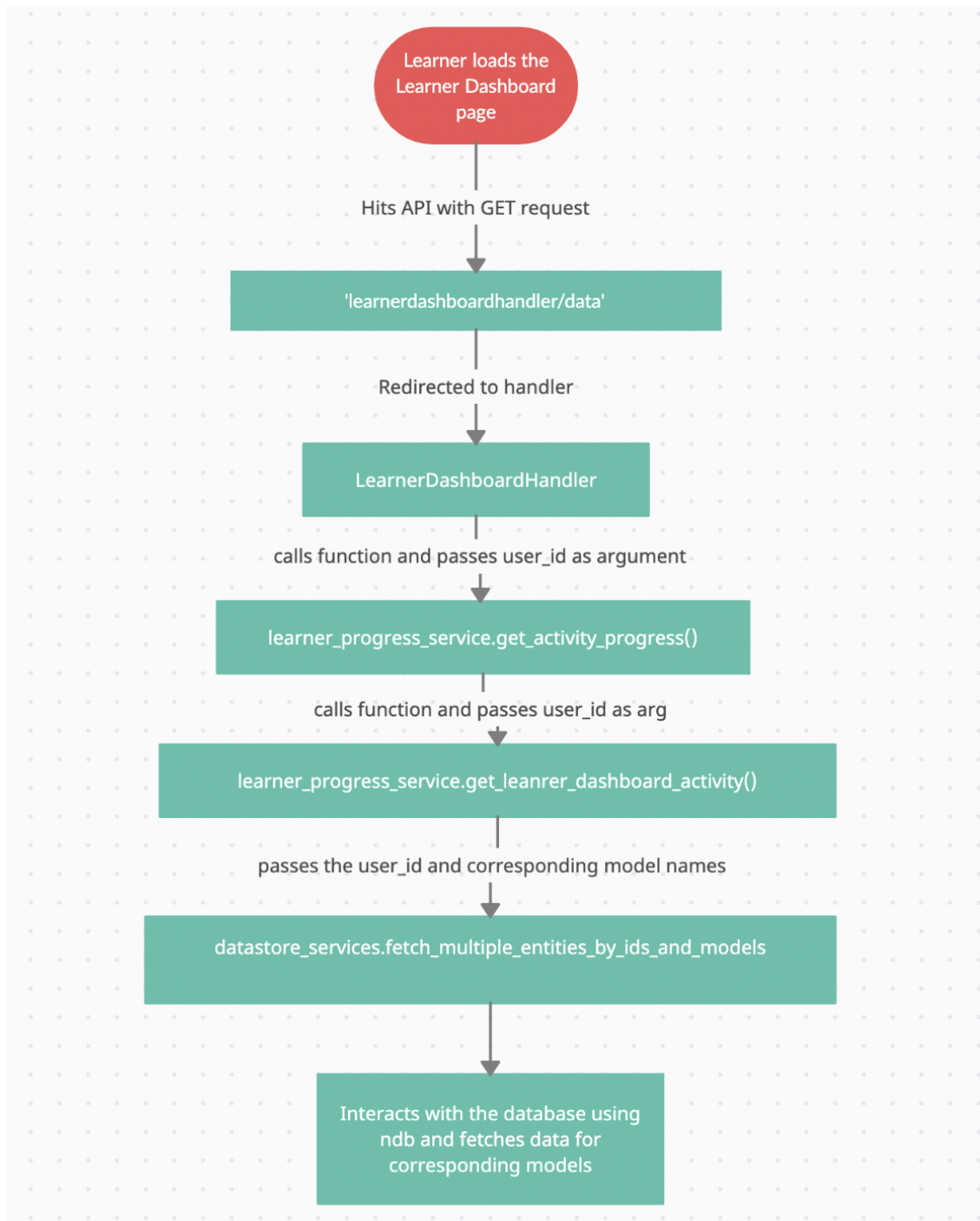
➤ POST request workflow when the learner has completed the exploration



➤ POST Request workflow when the learner has left the exploration halfway:



➤ GET Request workflow:



When an exploration is completed or left mid-way, the storage models are updated with a POST request.

When the Learner Dashboard page is loaded, it hits the '/learnerdashboardhandler/data' API endpoint with a GET request. This API fetches the data from the datastore corresponding to the Learner Dashboard page models. This data is then formatted and displayed to the learner in the frontend.

Implementation Approach

BACKEND (STORE AND FETCH DATA)

➤ For completed and incomplete Stories:

- A story will be marked as incomplete when the learner has completed/started at least one chapter(node) in it and as completed when the learner has completed all the nodes in it.
- To track all the stories currently being completed by the learner, a new field called `story_ids` will be added inside the `IncompleteActivitiesModel` storage model. The same field will also be added inside the `IncompleteActivities`, which is the domain object for the incomplete activities model.
- To track all the stories completed by the learner, a new field called `story_ids` will be added inside the `CompletedActivitiesModel` storage model. The same field will also be added inside the `CompletedActivities`, which is the domain object for the completed activities model.
- Both the Models are kept in sync with the `StoryProgressModel` which already tracks the chapter completion in a story.
- After adding `story_ids` inside `CompletedActivitiesModel` and `IncompleteActivitiesModel`, write a migration job inside `user_jobs_one_off` for it.
- **POST Request (Exploration Complete event):**
 - When the learner has reached the end of an exploration i.e. the next card in the exploration is the terminal card, it will hit the `'explorehandler/exploration_complete_event'` API endpoint with a POST request. The API will pass the `topic_url` and the `story_url` of the story to the `ExplorationCompleteEventHandler` inside `reader.py`.
 - If the exploration is being played inside a story, the `story_url` will be a string else it will be null.
 - Using `story_fetchers` and `topic_fetchers` inside the `ExplorationCompleteEventHandler`, will return domain objects representing the story and topic respectively.
 - Create a new log entry inside the `CompleteExplorationEventLogEntryModel`.
 - If the exploration is being played in context to a story, fetch details about the completed nodes and the ordered nodes of the story and check for the next incomplete node inside the story.

- If there is a `next_node_id`, then the story will be marked as incomplete by using the `mark_story_as_incomplete()` present in `learner_progress_service`.
- If there is no `next_node_id`, mark the story as complete by using the `mark_story_as_complete()` present inside `learner_progress_service`.

```
if user_id and story_id:
    completed_node_ids = [
        completed_node.id for completed_node in
        story_fetchers.get_completed_nodes_in_story(user_id, story_id)]

    ordered_nodes = a.story_contents.get_ordered_nodes()

    for node in ordered_nodes:
        if node.id not in completed_node_ids:
            next_exp_ids = [node.exploration_id]
            next_node_id = node.id
            break

    if not next_node_id:
        learner_progress_services.mark_story_as_completed(user_id, story_id)
    else:
        learner_progress_services.mark_story_as_incomplete(user_id, story_id)
```

- The `mark_story_as_completed()` will add the story id to the list of stories completed by the user and also remove the story id from the list of incomplete stories ids. It will take 2 arguments:
 - `user_id`
 - `story_id`
- If the story is not marked as already completed:
 - Remove the story from the incomplete story list
 - Update the `IncompleteActivitiesModel`.
 - Add the story inside the list of completed activities in the story ids list and save it inside the datastore.

```

def mark_story_as_completed(user_id, story_id):
    """Adds the story id to the list of storys completed by the user
    unless the story has already been completed or has been created/edited
    by the user. It is also removed from the incomplete list and the play later
    list (if present).

    Args:
        user_id: str. The id of the user who completed the story.
        story_id: str. The id of the completed story.
    """
    completed_activities_model = (
        user_models.CompletedActivitiesModel.get(
            user_id, strict=False))
    if not completed_activities_model:
        completed_activities_model = (
            user_models.CompletedActivitiesModel(id=user_id))

    activities_completed = _get_completed_activities_from_model(
        completed_activities_model)

    if (story_id not in activities_completed.story_ids):
        remove_story_from_incomplete_list(user_id, story_id)
        activities_completed.add_story_id(story_id)
        _save_completed_activities(activities_completed)

```

- Similarly, the `mark_story_as_incomplete()` will add the story id to the list of incomplete story ids.
- **POST Request (Exploration maybe left event)**
 - When the learner leaves an exploration without completing it, the 'explorehandler/exploration_maybe_leave_event' API endpoint is hit with a POST request. The API will pass the `topic_url` and the `story_url` of the story to the `ExplorationMaybeLeaveHandler` inside `reader.py`.
 - If the exploration is being played inside a story, the `story_url` will be a string else it will be null.
 - Using `story_fetchers` and `topic_fetchers` inside the `ExplorationMaybeLeaveHandler`, will return domain objects representing the story and topic respectively.

- If the `story_id` is not null, mark both the story and topic as incomplete.

```
if user_id and story_id:
    learner_progress_service.mark_story_as_incomplete(user_id, story_id)
    learner_progress_service.mark_topic_as_incomplete(user_id, topic_id)
```

- After adding the `story_id` and `topic_id` in the `IncompleteActivitiesModel`, create a new log entry inside `MaybeLeaveExplorationEventLogEntryModel`.

- **GET Request:**

- When the user loads the Learner dashboard page, its data is fetched by using the `learner-dashboard-backend-api-service`, a service to retrieve information for the learner dashboard from the backend.
- The `learner-dashboard-backend-api-service` uses a function called `_fetchLearnerDashboardDataAsync()` which returns a promise containing the data for the learner dashboard. When the user loads the learner dashboard page, the `_fetchLearnerDashboardDataAsync()` hits an API endpoint called `/learnerdashboardhandler/data` with a GET request.
- When this endpoint is hit, it calls the `LearnerDashboardHandler` inside the `learner_dashboard.py` which handles the GET request. This handler uses the `get_activity_progress()` inside the `learner_progress_services` to get information on the user's learner dashboard page.
- This function i.e. `get_activity_progress()` calls a function – `get_learner_dashboard_activities()`, which returns the ids of all the activities that are present in the various sections of the learner dashboard.
- The `get_learner_dashboard_activites()` uses the `fetch_multiple_entities_by_ids_and_models()` inside the `gae_datastore_services.py`. This function takes a list of tuples as its argument – `list(tuple(str, list(str)))`. The ids and their corresponding model names for which we have to fetch entities. This function uses `nbd` imported from the `google.appengine.ext()` to fetch the data from the datastore corresponding to the given ids and models.
- Using `datastore_services.fetch_multiple_entities_by_ids_and_models`, fetch the model instance of the `StorySummaryModel` corresponding to the ids.
- Filter this data by using:
 - `_get_filtered_incomplete_story_summaries()` inside the `learner_progress_service` which will return a list of summaries of

the incomplete story ids and the ids of story that are no longer present.

- `_get_filtered_complete_story_summaries()` inside the `learner_progress_service` which returns a list of summaries of the completed story ids and the ids of story that are no longer present.
- The completed story will be displayed inside the **Stories completed** section of the **Progress** tab. For this we require:
 - Story thumbnail
 - Story name
 - Topic name

➤ For all topics inside the **Todolist** section

- This will keep a track of the user progress in the **To-Do List** tab.
- A new storage model with the name `ToDoListModel` will be created inside `user/gae_model.py`.
 - This will be a **non-versioned** model which will preserve just the current version. Instances of this class will be keyed by the `user_id`.
 - It will have a:
 - `current_goals_to_learn` class variable which is a list of topic IDs of all the topics selected by the user for learn goals in the **Edit Goals** section. It will be a `StringProperty`.
 - `current_goals_to_master` class variable which is a list of topic IDs of all the topics selected by the user for master goals in the **Edit Goals** section. It will be a `StringProperty`. (**NOTE:** Master is not a part of this project and the field will be added in the future.)
 - `completed_goals_to_learn` class variable which is a list of topic IDs of all the topics completed (Here completed topic refers to a topic inside which the user has completed all the stories) by the user for learn goals in the **Current Goals** section. It will be a `StringProperty`.
 - `completed_goals_to_master` class variable which is a list of topic IDs of all the topics completed (Here completed topic refers to a topic inside which the user has completed all the subtopics) by the user for master goals in the **Current Goals** section. It will be a `StringProperty`. (**NOTE:** Master is not a part of this project and the field will be added in the future.)

- For the deletion policy: The model only belongs to one user, and should be deleted
- Takeout policy:
 - `get_model_association_to_user()`: Model is exported as one instance per user.
 - `get_export_policy()`: Model contains data to export corresponding to a user
- Add a `has_reference_to_user_id(cls, user_id)` method to the model. This method should return True when any of the model fields contains the given `user_id`.
- As the deletion policy is DELETE, Add an `apply_deletion_policy(cls, user_id)` method to the model
- Add an `export_data(user_id)` method to the model. This method returns the data fields that are associated with or refer to the given user.
- Write a `TodoListModelValidator` inside `user_validators.py` which is a class for validating `TodoListModels`.
- This model will have a domain object in `user_domain.py` called `TodoList`.

```
class TodoList(python_utils.OBJECT):
    """Domain object for a Goals."""

    def __init__(
        self, user_id, current_goals_to_learn, completed_goals_to_learn):
        self.user_id = user_id
        self.current_goals_to_learn = current_goals_to_learn
        self.completed_goals_to_learn = completed_goals_to_learn

    def add_topic_id_to_current_goals_to_learn(user_id, topic_id):
        self.current_goals_to_learn.append(topic_id)

    def remove_topic_id_from_current_goals_to_learn(user_id, topic_id):
        self.current_goals_to_learn.remove(topic_id)

    def add_topic_id_to_completed_goals_to_learn(user_id, topic_id):
        self.completed_goals_to_learn.append(topic_id)

    def remove_topic_id_from_completed_goals_to_learn(user_id, topic_id):
        self.completed_goals_to_learn.remove(topic_id)
```

- The **Edit Goals** section of the **To-Do List** tab will contain a list of all the topics in the server.
- If a `topic_id` is present in the `current_goal` list of the `TodoListModel`, fill the checkbox of that topic in the **Edit Goals** section with green.

- If a `topic_id` is present in the `completed_goal` list of the `ToDoListModel`, replace the checkbox of that topic in the **Edit Goals** section with a book icon.
- **POST Request(When the learner selects goal from Edit Goals):**
 - When the user selects a goal in the **Edit Goals**, check the length of `current_goal` in the `ToDoListModel`. If the length is less than 5, the `'todo_list_model/data'` API is hit with a POST request. It will pass the `topic_id` of the selected topic in the payload. This API will direct the learner to `LearnerDashboardToDoListHandler` inside the `learner_dashboard.py`.
 - This handler will receive the `topic_id` of the selected topic from the payload and add the `topic_id` to the list of `current_goal` in the `ToDoListModel` and save it.
 - If the length of `current_goal` in the `ToDoListModel` is 5, display a message that says that 'A max of 5 topics can be selected at a time'.
- **DELETE Request(When the user clicks on the delete icon)**
 - When the user clicks on the delete icon in the **Current Goals** section for a particular topic, the `'todo_list_model/data'` API is hit with a DELETE request. It will pass the `topic_id` of the deleted topic in the payload. This API will direct the learner to `LearnerDashboardCurrentGoalsHandler` inside the `learner_dashboard.py`.
 - This handler will receive the `topic_id` of the selected topic from the payload and will remove the `topic_id` from the list of `current_goal` in the `ToDoListModel` and save it.
- **POST Request(When the user has completed all the stories in the topic)**
 - When the user completes a chapter in the story, they are directed to the `ExplorationCompleteEventHandler` inside the `reader.py`. There we will check if all the stories inside the topic are completed or not. If all the stories are completed, we will remove the `topic_id` from the `current_goal` list of the `ToDoListModel`, if present and save it.
 - Along with removing the `topic_id` from the `current_goal`, we will add the `topic_id` to the `complete_goal_ids` list of the `ToDoListModel`.
- **GET Request**
 - When the user loads the Learner dashboard page, its data is fetched by using the `learner-dashboard-backend-api-service`, a service to retrieve information for the learner dashboard from the backend.

- The learner-dashboard-backend-api-service uses a function called `_fetchLearnerDashboardDataAsync()` which returns a promise containing the data for the learner dashboard. When the user loads the learner dashboard page, the `_fetchLearnerDashboardDataAsync()` hits an API endpoint called `/learnerdashboardhandler/data` with a GET request.
- When this endpoint is hit, it calls the `LearnerDashboardHandler` inside the `learner_dashboard.py` which handles the GET request. This handler uses the `get_activity_progress()` inside the `learner_progress_services` to get information on the user's learner dashboard page.
- This function i.e. `get_activity_progress()` calls a function – `get_learner_dashboard_activities()` which returns the ids of each of the activities that are present in the various sections of the learner dashboard.
- The `get_learner_dashboard_activites()` uses the `fetch_multiple_entities_by_ids_and_models()` inside the `gae_datastore_services.py`. This function takes a list of tuples as its argument – `list(tuple(str, list(str)))`. The ids and their corresponding model names for which we have to fetch entities. This function uses `nbd` imported from the `google.appengine.ext()` to fetch the data from the datastore corresponding to the given ids and models.
- Using `datastore_services.fetch_multiple_entities_by_ids_and_models`, fetch the model instance of the `ToDoListModel` corresponding to the ids.
- The topics and the stories of the topics present in `current_goal` will be displayed inside the **Continue where you left off** section of the **Home** tab. For this we require:
 - Next incomplete node's name and id.
 - Story thumbnail
 - Percent of the story completed
 - Name of the story
 - Name of the Topic.

These stories will be displayed in decreasing order of their percentage of completion. If there are more than three stories, they will be displayed using a carousel.
- The Topics in `completed_goal` will be displayed inside the **Completed Goals** section of the **To-Do List** tab.

➤ **For avg mastery in Subtopics:**

- Calculate the avg mastery of subtopics of the topics that the user has started (if subtopic is present) or the topics that the user has selected in the `ToDoList` section.

- To get the list of all the `topic_started_by_the_user`, combine the list of `completed_topic_ids` and `incomplete_topic_ids` with the `current_goal` inside the `learner_progress_service()`.
- Iterate over the `topic_started_by_the_user` and for each `topic_id` get the values for:
 - `Classroom_url_fragment`
 - Subtopics present inside the topic
 - `avg_degree_of_mastery` for each subtopic.

```

all_classrooms_dict = config_domain.CLASSROOM_PAGES_DATA.value

topic_user_has_started = list(set(incomplete_topic_ids + completed_topic_ids + current_goal_ids))

for topic_id in topic_user_has_started:
    Topic = topic_fetchers.get_topic_by_id(topic_id)
    classroom = classroom_service.get_classroom_url_fragment_for_topic_id(topic_id)
    subtopics = Topic.subtopics #List of subtopics inside the topic
    for subtopic in subtopics:
        skills = subtopic.skill_ids #List of skill ids inside the subtopic
        degrees_of_mastery = skill_services.get_multi_user_skill_mastery( #Dict {skill_id: degree_of_mastery}
            self.user_id, skills)
        avg_degree_of_mastery = (sum(degrees_of_mastery.values())/len(degrees_of_mastery.values())) * 100

        skill_proficiency.append({'classroom_url_fragment': classroom, 'topic_url_fragment': Topic.url_fragment,
            'topic_name': Topic.name, 'subtopic_id': subtopic.id, 'subtopic_name': subtopic.subtopic_name, 'skill_ids':
            skills, 'avg_degree_of_mastery': avg_degree_of_mastery})

```

- Append these values inside the `skill_proficiency` array.
- The subtopics for a topic will be displayed inside the **Skill Proficiency** section of the **Progress** tab. For this we require:
 - `classroom_url_fragment`
 - `topic_url_fragment`
 - `topic_name`
 - `subtopic_id`
 - `subtopic_name`
 - `avg_degree_of_mastery`
 - `skill_ids` in the subtopic

➤ For completed and incomplete Topics:

- A topic will be marked as incomplete when the learner has completed/started at least one chapter(node) inside a story and as completed when the learner has completed all the stories in the topic.
- To track all the topics currently being completed by the learner, a new field called `topic_ids` will be added inside the `IncompleteActivitiesModel` storage model. The same field will also be added inside the `IncompleteActivities`, which is the domain object for the incomplete activities model.

- To track all the topics completed by the learner, a new field called `topic_ids` will be added inside the `CompletedActivitiesModel` storage model. The same field will also be added inside the `CompletedActivities`, which is the domain object for the completed activities model.
- After adding `topic_ids` inside `CompletedActivitiesModel` and `IncompleteActivitiesModel`, write a migration job inside `user_jobs_one_off` for it.
- We need to check/update the topic completion status at only one point i.e. when the user completes an exploration. The topic will automatically marked as incomplete when the user leaves an exploration midway as described above for **POST Request (Exploration maybe left event)** in `story_ids`.
- **POST Request(Exploration Complete event):**
 - When the learner has reached the end of exploration, i.e. when the next card in the exploration is the terminal card, it will hit the 'explorehandler/exploration_complete_event' API endpoint with a POST request. The API will pass the `topic_url` and the `story_url` of the story to the `ExplorationCompleteEventHandler` inside `reader.py`.
 - If the exploration is being played inside a story, the `topic_url` will be a string else it will be null.
 - Using `story_fetchers` and `topic_fetchers` inside the `ExplorationCompleteEventHandle`, will return a domain object representing the story and topic respectively.
 - If the exploration is being played in context to a story, after marking the story as complete/incomplete, fetch details of the list of all the stories completed by the user using a function `get_all_completed_story_ids()` inside the `learner_progress_service`.
 - If all the story ids of a topic are present in the completed story ids list, then the topic will be marked completed by using the `mark_topic_as_complete()` present inside `learner_progress_service`.
 - If not, then the topic will be marked incomplete by using the `mark_topic_as_incomplete()` present inside the `learner_progress_service`.

the `get_activity_progress()` inside the `learner_progress_services` to get information on the user's learner dashboard page.

- This function i.e. `get_activity_progress()` calls a function -- `get_learner_dashboard_activities()` which returns the ids of each of the activities that are present in the various sections of the learner dashboard.
- The `get_learner_dashboard_activites()` uses the `fetch_multiple_entities_by_ids_and_models()` inside the `gae_datastore_services.py`. This function takes a list of tuples as its argument – `list(tuple(str, list(str)))`. The ids and their corresponding model names for which we have to fetch entities. This function uses `nbd` imported from the `google.appengine.ext()` to fetch the data from the datastore corresponding to the given ids and models.
- Using `datastore_services.fetch_multiple_entities_by_ids_and_models`, fetch the model instance of the `TopicModel` corresponding to the ids.
- Filter this data by using:
 - `_get_filtered_incomplete_topic()` inside the `learner_progress_service` which will return a list of the incomplete topic ids and the ids of topic that are no longer present.
 - `_get_filtered_complete_topic()` inside the `learner_progress_service` which returns a list of the complete topic ids and the ids of topic that are no longer present.
- The incomplete topic will be displayed inside the **Continue where you left off section** of the **Progress** tab. For this we require:
 - Topic thumbnail
 - Topic name
 - The number of stories inside that topic.

➤ **For all Topics present inside the server (Suggested for you):**

- To get the list of topics that are present on the server.
- To get the ids of all the topics inside a particular classroom, fetch the data of all the classrooms i.e. the `classroom_urls` and the list of `topic_ids` associated with the classroom. This will be done inside the `get_activity_progress()` present in `learner_progress_service`.

```

completed_story_ids = (
    activity_ids_in_learner_dashboard.completed_story_ids)
incomplete_story_ids = (
    activity_ids_in_learner_dashboard.incomplete_story_ids)
completed_topic_ids = (
    activity_ids_in_learner_dashboard.completed_topic_ids)
incomplete_topic_ids = (
    activity_ids_in_learner_dashboard.incomplete_topic_ids)
suggested_for_you = [] #list of all the topics on the server.
skill_proficiency = []

all_classrooms_dict = config_domain.CLASSROOM_PAGES_DATA.value

for classroom in all_classrooms_dict:
    suggested_for_you += classroom['topic_ids']

```

- Fetch the data of all the classrooms present on the server by using the CLASSROOM_PAGES_DATA declared in config_domain. From this data, store the topic_ids inside suggested_for_you.
- Using datastore_services.fetch_multiple_entities_by_ids_and_models, fetch the model instance of the Topic corresponding to the ids.
- Filter this data by using _get_filtered_suggest_for_you_topic() inside the learner_progress_service which returns a list of the suggested for you topic ids and the ids of topic that are no longer present.
- The suggested_for_you topics will be displayed inside the **Suggested for you** section of the **Home** tab. Display the topics corresponding to the related classrooms. For this we require:
 - Classroom name
 - Topic thumbnail
 - Topic name
 - The number of stories inside that topic.

FRONTEND

- **Home tab:** A new folder called home will be created inside the core/templates/pages/learner-dashboard-page folder which will contain the component for **Home** tab.
 - **For Greeting to the User:**
 - A function which checks the time and greets the learner.

```
var now = new Date();
var time = now.getHours();

if (time <= 12) alert("Good morning, Sean Lip!");
if (time <= 18 && time >= 12) alert("Good Afternoon, Sean Lip!");
if (time >= 18) alert("Good evening, Sean Lip!");
```

- **Continue where you left off:**
 - When the user loads the Learner Dashboard page, data will be fetched from the `TodoListModel` for topic ids in the `learn` of `current_goal` using the GET request as described above.
 - For each `topic_id` in the `current_goal`, we will create an instance of `TopicModel` and instances of `StorySummaryModel` for the incomplete stories in that `topic_id`.
 - This data will be used to create a new list of `StorySummary` and `Topic` in the frontend called `currentGoalsStoryList` and `currentGoalsTopicsList`.
 - If a story id is present inside the `currentGoalsStoryList`, it will be present inside the `canonical_story_ids` list of the `currentGoalsTopicsList`. Hence we will get the `topic_url` and the `classroom_url` from the `currentGoalsTopicsList` for the corresponding story id.
 - Using the data from the `StorySummary`, we will calculate the next incomplete node inside the story and the percentage of the story completed by the user.
 - To display this data in the format presented in the mocks, we will create a new directive called the `learner-dashboard-story-summary-tile`.
 - This directive will be passed:
 - Story summary.
 - `topic_url`
 - `classroom_url`
 - This directive will contain a continue button, when the user clicks on it they will be directed to the next incomplete node of the story. The url will be calculated by:

```
// Returns the exploration page URL for the provided chapter title. Kevin Thomas, 6 months
ctrl.getChapterUrl = function(node_id) {
  let node = this.storySummary.getAllNodes().find(node => {
    return node.getId() === node_id;
  });
  if (!node) {
    return '';
  }
  let urlParams = UrlService.addField(
    '', 'story_url_fragment', this.storySummary.getUrlFragment());
  urlParams = UrlService.addField(
    urlParams, 'topic_url_fragment', ctrl.topicUrlFragment);
  urlParams = UrlService.addField(
    urlParams, 'classroom_url_fragment', ctrl.classroomUrlFragment);
  urlParams = UrlService.addField(
    urlParams, 'node_id', node.getId());
  return (
    `${EXPLORE_PAGE_PREFIX}${node.getExplorationId()}${urlParams}`);
};
```

- The cards inside this section will be displayed in descending order ordered by the percentage of story completed by the user.
- If the `currentGoalsStoryList` is empty, we will instead display a message 'Hmm, this looks empty. Select a topic in the **To-Do List** to get started!'. The **To-Do List** link will direct the user to the **To-Do List** tab.

- **Suggested for you:**

- This will display three topics of a classroom.
- The data from the backend will contain a list of `TopicModel` instances. This data will be used to create a new list of `Topic` stored in `suggestedForYouList`.
- This data will be passed to `topic-summary-tile.directive`, which will display the data in card format, as presented in the mocks.
- Topics inside the same classrooms will be displayed together.
- This section will never be empty and will remain constant.

➤ **Progress tab:** A new folder called `progress` will be created inside the `core/templates/pages/learner-dashboard-page` folder which will contain the component for **Progress** tab.

- **Skill Proficiency:**

- This will display a list of all the subtopics of a topic started by the user or selected by the user in the **To-Do List** tab.
- The data from the backend will contain:
 - classroom_url_fragment
 - topic_url_fragment
 - topic_name
 - subtopic_id
 - Subtopic_name
 - avg_degree_of_mastery
- The avg_degree_of_mastery of the subtopic is displayed by a progress circle.
- If the avg_degree_of_mastery of a subtopic is between:
 - 70-79%, a bronze badge will be placed before the topic name.
 - 80-89%, a silver badge will be placed before the topic name.
 - 90-100%, a gold badge will be placed before the topic name.
- If the avg_degree_of_mastery of a subtopic is 0%, then no badge will be displayed.
- If the user selects more than one subtopic to practice on, then those subtopics will be played in succession. (Similar to practice-tab)
- **Topics in progress:**
 - This will display a list of all the topics that the user has started by not yet completed
 - When the user loads the Learner Dashboard page, data will be fetched from the IncompleteActivitiesModel for topic ids using the GET request as described above.
 - The data will contain a list of TopicModel instances of the incomplete topics. This data will be used to create a new list of Topic in the frontend called incompleteTopicsList.
 - This data will be passed to topic-summary-tile.directive, which will display the data in card format, as presented in the mocks.
 - If the suggestedForYouList is empty, we will instead display a message 'Hmm this looks empty. Try browsing our **library** for a lesson'. The library link will direct the learner to the default classroom page i.e. the math classroom.

- **Stories Completed:**
 - This will display a list of all the stories that the user has completed.
 - When the user loads the Learner Dashboard page, data will be fetched from the `IncompleteActivitiesModel` and `CompletedActivitiesModel` for story ids and `topic_ids` using the GET request as described above.
 - The data will contain a list of `StorySummaryModel` instances of the completed stories. This data will be used to create a new list of `StorySummary` in the frontend called `completedStoryList`. The incomplete topic ids will be stored inside the `incompleteTopicList` and `completedTopicList`.
 - If a story id is present in `completedStoryList`, that means it is also present inside:
 - `canonical_story_ids` field of `incompleteTopicList` OR
 - `canonical_story_ids` field of the `completedTopicsList`
 - Hence we will get the `topic_url` and the `classroom_url` from the `incompleteTopicList/completedTopicList` for the corresponding story id.
 - To display this data as presented in the mocks, we will create a new directive called `learner-dashboard-story-summary-tile` (as mentioned above for **Continue where you left off** section)
 - If the `completedStoryList` is empty, we will instead display a message 'Hmm this looks empty. Try browsing our **library** for a lesson'. The library link will direct the learner to the default classroom page i.e. the math classroom.

➤ **Todo List:**

- The **To-Do List** tab is divided into 3 sections namely: **Current Goals**, **Edit Goals** and **Completed Goals**.
- The **Edit Goals** section will contain data on all the topics that are present inside the server. This will be done inside the `learner_progress_service` as described above for **Suggested for you** section.
- For topics in the `current_goal` of the `TodoListModel`, the checkbox of that topic in the **Edit Goals** section will be filled with green.
- For topics in `completed_goal` of the `TodoListModel`, the checkbox of that topic in the **Edit Goals** section will be replaced with a book icon.

- The **Current Goals** section will contain the data present in the `current_goal` of the `TodoListModel`. The **Completed Goals** section will contain the data present in the `completed_goal` of the `TodoListModel`.

Third-party Libraries*

This project does not require any additional third party libraries.

Testing Approach

- Test files will be added whenever changes are made in `service.ts` files in the frontend along with respective PRs.
- All backend migration PRs will include test files with them.
- Unit test for the following [journey](#) or flow will be added:
- **Home** tab:
 - When the user loads the Learner dashboard page, the **Home** tab should open up.
 - It should start with a greeting to the learner.
 - It should display three incomplete stories in the **Continue where you left off** section.
 - It should direct the learner to the exploration when the user clicks on any story inside the **Continue where you left off** section.
 - It should display three topic cards and one grey card in the **Suggested For You** section.
 - It should direct the learner to the topic when the user clicks on any topic card inside the **Suggested For You** section.
 - It should direct the learner to the corresponding topic classroom when they click on the grey card in the **Suggested For You** section.
- **Progress** tab:
 - It should display all the topics in the incomplete and completed topic list inside the **Skill Proficiency** section.
 - It should display all the subtopics inside a topic when the learner clicks on the dropdown button inside the **Skill Proficiency** section.
 - It should direct the learner to correct subtopics when they select the subtopics and click the Start Practice button in the **Skill Proficiency** section.
 - It should display four incomplete topic cards in the **Topic in Progress** section.
 - It should direct the learner to the topic when they click on the topic card in the **Topic in Progress** section.
 - It should display all the stories (3 in a row) completed by the user in the **Stories Completed** section.
 - It should direct the learner to the story when they click on any story inside the **Stories Completed** section.

- **Todo-List** tab:
 - It should display all the topics that the learner selects in the **Edit Goals** section inside the **Current Goals** section.
 - It should display all the stories inside the topic when the learner clicks on the arrow button in the **Current Goals** section.
 - It should display all the topics present in the server in the **Edit Goals** section.
 - It should display all the topics completed by the user in the **Edit Goals** section inside the **Completed Goals** section.

- E2e tests will be added to check the flow of new tabs added inside the Learner Dashboard page as described in the Product design. This will include:
 - It should add the story selected by the user in the **Edit Goals** section of the **Todo-List** tab to the **Continue where you left off** section in the **Home** tab.
 - It should add three topics in the server to the **Suggested for You** section in the **Home** tab.
 - It should add all the incomplete and completed topics in the **Skill Proficiency** section of the **Progress** tab.
 - It should add all incomplete topics in the **Topic in Progress** section of the **Progress** tab.
 - It should add all the completed stories in the **Completed Stories** section of the **Progress** tab.
 - It should add all the topics selected by the user in the **Edit Goals** section in the **Current Goals** section of the **Todo-List** tab.
 - It should add all the topics present in the server in the **Edit Goals** section of the **Todo-List** tab.
 - It should add all the topics completed by the user in the **Edit Goals** section in the **Completed Goals** section of the **Todo-List** tab.

Milestones

I plan to have two big milestones based on the timeline provided by Google. Below are the detailed explanations of each milestone.

Community Bonding Period (May 17 - June 7)

During this period I will continue contributing to Oppia. Also, since I have been contributing to this community for more than 4 months now, I'm quite familiar with the codebase and the workflow. This is why I will start working on the project right away to avoid any unforeseen delay in the future.

The PR descriptions are as follows:

No.	Description of PR	Prereq PR numbers	Target date for PR submission	Target date for PR to be merged
0.1	Adding topic_ids and story_ids inside CompletedActivitiesModel and IncompleteActivitiesModel	None	19th May	21st May

Milestone 1 (June 7 - July 13)

Key Objective:

- Redesigning Learner Dashboard page to include **Home** tab and **Community Lessons**.
- This will include adding functions to store and fetch data from the storage models that are required to render and store data for learner dashboard pages.

No.	Description of PR	Prereq PR numbers	Target date for PR submission	Target date for PR to be merged
1.1	Create TodoListModel along with a validator, domain object.	None	7th June	13th June
1.2	<ul style="list-style-type: none"> • POST requests and GET requests to CompletedActivitiesModel IncompleteActivitiesModel for topic_ids and story_ids. • Along with the necessary functions for fetching, creating and deleting the data inside the models. 	0.1	12th June	18th June
1.3	<ul style="list-style-type: none"> • POST requests and GET requests to TodoListModel for topic_ids and story_ids. • Along with the necessary functions for fetching, creating and deleting the data inside the 	1.1	17th June	23th June

	models.			
1.4	<ul style="list-style-type: none"> • Add the Home tab inside the learner dashboard page. Fetching the data and displaying it to the user in Continue where you left off and Suggested for you sections. (The Continue where you left off section won't have any topics in it till the Todo-List section is built. So the Home tab component will be hidden under the ng-if tag based on the constant "HIDE_NEW_LEARNER_DASHBOARD". This will be removed once the TodoList section is built in the second milestone.) • This will include the necessary e2e, frontend and backend tests. 	1.1, 1.3	24th June	30th June
1.5	<ul style="list-style-type: none"> • Remove the In Progress and Completed tabs and make a new component called the Community Lessons in the learner dashboard. • This will include the necessary e2e, frontend and backend tests. 	None	1st July	7th July
1.6	Leaving the last week empty for solving any bug raised related to the project	N/A	N/A	N/A

Milestone 2 (July 16 - August 16)

Key Objective:

- Redesigning Learner Dashboard page to include **Progress** tab and **To-Do List** tab.

No.	Description of PR	Prereq PR numbers	Target date for PR submission	Target date for PR to be merged
2.1	<ul style="list-style-type: none">• Adding Progress tab in the learner dashboard page. Fetching the data and displaying it to the user in the Skill Proficiency section.• This will include the necessary e2e, frontend and backend tests.	0.1, 1.2	16th July	22th July
2.2	<ul style="list-style-type: none">• Further modifying the Progress tab in the learner dashboard page. Fetching the data and displaying it to the user in the Topics in Progress and Completed Stories section.• This will include the necessary e2e, frontend and backend tests.	0.1, 1.2, 2.1	21th July	27rd July
2.3	<ul style="list-style-type: none">• Adding the To-Do List tab inside the learner dashboard page. Fetching the data and displaying it to the user in Current Goals, Edit Goals and Completed Goals section.• This will include the necessary e2e, frontend and backend tests.	0.1, 1.3	30th July	7th August
2.4	Leaving the last week empty for solving any bug raised related to the project	N/A	N/A	N/A

Optional Sections

Future Work

In future, we can ask the learner beforehand about their preferences in the classrooms. This way the learner dashboard page can be customized according to the user's preference like suggesting topics of the classrooms that the user chooses or visits the most.

Additional Project-Specific Considerations

Documentation Changes

This project aims to add 4 new tabs in the learner dashboard page and will therefore need no additions to the Oppia wiki page.