

About You

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

I have been a part of the Oppia Organization for about an year and I can proudly say that Oppia is a very welcoming, supportive and noble Organization with an equally noble aim to make education accessible to all parts of the world and mainly those which really require that. I have chosen Editor Page Redesigns since I have worked with Oppia's Frontend Part quite a lot and secondly, I love creating user facing stuff.

Prior experience

I have worked on building a lot of web apps and websites(a few from scratch even) which is exactly what i need for this project too. [Codersera.com](https://codersera.com), this is a web-app build by me from scratch single handedly in Angular, and is fully optimized for mobile devices as well as tablets, is a PWA as well, is SEO optimized(used to score 100 on Google lighthouse and audits), has an AMP page too and has Google Analytics integrated and is optimized for speed.I have worked on quite some full stack projects and hence my UI/UX skills are good. I am a Microsoft Student Partner, and here's my [portfolio](#).

PR's :

Oppia Python script and a github action: <https://github.com/oppia/oppia/pull/8334>

The problem was that there was no way to be sure that the e2e tests and the e2e tests that run on Travis are in sync. Solved it by adding a script and a subsequent action that will be triggered every time code is pushed to the Oppia repo. It was a lot of fun since I used regex and capture groups.

Oppia Angular Migration: <https://github.com/oppia/oppia/pull/7935>

Frontend: Migrating all the frontend filter.ts files. Quite a big PR.

Oppia E2e test: <https://github.com/oppia/oppia/pull/8455>

Added E2e test for admin one off jobs. Adding this PR because this was quite a challenging one. Used Recursion to solve since the problem was that the iterative approach was failing.

Oppia UI blocking bug: <https://github.com/oppia/oppia/pull/8673>

Solved a blocking bug in the last testing release.

Oppia Topic Dashboard Redesign: <https://github.com/oppia/oppia/pull/8710>[Not yet merged]
This is a PR that is in the scope of this project. Made that to get a feel of the project and to know how much time is designing gonna take.

Contact info and timezone(s)

My email and hangouts: coolrishabhrawat@gmail.com

My time zone is: IST(Indian Standard Time GMT+5:30)

I have seen that the Oppia Community generally uses Hangout to talk and meet, and I am pretty comfortable with Hangout. But I can also work with Slack, Gitter and Zoom if the mentor suggests that.

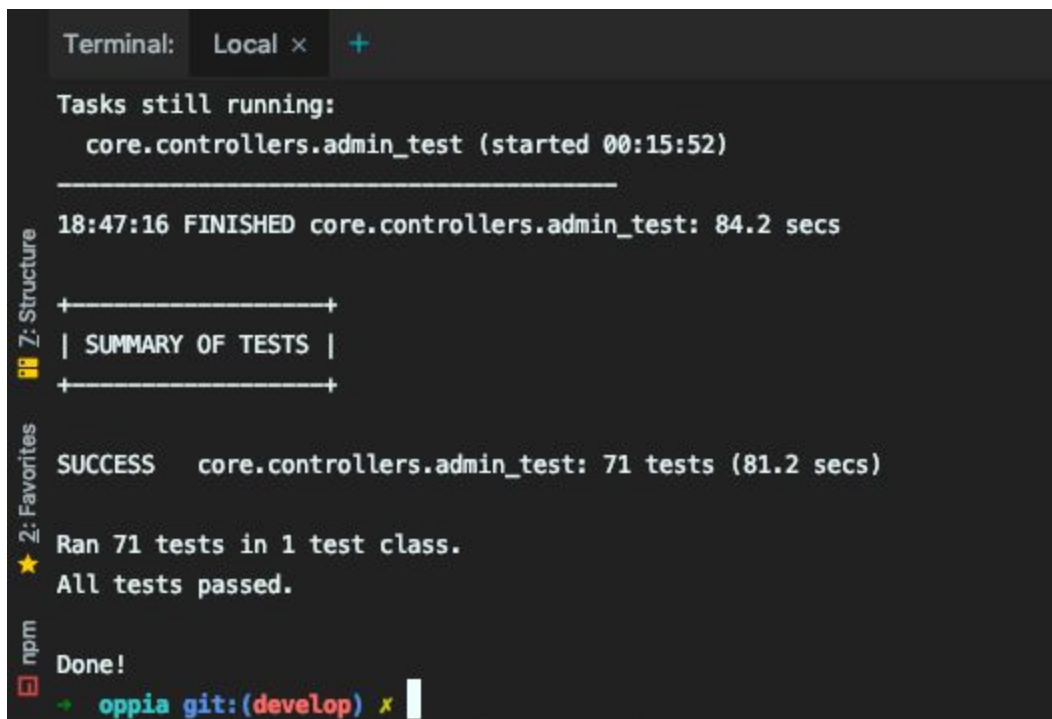
Time commitment

I think around 7-8 hours per day and about 50 hours per week(which I can extend upto 70 hours per week if the need arises.)

Essential Prerequisites

Answer the following questions:

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



```
Terminal: Local x +
Tasks still running:
  core.controllers.admin_test (started 00:15:52)
-----
18:47:16 FINISHED core.controllers.admin_test: 84.2 secs
+-----+
| SUMMARY OF TESTS |
+-----+
SUCCESS   core.controllers.admin_test: 71 tests (81.2 secs)
Ran 71 tests in 1 test class.
All tests passed.
Done!
+ oppia git:(develop) x
```

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

```
Terminal: Local x +
HeadlessChrome 80.0.3987 (Mac OS X 10.14.6): Executed 1965 of 1975 SUCCESS (0 secs / 39.267 secs)
ERROR: 'Error communicating with server.'
HeadlessChrome 80.0.3987 (Mac OS X 10.14.6): Executed 1966 of 1975 SUCCESS (0 secs / 39.286 secs)
HeadlessChrome 80.0.3987 (Mac OS X 10.14.6): Executed 1975 of 1975 SUCCESS (46.897 secs / 39.431 secs)
TOTAL: 1975 SUCCESS
TOTAL: 1975 SUCCESS
31 03 2020 00:04:41.983:WARN [launcher]: ChromeHeadless was not killed in 2000 ms, sending SIGKILL.
Done!
+ oppia git:(develop) |
```

- I am able to run one suite of e2e tests on my machine. (Show a screenshot of a successful test.)

```
/Users/apple/codebase/opensource/oppia_tools/google_appengine_1.9.67/google_a
many files in your application for changes in all of them to be monitored. Yo
'There are too many files in your application for '
*
screenreader and keyboard user accessibility features
  ✓ should skip to the main content element

Ran 1 of 2 specs
1 spec, 0 failures
Finished in 17.403 seconds

Executed 1 of 2 specs INCOMPLETE (1 SKIPPED) in 17 secs.
[00:14:12] I/launcher - 0 instance(s) of WebDriver still running
6: TODO 9: Version Control Terminal TypeScript
```

Other summer obligations

I am totally free this summer, except for my exams which might happen around July(can't say for sure due to CoronaVirus) but i'll make sure they don't prove to be a roadblock.

Communication channels

I think updates once in every three days would be good(daily updates can also be done in case the mentor insists). I am pretty comfortable with Google Hangouts which the Oppia Community

generally uses to meet, I am also comfortable with Slack and Zoom to Meet and Gmail, Hangouts, Gitter to chat.

Project Details

Product Design

The users for whom this project is concerned are the Lesson Creators that include (Topics, Skill and Explorations Creators.)

The mockups have been provided here:

Desktop:

<https://xd.adobe.com/view/a496f74b-80d1-4d8f-78b5-773b61ea1479-d2a7/grid>

Mobile:

<https://xd.adobe.com/view/84eb3b8d-3d19-4971-7e79-9ff756b25c83-f2da/grid>

Once this project is complete, I aspire to achieve that:-

1. The Topic Creation, Skill Creation, Exploration Editor, Topic and Skill Dashboard page are all in accordance to the new and updated mockups.
2. The pages like Topic and Skill Dashboard page are more accessible and useful since new components like Pagination, Filters, Total Count need to be added.
3. A lot of user journeys that are in the scope of this project(almost all) like topic creation, skill creation etc are not mobile friendly as of now. This project aims to make these journeys user friendly, so this is gonna be a huge plus point.

Technical Design

Architectural Overview

Since this is a redesign project majorly concerning the UI/UX, a lot of new structures or features are not supposed to be made. So the major changes are to be in the frontend part of the codebase that is in AngularJs and CSS(directive.html files, controller.ts files)

and in the Existing models and the corresponding controllers and services because some fields might need to be added or removed.(domain.py, controller.py files, gae_models) and subsequent migrations or One-off jobs.

Implementation Approach

1. Topic and Skill Dashboard and Topic and Skill Editor Page Redesign:

The screenshot displays the Oppia 'Topics and Skills Dashboard'. The header includes the Oppia logo and a 'CREATE' button. The main content area is divided into two tabs: 'TOPICS' and 'SKILLS'. The 'SKILLS' tab is active, showing a table of skills. The table has columns for 'Details', 'Worked Examples', 'Misconceptions', and 'Status'. Three skills are listed, all marked as 'UNASSIGNED'. To the right of the table, there are summary cards for '11 Topics' and '6 Skills', and a 'Filter Skills' section with dropdowns for 'Keywords', 'Status', 'Category', and 'Sort'.

Details	Worked Examples	Misconceptions	Status
1. Cross Cancelling UNASSIGNED ⚠️ Cross cancelling is a way to simplify before we multiply. This can save us from dealing with large numbers in our product.	1	2	UNASSIGNED
2. Multiplying Mixed Fractions UNASSIGNED ⚠️ Before multiplying, we need to write the mixed numbers as improper fractions.	1	1	UNASSIGNED
3. Subtracting Polynomials with Different Variables UNASSIGNED ⚠️ Replace given fractions with equivalent fractions in such a way as to produce an equivalent sum or	2	1	UNASSIGNED

Frontend Section: `core/templates/pages/topics-and-skill-dashboard-page/*`

Backend Section: `core/domain/topic_*.py`

`core/controller/topics_and_skills_dashboard*.py`

Frontend Changes:

1. The logic for Pagination needs to be implemented.

For the pagination logic, we need the topics in batches or groups like 0-10, 11-20, 21-30 and so and so forth.

This can be done in two ways:

1. Sending the data/topics in batches from the backend(using limit and offset).
2. Sending all of the topics and manipulating them in the frontend.

For the first approach, more bandwidth and more backend API calls will be required since every time the page is changed, an API call will be made. (I am implementing this for the Skills though since they can be large in number) But it will be easier for the device and less computational power will be required. One thing that can be done to improve this approach can be to cache the data and save the data corresponding to the page number so that no two identical calls are made.

For the second approach, API calls will be made just once and the topics are not going to be too large in numbers and since I think the devices are capable enough now to do the computations in the frontend almost instantaneously so I preferred this method.

The pagination for the Topic dashboard can be implemented in the frontend only, with the assumption that topic count is not going to be too large. And since the same dashboard is being used for both topics and skills, the state is also supposed to be remembered for a better UX like which page is the user on the Topics Dashboard and which page is the user on the Skills Dashboard.

```
ctrl.paginationHandler = function(e) {  
  if (ctrl.activeTab === ctrl.TAB_NAME_TOPICS) {  
    ctrl.totalCount = ctrl.totalTopicSummaries.length;  
    ctrl.topicPageNumber = e;  
    ctrl.pageNumber = ctrl.topicPageNumber;  
    ctrl.topicSummaries =  
      ctrl.totalTopicSummaries.slice(e * 10, (e + 1) * 10);  
  } else if (ctrl.activeTab === ctrl.TAB_NAME_SKILLS) {  
    ctrl.totalCount = ctrl.totalSkillSummaries.length;  
    ctrl.skillPageNumber = e;  
    ctrl.pageNumber = ctrl.skillPageNumber;  
    ctrl.SkillSummaries =  
      ctrl.totalSkillSummaries.slice(e * 10, (e + 1) * 10);  
  }  
};
```

For the Skills dashboard, since the quantity of skills can be large, the pagination logic needs to take place in the backend. So the changes that can be made to achieve this are

A new API should be added in the backend `topic_and_skills_dashboard.py` and the fetching of topics and skills should be separated so that if we need to retrieve some skills, we don't receive topics also and vice versa(since that will reduce speed and improve the response size unwantedly.)

These changes should be done in `feconf.py`

Current:

```
TOPICS_AND_SKILLS_DASHBOARD_DATA_URL = '/topics_and_skills_dashboard/data'
```

New:

```
TOPICS_AND_SKILLS_DASHBOARD_TOPIC_DATA_URL =
```

```
'/topics_and_skills_dashboard/data/topic'
```

```
TOPICS_AND_SKILLS_DASHBOARD_SKILL_DATA_URL =
```

```
'/topics_and_skills_dashboard/data/skill'
```

Then to map these variables to actual routes and endpoints, the subsequent changes need to be made in the `main.py` file:

Current:

```
get_redirect_route(  
    r'%s' % feconf.TOPICS_AND_SKILLS_DASHBOARD_DATA_URL,  
    topics_and_skills_dashboard.TopicsAndSkillsDashboardPageDataHandler),
```

New:

```
get_redirect_route(  
    r'%s' % feconf.TOPICS_AND_SKILLS_DASHBOARD_TOPIC_DATA_URL,  
    topics_and_skills_dashboard.TopicsAndSkillsDashboardPageTopicDataHandler),  
  
get_redirect_route(  
    r'%s' % feconf.TOPICS_AND_SKILLS_DASHBOARD_SKILL_DATA_URL,
```

```
topics_and_skills_dashboard.TopicsAndSkillsDashboardPageSkillDataHandler),
```

Then in class `TopicsAndSkillsDashboardPageDataHandler(base.BaseHandler)` present in the `topics_and_skill_dashboard.py`, the skills part needs to be moved from the original method to the new method that's created. The new method and api that is for skills specifically will have a limit as 10 and an offset so that the pagination works. This is going to be implemented in the following way:

I'll add a function in the [skill_services.py](#) to get the skills in batches with a specified offset(10,20,30 etc)

[topics_and_skills_dashboard.py](#)

```
skill_summaries = skill_services.get_skill_summaries_in_batch(offset)
```

[skill_services.py](#)

```
def get_skill_summaries_in_batch(offset=0):
    """Returns the summaries of all skills present in the datastore in batches.

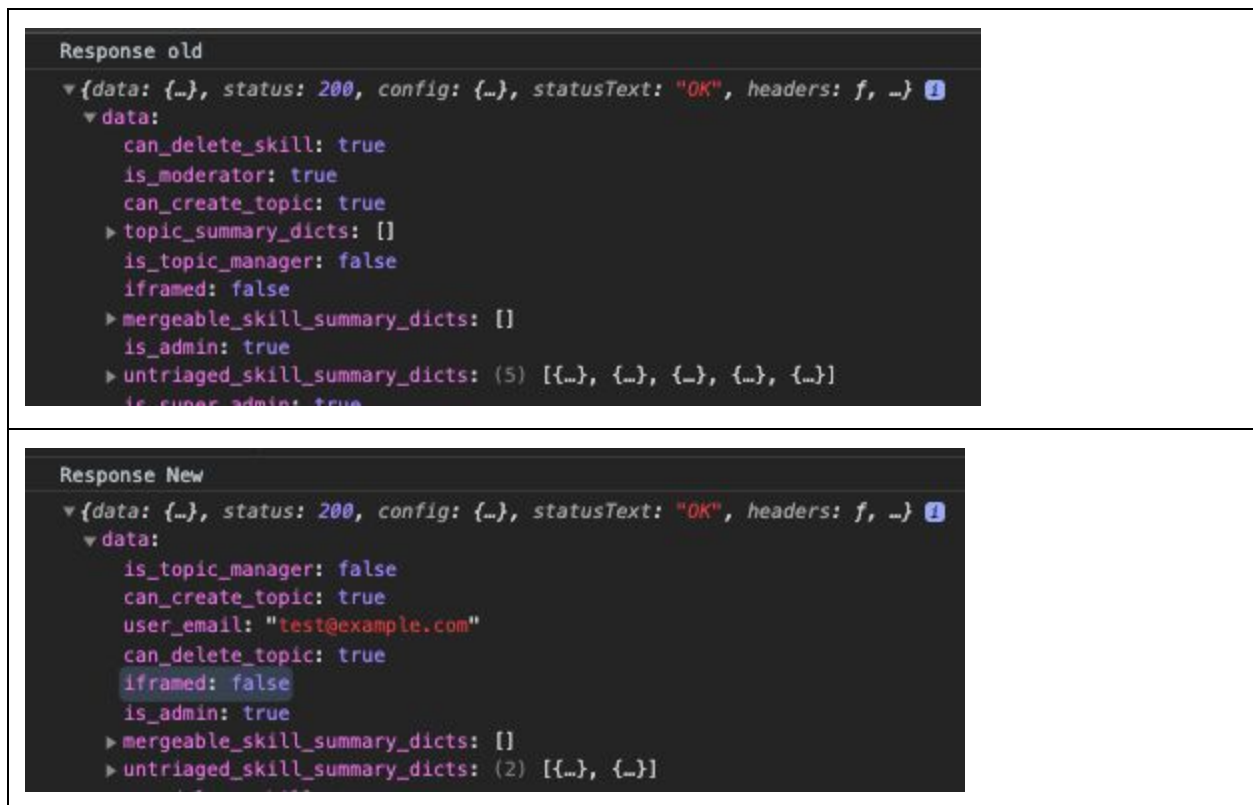
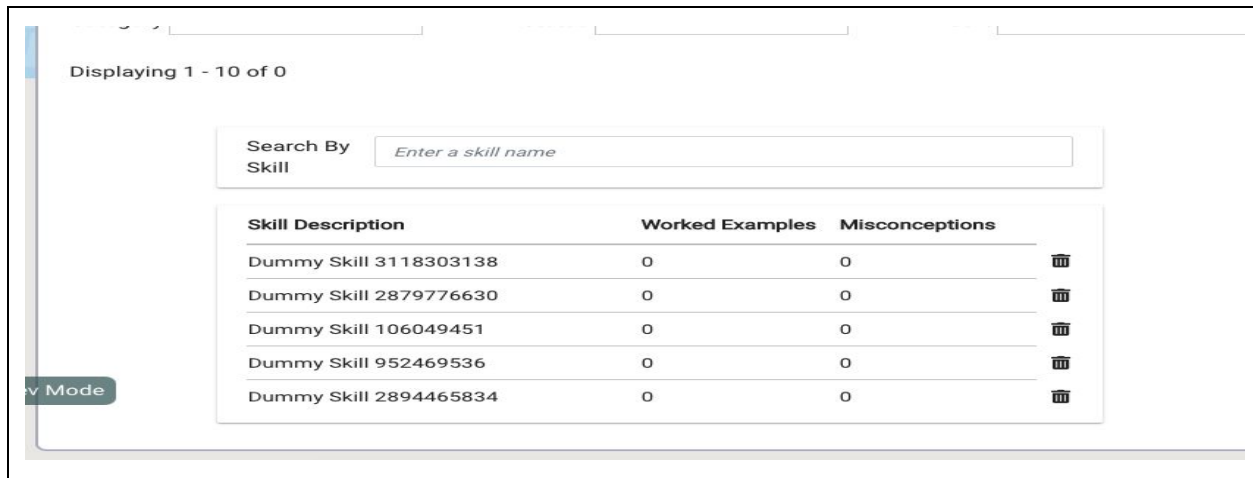
    Returns:
        list(SkillSummary). The list of summaries of all skills present in the
        datastore.
    """
    skill_summaries_models = skill_models.SkillSummaryModel.get_in_batches(offset)
    skill_summaries = [
        get_skill_summary_from_model(summary)
        for summary in skill_summaries_models]
    return skill_summaries
```

[base_model/gae_models.py](#)

```
@classmethod
def get_in_batches(cls):
    """Gets iterable of all entities of this class with the following offset.

    Returns:
        iterable. Filterable iterable of all entities of this class.
    """
    query = cls.query().fetch(limit=10, offset=3)
    return query
```


I have tested the above approach and found it be working. Firstly, I created 5 skills and fetched them all using the existing api then i tested it using the new api that i created as described above.



I made the original call and console.logged it then I made the call triggering the new function and used the offset as 3 so i got the last two responses.

Then, the changes needed to be reflected in the frontend also, so a new api needs to be added in the [topics-and-skills-dashboard-backend-api.service.ts](#) and the already existing one needs to be modified.

```
var _fetchDashboardTopicData = function() {  
    return $http.get('/topics_and_skills_dashboard/data/topic');  
};  
var _fetchDashboardSkillData = function() {  
    return $http.get('/topics_and_skills_dashboard/data/skill', {offset: 30});  
};
```

And the subsequent changes need to be made in the frontend part also, to be specific, [topics-and-skills-dashboard-page.controller.ts](#) needs to be changed to call both the modified api's(as shown above) i.e the `_initDashboard();` needs to be modified accordingly.

2. A Count needs to be maintained for the Total Topics and Total Skills for the right box.

```
ctrl.totalTopicCount = ctrl.topicSummaries.length;  
ctrl.totalSkillCount = ctrl.totalUntriagedSkillSummaries.length;
```



3. A new Modal needs to be made for the Topic and Skill Creation. \$uibModal can be used for this. (Existing Modals can be edited for this section whenever applicable, but the button shown above requires a modal with both the Create New Topic and Create New Skill together, and hence new Modal is needed.)
4. Filters: To implement filters in the Topic and Skill Dashboard, and anywhere else I think a [generic filter service](#) would be really helpful. Something like this:

```
// topics, ['category', 'name', 'author'] ['english', 'rishabh', 'richard']

filterArray(originalArray, fieldsToSearch, valuesToSearch){
  int i = valuesToSearch.length;
  originalArray.filter((ele) => {
    for(i=0;i<len;i++){
      if(fields[i]==values[i]){
        return true;
      }
    }
    return false;
  })
}
```

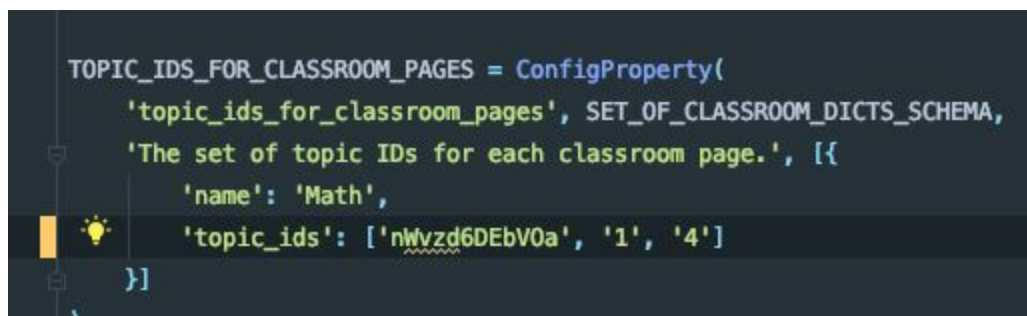
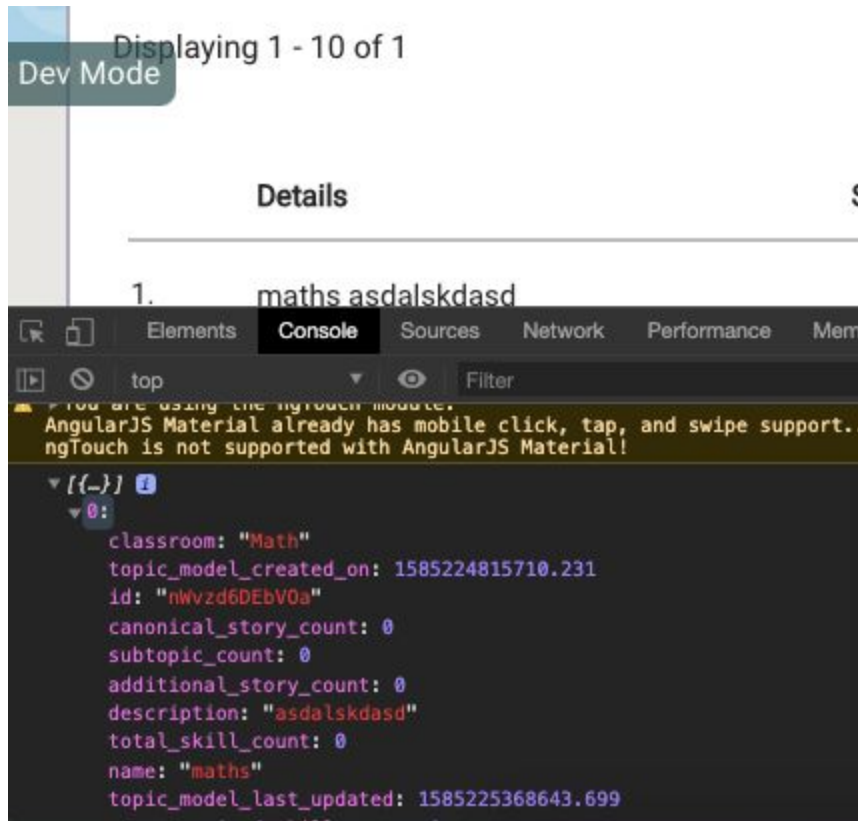
Backend Changes:

1. For the category filter, we need the classroom name for which the topic belongs so we need to add that value in the topic summary dict and return that in the api request, for the topic and skill dashboard in the frontend. So for this, we can import [config_domain from core.domain](#), and iterate over the [TOPIC_IDS_FOR_CLASSROOM_PAGES](#) and the topic_summaries and compare their topic_ids, and if the topic_id are same, then it signifies that this particular topic belongs to this class and then the topic dict can be update appropriately.

This code below can be added to the [topics_and_skills_dashboard.py](#):

```
from core.domain import config_domain
```

```
for topic_summary_dict in topic_summary_dicts:  
    for classroom_dict in config_domain.TOPIC_IDS_FOR_CLASSROOM_PAGES.value:  
        for topic_id in classroom_dict['topic_ids']:  
            if topic_id == topic_summary_dict['id']:  
                topic_summary_dict['classroom'] = classroom_dict['name']
```



I found out that this approach is working.

Html and CSS Changes:

1. A new modal needs to be added that has the Create Topic and Create Skill button.



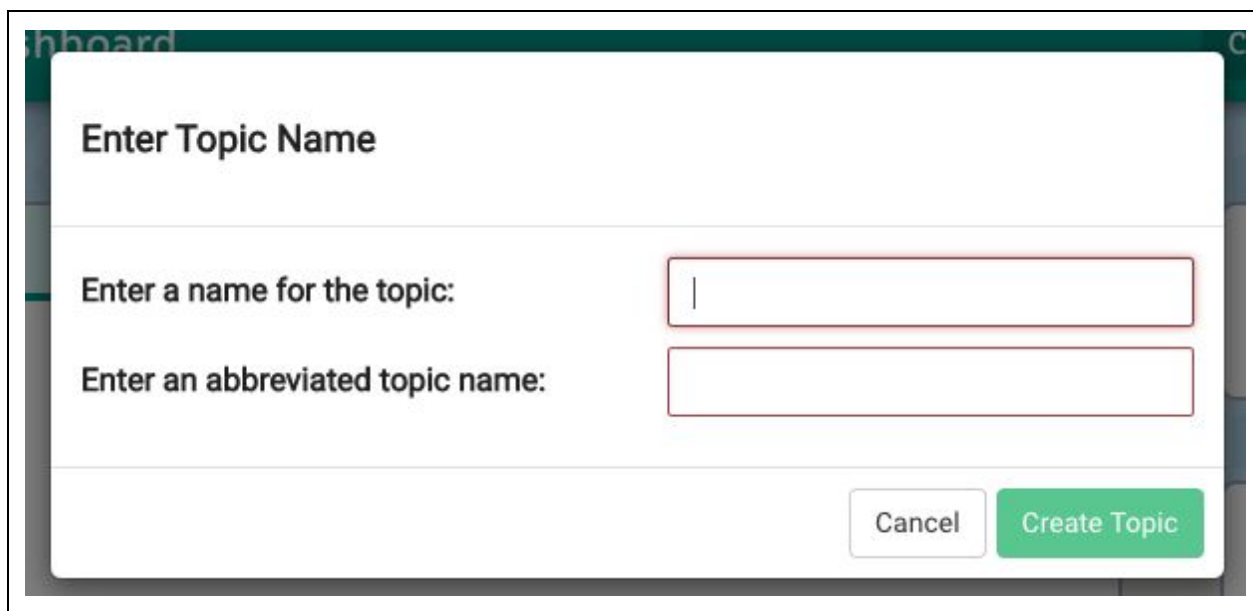
```
<div class="create-topic-skill-modal">
  <div class="create-topic-skill-modal-buttons">
    <button class="btn oppia-dashboard-intro-button oppia-transition-200
protractor-test-create-topic-button"
      style="color: white; text-decoration: none;"
      ng-click="createTopic()"
      ng-if="userCanCreateTopic">
      Create Topic
    </button>
    <button class="btn oppia-dashboard-intro-button oppia-transition-200
protractor-test-create-skill-button"
      style="color: white; text-decoration: none;"
      ng-click="createSkill()"
      ng-if="userCanCreateSkill">
      Create Skill
    </button>
  </div>
  <div class="modal-footer create-topic-skill-modal-footer">
    <button class="btn btn-secondary" ng-click="cancel()">Cancel</button>
  </div>
</div>
```

And it can be linked to the Topic & skill dashboard in the following manner:

```
$uibModal.open({
  templateUrl: UrlInterpolationService.getDirectiveTemplateUrl(
    '/pages/topics-and-skills-dashboard-page/templates/' +
    'create-new-topic-or-skill-modal.template.html'),
  backdrop: 'static',
  controller: [
```

```
'$scope', '$uibModalInstance',  
function($scope, $uibModalInstance) {  
  $scope.userCanCreateTopic = ctrl.userCanCreateTopic;  
  $scope.userCanCreateSkill = ctrl.userCanCreateSkill;  
  
  $scope.createTopic = ctrl.createTopic;  
  $scope.createSkill = ctrl.createSkill;  
  $scope.cancel = function() {  
    $uibModalInstance.dismiss('cancel');  
  };  
}  
});
```

Topic and Skill Editor:



The screenshot shows a modal dialog titled "Enter Topic Name". It contains two input fields: "Enter a name for the topic:" and "Enter an abbreviated topic name:". At the bottom right, there are two buttons: "Cancel" and "Create Topic".

Enter Topic Name

Enter a name for the topic:

Enter an abbreviated topic name:

New Topic

Category*

View an example [View an example](#)

Select one

Name*

View an example [View an example](#)

Description*

View an example [View an example](#)

CANCEL

CREATE TOPIC

Editor: alpha (v2)

Save Changes

Publish Topic


Topic

Subtopics

Questions

Preview

Topic Thumbnail



Topic Name

alpha

Abbreviated Topic Name

alpha

Description

asdasdasd

Canonical Stories

Click on Create Story to create a new canonical story for the topic

Create Story

Topic Editor

Polynomials

Details

Category*

View an example

Mathematics

Name*

View an example

Polynomials

Description

View an example

We tackle advanced topics like the Binomial Theorem and the Fundamental Theorem of Algebra. We will also perform more challenging polynomial division and dive deeper into the graphical method for solving equations.

Thumbnail

Stories

+ ADD STORY

	Name	
1.	Sydney's Ball Gown	
2.	Cooking Conundrum	
3.	Lost in the Woods!	

0

Stories

0

Subtopics

0

Skills

SIMULATE FILLED

The New Topic and New Skill modals, delete modals needs to be edited(which more or less have the same functionality as of the current modals)

Topic Editor

Polynomials

Details

Category*

View an example

Mathematics

Name*

View an example

Polynomials

Description

View an example

We tackle advanced topics like the Binomial Theorem and the Fundamental Theorem of Algebra. We will also perform more challenging polynomial division and dive deeper into the graphical method for solving equations.

Thumbnail

Stories

+ ADD STORY

	Name	
1.	Sydney's Ball Gown	
2.	Cooking Conundrum	
3.	Lost in the Woods!	

0

Stories

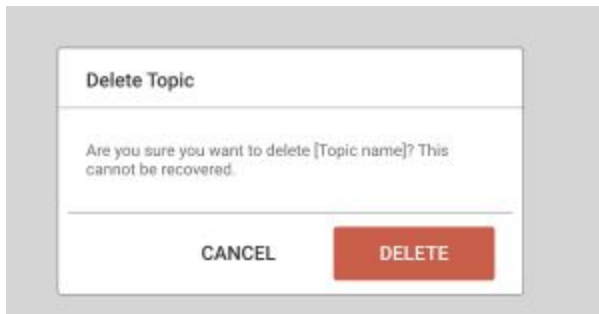
0

Subtopics

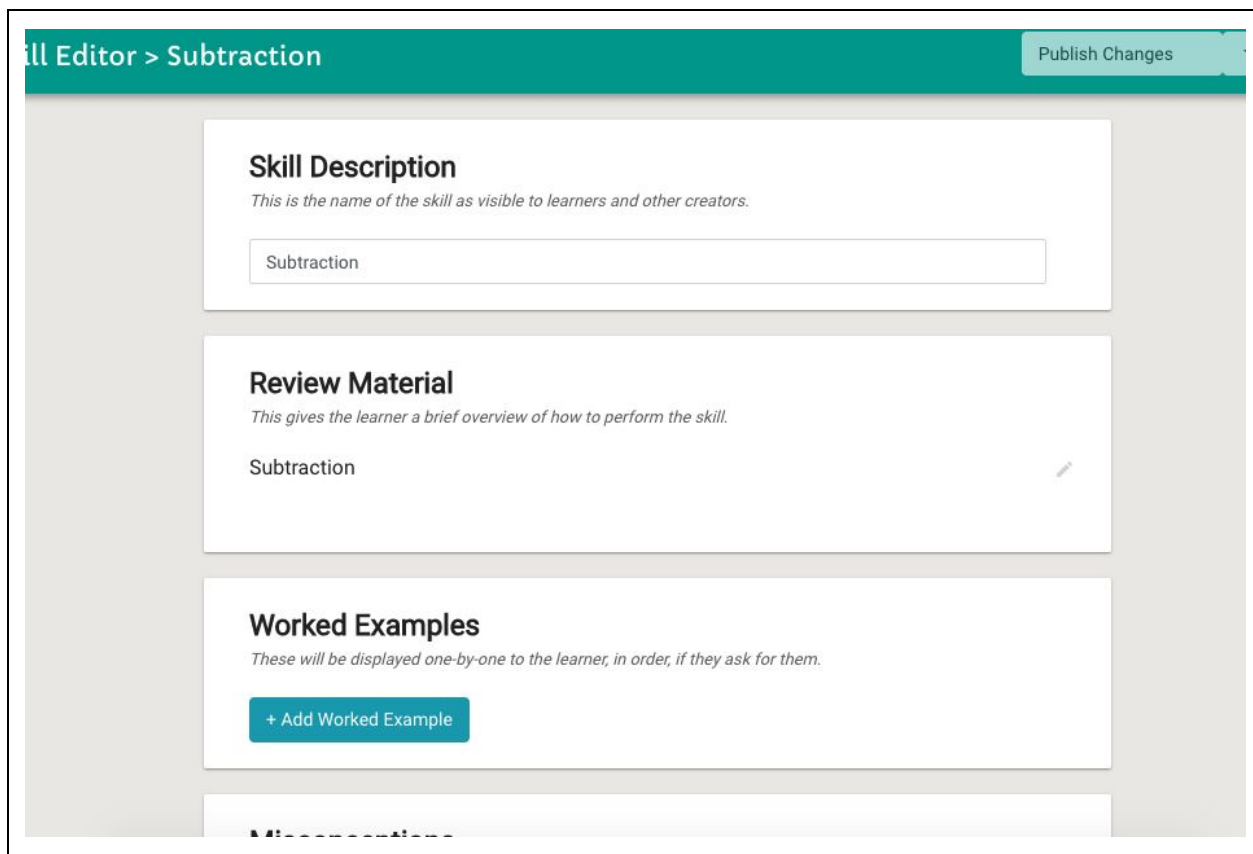
0

Skills

SIMULATE FILLED FIEL



The Delete Topic modal needs to be made as the mockup above.



The Topic Editor page needs to be modified as above .

The Skill Editor needs to be reduced by width, a right modal with Worked Example count, Misconception Count needs to be added. The UI needs to be improved and The Skill Editor page needs to be modified too to look like the above mock.

The delete skill modal needs to be updated according to the mockup above.

Question Editor:

The Question Editor needs to be broken down into modals, and each modal represent a step in the question. Implementing this is a bit tricky, but my approach is gonna be to create a object and store all the inputs for the question like difficulty, hints and answers

in that object and once it's done(that is the author clicks the final SAVE button) then i'll call the appropriate functions of the state editor. This is because ultimately, I'll use the state-editor behind the scenes, but the UI needs to be different from the state-editor UI.

I'll just hide and show stuff based on the current step(step1, 2 or 3) in the question editor and separate directives would not be required.

Majority of the changes would be done in [question-editor.directive.html](#) and [question-editor.directive.ts](#). This can be done by firstly, implementing the html and then creating different functions to update different properties of the [questionObject](#). Once the save button is clicked then the original functions will be called.(Rough implementation below)

```
ctrl.updateQuestionObject = (key, value) => {  
  ctrl.questionObject[key] = value; // the template will be created accordingly  
}  
  
ctrl.questionObject = { // The final Question Object  
  hints: hints,  
  answers: answers,  
  outcome: outcome  
}  
  
onSaveButtonClicked = () => {  
  StateEditorService.setInteractionDefaultOutcome(  
    angular.copy(questionObject.outcome));  
  });  
  StateEditorService.setInteractionCustomizationArgs(  
    angular.copy(questionObject.CustomizationArgs));  
  });  
  StateEditorService.setInteractionCustomizationArgs(  
    angular.copy(questionObject.CustomizationArgs));  
  });  
}
```

New Question

**required*

1 Problem ————— 2 Answers ————— 3 Responses

Select Difficulty*

- ☐ Easy: Identify coefficients and like terms.
- ☐ Easy: Break down monomials by factoring.
- ☐ Medium: Solve multistep problems, including word problems, by adding, subtracting, multiplying, and dividing polynomials.
- ☐ Medium: Factor quadratics to be the product of two or more numbers, monomials, or polynomials.
- ☐ Medium: Add/Subtract the polynomial equations using the column method.

CANCEL

NEXT

New Question

**required*

✓ Problem ————— 2 Answers ————— 3 Responses

Multiple Choice Answers

Fill out at least 2 multiple choice options.

Answer 1

Answer 2

CANCEL

NEXT

New Question *required

✓ Problem
✓ Answers
3 Responses

Responses

If the learner answers:

Select multiple choice option

Oppia tells the learner:

B I [Rich Text Editor Icons]

CANCEL SAVE

1. Question Editor Implementation

To update the question editor, my approach is to change the UI and the code in the frontend and not to change any API's or their fields. To achieve this the following needs to be done:

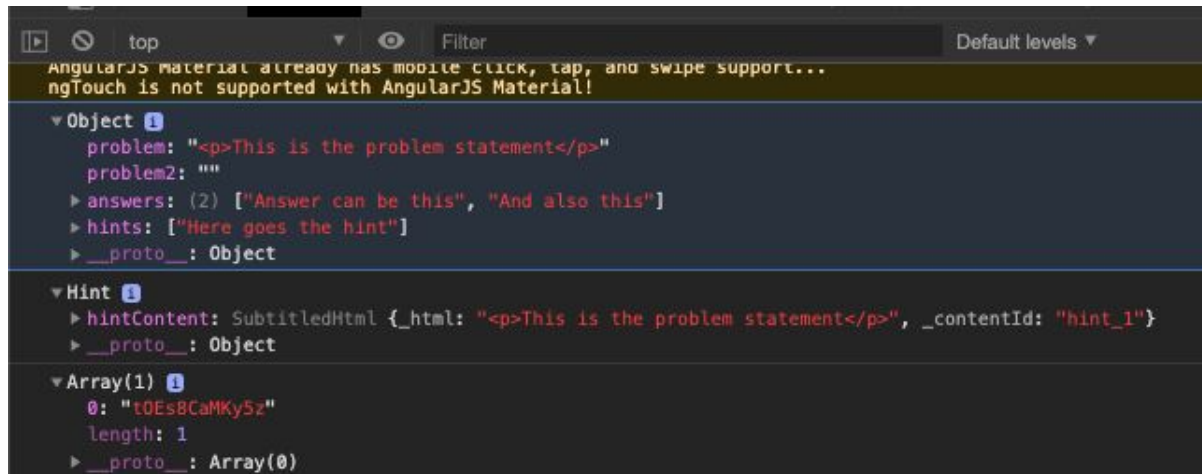
- The first step will be to update the UI for the question editor modal and show different screens inside the modal on the basis of the **step**(step1, 2 or 3).

```
$scope.changeStep = function(step) {
    $scope.step = step;
};
```

- A new Service will be created(similar to state-editor) which will be linked to HintObjectFactory, SolutionObjectFactory, AnswerGroupObjectFactory etc to provide all the parts that a Question requires and this service will be injected to the question-list directive from where the question editor modal is called.

Tried to implement this partially, Video attached [here](#).

The console.log() from the above video(Tried to assemble all the objects required like Hint with the manual UI that I created)



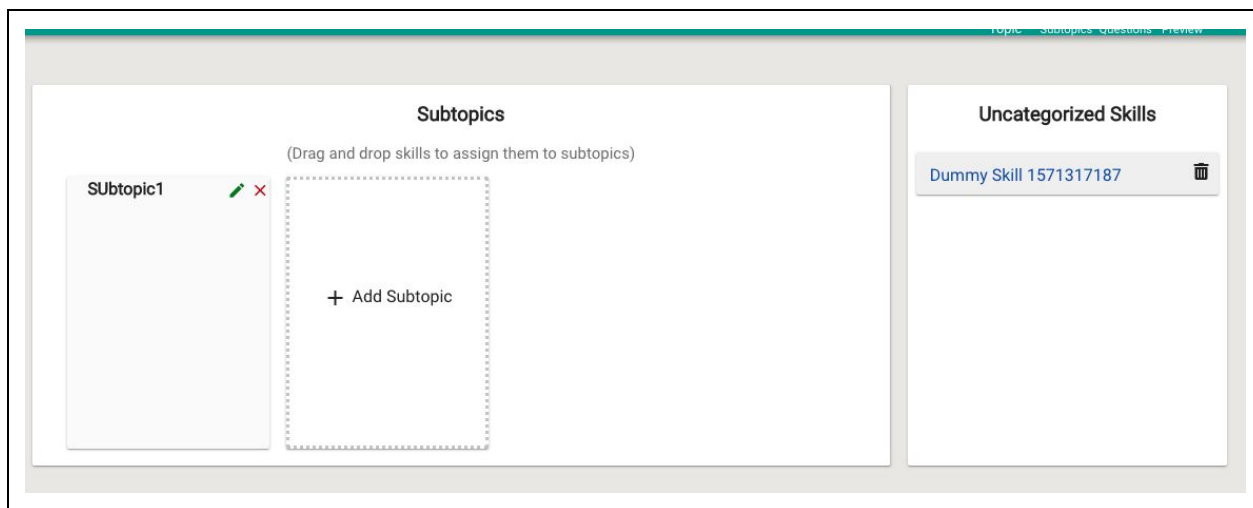
```
AngularJS Material already has mobile click, tap, and swipe support...
ngTouch is not supported with AngularJS Material!

▼ Object
  problem: "<p>This is the problem statement</p>"
  problem2: ""
  ▶ answers: (2) ["Answer can be this", "And also this"]
  ▶ hints: ["Here goes the hint"]
  ▶ __proto__: Object

▼ Hint
  ▶ hintContent: SubtitledHtml {_html: "<p>This is the problem statement</p>", _contentId: "hint_1"}
  ▶ __proto__: Object

▼ Array(1)
  0: "t0EsBCaMKy5z"
  length: 1
  ▶ __proto__: Array(0)
```

Subtopic Editor:



Subtopic Editor

Polynomials / Adding and Subtracting Polynomials

Review Material

Title*

View an example

Adding and Subtracting Polynomials

Explanation of the Subtopic*

Gives learners a brief overview of how to perform a skill. View an example:

B

I

$\frac{1}{x}$

$\frac{1}{x^2}$

$\frac{1}{x^3}$

$\frac{1}{x^4}$

$\frac{1}{x^5}$

$\frac{1}{x^6}$

$\frac{1}{x^7}$

$\frac{1}{x^8}$

$\frac{1}{x^9}$

$\frac{1}{x^{10}}$

To subtract polynomials, first reverse the sign of each term we are subtracting (in other words turn "+" into "-", and "-" into "+"), then add as usual.

Thumbnail Image*

Status

Topic

Polynomials

0

Skills

0

Questions

+

CREATE NEW SKILL

2. Story editor page redesign:

This is how the Story editor page looks as of now:

Return to topic

Story Title

File1

Story Description

Enter the description of the story

Notes

Add notes about the story to help other contributors.

Chapters

Select Initial Chapter

File1

File1

↓

File2

Chapter Title

File1

Chapter Outline

Outline Finalized ☐

B

I

$\frac{1}{x}$

$\frac{1}{x^2}$

$\frac{1}{x^3}$

$\frac{1}{x^4}$

$\frac{1}{x^5}$

$\frac{1}{x^6}$

$\frac{1}{x^7}$

$\frac{1}{x^8}$

$\frac{1}{x^9}$

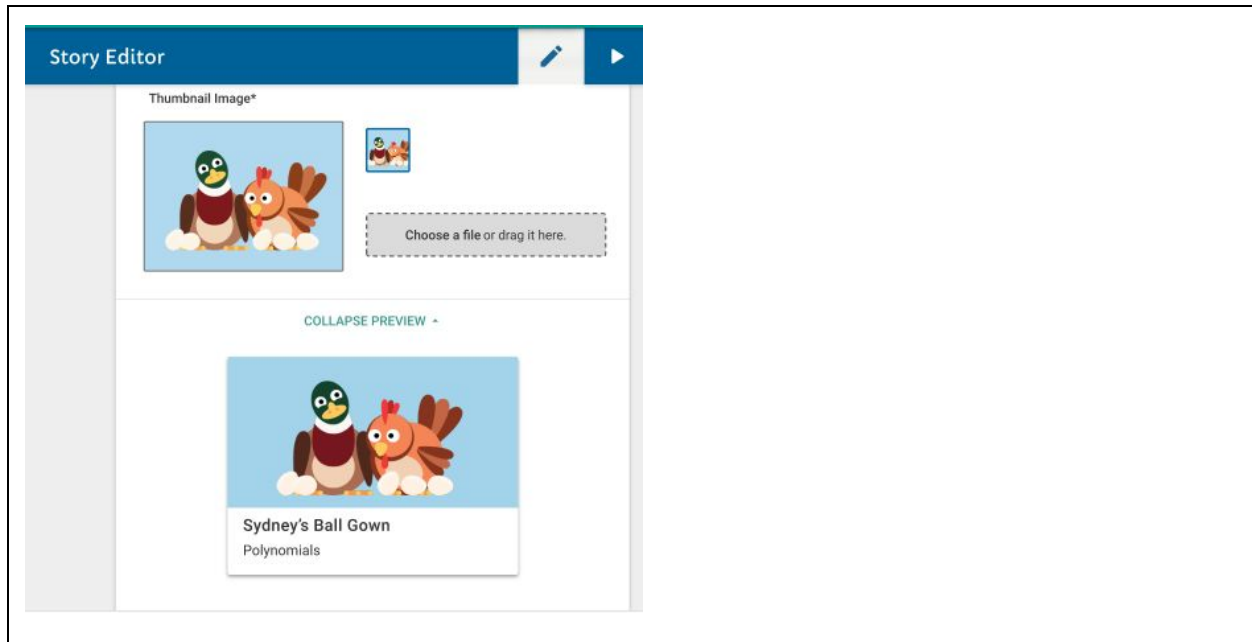
$\frac{1}{x^{10}}$

Exp 1sadasd

Save

Exploration ID

The major changes that seems to be done are some added functionality, some added modals and a lot of UI changes,



The support for ~~Thumbnail functionality~~ needs to be added which is shown in the new mocks. This does not require a lot of brainstorming since we have an identical feature in the Create Topic part and a subsequent Preview part needs to be created which will Display the Image, Chapter Name and Topic Name.(Supposed to be completed before GSoc)

1. Making the story inputs(upper left part) condensed into a card which can be easily done using md-card and some CSS.The chapter list should be draggable(take hint from subtopics tab in topic editor).To implement this(below is the approach, not the actual implementation)

```
<div ng-repeat="node in linearNodesList">
  <md-card class="story-editor-node"
    ng-class="{ 'selected-node': (node.getId() === idOfNodeToEdit)}">
```



```
ng-click="setNodeToEdit(node.getId())"  
dnd-draggable="node"  
dnd-effect-allowed="move"  
dnd-dragstart="rearrangeNodeId(currentPrevious, finalPrevious)"  
>
```

And a new rearrange function can be created to handle this shift and update the StoryNode Array.

Making the chapter list draggable implementation

For this, the **dnd package** can be used (we already have that) and for making the chapter list draggable, three things need to be tracked, the chapter/node that is being dragged, its original index (where it's being dragged from) and its new index (where it's being dropped). I implemented this as shown below and also uploaded a screen recording for the same.



```
</div>  
<div class="story-node-editor" ng-if="linearNodesList.length > 0">  
  <div class="story-nodes-container"  
    dnd-list="linearNodesList"  
    dnd-drop="endMoveChapter(index)"  
  >  
    <div ng-repeat="node in linearNodesList"  
      dnd-draggable="node"  
      dnd-effect-allowed="move"  
      dnd-dragstart="startMoveChapter(node)"  
    >  
  </div>  
</div>
```

```

    });
    $scope.startMoveChapter = function(node) {
        $scope.chapterIndexOldPosition = $scope.linearNodesList.findIndex((
            nodes) => {
                return nodes._id === node._id;
            });
    });
    $scope.endMoveChapter = function(newIndex) {
        $scope.chapterIndexNewPosition = newIndex;
        var oldChapter =
            $scope.linearNodesList[$scope.chapterIndexOldPosition];
        $scope.linearNodesList.splice($scope.chapterIndexOldPosition, deleteCount: 1);
        $scope.linearNodesList.splice(
            $scope.chapterIndexNewPosition, deleteCount: 0, oldChapter);
    });

```

I found this to be working nicely, video uploaded [here](#)

These changes needs to be made in:

1. Story-editor.directive.html
2. Story-editor.directive.ts
3. StoryUpdateService

In StoryUpdateService an Rearrange function can be made for the rearranging the destination nodes and the nodeArray

1->2->3->4->5->NULL Suppose 5 needs to be dragged and dropped as a destination node for the first node.

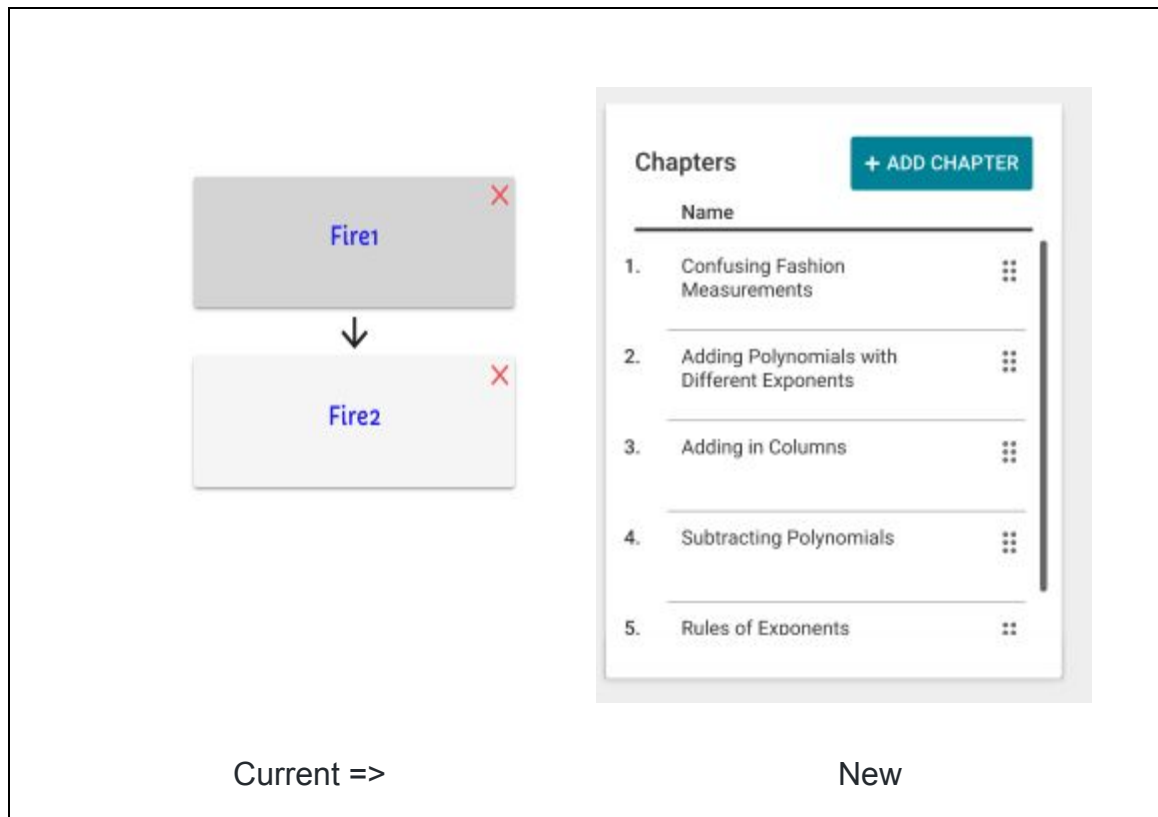
So the new sequence becomes 1->5->2->3->4->NULL

```

rearrangeNodes() {
    StoryUpdateService.removeDestinationNodeIdFromNode(
        $scope.story, nodes[i].getId(), nodeId);

    StoryUpdateService.addDestinationNodeIdToNode(
        $scope.story, $scope.getId(), nodeId);
}

```



2. Shifting the bottom left [Chapter roadmap](#) to the top right with some UI Changes.(as depicted above)

Chapter Editor

Polynomials / Confusing Fashion Measurements


Chapter Card

Title*

This will be shown to the learner. It overrides the title of the linked exploration.



Confusing Fashion Measurements

Exploration ID*

What is an Exploration ID? 

AsDf1234gH

Thumbnail Image*



Choose a file or drag it here.

To-Do List

✓ Complete Chapter Outline

✓ Link Exploration ID

✓ Create a Goal

SIMULATE INCC

3. The Chapter inputs(the whole bottom right part) needs to be made into a separate page(as depicted above).


New Chapter

*required

Title*

This will be shown to the learner. It overrides the title of the linked exploration.

Exploration ID

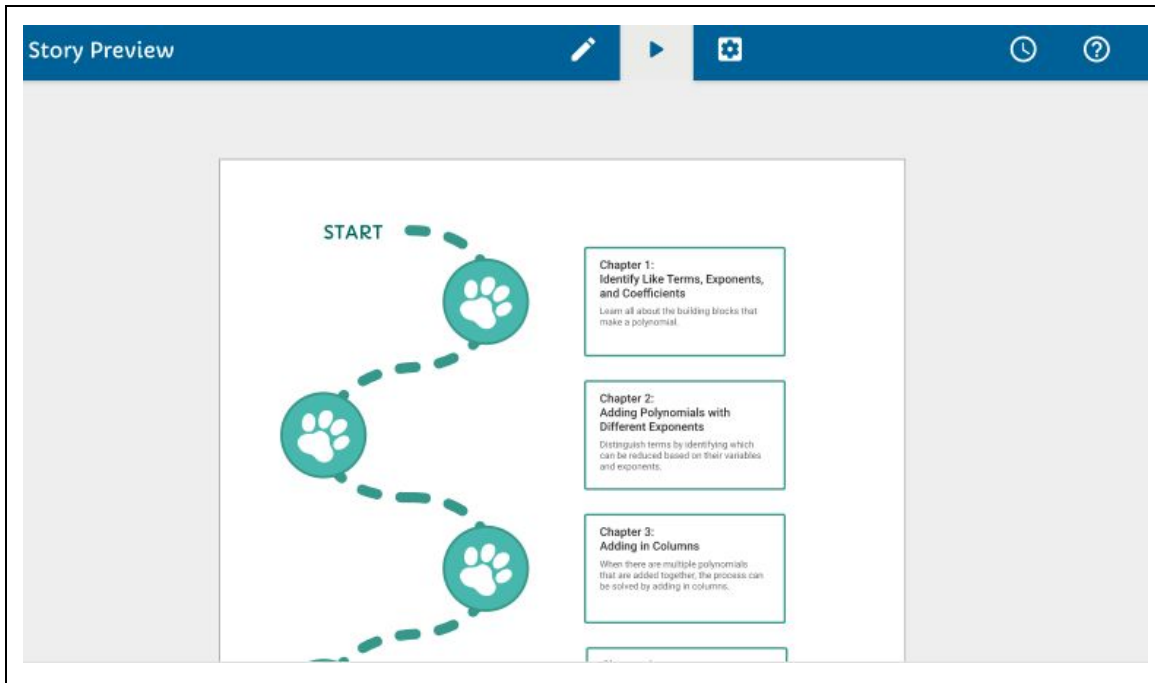
What is an Exploration ID? 

Thumbnail Image*

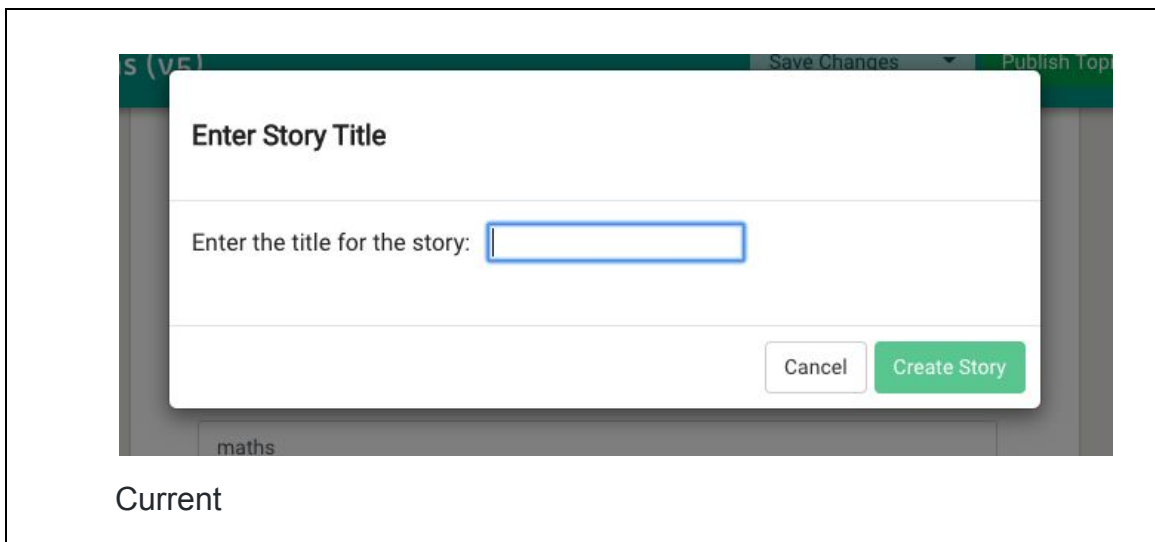
CANCEL

CREATE CHAPTER

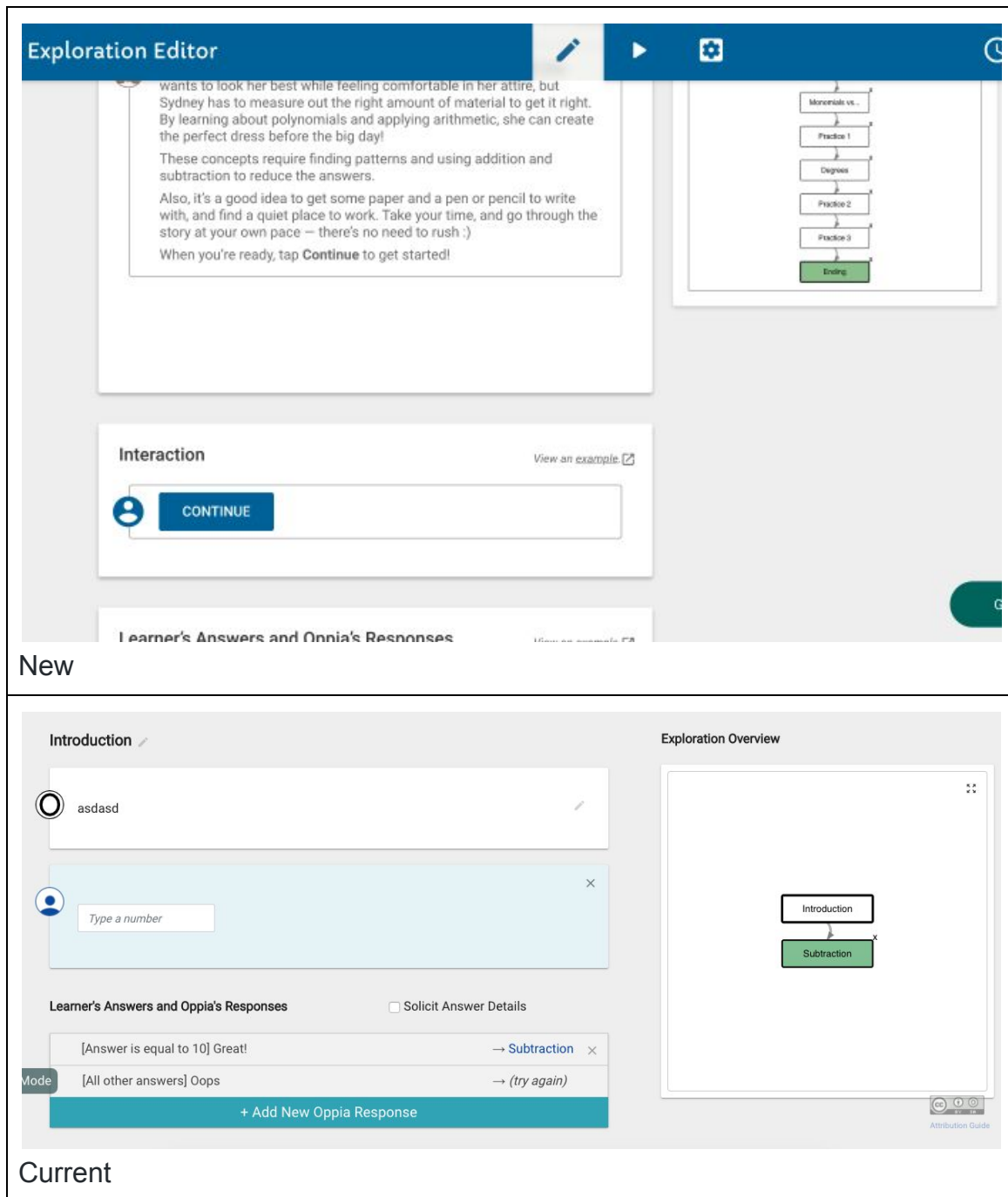
4. A new Create Chapter Modal needs to be made which will link us to the above Chapter Editor Page.



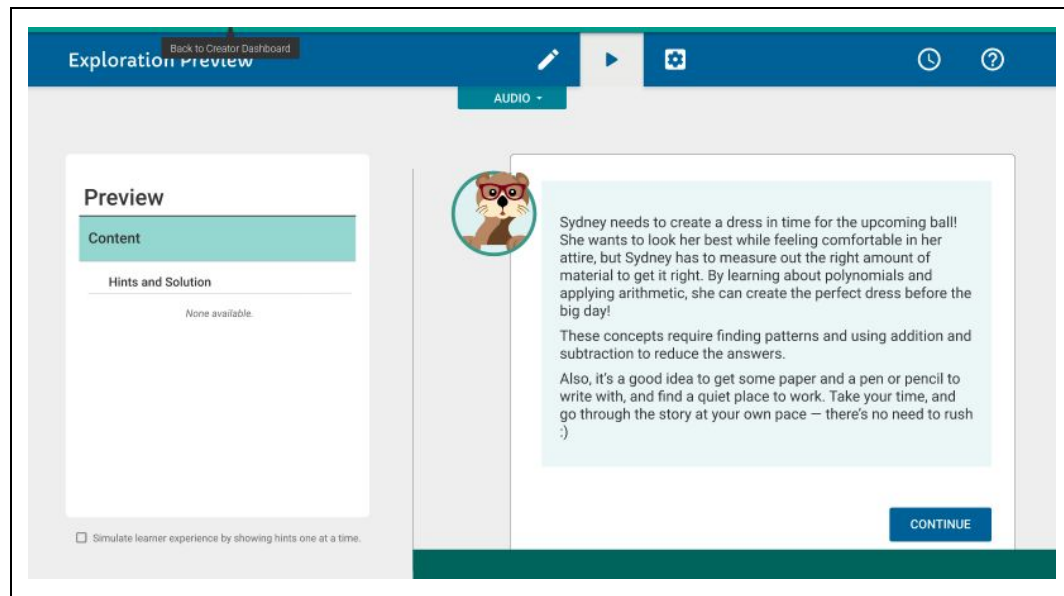
5. A new tab in the newly [Create Story](#) page needs to be added(as depicted above).This might have been challenging but can be implemented easily too since we have a very much identical structure in the [collection preview](#) too.



3. Exploration Editor Page Redesign:



1. This is the easiest among the three, the major changes are some the UI needs to be improved.



2. The Exploration Preview UI needs to be reworked.

Mobile View Implementation(Common to all three):

Well, My general approach will be to Finish the desktop view first like:

Step1: Topic and Skill Dashboard Page Redesign(Desktop)

Step2: Topic and Skill Dashboard Page Redesign(Mobile)

Step3: Topic Editor Page Redesign(Desktop)

Step4: Topic Editor Page Redesign(Mobile)

Step5: Skill Editor Page Redesign(Desktop)

Step6: Skill Editor Page Redesign(Mobile)

Step7: Subtopic Editor Page Redesign(Desktop)

Step8: Subtopic Editor Page Redesign(Mobile)

Step9: Question Editor Redesign(Desktop)

Step10: Question Editor Redesign(Mobile)

Step11: Story Editor Page Redesign(Desktop)

Step12: Story Editor Page Redesign(Mobile)

Step13: Chapter Editor Page Redesign(Desktop)

Step14: Chapter Editor Page Redesign(Desktop)

Step15: Exploration Editor Page Redesign(Desktop)

Step16: Exploration Editor Page Redesign(Mobile)

This is the most optimal and obvious approach since while working on the desktop, all the components, functionality, api changes would have already been made and implemented and the major changes that are required in mobile view are to increase the main content's width and

size, and hide the unnecessary stuff which will be relatively easier and straight forward. The navbar will also be modified in the above manner(desktop and then mobile.)

Testing Approach

Oppia follows three testing procedures basically,

1. Python Backend Code: Since having the backend code 100% covered is a must at any point of time, the backend changes would surely be tested. The backend changes, in general, will be changes in the api's and gae_models and domain files, so for testing them the already present tests need to be modified.
2. E2e Test: Since a lot of modals and pages need to be added, e2e needs to be edited to test various user flows and their proper workings.
3. Karma Tests: The logic/services part needs to be tested by Karma in the frontend.(Like the generic filter service.)
4. The fourth one is the TypeScript Checks, so if in case I need to work with a service that is in Angular, then to test that proper types have been added, TypeScript checks will come into play.

1. E2e tests that are going to be written/modified:

- a. I believe, all the user journeys that are being updated and pages where modals are being created or updated or modified, need to be covered with e2e tests.
- b. Topic Dashboard, skill dashboard, topic creation, skill creation, question creation, chapter creation, exploration editor, topic deletion, assigning skill to a topic, etc will be covered with e2e tests.

c. Topic Dashboard

- i. Login to Oppia, as an admin.
- ii. Click on the profile icon dropdown, check that no Topic and Skill Dashboard link is present in the dropdown.
- iii. Go to the admin page, assign your role as an admin.
- iv. Go back to the splash page, and click on the profile icon dropdown. Now the Topic and Skill Dashboard link should be visible. It should be clickable and should redirect to the topic and skill dashboard which a div that says "Create Topic", "Create Skill"
- v. Create and publish a topic.
- vi. Go back to the topic and skill dashboard.
- vii. In the topics tab, a topic should be listed which was created in the earlier step.

d. Skill Dashboard

- i. Step(1-6) same as above.

- ii. In the skills tab, skill should be listed which was created in the earlier step.

e. Topic Creation

- i. Step(1-4) same as above.
- ii. On the Topic and Skill dashboard, Click on create Topic. A new modal should appear.
- iii. Fill in the topic name, abbreviated name, description, and thumbnail. Upon submitting, the topic should get published and the page should be redirected to the topic editor. The topic editor should have all the information pre-filled.

f. Skill Creation

- i. Step(1-4) same as above.
- ii. On the Topic and Skill dashboard, Click on create Skill. A new modal should appear.
- iii. Fill in the skill name and review material. Upon submitting, the skill should get published and the page should be redirected to the skill editor. The skill editor should have name and review material pre-filled.

g. Topic deletion

- i. Create and publish a topic as described above.
- ii. Go to the Topic dashboard and click on the deletion icon in the topic row.
- iii. Verify that the topic is deleted.

h. Skill Deletion

- i. Create and publish a Skill as described above.
- ii. Go to the skill dashboard and click on the delete icon in the skill row.
- iii. Verify that the skill is deleted.

i. Subtopic Creation

- i. Create a topic and in Topic editor click on subtopic tab.
- ii. Click on save changes and then publish.
- iii. Go to the Topics dashboard and verify the subtopic count to be 1.

j. Assign skill to topic

- i. Create and publish a topic with a subtopic and create and publish a skill.

- ii. Go to skill dashboard and click on assign skill to the topic button. A new modal will appear with the list of topics that were published previously.
- iii. Select topic 1 by clicking on it and then click Save button. This will redirect to the Topic Dashboard.
- iv. Select Topic 1 and click on the Subtopic tab.
- v. Click on Subtopic tab and then drag and drop the skill to the subtopic.
- vi. Click on save changes and then publish.
- vii. Go to the Topics dashboard and verify the subtopic count and skills count to be 1.

k. Misconception Creation

- i. Create a skill then it will be redirected to Skill editor.
- ii. Click on Add Misconception.
- iii. Enter misconception name, Notes for Creators and default feedback and click save.
- iv. Click publish in the top navbar.
- v. Navigate to skill dashboard and verify the misconception count to be 1.

l. Worked Examples

- i. Create a skill then it will be redirected to Skill editor.
- ii. Click on Add Misconception.
- iii. Enter Example question and answer.
- iv. Click publish in the top navbar.
- v. Navigate to skill dashboard and verify the example count to be 1.

m. Story Creation

- i. Create a topic, then in topic editor, click on create story. A modal will appear asking for story name and description. Enter them and click Save. The page will be redirected to story editor.
- ii. Click on create chapters and add explorations to chapters.
- iii. Save and publish, then verify that the story count is 1 in the topic dashboard.

n. Question Editor

- i. Create a skill and then in skill editor, click question tab.
- ii. Click on create question button. Enter the difficulty, question, answer and responses.
- iii. Save and publish the question.

- iv. The question should appear in the question tab of the skill.

Milestones

Milestone 1

Key Objective: Complete the Topic and skill dashboard, topic and skill editor page redesign(both mobile and desktop)(Step1-6).

No.	Description of PR	Prereq PR numbers	Target date for PR submission	Target date for PR to be merged
1.1	The topic and dashboard editor redesign (desktop)		June 6	June 10
1.2	The topic and skill dashboard redesign (mobile)	1.1	June 10	June 15
1.3	Topic Editor Page(Desktop)		June 15	June 20
1.4	Topic Editor Page(Mobile)	1.3	June 20	June 23
1.5	Skill Editor Page(Desktop)		June 17	June 23
1.6	Skill Editor Page(Mobile)	1.5	June 24	June 28

Milestone 2

Key Objective: Complete the Subtopic and Question editor page redesign(both desktop and mobile)(Step 7-10)

No.	Description of PR	Prereq PR numbers	Target date for PR submission	Target date for PR to be merged
2.1	Subtopic Editor Page Redesign(Desktop)		July 10	July 14
2.2	Subtopic Editor Page Redesign(Mobile)	2.1	July 15	July 19
2.3	Question Editor Redesign(Desktop)		July 20	July 24
2.4	Question Editor Redesign(Mobile)	2.4	July 24	July 27

Milestone 3

Key Objective: Complete the Story editor page redesign(both desktop and mobile) and Exploration editor page redesign(both desktop and mobile)(Step 11-16)

No.	Description of PR	Prereq PR numbers	Target date for PR submission	Target date for PR to be merged
3.1	The story editor pages redesign(desktop)		August 4	August 8
3.2	The story editor pages redesign(Mobile)	3.1	August 8	August 10
3.3	Chapter Editor Page Redesign(Desktop)		August 9	August 12
3.4	Chapter Editor Page Redesign(Mobile)	3.3	August 13	August 16
3.5	The exploration editor redesign(Desktop)		August 16	August 20
3.6	The exploration editor redesign(Mobile)	3.5	August 20	August 24

Optional Sections

Future Work

Since Oppia's codebase is very huge, and we have a lot of specific colors defined, and specific fonts, I'd definitely propose that we migrate from css to [scss](#). Since Scss comes with a lot of powers, I think Oppia's frontend codebase will benefit definitely from that.

Additional Project-Specific Considerations

Accessibility (if user-facing)

For this i think a full pass of codebase is required and the html tags should be changed to semantic tags wherever applicable which will help with the screen readers for the users with eyesight issues. The Google Lighthouse tools and other devtools can be used to convert the Oppia webApp to a PWA(Progressive Web App)