Section 2: About Your Project

Project Details

Project title	Improvements for translation reviewer experience
Project size	Large (~350 hours)
Why did you choose this project?	As a current member of the Contributor Dashboard team, I am already quite familiar with the codebase related to this project. By working on this project, I believe I can further develop my skills and gain confidence in contributing to Oppia's mission of providing high-quality education worldwide. This project is particularly exciting to me as it has the potential to significantly improve the productivity of translation reviewers, making it possible to bring Oppia's lessons to even more people in multiple languages quickly. By contributing to this project, I can use my technical skills to make a meaningful impact on Oppia's goals and ultimately help more people access quality education.

Project Timeframe

Note*: Oppia will only be offering a single GSoC coding period timeframe this year, starting on **May 29**. All work for Milestone 1 must be completed and submitted by **July 14**, and all work for Milestone 2 must be completed and submitted by **Sept 15**. We will not be able to extend these deadlines.*

Coding period	● I will adhere to the above deadlines.
Planned time commitment	● May 25 - June 10: Semester exams ● June 11 - July 31: Summer break ● Aug 1 - New semester begins During my exam period, I won't be much active and will dedicate 1-2 hrs per day. During my semester break, I can easily dedicate 6-7+ hours per day. After commencement of my new semester, I can dedicate 3-5 hours per day. I will take Thursdays/Saturdays off to relax myself. *I am flexible with my workload and can adapt to the demands of the project.
What other	None

obligations might you need to work around during the summer?	

Communication Channels

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

I can commit to sending daily updates to my mentor by email, each day I work during the GSoC period.	• Yes
In addition to the above: how often, and through which channel(s), do you plan on communicating with your mentor?	I will be able to meet with my mentors biweekly using video conferencing tools such as Google Meet, Zoom, MS Teams, or Discord. Additionally, I will provide regular progress updates to my mentors via email or chat, at least every other day or as often as needed.

Section 3: Proposal Details

Problem Statement

Link to PRD (or N/A if there isn't one)	PRD: Contributor Dashboard Translation Reviewer Experience
Target Audience	Volunteers who review lesson translations on the contributor dashboard. These volunteers are fluent in both English and their chosen language, and have typically already submitted multiple translations in their chosen language. Since access to review translations is invite-only, these volunteers typically represent some of our most committed ones.
Core User Need	As a translation reviewer, I want to review translations effectively and quickly so that translation contributors get quick feedback and lessons can be quickly translated, which will help students who don't speak English. This is not possible on the current website.

	Background: This issue arose because the contributor dashboard displays translations in random order and only shows a limited number of translations to review. The expectation was that reviewers would work from the top down, with whatever translation submissions are displayed. However, the Language Accessibility team has found that translators prefer to have the context that arises from reviewing translations corresponding to a specific lesson, since it helps them ensure consistency in the language used and provides them with continuity of setting. This difference in fundamental assumptions means that the current review flow on the website is clunky and doesn't satisfy the needs of the workflow adopted by the Language Accessibility team. **Impact**: 20+ reviewers are directly affected by this issue. Without a resolution, submitters do not see their translations submitted, lose motivation, and drop off. Reviewers may also drop off because the workflow is too cumbersome, and they spend more time fighting with the tool than reviewing translations (which is what they originally signed up to do). This isn't just theoretical -- we're currently seeing attrition[1], both from reviewers and submitters, due to the difficulty of reviewing translations. This results in lots of additional effort on the part of translation coordinators to bring in new translators and reviewers, and the lack of continuity of volunteers puts strain on the Language Accessibility team and makes it harder for them to scale operations to multiple languages. This, in turn, slows down the language translation effort, and means that Oppia's lessons will take longer to be available in the languages that users need, thus retarding progress towards our core mission.
What goals do we want the solution to achieve?	● Increased review throughput ● Increased reviewer engagement ● Can efficiently keep lessons at "100% translated" status after edits.

Section 3.1: WHAT

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

1

Key User Stories and Tasks

#	Title	User Story Description (role, goal, motivation) *"As a ..., I need ..., so that ...."*	Priority [2]	List of tasks needed to achieve the goal (this is the "User Journey")	Links to mocks / prototypes, and/or sections spec'cing[3] out the task's user flows and product requirements.[4]
1	Efficient Lesson Status and Navigation for Translation Reviewers	As a translation reviewer, **I need** to see the translation status of the lessons in the topic I'm responsible for, **so that** I can figure out which lesson to review now.	Must have	- Access the "Review Translations" dashboard - User is prompted to pick a topic to view if none has been selected yet after a new login session. - Users can, subsequently, select a different topic (if they wish). - Default will show all topics and lessons without filtering, following the same order as the learner's view.	Mock
		As a translation reviewer, **I need** to review a lesson's translation cards in the order according to the content flow, **so that** I will know the context for each of the translations.	Must have	On the "Review Translations" dashboard (possibly after filtering by topic), click on a specific lesson index card. The translation cards can be seen in order according to the content flow.	
				- Click on a specific card in the list to review it. - Click on a link that will lead to the Exploration Editor in a new tab, showing the original lesson card flows.	Mock Exploration Editor does not allow edit access to the Lesson,
				Click "Accept" or "Reject".	

[2] Use the **M**o**SC**ow system ("Must have", "Should have", "Could have"). You can read more here.
[3] Specs are important if the user journey has multiple cases (e.g. error types for wrong answers; different choices a user can make; etc.) and these need detailed enumeration.
[4]

		As a translation reviewer, I **need to** identify short translation cards **so that**, if I'm on the move, I can prioritize reviewing them as "quick wins".	Could have	After clicking on a lesson to review, read the list of translation cards.	Each translation card displays "short" if the word count < 20.
		As a translation reviewer, I **need to** see the image alt text translation below the image by default **so that I** can review the image alt text translation without additional clicks.	Should have	- Click on the translation card with the image to review.	Mock
2	Continuity	As a reviewer, **I need** to be able to pin a specific lesson **so that** I can continue reviewing it later.	Must have	Access "Review Translations" dashboard.	
				- While at the lesson index dashboard, click to pin a lesson. - Click another lesson from the list - Click to pin this lesson as well	Prototype
		As a reviewer, I **need to** be able to unpin a lesson **so that** I can remove it from the default view in the dashboard.	Must have	- Access "Review Translations" dashboard - Toggle the pin to unpin the lesson in the dashboard or within the lesson view	Prototype
		As a reviewer, I **need to** be able to revert the acceptance/rejection of the translation **so that** I can redo the review of the translation.	Could Have	- Take action (accept/reject) a translation suggestion - Click on "Undo" button of the same tile within 30 seconds for reverting the status back to unreviewed	
		As a reviewer, I need to be able to navigate through any page I want using	Must Have	- Access "Review Translations" dashboard. - Navigate through pages	

		just one click.		using goto dropdown.	
3	Notification	As a reviewer, I **need** to know when new submissions are ready for me to look at or if there are any outstanding reviews **so that** I can act on them.	Should have	- Receive an email on lessons that still need to be reviewed - Reviewer has option to click on a link to go directly to the lesson view dashboard	Notification - E... Email Mock

Technical Requirements

Additions/Changes to Web Server Endpoint Contracts

#	Endpoint URL	Request type (GET, POST, etc.)	New / Existing	Description of the request/response contract (and, if applicable, how it's different from the previous one)
1.	/getreviewableopportunitieshandler	GET	Existing	Request currently returns the opportunities list of lessons which has to be reviewed. It will additionally fetch pinned opportunities and to let them get displayed at top.
2.	/manage_pinned_opportunities/{exploration_id}/{language_code}	POST	New	This request will be responsible for adding/removing pinned lessons in new storage models (say **PinnedOpportunitiesModel**) which will keep track of pinned lessons.
3.	/cron/mail//reviewers/contributor_dashboard_new_suggestions	GET	New	This request is used to trigger a cron job that will send email notifications to translation reviewers about new suggestions that are waiting for review on the contributor dashboard.

Calls to Web Server Endpoints

#	Endpoint URL	Request type (GET, POST, etc.)	Description of why the new call is needed, or why the changes to an existing call is needed
1.	/reviewable_suggestion_pinned_task/{exploration_id}/{language_code}	POST	We will introduce a new storage model **`PinnedOpportunitiesModel`** to keep track of pinned lessons by lesson and user. PinnedOpportunitiesModel will have • User_id (string): The ID of the user who pinned the opportunity. • Lang_code (string): The language code of the pinned opportunity. • opp_id (string): The ID of the pinned opportunity. • pinned_on (datetime): The timestamp when the opportunity was pinned. This new request will be responsible for pinning/unpinning an opportunity
2.	/getreviewableopportunitieshandler Frontend Backend	GET	Currently when the user navigates to the translation review dashboard, the request returns a list of translation opportunities that are to be listed in the translation review dashboard. We will have to modify the fetch query in controller returning logic so that pinned lessons are always at top.
3.	/cron/mail/reviewers/contributor_dashboard_new_suggestions	GET	It is used to trigger a cron job that will send email notifications to translation reviewers about new suggestions that are waiting for review on the contributor dashboard. The cron job will run every 24 hours and send emails to reviewers based on their reviewing permissions. The email will contain a list of new suggestions that need to be reviewed. This endpoint URL is tied to the CronMailReviewersNewSuggestionsHandler handler function in the cron.py file and is defined in the main.py file.
4.	create/<exploration_id>/translations	GET	This endpoint fetches data for the exploration page for the given exploration id. (Handler)

			The exploration id can be obtained from the target_id of the suggestion. To land on the correct card in the translation tab, we need to pass state_name and content_id as hash values.

UI Screens/Components

#	ID	Description of new UI component	i18n required?	Mock /spec links	A11y requirements
1.	Pin / Unpin icon	Pin and unpin icon shall be added accordingly in the opportunity-list items Pin icon will be on unpinned lesson and vice versa	No	Mock	Ensure that the pin and unpin icons have appropriate alt text that describes their functionality when using screen readers. Use visually distinguishable colors and contrast ratios to ensure the icons are perceivable for users with visual impairments. Implement proper keyboard navigation and focus management so that users can interact with the icons using keyboard-only inputs.
2.	Tooltip for Pin icon	Upon hovering on pin icon we shall show a tooltip `Pin the lesson`	No	Mock	Ensure that the tooltip is accessible to screen readers by providing an accessible name

					and description for the tooltip. Make sure the tooltip appears and disappears correctly when users interact with the pin icon using keyboard or mouse inputs. Provide a mechanism for users to dismiss or close the tooltip, especially for users who rely on keyboard navigation.
3..	Image alt text translation	Image alt text will be shown below the image for reviewing in Translation review modal.	No	Mock	Make sure the modal is accessible with proper focus management and keyboard interactions.
4.	Word count for translation length	If the word count of translation is less than 20 words then it will be short otherwise it will be considered as long.	No	Mock	Ensure that the word count information is perceivable and understandable by all users, including those using assistive technologies. Use appropriate labels or text descriptions to indicate whether the translation is short or long, making it clear and meaningful . Ensure that the word count information is available to screen

					readers and other assistive technologies.

Data Handling and Privacy

#	Type of data	Description	Why do we need to store this data?	Anonymized?	Can the user opt out?	Wipeout policy	Takeout policy
1.	Pinned Opportunities Modal user_id: string Exp_id: string language_code:string	A new storage modal to track pinned lessons by user and language.	We need to store a list of pinned lessons which will be displayed at the top of the review dashboard.	No, data will be explicitly tied to a user.	Yes, the user has the freedom to pin/unpin the lessons.	If a user requests account deletion, all pinned lessons associated with their user ID will be deleted	Users can export their pinned lessons via the export data feature

Other Requirements

N/A

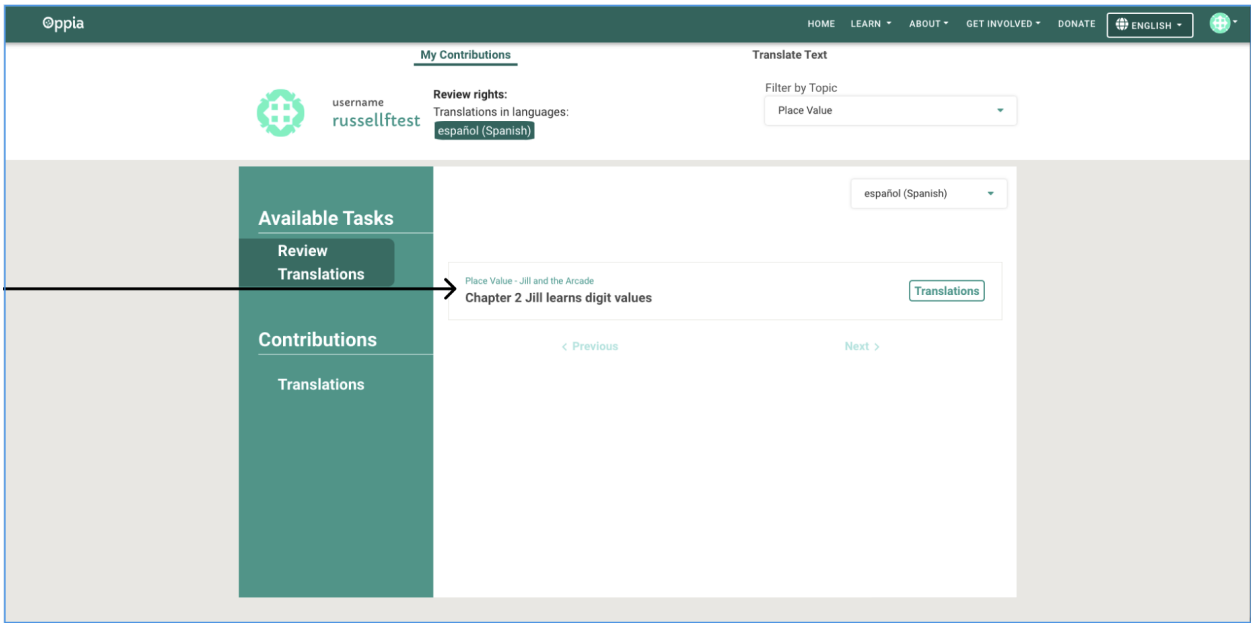
Section 3.2: HOW

Existing Status Quo

Translation reviewers currently page through the contributor dashboard to find items to review. However, there is no way to bookmark or pin the lesson they were reviewing. As a result, they have to search for the lesson again to continue their work. This leads to frustration and they complain to the Language Accessibility leads, who raise concerns to the technical team. Additionally, the page takes more than 10 seconds to load, furthering the frustration.

There is currently no process for prioritizing the translation cards that reviewers want to work on. Additionally, once they have found a lesson for review, there is no option to save it for later, requiring them to search for it again. Furthermore, if a reviewer takes an action on a suggestion and later notices minor mistakes, they are unable to undo it themselves.

Translation reviewers do not currently have a straightforward way to get out of this loop, because the tool does not allow it. In the meantime, translation submissions do not get reviewed.



Picture of current Translation review dashboard

*** Here is a video of the current user experience.**

Barriers/Dependencies: There are no barriers/dependencies to changing the status quo.

Note: This is our first time tackling this specific problem. We became aware of the mismatch in expectations in late 2021, as the Language Accessibility team started standardizing review processes across the translation teams.

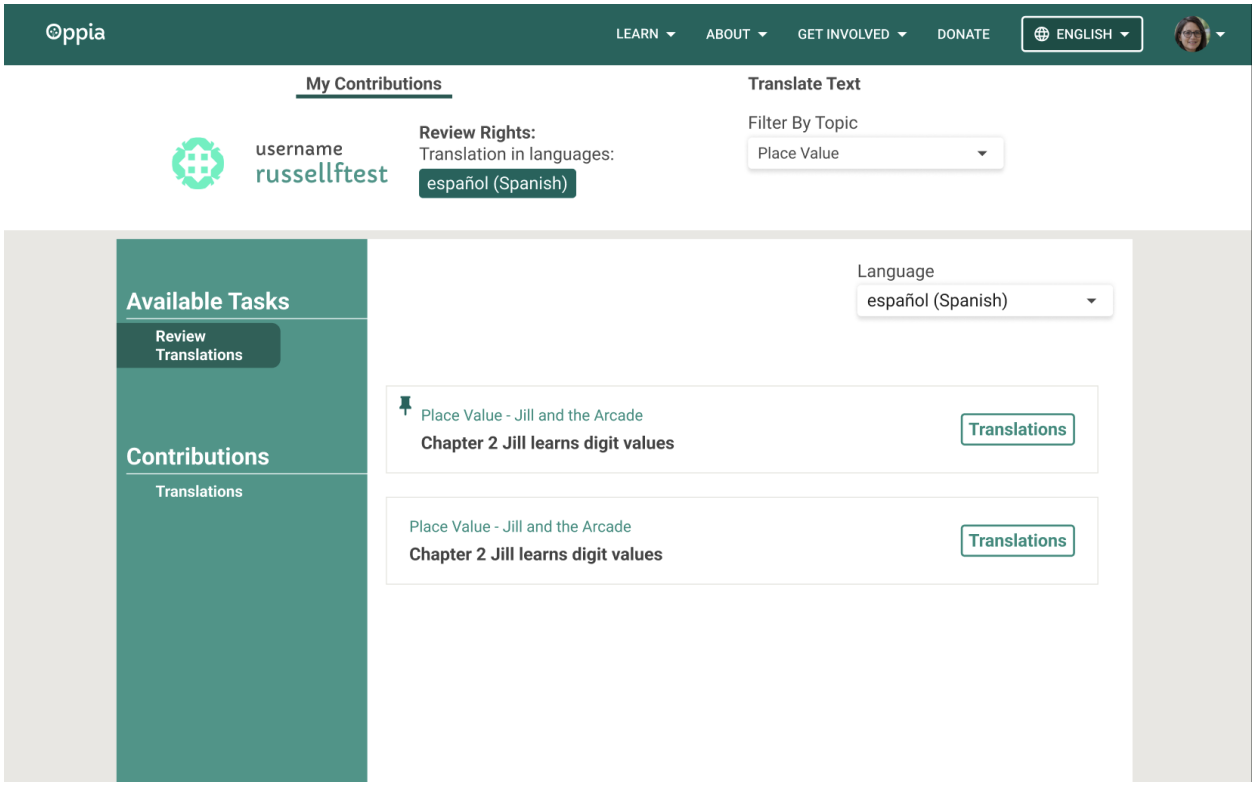
Solution Overview

The current workflow of the translation review dashboard was designed with the assumption that reviewers would review translation cards from top to bottom. However, as discussed in the "What" section and the "Existing Status" section, this is not always the case.

To address the current issues faced by translation reviewers, we propose modifying the current workflow and implementing new features that will ultimately increase their effectiveness and productivity. This will enable quick translations of lessons in multiple languages, which helps support Oppia's core aim of bringing lessons to learners around the world.

In this section, we provide an overview of what the changes might look like and what impact they could potentially have. Their implementation will be discussed in the "Implementation" section.

- Currently, translation reviewers are unable to pin important lessons or those they may want to revisit in the near future. Therefore, we propose the introduction of a **pinning/unpinning** feature for lessons in the reviewer's dashboard. Please refer to the attached image for an example of how this feature might look.



User will be able to click on the pin icon to pin/unpin according to his will. We will also keep in mind the keyboard users and let them access it using the keyboard. If a user has manually pinned a lesson before then that lesson will appear at the top.

We will allow pinning of one lesson per language and topic per user . (Implementation Plan)

Rationale:

This decision made as per the pre-user survey, following users were involved in our Survey : **Anshuman, Yiga, Bhaskar**

Initial Plan No Restriction: Initial Plan was to allow any number of pinned lessons, however this led to a thought would this be able to fulfill the requirement to allow users to prioritize the lessons to review, since they can just pin any number of lessons in this case.

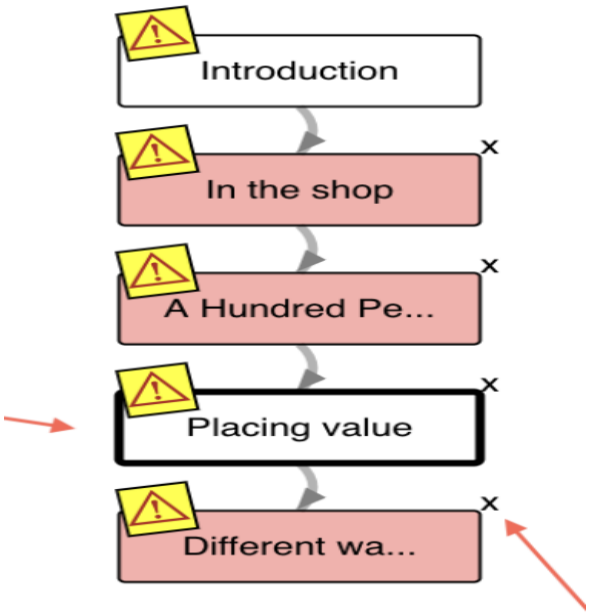
Allow only Pinned Lesson: Upon discussing with Anshuman, we came to know about the translation reviewer's workflow and he suggested to allow only one pinned lesson throughout contributor dashboard, here is his statement,

In my opinion, pinning just one lesson is plenty because we review in a regular sequence in the team so that we don't get reviews half-done or miss some.
For your convenience, an example of a sequence is provided.
We simply finish one lesson before going on to the next, which gives the impression that "yay! I finished one lesson and may now move on to the next".

One Pinned Lesson per language and topic_id: When we were about to finalize the structure with the above approach, an interesting discussion came up "Is just One Pinned suggestion throughout the CD is enough" or should we consider "One Pinned Opportunity per topic and language". We once again reached out to our users, this time several of them and gathered their thoughts, upon further discussion with tech team, we came to a mutual decision of "Allowing **One Pinned Lesson per language and topic_id**"

- Currently the translation cards in lessons are shown in random order. The aim is to display them in a list according to the content flow of the lesson to which they correspond. By showing the translation cards according to the content flow, it will be easier for the reviewer to follow along and understand the material.

  Below we have added an example to clear what we mean by `**content flow**`.



Source - google

Let's say we have a lesson named `Place Values` which comes with 5 cards with it as shown in the above image.

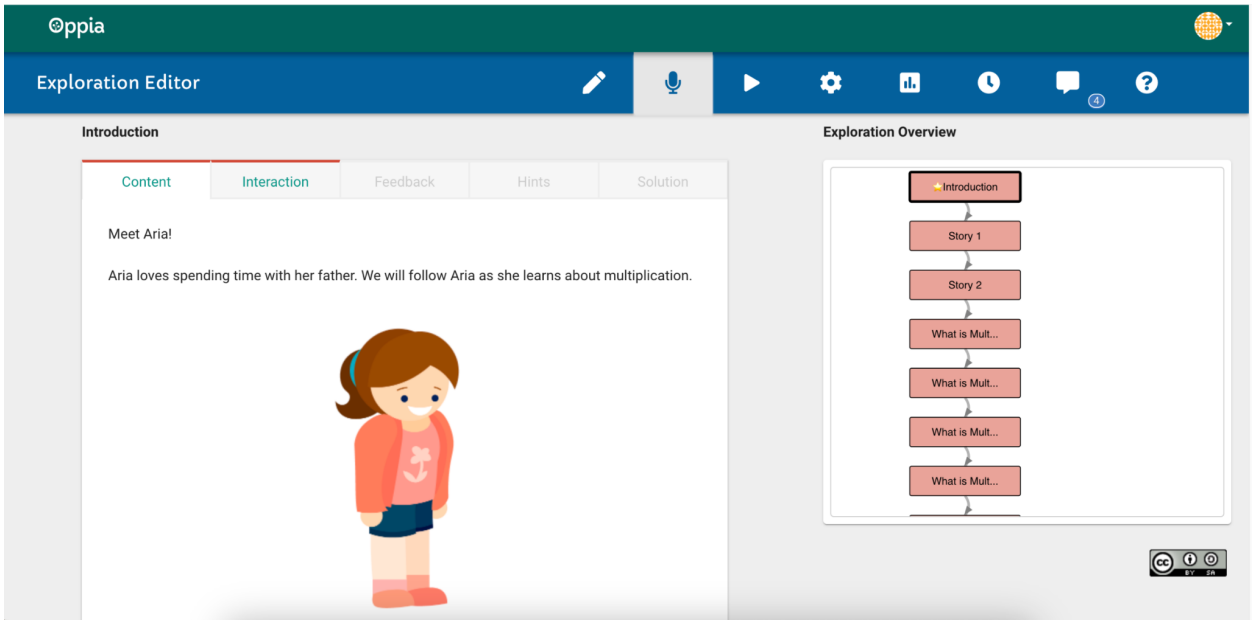
The content flow for the learner here would be

- Introduction → In the shop → A hundred people → Place value → Different waivers.

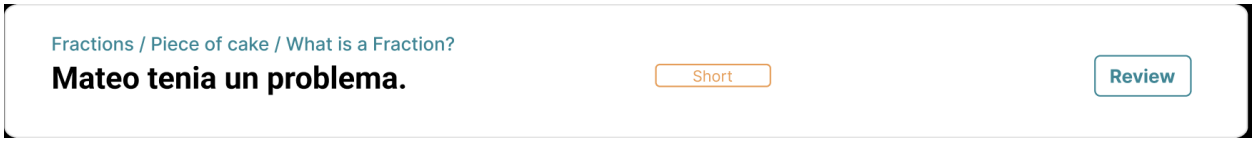
Our goal is to achieve the same context flow for the translation reviewers to make it easy for them to follow along. (Implementation Plan)

- We will add a link in the translation-review modal as shown in the mocks which will navigate the user to the "**Exploration-Editor**" page in a new tab. (Implementation Plan)

Below is an image for reference.



- To let the translation reviewers pick up **quick wins** we will add a label which will show whether the content to be translated is short or long. As per PRD we will consider it short if the word count is less than 20 and long for else.



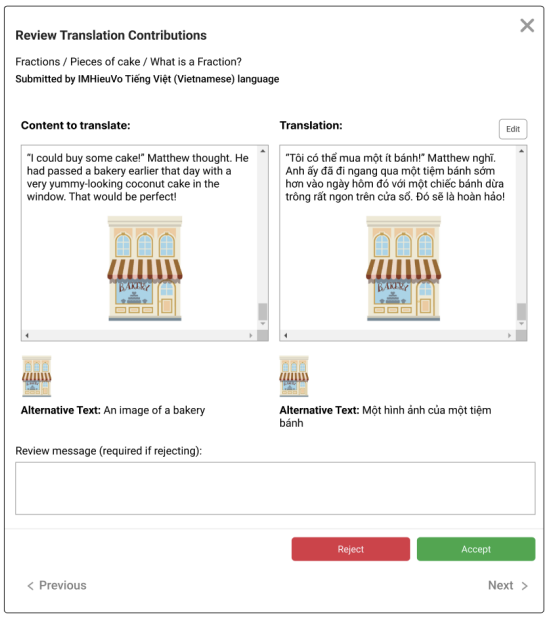
This label will be displayed in Translation cards so that users can know the complexity without opening the modal as well.(Implementation Plan)

- Currently, reviewers have to perform some additional clicks to review the image alt text, if there are any images in the content to be reviewed.
To smoothen the workflow, we will show the **image alt text** right below the image as shown in images. (Implementation Plan)

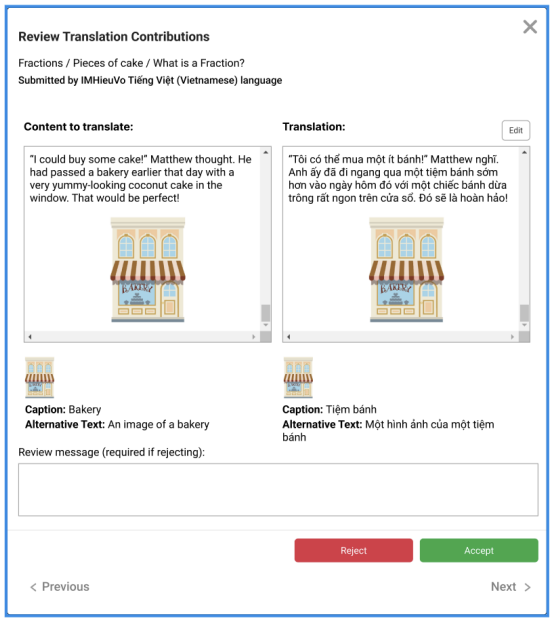
Oppia

Proposed User Flow (Translation Review Screens)

1. With No Caption



2. With Caption

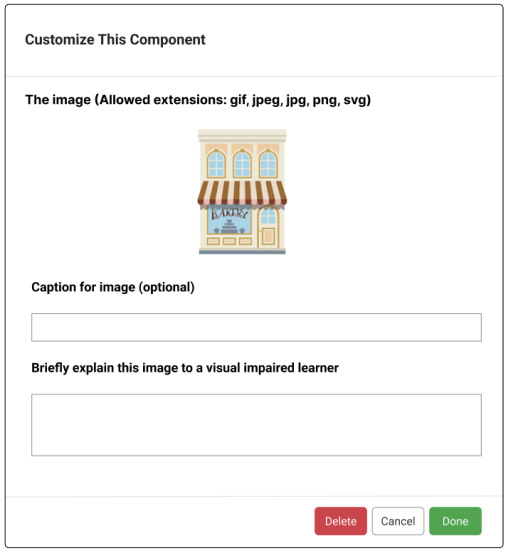


- **Reviewer wants a quick way to view the translation of the alternative text.**
- **Proposed: modal shows the alternative text with image thumbnail under content window**

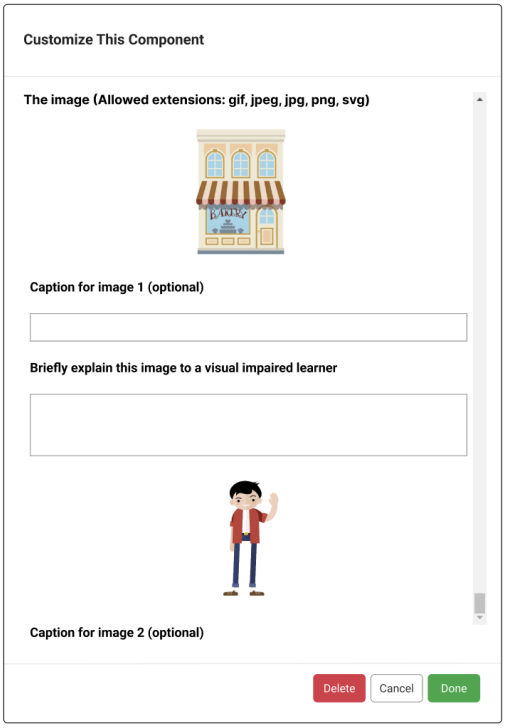
Oppia

Proposed User Flow (Edit Screen Mockups)

1. With one image (unchanged)



1. With 2+ images (Add scroll)



- The proposal suggests implementing a notification system that will notify translation reviewers when there are new suggestions to review. The notification system will be designed to work via email. Currently the notification system is implemented to send emails to reviewers for the suggestions that have been waiting for the longest. (Implementation Plan)

- We would like to add a feature that allows translation reviewers to undo the acceptance or rejection of a recently reviewed translation, so that they can redo the review if necessary.
  I am planning to enqueue the suggestion acceptance/rejection in the front end itself. Below is a mock workflow for it.
  Status property of suggestion object keeps track of a suggestion's status (either "accepted", "rejected" or "pending")

1. When the user selects a translation card to review, the translation suggestion modal appears. The modal contains two buttons: "Accept" and "Reject".
2. When the user clicks on either button, the suggestion is marked as "accepted" or "rejected" respectively, the status property is updated accordingly and a timer is started.
3. A pop up bar appears with two action buttons “Undo” and “Dismiss”.
4. If the user doesn't click the "Undo" button within 30 seconds, the status of the suggestion is committed to the backend, and the translation card tile vanishes from the list of to-be-reviewed cards in the dashboard.
5. If the user clicks on undo, the pop-up bar’s message changes to “Action Undone”.

Exception: If a user tries to close the browser tab while there are still suggestions in the queue, a warning message will appear in a modal with text “You have some pending review approvals/rejections that will be lost if you exit this page. To fully apply them, please close the translation review modal or dismiss the pop-up bar.” along with two action buttons “close anyway” and “stay on page”.

This feature has the potential to improve the efficiency of the translated material, as discussed in the PRD, and the undo action can only be done within a time frame of 30 seconds. (Implementation Plan)

- We would also like to change the pagination method of the contributor dashboard and would like to have a goto dropdown menu in which the user can select any page number he wants to navigate to along with pre-existing next and previous buttons. (Implementation Plan)

Third-Party Libraries

N/A

“Service” Dependencies

N/A

Impact on Other Oppia Teams

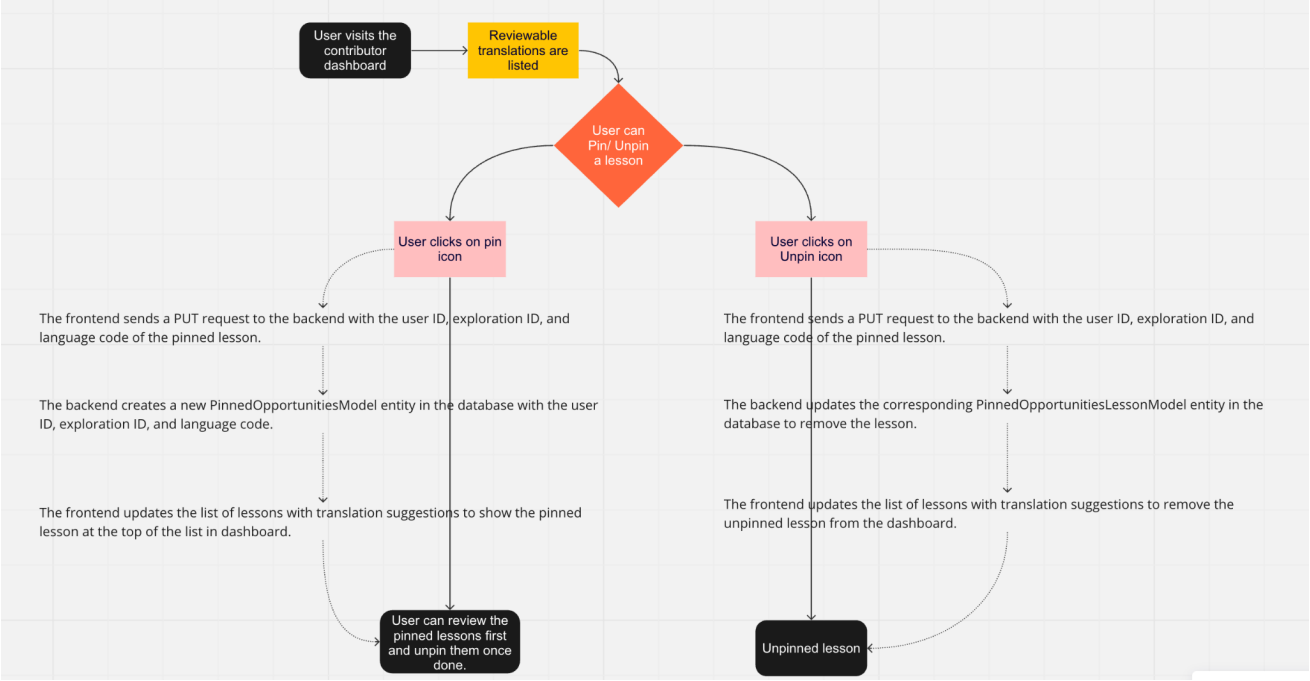
Translation Review Team → Upon release of this feature reviewers can prioritize their work and also with a better workflow, it will be a smoother process than now it currently is.

Release Testing Team → With new features, comes new bugs too. Thus the testing team should be thoroughly testing this feature before launching this for the users.

CD Team → The Contributor Dashboard Team will make efforts by sending emails to users to popularize the feature.(Mock Template)

Key High-Level and Architectural Decisions

Proposed Workflow for pinning of lessons:



For pinning of lessons we need to create a new model **(PinnedOpportunityModel)** to track the pinned lessons.

I have considered the following alternatives for it:

1. Option 1: Model keyed by language_code/user_id

   This option uses a model structure where the key is a combination of language_code and user_id. Each key points to a dictionary that stores the pinned lessons for that user and language. Here's an example of how the data structure might look:

```
{
   (user_id_1, language_code_1): {
      topic_id_1: lesson_id_1,
      topic_id_2: lesson_id_2,
      ...
   },
   (user_id_1, language_code_2): {
      topic_id_1: lesson_id_3,
      topic_id_2: lesson_id_4,
      ...
   },
   ...
}
```

In this structure, each user and language combination has a separate dictionary containing the pinned lessons for each topic. The lesson_id represents the ID of the pinned lesson.

2. Option 2: Model keyed by language_code/user_id/topic_id

   This option uses a model structure where the key is a combination of language_code, user_id, and topic_id. Each key directly points to the lesson_id of the pinned lesson. Here's an example of how the data structure might look:

```
{
   (language_code_1, user_id_1, topic_id_1): lesson_id_1,
   (language_code_1, user_id_1, topic_id_2): lesson_id_2,
   (language_code_1, user_id_2, topic_id_1): lesson_id_3,
   (language_code_1, user_id_2, topic_id_2): lesson_id_4,
   ...
}
```

   In this structure, each combination of language, user, and topic has a separate key-value pair, where the key represents the language, user, and topic combination, and the value is the lesson_id of the pinned lesson.

Factor	Option 1	Option 2

Model's Structure	Keyed by (language_code, user_id) mapping to dictionary of (topic_id, lesson_id)	Keyed by (language_code, user_id, topic_id) mapping to lesson_id
Individual entry size	Larger, as it includes all pinned lessons for a user	Smaller, as it only includes one pinned lesson per entry
Query complexity for 1 Topic	One GET call to fetch the dictionary for a given user id and lang code, and then using topic_id as key fetch pinned lesson.	One GET as the topic_id is a part of the key, so we don't need any further filtering
Query complexity for All Topic	One GET to fetch the dictionary for a given key (user id and lang code).	One GET_MULTI to fetch all the pinned lessons for given (user_id and lang_code)
Code and system maintainability and readability	Slightly more complex, requires additional filtering logic	Simpler and more straightforward

Option 1 (Model keyed by language_code/user_id):

- Advantages:
  - Smaller individual entry size, as it only includes one pinned lesson per entry.
  - **Efficient GET calls for fetching data for a specific topic and fetching data for all topics.**
- Disadvantages:
  - Limited ability to filter/index by topic_id when fetching data for a specific topic.

Option 2 (Model keyed by language_code/user_id/topic_id):

- Advantages:
  - **Ability to filter/index by topic_id when fetching data for a specific topic.**
- Disadvantages:
  - Larger individual entry size, as it includes all pinned lessons for a user.
  - Fetching data for all topics requires a GET_MULTI call, which may have a slight performance impact compared to Option 1.

Option 1 allows us to fetch all the pinned translations for a given user + language in one GET and for topic_filtering we can fetch the dictionary in one GET and then do a key-value lookup for given topic id but it will be tricky to delete all the completed lesson's model as we are storing the topic_id_to_opportunity_id as a JSON property. Thus we are going with Option 2, the only drawback in this option is that we will have to make a **GET_MULTI** query to fetch lessons of all topic, but considering overall requirements Option 2 is the optimal approach.

Risks and mitigations

Potential Risk	Mitigation
Difficulty in implementing the pinning feature for lessons and translation cards.	Conduct a thorough analysis of the existing codebase to identify areas where the new feature can be added without causing conflicts. Implement the feature in small, incremental steps and test each step thoroughly before moving on to the next.
Lack of user engagement with the pinning feature.	Notify users of the new feature through emails and in-app notifications. Provide clear instructions on how to use the feature and the benefits it offers. Monitor user engagement with the feature and make improvements based on feedback. Release team can send an email to all the users who are translation reviewers regarding the new features for them. Mock Template

Implementation Approach

In this section, I have mentioned an overview of the changes that I might make in each of the 5 aspects, namely Storage Models, User flows(Controllers and Services), Web frontend changes(if any), for each feature of this project.

Pinning/Unpinning lessons

Storage Model Layer Changes

In gae_models.py

- Create a new Datastore model to represent a pinned opportunity. The model will be named as PinnedOpportunityModel.
- user_id (StringProperty): Represents the user ID.
- language_code (StringProperty): Represents the language code.
- topic_id (StringProperty): Represents the topic_id.
- opportunity_id (StringProperty): Represents the opportunity_id of the pinned lesson

  Key will be **user_id.lang_code.topic_id**

This model class will have two methods which are mentioned below:

- get_model(): It will take three parameters: user_id, language_code, topic_id and call a helper method **_generate_id** which will take these parameters to form a key and then fetch the model using a **get_by_id** call.
- create(): It will take four parameters, user_id, language_code, topic_id and opportunity_id and will create a new model for pinned opportunities of a user if it doesn't exist.
  If the model already exists, we can update the previous opportunity_id or None to the new opportunity id.

User Flows (Controllers and Services)

We will need to modify the existing controller function that handles fetching reviewable translation opportunities to include pinned lessons in the reviewer dashboard. Specifically, we will modify the **_get_reviewable_exploration_opportunity_summaries** to include the pinned lesson at top.

For pinning/unpinning lessons we will be introducing a new **endpoint URL** for which we need to have a new handler function. These handler functions will call the corresponding service functions to pin and unpin lessons.
We will club (requests for pinning, unpinning and fetching) into a single URL by using HTTP methods (PUT & GET).

A new handler class will be created (PinLessonsHandler) which will be exposed to URL `/pinned-lessons`

Updating (Pinning/Unpinning) a Lesson:

- URL: /pinned-lessons
- HTTP Method: PUT
- Request Data:
  - For Pinning:
    - user_id=123, language_code=en, topic_id=topic123, opportunity_id=lesson456
    - This action will pin the opportunity with ID lesson456 for the user with ID 123, language code en, and topic ID topic123.
  - For Unpinning:
    - user_id=123, language_code=en, topic_id=topic123, opportunity_id=None
    - This will unpin any currently pinned lesson for the user with ID 123, language code en, and topic ID topic123.

Fetching Pinned Lessons:

- URL: /pinned-lessons
- HTTP Method: GET
- Request Data: user_id=123, language_code=en, topic_id = topic123
- This will fetch the pinned lesson for the user with ID 123 and language code en and topic id as topic 123

In suggestion_services.py we will add new methods for pinning, unpinning and fetching all pinned lessons.

1. Create a service function to update the pinned opportunity model : Pinning and Unpinning
   Input: user_id, language_code, topic_id, and lesson_id. Note: for unpinning, lesson_id will be None.
   a. Fetch the PinnedOpportunityModel using the combination of user_id, language_code, and topic_id.
   b. If the model exists:
      i. Set the model's opportunity_id to the given lesson_id.
      ii. Note that this includes both unpinning (setting to None) and pinning a new one(replacing previous lesson_id with new one).
   c. If no model exists and lesson_id is provided: Create a new model with the provided parameters.
   d. Save the model to the datastore.

2. Create a service function for fetching the pinned lesson (if any) for a user with given language and topic:
   a. Model Exists with an Opportunity ID: If the model exists and has a valid opportunity_id, then it indicates that the user has pinned a specific lesson. In this case, the service function will return the opportunity_id.
   b. Model Exists with a Value of None: If the model exists but the opportunity_id is None, it denotes that the user had previously pinned a lesson but has since unpinned it. Even though the record exists, the absence of an active pinned lesson is reflected through the None value. Here, the service function will return None.
   c. Model Doesn't Exist: If there's no model corresponding to the provided parameters (i.e., user, topic_id, and language_code), it suggests that the user hasn't interacted with the pinning functionality for the given topic and language. In this scenario, the absence of the model is treated similarly to an unpinned state. The service function will also return None.
3. Create a service function to delete the completed pinned lesson for all the users.

a. Input: topic_id, language_code and opportunity_id.
b. Fetch all the PinnedOpportunityModels for given parameters, and delete it for all the users, using a delete_multi.

Web frontend changes

As for front end changes we will add icons in lesson tiles based on if they are pinned or not.

In opportunity-list-item.component.ts we will add a new property **Pinned?: boolean** property to the `ExplorationOpportunity` interface. We will be limiting this variable to be not null for opportunities whose type are translation_review.

```
export interface ExplorationOpportunity {
  id: string;
  labelText: string;
  labelColor: string;
  progressPercentage: number;
  subheading?: string;
  inReviewCount: number;
  totalCount: number;
  translationsCount: number;
  heading?: string;
  actionButtonTitle?: string;
}
```

When the user clicks on the pin/unpin icon, we will add action methods for pinning and unpinning which will be responsible for making requests to the server and calling respective functions which are discussed in above sections.

Exceptions/Corner Cases:

- When a User tries to pin another lesson of the same language and topic id while one is already pinned.
    - In the pinning service function, before updating the PinnedOpportunityModel, perform a query to check if there is already a pinned lesson for the given user, language and topic combination.
    - If a pinned lesson exists, throw an exception or return an error indicating that a lesson is already pinned and the user needs to complete it or unpin it before pinning another lesson.
    - This check can be implemented using a datastore query to retrieve the PinnedOpportunityModel based on the user ID, language code and topic id.
- When the lesson gets completely reviewed, we can unpin the lesson for reviewers.
    - Identify Completed Lesson:
        - Once a suggestion has been accepted/rejected we will check if the lesson has been completed in the backend. The logic to check if a lesson has been completed is in opportunity_services.py. This method gets invoked whenever a new suggestion is accepted.
    - Initiate the clean up process
        - In the above step, once the completed lesson has been identified we can use its exp_id, topic_id and language_code to fetch all the matching modals and then delete them using **DELETE_MULTI.**

Link to the Exploration Editor Page

Exp_id is stored in the target_id property of Active suggestion Dict. Thus we will need to pass this target id as a parameter for our endpoint to navigate to the exploration editor page.

Format of URL : /create/{exploration_id}#/translation/{state_name}/{content_id}

To handle navigation to the translation editor and specific cards within it, we need to extract state_name and content_id from the path in router.services.ts. State name will allow us to navigate to a particular state and content id to navigate to the correct section for that particular state.
We will extend the _doNavigationWithState() method for the translation tab as well. We will also need to update the navigateToTranslationTab() method to call this **_doNavigationWithState()** method when **state_name and content_id** is provided.
We want to ensure that the existing routing in the exploration editor, specifically switching among tabs, is not broken. To achieve this, we will pass null values to both the query parameters in the URL. This will ensure that the handling of the URL remains the same as it currently is.

If the state name or content ID provided in the URL is invalid, the router/navigation service should handle these scenarios to ensure smooth navigation and provide a fallback mechanism. Here's how it will be handled:

(a) Invalid State Name:
If the state name specified in the URL is invalid or does not exist in the exploration, the router/navigation service will default to a sensible fallback state, as follows:

- The service can have a predefined fallback state that is used when an invalid state name is provided.
- It can be the initial/first state of the exploration.
- The router/navigation service should update the URL to reflect the fallback state, allowing users to share or bookmark the corrected URL.

(b) Invalid Content ID:
If the content ID specified in the URL is invalid or does not correspond to any valid content within a state, the router/navigation service will default to the first available content, as follows:

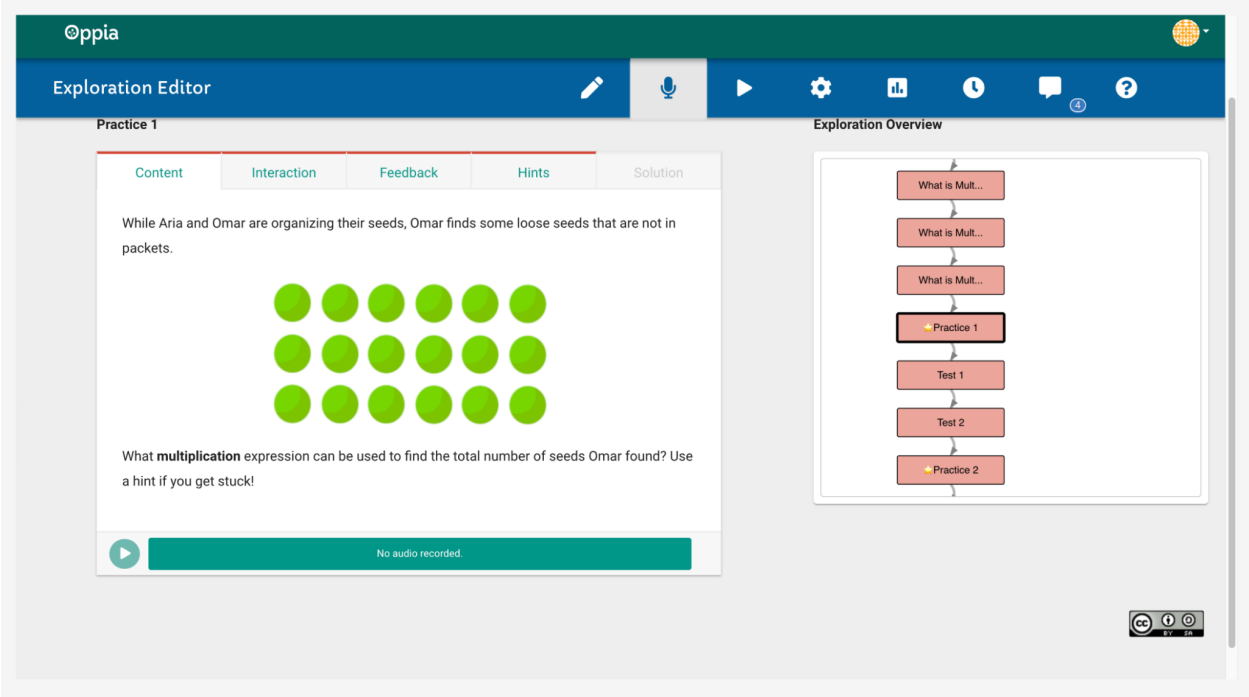
- If the content ID is invalid, the service can default to the first available content in the state.
- This ensures that there is always valid content displayed to the user.
- The router/navigation service should update the URL to reflect the corrected content ID, allowing users to share or bookmark the updated URL.

We will prevent the default behavior of links opening in the same tab using target =’_blank’ for link tag.

Frontend Changes

There won’t be much front end changes as our plan is to navigate the user to the exploration editor page in a new tab.

In the translation review modal template we will add a new link which will open the Exploration Editor in a new tab. When this link is clicked we will call the endpoint to open up the exploration editor page along with query parameters to auto navigate to the card which was being reviewed.

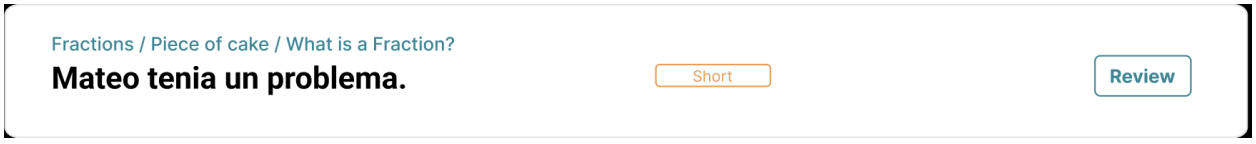


Size of Translation

Backend Changes / None

Frontend Changes

In the [translation review model], we will add a new label in the translation card. This label will display whether the content to be reviewed is short or long.



Additionally, when a user hovers over this label, a tooltip will appear displaying the number of words in a card which needs to be reviewed.

We will create a new utility service which will be used to calculate the length of Html string. The utility service will be called **HtmlLengthService** and will include two functions **computeHtmlLengthInWords()** and **computeHtmlLengthInCharacters().**

computeHtmlLengthInWords(): This function takes a HTML string as input and returns the length of the content in the words, including any non-text elements like SVGs or images.

computeHtmlLengthInCharacters(): This function will take an HTML string as input and return the length of the string in characters, including non-text elements.

To calculate the length of the html string we will assign weights to the tags and return the sum of the weights. For non text components we have <oppia-noninteractive> like math and image. (e.g. oppia-noninteractive-math)

In order to assign weights to each of these tags, we can follow the following criteria:

- For normal words or text inside <p> tag, a weight of (words inside p tag) will be assigned.
- For math functions, a weight of 1 will be applied as the user can use a copy tool and there's no need to translate it.
- For images, we can check with the alt text tags and assign the weight accordingly (1 x words inside (alt text tag)).
  Additionally, for each image instance we will add a small premium of 2 as there are occasionally such cases where images have text which need to be translated as well.

Sample e.g:

```
<oppia-noninteractive-image alt-with-value=\"&amp;quot;This is svg demo
file&amp;quot;\" caption-with-value=\"&amp;quot;svg file demo&amp;quot;\"
filepath-with-value=\"&amp;quot;img_20230523_160213_b3btzp9m4l_height_24_wi
dth_24.svg&amp;quot;\"></oppia-noninteractive-image>\n<p><oppia-noninteract
ive-math
math_content-with-value=\"{&amp;quot;raw_latex&amp;quot;:&amp;quot;22/32&am
p;quot;,&amp;quot;svg_filename&amp;quot;:&amp;quot;mathImg_20230523_160234_
ag5enyibhr_height_2d731_width_5d865_vertical_0d833.svg&amp;quot;}\"></oppia
-noninteractive-math> The fractions is 22 by
32. <oppia-noninteractive-math
math_content-with-value=\"{&amp;quot;raw_latex&amp;quot;:&amp;quot;21&amp;q
uot;,&amp;quot;svg_filename&amp;quot;:&amp;quot;mathImg_20230523_160253_7rb
i5oru2b_height_2d021_width_2d346_vertical_0d241.svg&amp;quot;}\"></oppia-no
ninteractive-math> The integer is 21.</p>
```

The above is a html string which was extracted from **exploration_content_html** of a particular card. Now below is a mock on how this string will be weighted:

- **Oppia-noninteractive-image = 1 * (no of words in alt property of this img) + 2**
- **Oppia-noninteractive-math = 1 (1 for each math component for copying)**
- **Paragraph tag = 1 * (no of words inside p tag)**

Thus based on values mentioned here's the calculation for the mock data:

- Oppia-noninteractive-image = 1 * (count(This is svg demo file)); = (5 * 1) + 2 = 5 +2 = 7
- Paragraph tag = 1 * count(<oppia-noninteractive-math> The fractions is 22 by 32.<oppia-noninteractive-math>The integer is 21.)
  This string will be cleaned and thus it will be counted as
  After Cleaning:
  Paragraph tag = 1 * count(The fractions are 22 by 32. The integer is 21.) = 1* 10 = 10;
- <oppia-noninteractive-math> = 1*2 = 2

  Overall weight of the string becomes = 7 + 10 + 2 = **19**

To ensure the accuracy of this function, we will create tests to cover a range of scenarios and edge cases.

- Test that the computeHtmlLength() function returns the correct length for a simple HTML string containing only text.
- Test that the computeHtmlLength() function handles HTML strings with multiple text nodes correctly.
- Test that the computeHtmlLength() function assigns the appropriate weight to non-text elements like images and math components.
- Test that the computeHtmlLength() function handles HTML strings with special characters or entities correctly.
- Test that the computeHtmlLength() function returns 0 for an empty HTML string.

Once the HtmlLengthService is implemented and tested, it will be used by the new **computeTranslationLengthLabel()** function that will be added to contributor-and-review.component.ts.

```
export interface Suggestion {
  change: {
    skill_id: string;
    content_html: string;
    translation_html: string | string[];
    question_dict: QuestionBackendDict;
    skill_difficulty: string[];
  };
  status: string;
  suggestion_type: string;
  target_id: string;
  suggestion_id: string;
  author_name: string;
  exploration_content_html: string | null;
}
```

The **computeTranslationLengthInWords()** function will take exploration_content_html as a parameter, and use HtmlLengthService to compute the length of the content. From the weight returned by the utility service we can determine whether translations are short or long.
If the overall length is less than 20, the translation will be labeled as short, otherwise it will be labeled as long.

Finally, the key labelText will be populated with “long” or “short” in the **getTranslationContributionSummary()** function of **contributor-and-review.component.ts.**

Image Alt Text

Storage Model Layer Changes

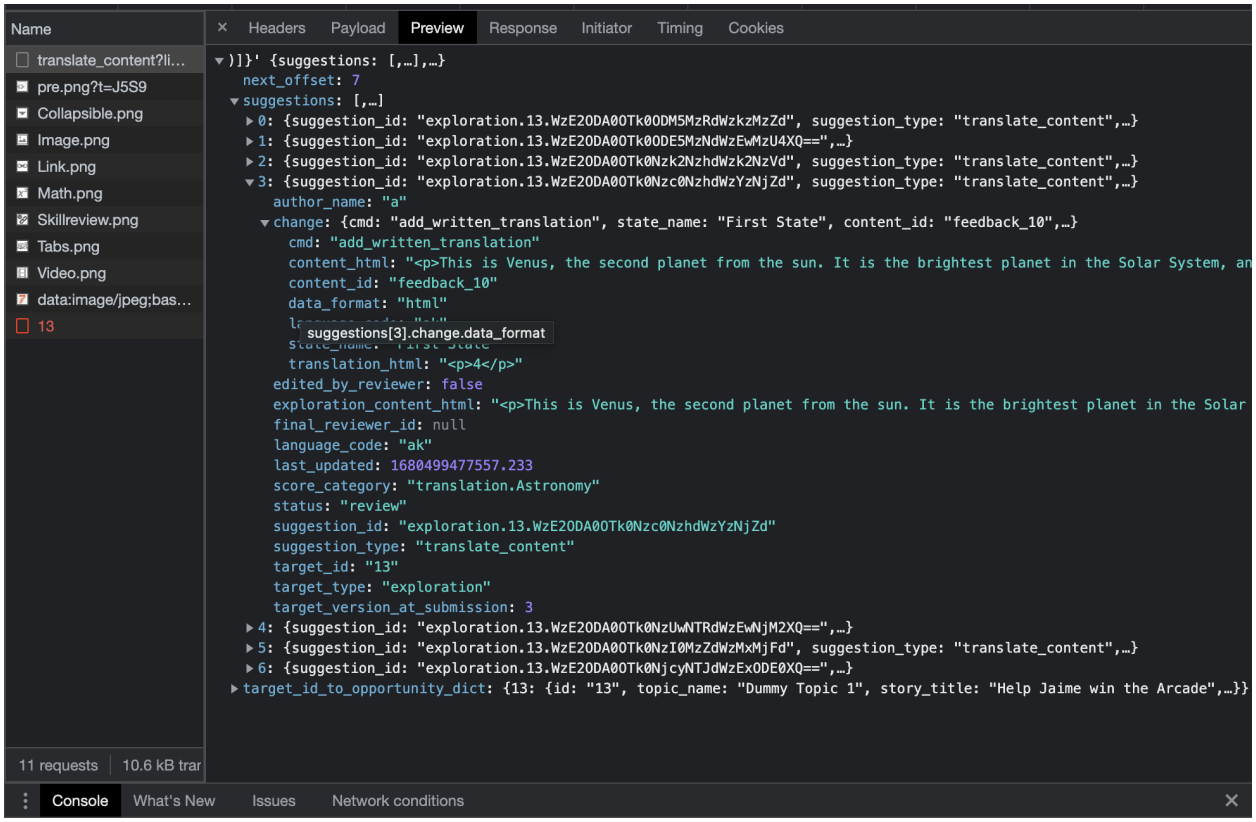
None

Domain Objects

None

User Flows (Controllers and Services)

Since all the necessary data is received from the backend in existing API calls, there's no need for change in it.



Frontend Changes

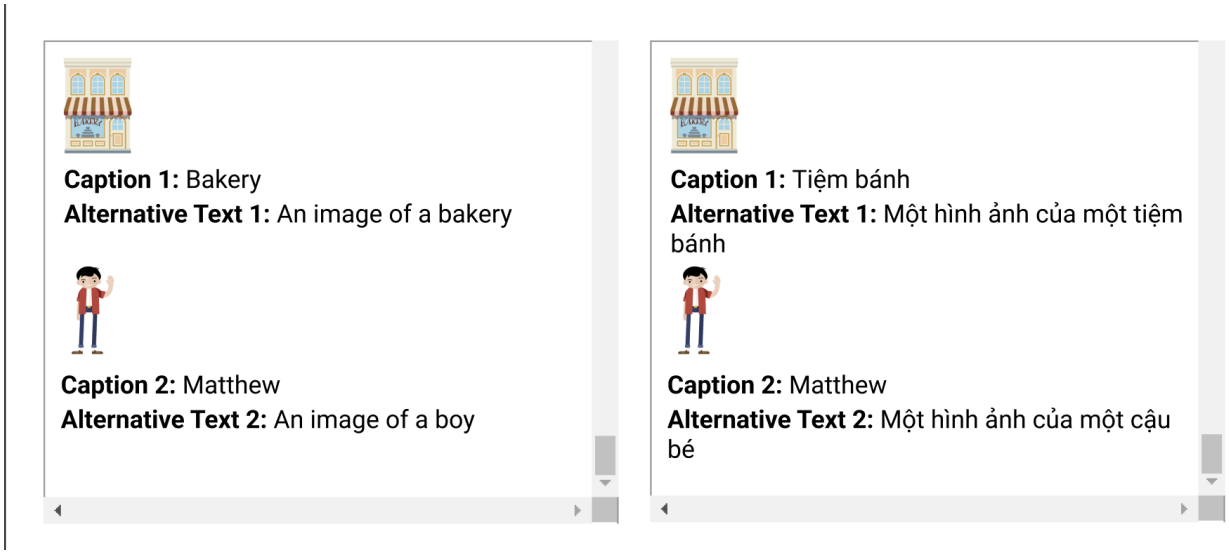
As discussed in the above section, the reviewer has to perform additional clicks to view alt text. In modal-body of translation suggestion review modal we will add another section which will show all the image alt texts as discussed in the mocks using ***ngFor.**

To identify images within the text, we will utilize the DOMParser to parse the HTML content. This will allow us to traverse the DOM tree.

- Use DOMParser to parse the HTML content (Parse the HTML content string using the parseFromString method of DOMParser.)
- Traverse the document object to identify image tags (oppia-noninteractive-img)
- Iterate over the collection of image tags.
- Retrieve the source URL and alt text of the image.

We will also use Angular's inbuilt DOMSanitizer to sanitize the HTML content and ensure that it is safe to work with, minimizing the risk of any security concerns.

By incorporating DOMParser and DomSanitizer, we ensure that the HTML content is safely parsed, sanitized, and displayed in the user interface, providing an enhanced user experience for reviewing image alt text translations while maintaining security best practices.



For example :

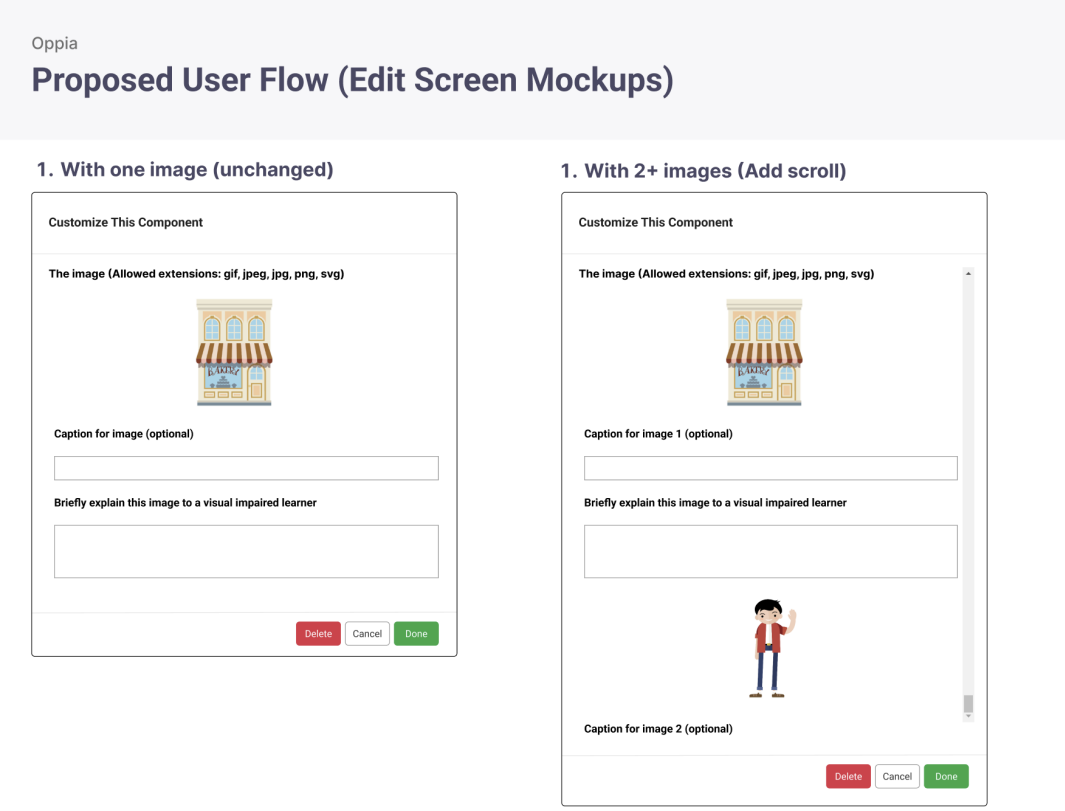
```
<div class="image-info" *ngIf="suggestion.image">
  <div class="image-container">
    <img [src]="suggestion.image" alt="{{suggestion.altText}}">
  </div>
  <div class="image-description">
    <p>{{suggestion.imageDescription}}</p>
  </div>
  <div class="alt-text-container">
    <p *ngFor="let altText of suggestion.altTexts">{{altText}}</p>
  </div>
```

```
</div>
```

Edit Mode:

When the user clicks on the edit button and then clicks on any image to edit we can show the modal which we use currently to allow reviewers to edit for images.

However instead of opening up the modal for only one image, the updated modal will have all the images along with their caption and alt text which users can review and edit accordingly.



Notifying Translation Reviewers about new Suggestions by Email

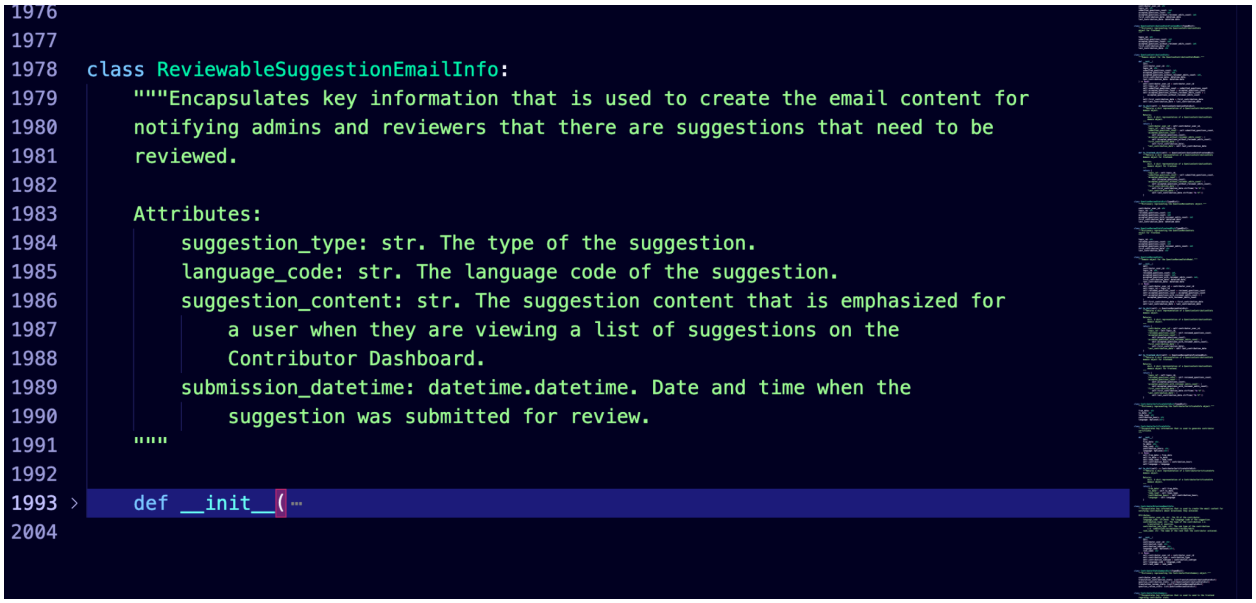
Here's the current notification system (Current Email Notification System). We will be extending it to send email notifications for the suggestions that are newly added to the contributor dashboard.

Notification - Email Mocks

Storage Model Layer Changes

None

Domain Objects



```
1976
1977
1978  class ReviewableSuggestionEmailInfo:
1979      """Encapsulates key information that is used to create the email content for
1980      notifying admins and reviewers that there are suggestions that need to be
1981      reviewed.
1982
1983      Attributes:
1984          suggestion_type: str. The type of the suggestion.
1985          language_code: str. The language code of the suggestion.
1986          suggestion_content: str. The suggestion content that is emphasized for
1987              a user when they are viewing a list of suggestions on the
1988              Contributor Dashboard.
1989          submission_datetime: datetime.datetime. Date and time when the
1990              suggestion was submitted for review.
1991      """
1992
1993 >    def __init__(…
2004
```

We don't have to create any new domain objects, instead we can make use of ReviewableSuggestionEmailInfo to create the email content.

User Flows (Controllers and Services)

Setting up the cron job for handling the reviewer emails:

In cron.py we will add a new handler for mailing reviewers when there are new suggestions which need to be reviewed.

```
class CronMailReviewersNewSuggestionsHandler(
    base.BaseHandler[Dict[str, str], Dict[str, str]]
):
    """Handler for mailing reviewers when there are new suggestion
    that need review on the Contributor Dashboard.
    """

    GET_HANDLER_ERROR_RETURN_TYPE = feconf.HANDLER_TYPE_JSON
    URL_PATH_ARGS_SCHEMAS: Dict[str, str] = {}
```

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

@acl_decorators.can_perform_cron_tasks
def get(self) -> None:
    """Sends each reviewer an email with a list of new suggestion
    that need review on the Contributor Dashboard, based on their
    reviewing permissions.
    """
```

In main.py we will tie this handler function to the new endpoint **/cron/mail/reviewers/contributor_dashboard_new_suggestions**

For the handler defined above, there will be following in cron.yaml

- Task_description
- The corresponding URL
- The schedule, which will be set to every 24 hours

Frontend Changes / None

Translation Cards in the Order of Content Flow

To get the data for one lesson, it takes one model fetch (the exploration and its state).
We will do the sorting at the time the lesson tile is opened/clicked, rather than preemptively for all the lessons. While the opportunity-list is being sorted we will use lazy loading which we already use when the data is being loaded. (sample video of lazy loading)

From the backend we will fetch all the cards for a lesson tile and then sort and paginate it in frontend.
For sorting translation cards in the order of content flow in the lesson, we can make use of the **computeBfsTraversalOfStates** function in core/templates/services/compute-graph.service.ts to obtain a lesson's state names in order. We can then sort the translation cards by lesson state.

Currently, getReviewableTranslationSuggestionsAsync function makes a call to fetch translation cards which are then populated. Thus we will call our function here.

```
▼ 11: {suggestion_id: "exploration.13.WzE2ODUxNzI1ODA0OTZdWzEzODY2XQ==",…}
    author_name: "b"
  ▼ change: {cmd: "add_written_translation", state_name: "First State", content_id:
      cmd: "add_written_translation"
      content_html: "<p>Jupiter, the fifth planet from the sun, is a huge planet tha
      content_id: "feedback_13"
      data_format: "html"
      language_code: "ak"
      state_name: "First State"
      translation_html: "<p>Jupiter</p>"
    edited_by_reviewer: false
    exploration_content_html: "<p>Jupiter, the fifth planet from the sun, is a huge
    final_reviewer_id: null
    language_code: "ak"
    last_updated: 1685172580707.428
    score_category: "translation.Astronomy"
    status: "review"
    suggestion_id: "exploration.13.WzE2ODUxNzI1ODA0OTZdWzEzODY2XQ=="
    suggestion_type: "translate_content"
    target_id: "13"
    target_type: "exploration"
    target_version_at_submission: 3
```


Here's a rough implementation plan:

1. In contribution-and-review.service.ts add a new function sortTranslationCardsByState() to the service. This function should take a list of translation cards and a list of states for the lesson as parameters.
2. Inside **sortTranslationCardsByState()**, use the **computeBfsTraversalOfStates()** function from **core/templates/services/compute-graph.service.ts** to obtain the state names in the order of the content flow in the lesson.
3. Create an empty map to store the translation cards for each state. The keys of the map will be the state names.
4. Iterate over each translation card and assign it to the corresponding state in the map based on its state name.
5. After iterating over all the translation cards, iterate over the state names obtained in step 3. For each state name, retrieve the corresponding translation cards from the map.
6. Within each state, apply the following sorting criteria:
    - Sort the translation cards based on the content_id property.
    - The content_id can be used to determine the type of card (content, feedback, etc.) and its index within the state.
    - Assign a numeric value or a sorting key to each card type to establish the desired order **(content cards come first, followed by interaction, feedback, hints, solutions)**.

7. Concatenate the lists of translation cards obtained in step 6, preserving the sort order within each state, to create a sorted list of translation cards according to the content flow in the lesson.
8. Return the sorted list of translation cards from the sortTranslationCardsByState() function.

User Flows (Controllers and Services)

Discussed above.

Frontend Changes / None

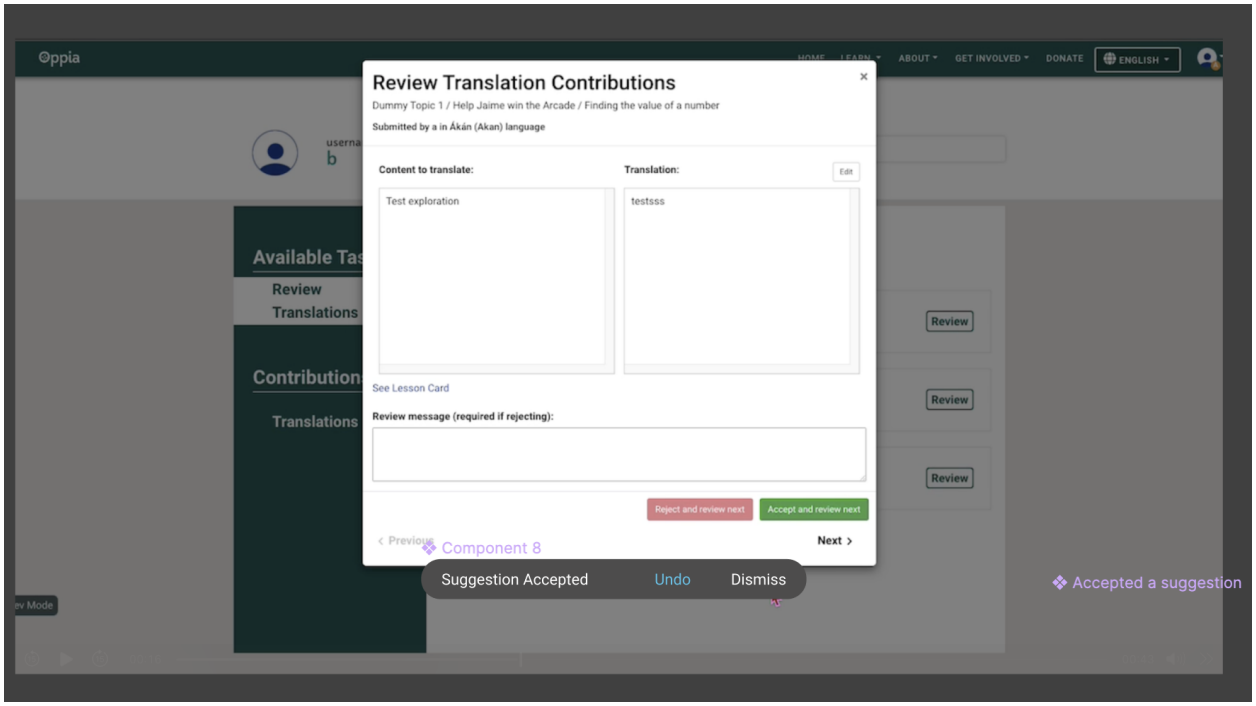
Undo Reviewed Suggestion's Status

The idea is to queue the suggestion in the frontend and whenever the user takes an action on suggestion (accepts or rejects it), commit it to the backend after 30 seconds. If within this time the reviewer wants to undo he can or else the changes will be committed to the backend.

Backend Changes / None

Frontend Changes

Proposed User flow:



Workflow will be similar to the video attached (Designs will be similar to the screenshot attached above)→ Working Prototype (Link to Video)

Review Action Queue:

When a user reviews a translation suggestion (accepts or rejects it), we will use a 30-second delay before the call to the service method is made.

Undo Pop-Up Bar:

As soon as the user takes a review action, a pop-up bar will appear on the bottom of the page. The bar will display a message indicating the action taken (e.g., "Accepted a translation" or "Rejected a translation").

The pop-up bar will also include two interactive elements:

Undo Button: This button allows the user to undo the review action and revert the suggestion's status.

Dismiss Icon: By clicking this icon, the user can immediately commit the review action to the backend.

30-Second Undo Window:

The suggestion will be queued for a 30-second window. During this time, the user can choose to either undo the action or dismiss the pop-up bar.

If no action is taken within the 30-second window, the review action will be automatically committed to the backend.

Queue Management:

If the user reviews another suggestion while there is already an item which is queued, the previous one will be committed and the new suggestion will be queued.
A new pop-up bar corresponding to the most recent action will appear.

Implementation Details:

```
acceptAndReviewNext(): void {
  this.finalCommitMessage = this.generateCommitMessage();
  const reviewMessageForSubmitter = this.reviewMessage + (
    this.translationUpdated ? (
      (this.reviewMessage.length > 0 ? ': ' : '') +
      '(Note: This suggestion was submitted with reviewer edits.)') :
    '');
  this.resolvingSuggestion = true;
  this.siteAnalyticsService.registerContributorDashboardAcceptSuggestion(
    'Translation');

  this.contributionAndReviewService.reviewExplorationSuggestion(
    this.activeSuggestion.target_id, this.activeSuggestionId,
    AppConstants.ACTION_ACCEPT_SUGGESTION,
    reviewMessageForSubmitter, this.finalCommitMessage,
    this.resolveSuggestionAndUpdateModal.bind(this),
    (errorMessage) => {
      this.rejectAndReviewNext(`Invalid Suggestion: ${errorMessage}`);
      this.alertsService.clearWarnings();
      this.alertsService.addWarning(
        `Invalid Suggestion: ${errorMessage}`);
    });
}

rejectAndReviewNext(reviewMessage: string): void {…
}
```

Methods in the code are responsible for committing changes to the backend. Previously, changes were immediately committed to the backend whenever the user took an action.
With the new approach:

- When a user takes an action for a suggestion, we will store the suggestion's details in an object and start a 30-second timer using setTimeOut. Once this time passes, we will commit the object that we stored and set the object to null.
- Meanwhile, while the task is queued, we will show the pop-up bar until it is either dismissed or the time limit is reached.

Meanwhile while the task is queued we will show the pop-up bar (until it's dismissed or time ticks away)

Exceptions :

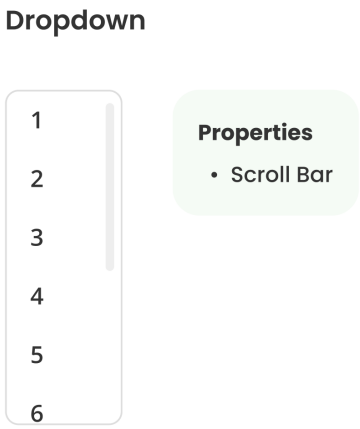
- User tries to refresh/close the browser tab when the queue is not empty :
    - A warning message will appear in a modal with text "You have some pending review approvals/rejections that will be lost if you exit this page. To fully apply them, dismiss the pop-up bar " along with two action buttons "close anyway" and "stay on page ".

Feedbacks on MVP:

- **Design Team (Hieu):**
    - Probably the modal should navigate back to the queued suggestion when we undo the suggestion.
    - If the user clicks on Undo button, the modal window should jump back to the previous translation(The one which was undone). It makes more sense because there is a direct correlation between clicking the button = jumping backing the previous button (action and reaction)
- **Users(Yiga & Bhaskar):**
    - Action buttons in the pop up bar should be differently coloured so that it pops out more, similar to the "Accept" and "Reject" buttons in translation-review-modal.

Update Pagination to Dropdown

I will update the pagination to have a drop down menu along with present "Next" and "Previous buttons".



In [opportunity-list-component.ts](), I will add a new function which will populate the drop down numbers with the available page numbers.
In order to populate the drop down menu (number of pages), we would need to fetch the number of matching models using a database query so that we can use it to calculate the page numbers.
(Total Page Numbers = Fetched_Number/OPPORTUNITY_PAGE_SIZE)

Note: If remainder of division is >=1 then an extra page will be added.

Pagination for translation_review dashboard:

In general, opportunities are fetched in batches from the backend. However, for the **"translation_review"** dashboard only, the translation cards for a particular lesson will instead be fetched all together from backend and then sorted upon performing bfs to show them in context flow.

When a user clicks a lesson tile, translation cards corresponding to it will be sorted and then paginated.

- Calculate the total number of pages (totalPages) based on the number of translation cards and cards per page

- Determine the current page number (currentPage) based on user input or the default initial page(1).
- Calculate the start index (startIndex) and end index (endIndex) of the translation cards to be displayed on the current page.

  Note: Adjust the calculations to handle edge cases like the last page having fewer cards.

- Extract the translation cards for the current page from the sorted list using the start and end indices.
- Display the extracted translation cards to the user.

This function will take the current selected page number from the dropdown menu and show the opportunities accordingly. (**Default selection will be 1)**

We also need to update the HTML for the pagination container. Specifically, we need to modify the div to include a dropdown option in addition to the existing next and previous buttons.

Testing Plan

Acceptance testing plan

#	Test name	Initial setup step	Step	Expectation
1.	Link to Exploration Editor Page Feature test.	Create a exploration “Demo exp” and have a single state in it with state_name: first state Make translation suggestions for that card in “akan” language. Login with a different account with translation reviewer rights for “akan” language.	Go to translation review dashboard Click on the “Demo exp”: then click on the translation card, which is a content card	Translation review modal appears and the link to exploration editor page should be present.
			In translation review modal click the link which navigates to the exploration editor page.	The Exploration Editor page should open to a new tab. The URL of that tab should be **/create/{exp_id}#/translation/first%20state/content%201** and the translation tab should be the active tab.

				The control should auto navigate to the content card for the state, which the reviewer was reviewing.
2 ..	Translation Length feature test	Create an exploration “Demo exp” and have a single state with two content cards in it with state_names: first state and second_state respectively. { Card with a content id (content_1) should have 17 words. Card with a content id (content_2) should have 32 words. } Make translation suggestions for the two cards in “akan” language. Login with a different account with translation reviewer rights for “akan” language.	Go to translation review dashboard Click on the lesson tile, there we can see two translation cards.	The translation card tile with content_id “content_1” should be labeled as short. The translation card tile with content_id “content_2” should be labeled as long.
			Hover over the label of first to trigger the tooltip.	The tooltip should display {Word Count : 17}. .
			Hover over the label of the second card to trigger the tooltip.	The tooltip displays {Word Count : 32}

3.	Image Alt Text section displayed correctly	Submit 3 translations suggestions: - First suggestion has a translation with one image as content. - Second suggestion has a translation with 2 images as content - Third suggestion has a translation with 1 image and 1 math function img as content.	Click on the lesson to review the three cards. Then, click on the translation card tile that contains one image as its content. Click on the edit icon and click on the image.	Translation-modal appears and in the bottom section of the modal there should be a mini image of the main image along with its alt text. Modal should open up for the image to be reviewed.
			Click on the lesson to review the three cards. Then, click on the translation card tile that contains two images as its content. Click on the edit icon and click on the image.	In the bottom section of the translation-review modal there should be a scrollable section which shows mini images both the image along with their alt text. Modal should open up for the image to be reviewed with a scrollable modal which when scrolled shows the second image.
			Click on the third translation card tile which has one image and one math image as content.	In the bottom section of the translation-review modal there should be a section which shows a mini image of the image along with their alt text, but it should not show math img in that section.
4.	Translation cards displayed in the order of content flow	- Create a lesson with card A with content “YYY”. This card A has a multiple choice interaction which links to card B and C. The choices for interaction are say “Opt1 and Opt2”.	Go to the translation review dashboard in the contributor dashboard. Click on the lesson tile created during the setup.	Verify that the first card in the review dashboard is the content of Card A, followed by its interaction options "Opt1" and "Opt2". After that, card B should appear, followed by card C.
5..	Pinning/Unpinning feature test	Login with account the role of translation reviewer.(Login as super admin and give	Go to the contributor Dashboard then go to the translation review dashboard from the left	Both the lesson tiles should appear in the dashboard with none of them being pinned.

		yourself a translation admin role from /admin page.) Assign translation reviewer and choose the language from /contributor-dashboard -admin page Generate dummy data which has 2 lessons. Make translation suggestions for both the lessons.	column.	
			Click the "Pin" icon for the second lesson in the list.	Pinned lesson tile should move at the top of the list, along with a success toast indicating that lesson has been pinned. Icon changes to “Pinned” Icon
			Click on the pin icon for the remaining lesson (i.e now the second one of the list).	A toast appears and let’s know that a lesson has already been pinned.
			Click on the unpin icon of pinned lesson. Click on “Unpin”.	A modal appears to confirm if the user wants to unpin the lesson. The icon changes from pinned to unpinned. A toast appears and let’s know that the lesson has been unpinned.
			While the first lesson is pinned, click on it to view translation cards of that lesson. Review all the cards in it.	Pinned lesson’s translation cards are listed in review_dashboard. All the translation cards are reviewed for that lesson, so the lesson tile should get unpinned and vanish. A toast message appears indicating completion of review of pinned lesson.
6.	Dropdown Pagination feature test	Create an exploration “Demo exp” and have three states with five content cards each in it. Make suggestions for those content cards in “akan” language. Login with a different	Click on Demo exp lesson tile.	The translation cards should be listed and the first ten cards should be visible along with pagination buttons and dropdown in the bottom of the list. The drop down when clicked should be numbered from 1 to 2 and the active selection should be “1”.

		account with translation reviewer rights for “akan” language.	Click on the next button icon.	The remaining five translation cards should appear. The drop down’s active selection should be “2”.
			Click on the dropdown icon and select “1” from the dropdown menu.	The first ten cards should be visible and the active selection for the dropdown should become “1”.
7.	Revert Translation Status Feature test	Login with the role of translation reviewer. (Login as super admin and give yourself a translation admin role from /admin page.) Assign translation reviewer and choose the language from /contributor-dashboard -admin page	Go to the translation review dashboard and click on any lesson tile. Click on any card and review it.	Test that the "Undo" button appears on each translation card that has been reviewed, and disappears after 30 seconds. Test that when the user clicks "Accept" or "Reject", the suggestion is added to the queue with a timer set for 30 seconds.
			User realizes that there was something wrong in translation and thus clicks the Undo button.	The suggestion is removed from the queue and the status is reset to "Pending".
				If the user does not click "Undo" within the 30-second timer, the suggestion is automatically committed to the backend and removed from the queue. When the suggestion is committed to the backend, the status is updated to reflect the reviewer's decision.
			User tries to close the page, while there is still stuff in the queue.	A confirmation dialog appears asking if they want to continue and potentially lose changes, or stay and commit the remaining suggestions.
			User attempts to refresh the page, while the queue is not empty.	

Tests for Notifying Translation Reviewers about new Suggestions by Email

***Note**: We can't do these tests using the automated test framework because it does not send emails. So we need to validate this flow manually.*

Setup:

1. **Enable sending emails:** Set feconf.CAN_SEND_EMAILS to True
2. **Enable sending the Contributor Dashboard emails:** Go to the admins config page and enable sending Contributor Dashboard reviewer email notifications
3. **Create an admin:** On one browser, sign in as an admin (username A)
4. **Give the user admin rights:** Go to the admins role page and update A's role to an admin
5. **Create a reviewer:** On a separate browser sign in as a normal user (username B) click send me news and updates about the site (this will allow the user to get emails)
6. **Give reviewer reviewing rights:** Switch back to A's browser, go to the admin roles page and add B as a question and translation reviewer
7. **Populate the Contributor Dashboard with suggestion opportunities:** For translation opportunities: Create a story with a chapter and an Exploration for question opportunities create a new skill. Everything should be published.

Test steps/expectations:

8. **Submit translation suggestions on the Contributor Dashboard:** Go to A's Contributor Dashboard (http://localhost:8181/contributor-dashboard) and submit some translations in the languages that B can review. The suggestions need to be created by a user that isn't going to receive the emails about them because reviewers can't review their own suggestions.
9. **Verify that there are suggestions for the reviewer to review:** Go to B's Contributor Dashboard and see the suggestions waiting for review. This confirms that B has the right reviewing permissions.
10. **Run the cron job:** Go to the app engine's cron page and click on the CronMailReviewersContributorDashboardSuggestionsHandler to trigger sending the emails.
11. **Verify that the sent email was correct:** The email should have new translation suggestions. Also verify that the email reads correctly (no grammatical errors etc)
12. **Verify that the model created for the sent email was correct:** If possible, look at the datastore entries in the app engine and verify that the created SentEmailModel contains the correct email data.

Backend tests:

All the backend files will be accompanied by their test files.

Feature Gates-

I plan to make five feature flags which are mentioned below:

- Undo translation status will be gated behind undo_translation_status feature gate.
- Pinning/unpinning of lesson tiles will be gated behind allow_translation_reviewers_to_pin_opportunities feature gate.
- Image alt text and exploration page link will be gated behind improved show_enhanced_translation_review_modal feature gate.
- Translation size feature will be gated behind show_translation_size.
- Updated Pagination will be gated behind show_dropdown_pagination_in_contributor_dashboard.

Implementation Plan

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

Milestone 1

Key objectives for this milestone:

- Notify translation reviewers about new submissions by email.
- Link to the exploration editor so that the original lesson can be seen.
- Show the size of the translation.
- Allow reviewers to undo the acceptance or rejection of a translation card for up to 30 seconds.

Note: All user flows will work on both desktop and mobile.

Each PR deadline includes the time to add the corresponding unit tests.

No.	Description of PR / action	Prereq PR numbers	Target date for PR creation	Target date for PR to be merged

1	Add feature flag **show_translation_size**		June 1	June 2
2	Introduce new service HtmlLengthService which will be used to calculate lengths of html strings.	-	June 5	June 10
3	Modification in translation review modal to include size of translation and write acceptance tests for the same.	1,2	June 8	June 15
4	Show translation cards according to the order of the content flow in the lesson.		June 23	June 29
5	Add link to exploration page in translation review modal.	5	July 2	July 8
6	Modify the translation reviewer modal to show the alt text below the image.	5	July 8	July 13
	PM demonstration for Milestone 1		**July 12**	
7	Bug fixes to above milestones (if required)		July 17	July 22

Milestone 2

Key objectives for this milestone:

- Show image alt text below the image.
- Allow translation reviewers to pin/unpin.
- Show translation cards according to the context flow of a lesson.
- Update the pagination method to dropdown menu.

Note: All user flows will work on both desktop and mobile.

Milestone	Description of PR	Expected date	
		PR by	Merged by
1	Add link to exploration editor page in translation review Modal	Aug 4	Aug 24

2	Enqueue the suggestion creation in a front-end queue scheduled for 30 seconds in future and enable Undo feature for translation reviewers to let them undo status of the translation reviewed within 30 seconds.	Aug 28	Sep 15
3	Add feature flag: allow_translation_reviewers_to_pin_opportunities	Sep 9	Sep 13
4	Create new models for pinning/unpinning lessons and backend tests for the same.	Sep 15	Sep 20
5	Create Service methods and handler for pinning	Sept 22	Sept 28
6	Modify the front end files to reflect pinning/unpinning of lessons and acceptance tests for the same.	Sep 29	Oct 10
7	Set up a cron job for the email notification system and write backend tests.	Oct 12	Oct 20

Future Work

***Note**: This section is mainly for reference (since it is understood that items in this section will not be part of the GSoC project). Proposals will primarily be evaluated based on the implementation plan above.*

- In this proposal we are focusing on translation reviewers, and this section gives a summary on how we can extend the features for question contributors.

Features being proposed by this proposal	Summary of how are we going to extend the features to practice question
1. Pinning/Unpinning: Users can pin/unpin opportunities (translations or practice questions) for further review or exploration.	N/A, confirmed with the question team that this feature is not required for the questions tab.
2. Link to Exploration Editor Page.	N/A, for questions we don't need to see the

	content flow of the lesson.
3. Size of translation	N/A
4. Notifying translation reviewers about new suggestions by email	In cron.py we will add a new handler function **CronMailReviewersNewSuggestionsHandler** as part of this proposal along with the endpoint **`/cron/mail/reviewers/contributor_dashboard_new_suggestions.`** ` Thus we don't have to make any significant changes to extend it for practice questions. We can make use of ReviewableSuggestionEmailInfo to create the email content.
5. Translation Cards in the Order of Content Flow	N/A

6. Undo Reviewed Translation Status	When the reviewer takes action on practice questions, this handler gets called. This endpoint is tied to the controller function which is thus responsible for changing the status of the suggestion.
	
	Thus when the action is taken we would enqueue this suggestion in a queue for 30 seconds, and once the timer is over we can then commit the changes to the backend calling the same endpoint.
	
	Here's what our queueItem look like:
	
	`interface QueueItem {`
	`  opp_id: string;`
	`  opp_type: 'translation' | 'practiceQuestion';`
	`  item: Suggestion;`
	`  action: 'accept' | 'reject';`
	`  timestamp: number;`
	`}`

	Item is a property which is of type Suggestion which holds the suggestion object. Note: Suggestion is a predefined interface
7. Update Pagination to Drop-Down Menu	This feature will be implemented for the entire Contributor Dashboard , that means for all the four opportunity types (“Translation Review”, “Question_Review”, “Question_Suggestions”, “Translation_ Suggestions”). Thus there’s no need to implement it for the future.
8. Image Alt Text	N/A

- To provide users with the ability to search for specific translation cards within each lesson would be to add a search bar or input field within the lesson view, where users can enter keywords or phrases to search for