About You

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

I've always been fascinated by open source and its interesting organizational structure. I think it is quite cool how people voluntarily and collectively create and improve software to help make people's lives better. I thought applying for GSOC would be an excellent way to learn more about open source and learn its best practices.

I was initially attracted to working in Oppia because I thought an open source online educational resource was awesome. I personally have been saved many times from well made online tutorials, and I also had some experience with web development projects, so I thought I would be a good fit.

As I worked on Oppia more, I was impressed with its structure and the resources it gave to new contributors. Not only did Oppia have many tutorials and great documentation, but also had multiple avenues to ask for help. Because of this, even though I unfortunately missed the email notifying me of having an onboard mentor. Other members of Oppia would answer my questions, help me with my issues, and review my code. With all of these resources I was able to make a successful pull request to Oppia in my first week.

I picked project 2.4 Ensure that Oppia is Fully Accessible because it interested me the most and I believed I had the skill set to complete it. I had experience making tests in Oppia and had modified html and javascript on previous projects. In addition, making sure that Oppia is accessible to all users is an important step for its future, and I hope I could make that a reality.

Prior Experience

I'm a third year computer science student at UC Irvine. One relevant coursework I completed was a full stack project to create a blockbuster rental site. Users could pick through thousands of movies and purchase them through credit card or paypal. I learned many relevant skills for Oppia like Javascript, HTML, CSS, Bash, Git. I also learned how to navigate the various layers of website development through this. I also have three years of experience of coding with Python and Java.

So far I have made two merged pull requests that are related to testing Oppia's frontend
#8916
#8934
I will also keep contributing to Oppia after submitting this proposal. Here is a link of all my PRs:
https://github.com/oppia/oppia/pulls?q=is%3Apr+author%3Ajimbyo+

Contact info and timezone(s)

Jim Zhan
Email: jimz@uci.edu
Github: https://github.com/jimbyo
Preferred Method: Email or Github are both fine
Timezone: I will be in California over the summer (Pacific Time Zone)

Provide your contact information, preferred method of communication, and which timezone(s) you will primarily be in during the summer.

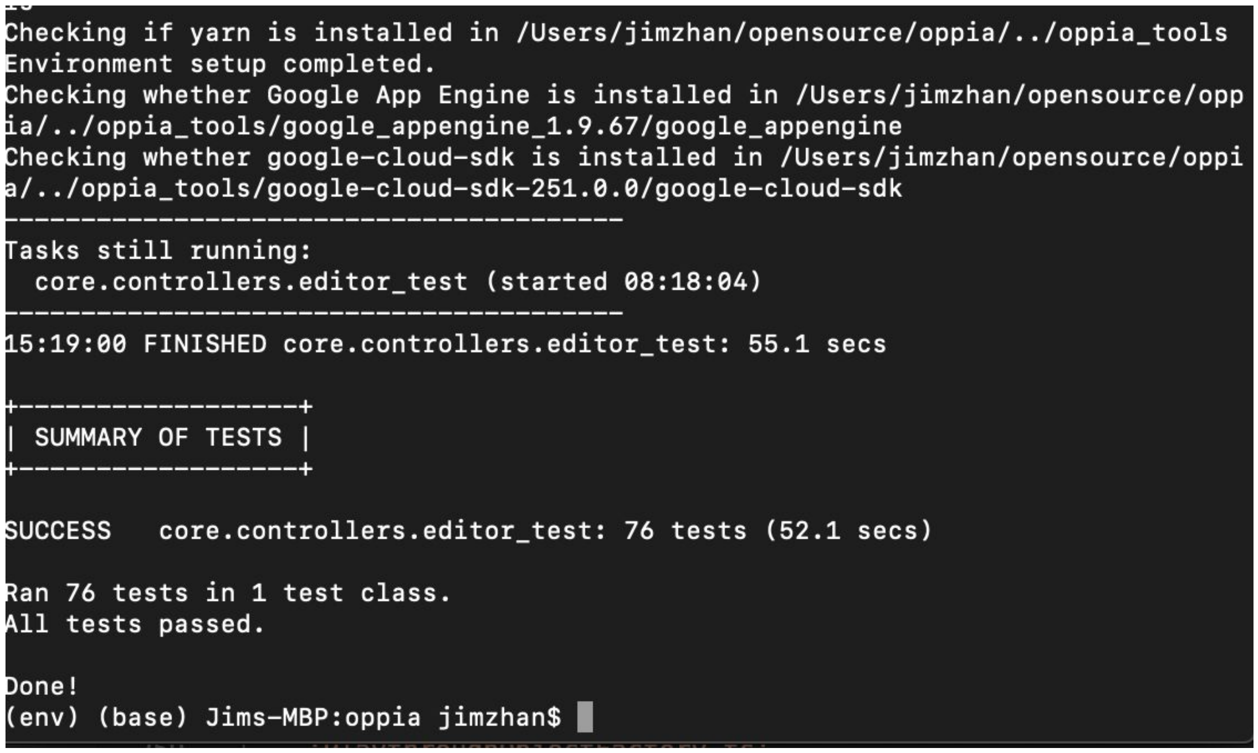
Time commitment

I can spend 40 hours a week, monday through saturday

Essential Prerequisites

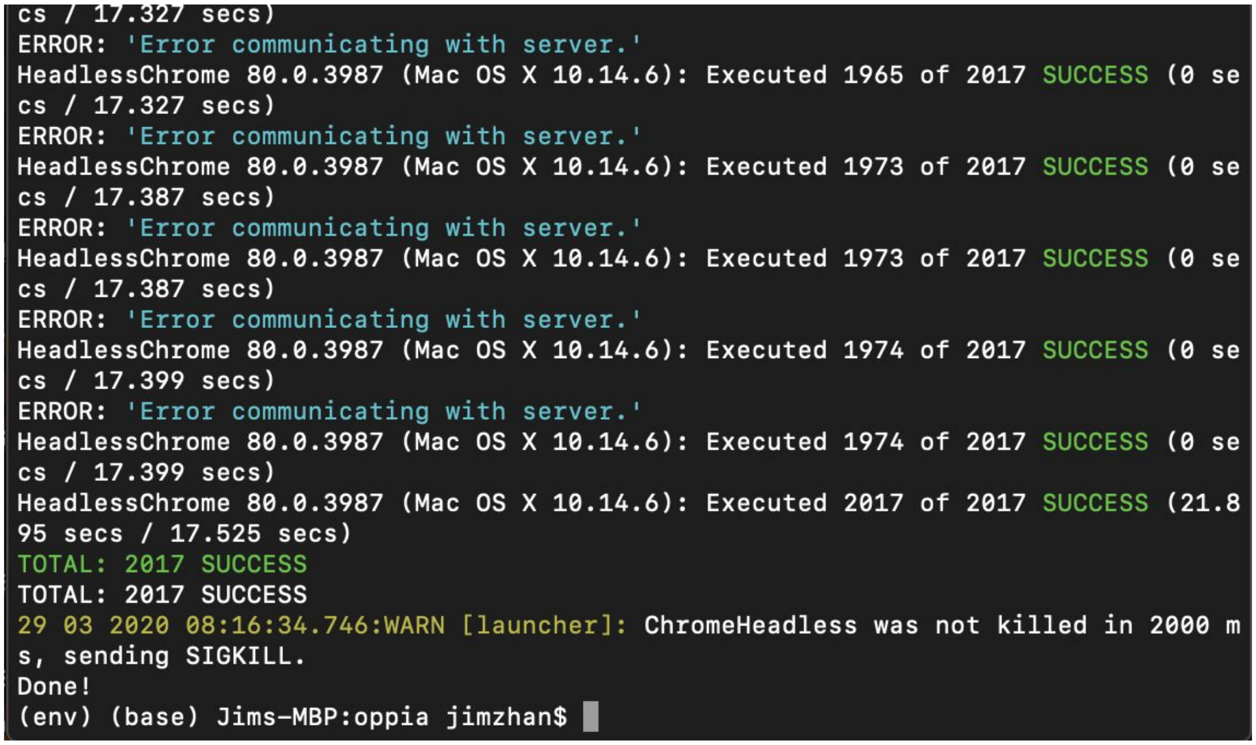
Answer the following questions:

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



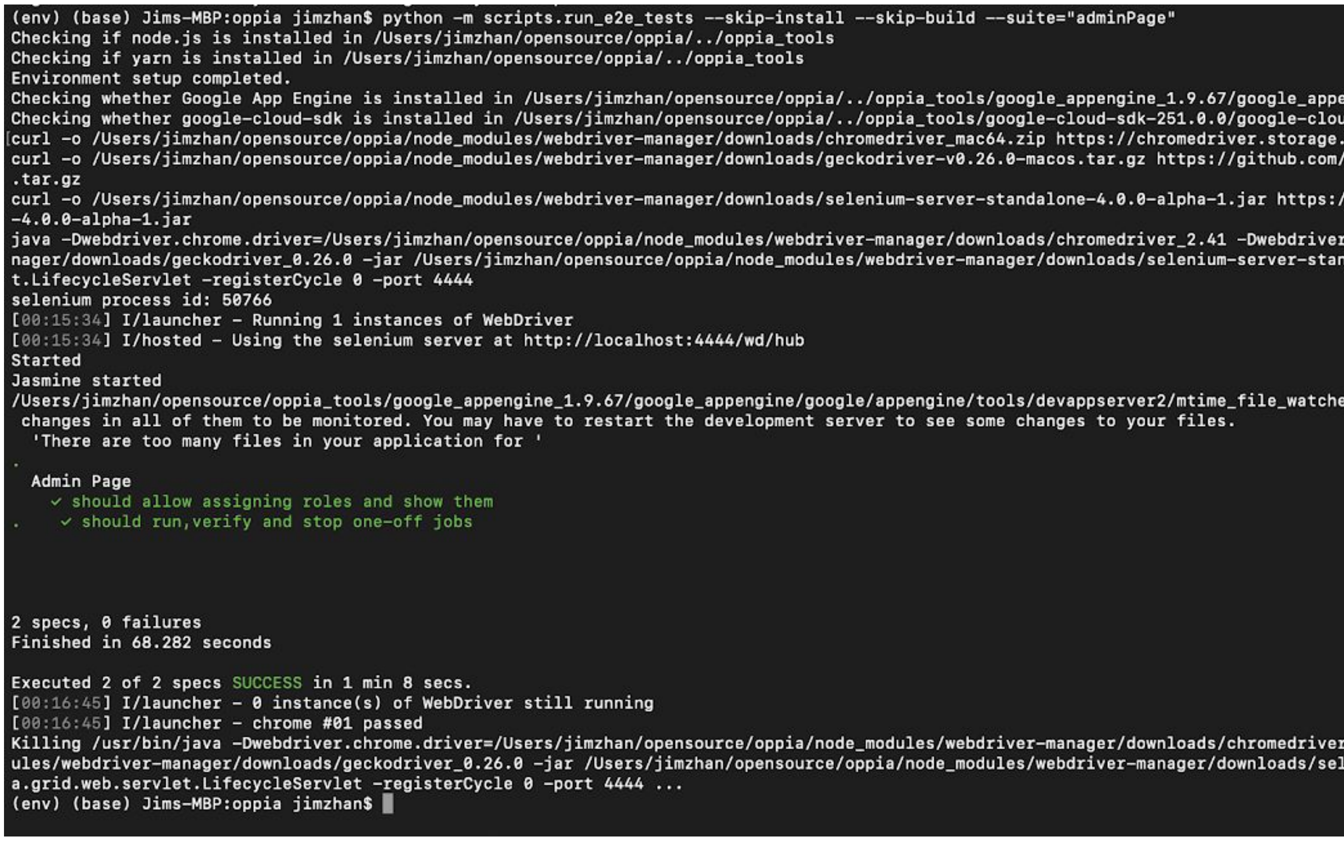
```
Checking if yarn is installed in /Users/jimzhan/opensource/oppia/../oppia_tools
Environment setup completed.
Checking whether Google App Engine is installed in /Users/jimzhan/opensource/opp
ia/../oppia_tools/google_appengine_1.9.67/google_appengine
Checking whether google-cloud-sdk is installed in /Users/jimzhan/opensource/oppi
a/../oppia_tools/google-cloud-sdk-251.0.0/google-cloud-sdk
----------------------------------------
Tasks still running:
  core.controllers.editor_test (started 08:18:04)
----------------------------------------
15:19:00 FINISHED core.controllers.editor_test: 55.1 secs

+------------------+
| SUMMARY OF TESTS |
+------------------+

SUCCESS   core.controllers.editor_test: 76 tests (52.1 secs)

Ran 76 tests in 1 test class.
All tests passed.

Done!
(env) (base) Jims-MBP:oppia jimzhan$
```

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



```
cs / 17.327 secs)
ERROR: 'Error communicating with server.'
HeadlessChrome 80.0.3987 (Mac OS X 10.14.6): Executed 1965 of 2017 SUCCESS (0 se
cs / 17.327 secs)
ERROR: 'Error communicating with server.'
HeadlessChrome 80.0.3987 (Mac OS X 10.14.6): Executed 1973 of 2017 SUCCESS (0 se
cs / 17.387 secs)
ERROR: 'Error communicating with server.'
HeadlessChrome 80.0.3987 (Mac OS X 10.14.6): Executed 1973 of 2017 SUCCESS (0 se
cs / 17.387 secs)
ERROR: 'Error communicating with server.'
HeadlessChrome 80.0.3987 (Mac OS X 10.14.6): Executed 1974 of 2017 SUCCESS (0 se
cs / 17.399 secs)
ERROR: 'Error communicating with server.'
HeadlessChrome 80.0.3987 (Mac OS X 10.14.6): Executed 1974 of 2017 SUCCESS (0 se
cs / 17.399 secs)
HeadlessChrome 80.0.3987 (Mac OS X 10.14.6): Executed 2017 of 2017 SUCCESS (21.8
95 secs / 17.525 secs)
TOTAL: 2017 SUCCESS
TOTAL: 2017 SUCCESS
29 03 2020 08:16:34.746:WARN [launcher]: ChromeHeadless was not killed in 2000 m
s, sending SIGKILL.
Done!
(env) (base) Jims-MBP:oppia jimzhan$
```

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



```
(env) (base) Jims-MBP:oppia jimzhan$ python -m scripts.run_e2e_tests --skip-install --skip-build --suite="adminPage"
Checking if node.js is installed in /Users/jimzhan/opensource/oppia/../oppia_tools
Checking if yarn is installed in /Users/jimzhan/opensource/oppia/../oppia_tools
Environment setup completed.
Checking whether Google App Engine is installed in /Users/jimzhan/opensource/oppia/../oppia_tools/google_appengine_1.9.67/google_appe
Checking whether google-cloud-sdk is installed in /Users/jimzhan/opensource/oppia/../oppia_tools/google-cloud-sdk-251.0.0/google-clou
curl -o /Users/jimzhan/opensource/oppia/node_modules/webdriver-manager/downloads/chromedriver_mac64.zip https://chromedriver.storage.
curl -o /Users/jimzhan/opensource/oppia/node_modules/webdriver-manager/downloads/geckodriver-v0.26.0-macos.tar.gz https://github.com/
.tar.gz
curl -o /Users/jimzhan/opensource/oppia/node_modules/webdriver-manager/downloads/selenium-server-standalone-4.0.0-alpha-1.jar https:/
-4.0.0-alpha-1.jar
java -Dwebdriver.chrome.driver=/Users/jimzhan/opensource/oppia/node_modules/webdriver-manager/downloads/chromedriver_2.41 -Dwebdriver
nager/downloads/geckodriver_0.26.0 -jar /Users/jimzhan/opensource/oppia/node_modules/webdriver-manager/downloads/selenium-server-star
t.LifecycleServlet -registerCycle 0 -port 4444
selenium process id: 50766
[00:15:34] I/launcher - Running 1 instances of WebDriver
[00:15:34] I/hosted - Using the selenium server at http://localhost:4444/wd/hub
Started
Jasmine started
/Users/jimzhan/opensource/oppia_tools/google_appengine_1.9.67/google_appengine/google/appengine/tools/devappserver2/mtime_file_watche
 changes in all of them to be monitored. You may have to restart the development server to see some changes to your files.
  'There are too many files in your application for '
.
  Admin Page
    ✓ should allow assigning roles and show them
.    ✓ should run,verify and stop one-off jobs




2 specs, 0 failures
Finished in 68.282 seconds

Executed 2 of 2 specs SUCCESS in 1 min 8 secs.
[00:16:45] I/launcher - 0 instance(s) of WebDriver still running
[00:16:45] I/launcher - chrome #01 passed
Killing /usr/bin/java -Dwebdriver.chrome.driver=/Users/jimzhan/opensource/oppia/node_modules/webdriver-manager/downloads/chromedriver
ules/webdriver-manager/downloads/geckodriver_0.26.0 -jar /Users/jimzhan/opensource/oppia/node_modules/webdriver-manager/downloads/sel
a.grid.web.servlet.LifecycleServlet -registerCycle 0 -port 4444 ...
(env) (base) Jims-MBP:oppia jimzhan$
```

Other summer obligations

No other summer obligations currently

Communication channels

Google hangouts would be a good communication channel. I think maybe a weekly or biweekly meeting, with google hangouts chat text messages for smaller issues.

Product Design

This project will improve Oppias ability to support users with physical, visual, or hearing disabilities. It aims to fix any issue caught by audits and improve the accessibility of the site. For physically impaired users, it aims to implement new and more advanced keyboard navigation features based off of facebook's. To help the developer maintain this standard, this project will create automated tests to ensure that any future frontend changes will not lower Oppia's accessibility score below an acceptable score.

1. Oppia's frontend will be redesigned to aid users with disabilities by achieving a score of 100 percent accessibility on GoogleChrome Lighthouse Audit.
2. A manual audit will further refine the website's accessibility on issues that can't be detected via Lighthouse
3. LightHouse automated tests will prevent regressions in accessibility.
4. Keyboard navigation will be introduced

GoogleChrome Lighthouse Audit

Lighthouse is a good tool for checking the accessibility of Oppia. It establishes a baseline score of the accessibility of the website, and then gives you a to-do list to improve the accessibility of the website. By getting 100 percent accessibility on GoogleChrome's lighthouse audit, issues that burden people with disabilities can be solved. The significance of these issues is explained in (Figure 1) and the current lighthouse scores are shown in (Figure 2) and a few common issues stood out:

- Low contrast text that is difficult to read
- Screen Readers having difficulty accessing information
- Incorrect or vague information being given to screen readers

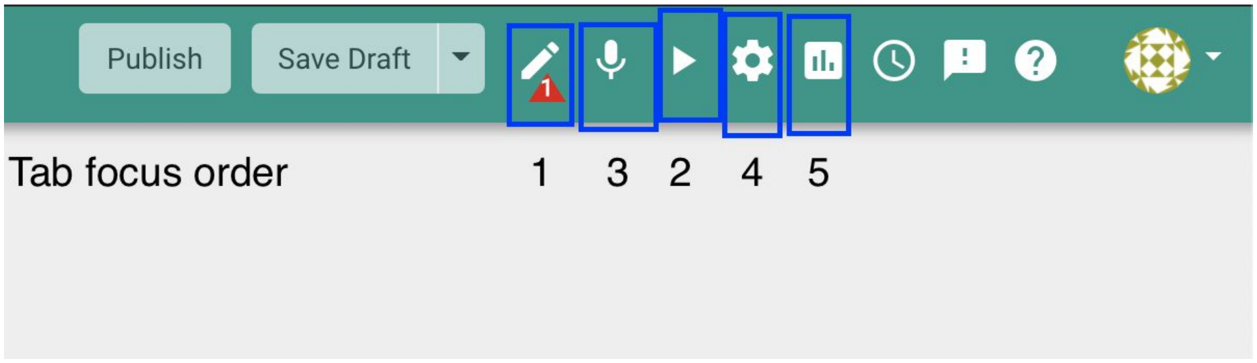
The latter two issues really need to be addressed or screen readers will be unable to use Oppia.

Manual Audit

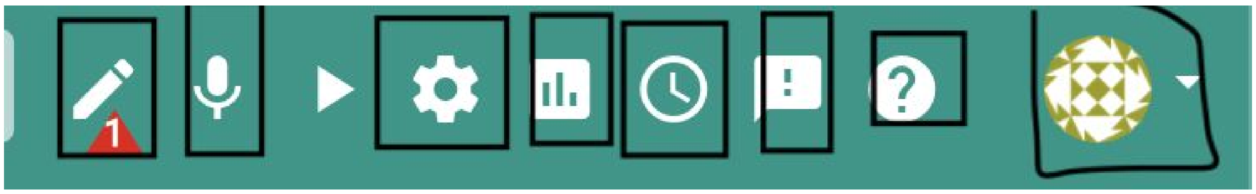
Lighthouse cannot address all areas of accessibility, and manual testing is needed to further improve the experience of the user.

A few issues users deal with below that can't be caught by lighthouse are:

A. The page has an illogical tab order



B. Interactive controls are not keyboard focusable



While perhaps not as critical as some of the issues caught by lighthouses. Testing these issues is important so that users can have a more streamlined and positive experiences on Oppia. A few strategies to test and fix these issues are below:

Google Chrome Automated tests

GoogleChrome Lighthouse will be used to also generate a lighthouse report alongside every PR. Developers could recognize when their PR reduces the accessibility of the site and could prevent regressions.

Issue Type Significance (Figure 1)

Issue Type	Exact issues found so far	Issue Significance
Contrast	1.) Background and foreground colors do not	1.) Text that has a low contrast ratio can be hard to

	have a sufficient contrast ratio	read. This is particularly troublesome for people with low vision
Tables and lists	1.) Lists do not contain only <li> elements and script supporting elements (<script> and <template>)	1.) Screen readers and other assistive technologies depend on lists being structured properly to keep users informed of content within the lists.
Internalization and localization	1.) <html> element does not have a [lang] attribute	1.) If a page doesn't specify a language for the <html> element, a screen reader assumes the page is in the default language that the user choose when setting up the screen reader, often making it impossible to understand the content.
ARIA	1.) [role]s are not contained by their required parent element 2.) [aria-*] attributes do not match their roles	1.) If the required parent role isn't present, assistive technologies may announce the child roles as plain text content rather than the intended control. 2.) An unsupported attribute generally won't break an element's role.However, this issue is still important to fix and probably indicates a mistaken assumption in your code.
Buttons and labels	1.) Buttons do not have an accessible name	1.)This issue was very noticeable in the exploration editor page. All the interactions would simply be called buttons. No one using a voice reader would be able to navigate the

		website.

Oppia Lighthouse Scores (Figure 2)

Webpage	**Current Score**	Issues discovered
Exploration editor page	67	*ARIA *Names and Labels *Contrast * Internalization and localization *Tables and lists
Creator Dashboard	88	*ARIA *Contrast *Tables and Lists
Learner Dashboard	88	*ARIA *Contrast *Tables and Lists
Preferences	88	*ARIA *Contrast *Tables and Lists
Library	88	*ARIA *Contrast *Tables and Lists
Topics and Skills Dashboard	92	*ARIA
Classroom page	88	*ARIA *Contrast *Tables and Lists
Editor Page(stats, history,feedback)	77	*ARIA *Contrast *Tables and Lists *Internationalization and localization
Topics Editor	92	*ARIA

Skills Editor	92	*ARIA
Story Editor	92	*ARIA
Splash	88	*ARIA *Contrast *Tables and Lists

Ensure Oppia stays Accessible

After this GSOC project is complete there will be three ways of ensuring Oppia continues to be accessible

- First is the lighthouse-ci assert functionality, this will fail any pull request that violates an accessibility requirement
- Second is updated accessibility documentation. This will help developers learn good practices and know what to do when their lighthouse audits fail.
- Finally release testers must be given new accessibility testing instructions,to test for anything that the audits cannot catch. Most critically, Every user must be able to finish each critical user journey with only a keyboard, and the screen reader must be able to give useful and not vague information about every focusable element

How developers will be trained

- **How developers will be trained(so that developers don't just modify the accessibility e2e tests if their changes break them). There should also be a documentation plan.**

1.) Developer Training Plan

- Developers will be trained from the documentation. Updates to the Testing for Accessibility page will provide comprehensive descriptions of each accessibility audit and links to potential solutions.

2.) Documentation Plan

1.) Update the documentation for Testing For Accessibility Page

- Add documentation describing each Lighthouse audit and common solutions for each of the audits.
  - This will be the primary way to train developers in dealing with accessibility issues discovered by Google LightHouse
- Add documentation describing each manual audit and their methodologies

- Link to the Google LightHouse Audit documentation

2.) Update the documentation for Running Test Page
- Add instructions for running the Lighthouse Audits with `lhci autorun`
  - Explain different configurations of the command such as running lighthouse only on a specific URL.

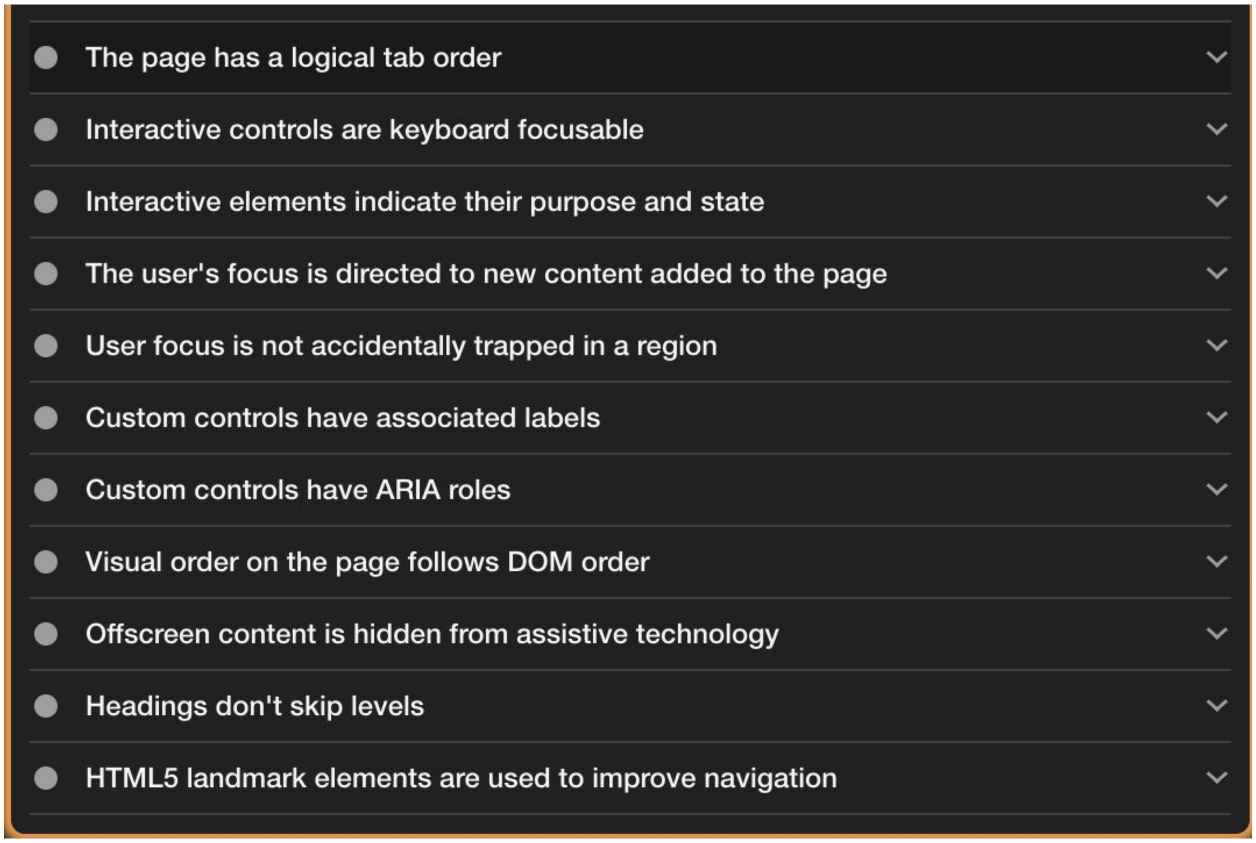
3.) Create new documentation page titled Writing lighthouse tests
- This will explain the lighthouse config to new developers and allow them to test newly created pages.

Manually check accessibility

- **Enumerate all the accessibility requirements that will not be covered by automatic checks after the GSoC project ends. Find ways to automatically check them, or propose an alternative solution that is not too resource intensive.**

Accessibility enumeration

  - These are the accessibility requirements that can't be covered by lighthouse after the GSOC project.


- The page has a logical tab order
- Interactive controls are keyboard focusable
- Interactive elements indicate their purpose and state
- The user's focus is directed to new content added to the page
- User focus is not accidentally trapped in a region
- Custom controls have associated labels
- Custom controls have ARIA roles
- Visual order on the page follows DOM order
- Offscreen content is hidden from assistive technology
- Headings don't skip levels
- HTML5 landmark elements are used to improve navigation

    - I plan to write documentation outlining how to recognize the issues above, and what good practices to make a habit.
    - Links to documentation on how to test and fix those issues will also be provided.
- I also plan to update the critical user journeys to help prevent regressions on these issues

Planned changes to the critical user journeys:

- Every critical user journey will be required to test with a keyboard and screen reader at least once per release and fulfill two key requirements
    1. Each critical user journey must be able to be finished with only a keyboard and screen reader
    2. Each element announced by the screen reader must have relevant and non vague information.
       E.g No buttons being announced as button

  The release testers could find experience breaking issues and report them.

  The less critical accessibility requirements will be documented and explained to aid developers, but release testers would not be required to test them.

  Note: I need to have a discussion about the release testing team about this proposed change and If you dislike the current proposed changes I was also brainstorming alternative ideas below.

Keyboard Navigation

Oppia will now have more keyboard shortcuts for people who rely on keyboards to navigate. These new keyboard shortcuts follow Microsoft's Guidelines For Keyboard User Interface Design for PC and macOS Human Interface Guidelines for Keyboard Shortcuts. This will enable them to avoid conflicts with existing commands and be intuitive to learn.

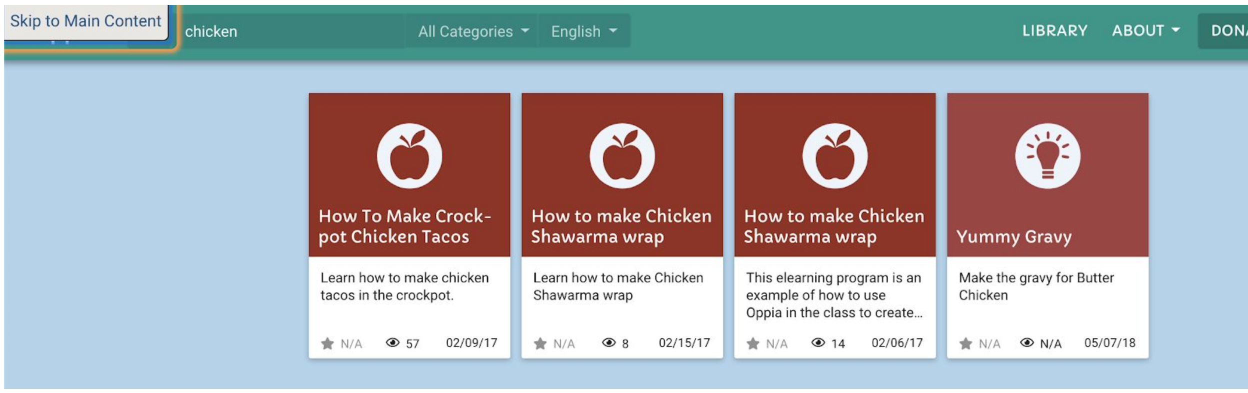
Action Shortcuts

Action shortcuts help you do things on particular pages. Action shortcuts will be limited to the Library Webpage and the Explorer Viewer. This decision is due to the importance of these two web pages and the difficulty of navigating these pages using screen readers.

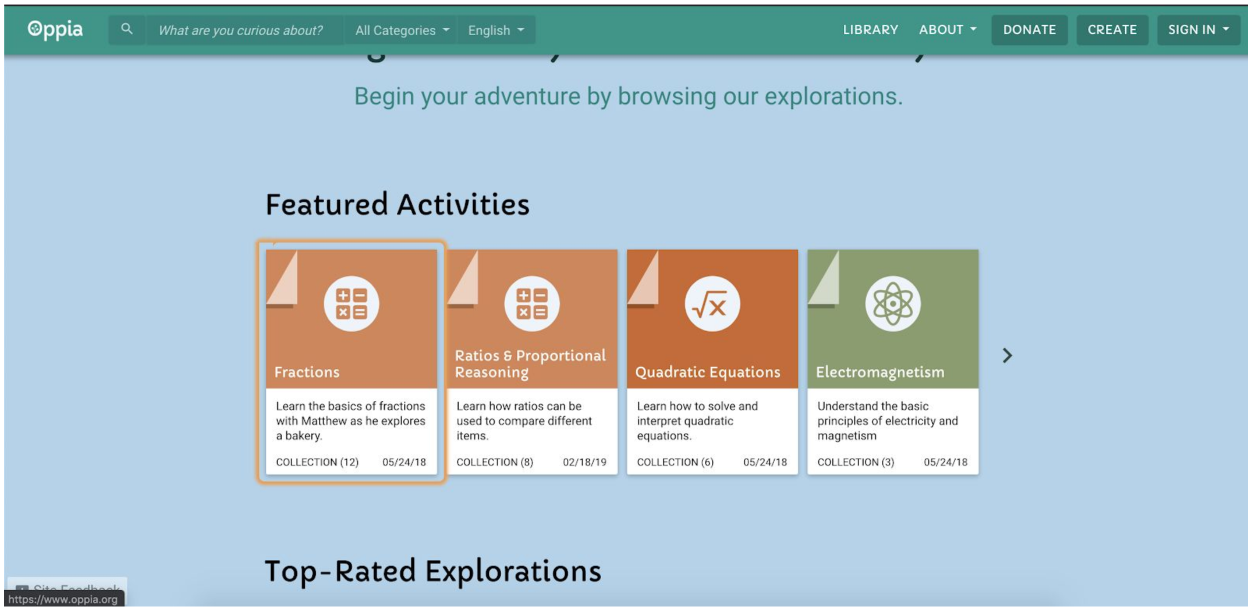
Library Page

These Action Shortcuts are based off of the Facebook Keyboard UI

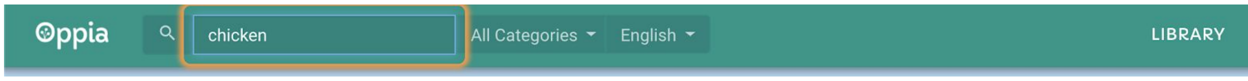
- S - Move focus to Skip to Main Content button
  - This keyboard shortcut was created to help screen readers specifically. Currently, screen readers either need to manually tab to the explorations, or click enter on the Skip to Main Content button


Skip to Main Content
chicken
All Categories
English
LIBRARY
ABOUT
How To Make Crock-pot Chicken Tacos
Learn how to make chicken tacos in the crockpot.
N/A 57 02/09/17
How to make Chicken Shawarma wrap
Learn how to make Chicken Shawarma wrap
N/A 8 02/15/17
How to make Chicken Shawarma wrap
This elearning program is an example of how to use Oppia in the class to create...
N/A 14 02/06/17
Yummy Gravy
Make the gravy for Butter Chicken
N/A N/A 05/07/18

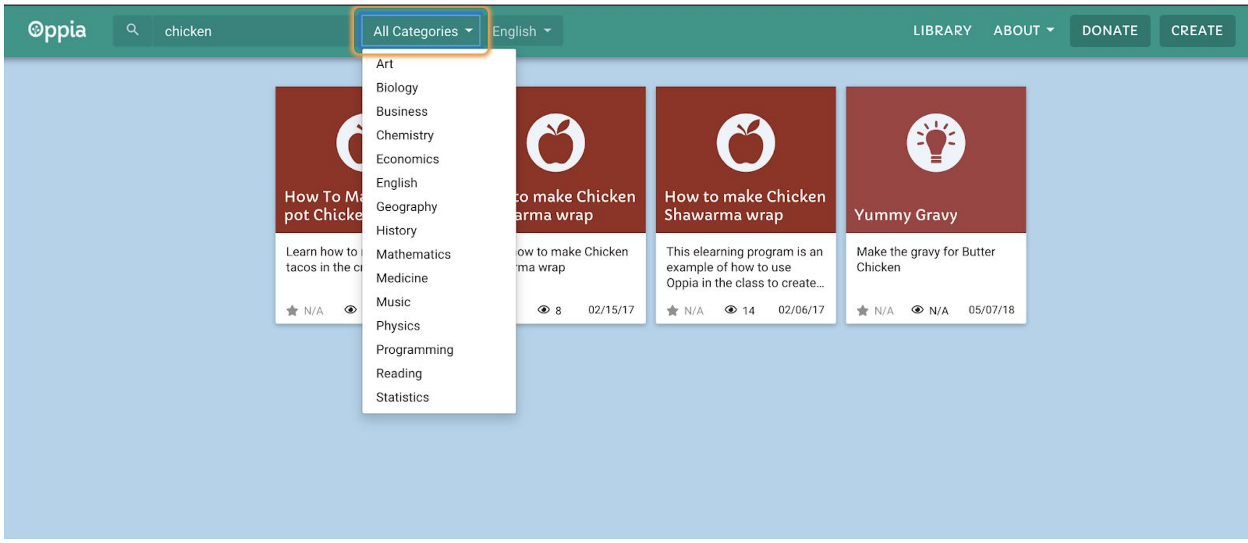
  - Pressing the Skip to Main Content button would send the focus to the entire screen, and the next tab would access the first exploration


Oppia
What are you curious about?
All Categories
English
LIBRARY
ABOUT
DONATE
CREATE
SIGN IN
Begin your adventure by browsing our explorations.
Featured Activities
Fractions
Learn the basics of fractions with Matthew as he explores a bakery.
COLLECTION (12) 05/24/18
Ratios & Proportional Reasoning
Learn how ratios can be used to compare different items.
COLLECTION (8) 02/18/19
Quadratic Equations
Learn how to solve and interpret quadratic equations.
COLLECTION (6) 05/24/18
Electromagnetism
Understand the basic principles of electricity and magnetism
COLLECTION (3) 05/24/18
Top-Rated Explorations
https://www.oppia.org

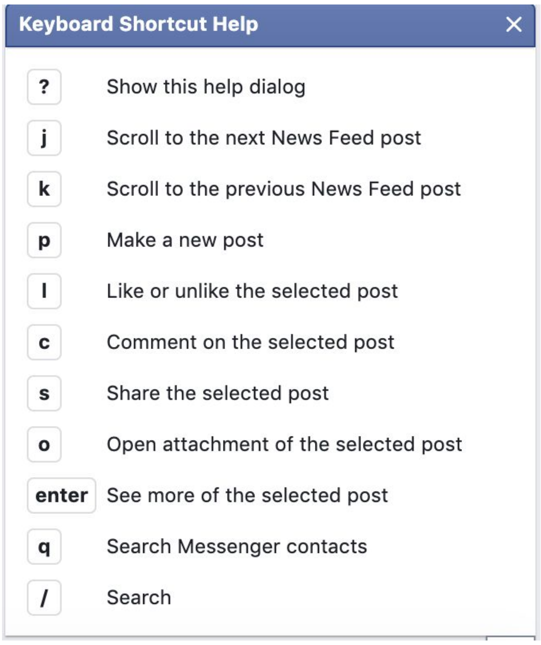
  - This approach is somewhat inflexible, especially if the focus is far away from the Skip to Main Content button. However, after implementing this shortcut by tapping S, a screen reader could go straight to the main content.
- / - Move focus to search bar in library page
  - Quick way to search up specific explorations



- C - Move focus to category bar in library page
  - Quick way to pick exploration categories



- ? - Open a list of these keyboard shortcuts while in Library Page
    - Similar to this Facebook help page

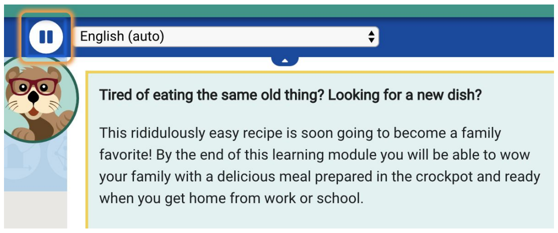


Keyboard Shortcut Help

?	Show this help dialog
j	Scroll to the next News Feed post
k	Scroll to the previous News Feed post
p	Make a new post
l	Like or unlike the selected post
c	Comment on the selected post
s	Share the selected post
o	Open attachment of the selected post
enter	See more of the selected post
q	Search Messenger contacts
/	Search

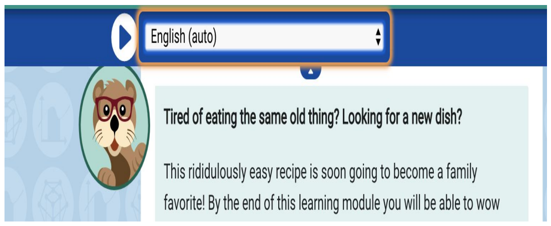
Exploration Viewer

- S - Move focus to Skip to Main Content button
- J - Move focus to continue button or interaction(multichoice, text box, etc)
    - Currently navigating the exploration viewer with a screen reader is somewhat tedious. In order to get to the continue button or interaction, you have to go through multiple elements each time. In the example below three tabs are necessary to reach the continue button



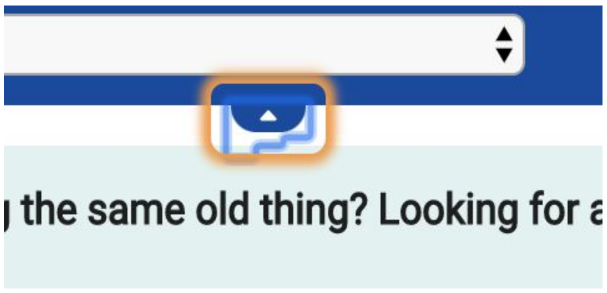
Tired of eating the same old thing? Looking for a new dish?

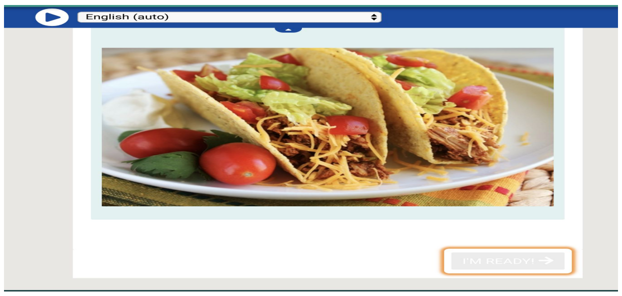
This rididuously easy recipe is soon going to become a family favorite! By the end of this learning module you will be able to wow your family with a delicious meal prepared in the crockpot and ready when you get home from work or school.



Tired of eating the same old thing? Looking for a new dish?

This rididuously easy recipe is soon going to become a family favorite! By the end of this learning module you will be able to wow


the same old thing? Looking for a



- ○ J would immediately move the focus to an interaction or the continue button
- ○ This can also help general users to quickly go through the explorations

- K - Move focus to the “previous page” button
  - Would offer similar utility as J
  - These keyboard shortcuts are heavily inspired by the Facebook J - K News Feed shortcuts
- ? - Open a list of these keyboard shortcuts while in Exploration Page

  Note: These shortcut keypresses won’t be registered in a form. “If a form element is focused the browser is capturing keypresses because that is the native behavior.” So when users focus on an input form the keyboard shortcuts won’t get executed.

  Note: These buttons won’t automatically switch to the previous or the next exploration card. Instead, it will only change the position of the focus area. Users will have to tap enter to keep going.

Navigation Shortcuts

Navigation shortcuts allow for quick traversal between pages.

These key combinations were selected to mirror the Facebook Keyboard UI navigation shortcuts. They also follow the principles of Microsoft's Guidelines For Keyboard User Interface Design for PC and macOS Human Interface Guidelines for Keyboard Shortcuts to avoid conflicts with existing commands.

- Internet Explorer for PC: Alt + #

Key combinations for Web Browsers

- Internet Explorer for PC: Alt + #
- Firefox for PC: Shift + Alt + #

- Chrome for PC: Alt + #
- Safari for Mac: control + option + #
- Firefox for Mac: control + option + #
- Chrome for Mac: control + option + #

The rationale behind the given keyboard shortcuts mapping is based off of the Oppia's navigation side-bar, facebook navigation shortcuts, and oppia's menu button.

Facebook Navigation Shortcuts

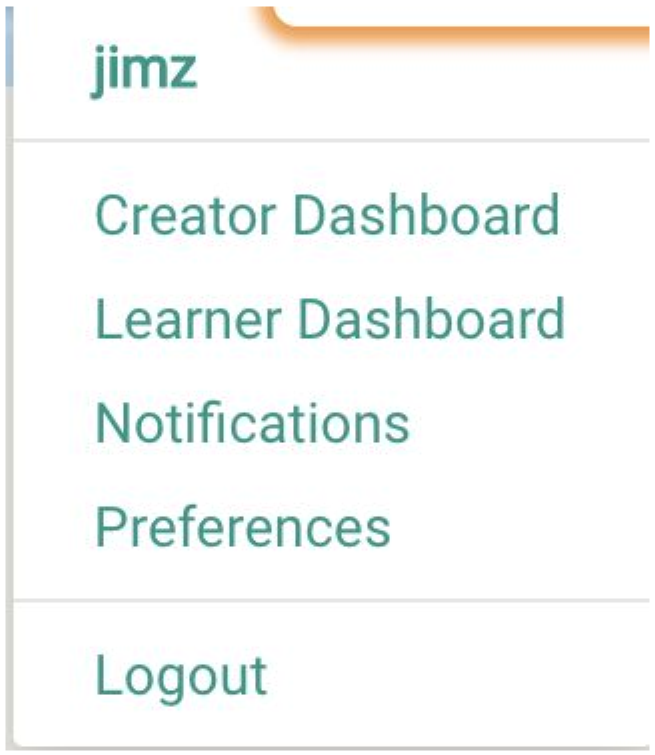
0 - Help.
1 - Home.
2 - Timeline.
3 - Friends.
4 - Notifications.
5 - Settings.
6 - Activity Log.
7 - About.
8 - Terms.

Side Bar Order

(Code is in the side-navigation-bar.directive.html)

1. Library
2. About
3. Get Started
4. Teach with Oppia
5. Donate
6. Oppia Blog
7. Contact us
8. Oppia Foundation

Oppia Menu Button



My Current Planned Order:

0 - Get Started page
- Get started is mapped to zero because facebook has 0 mapped to their help button and I think Oppia's get started page is analogous to their help page

1 - Library page
- The easiest to use shortcuts will be 1,2,3 and I definitely think Library warrants the first position as it is the hub area to all of the explorations. This logic is also seen in Oppia's navigation sidebar with the Library being the first element

2 - Creator Dashboard
- I think having a keyboard shortcut to the Creator dashboard would be very useful, this can allow creators to get a quick overview of all their explorations.

3 - Learner Dashboard
- I think having access to the learner dashboard would also be very useful, this can allow learners to get a quick overview of lessons they have shown interests in and might want to complete.

4 - Profile page
- Mirrors Facebook's Shortcuts and is on the Oppia's menu button

5 - Notification page
- Mirrors Facebook's Shortcuts and is on Oppia's menu button

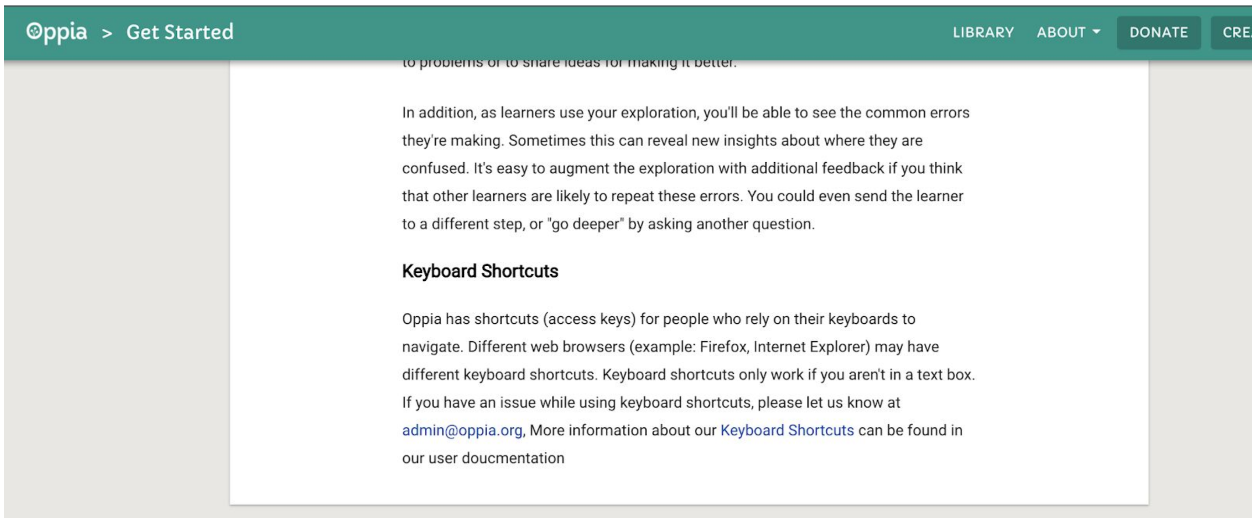
6 - Preferences page
- Mirrors the setting option on Facebook and is on Oppia's menu button

Shortcut Discovery

I believe an appropriate place for users to discover the keyboard shortcuts is in the get started page.

We could add a paragraph to mention the existence of keyboard shortcuts and link them to our user documentation page which will describe it in more detail.

Mock Ups

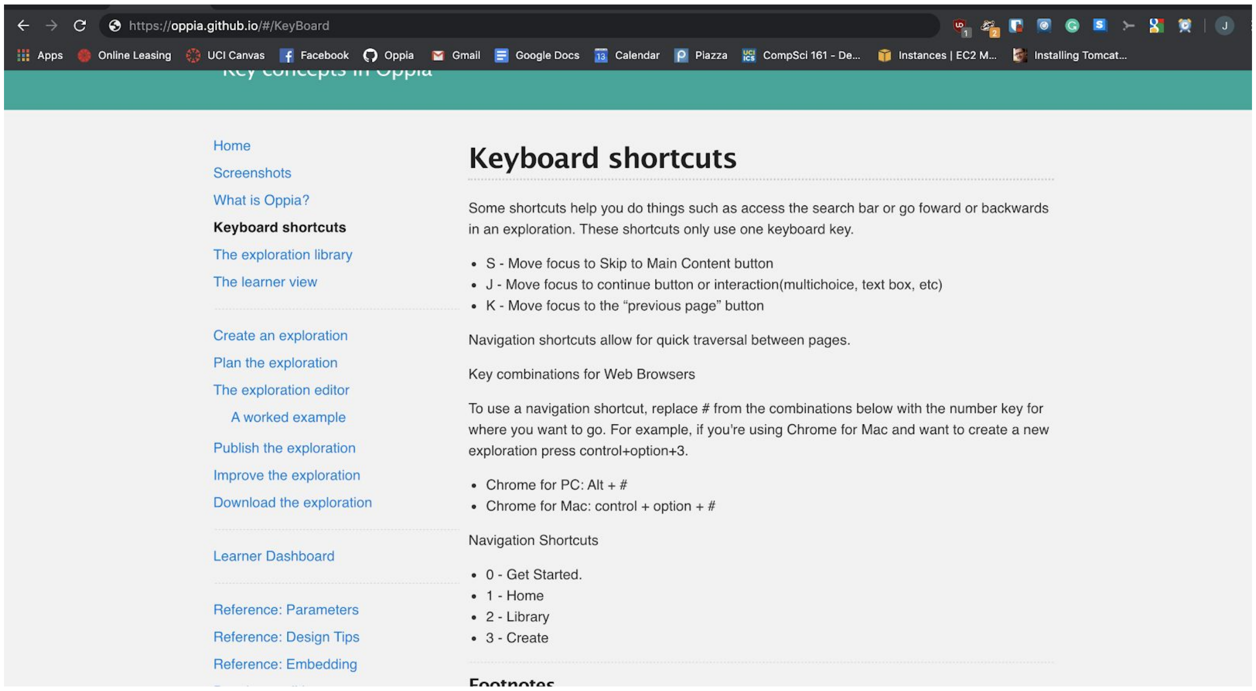


to problems or to share ideas for making it better.

In addition, as learners use your exploration, you'll be able to see the common errors they're making. Sometimes this can reveal new insights about where they are confused. It's easy to augment the exploration with additional feedback if you think that other learners are likely to repeat these errors. You could even send the learner to a different step, or "go deeper" by asking another question.

Keyboard Shortcuts

Oppia has shortcuts (access keys) for people who rely on their keyboards to navigate. Different web browsers (example: Firefox, Internet Explorer) may have different keyboard shortcuts. Keyboard shortcuts only work if you aren't in a text box. If you have an issue while using keyboard shortcuts, please let us know at admin@oppia.org, More information about our Keyboard Shortcuts can be found in our user doucmentation



Keyboard shortcuts

Some shortcuts help you do things such as access the search bar or go foward or backwards in an exploration. These shortcuts only use one keyboard key.

- S - Move focus to Skip to Main Content button
- J - Move focus to continue button or interaction(multichoice, text box, etc)
- K - Move focus to the "previous page" button

Navigation shortcuts allow for quick traversal between pages.

Key combinations for Web Browsers

To use a navigation shortcut, replace # from the combinations below with the number key for where you want to go. For example, if you're using Chrome for Mac and want to create a new exploration press control+option+3.

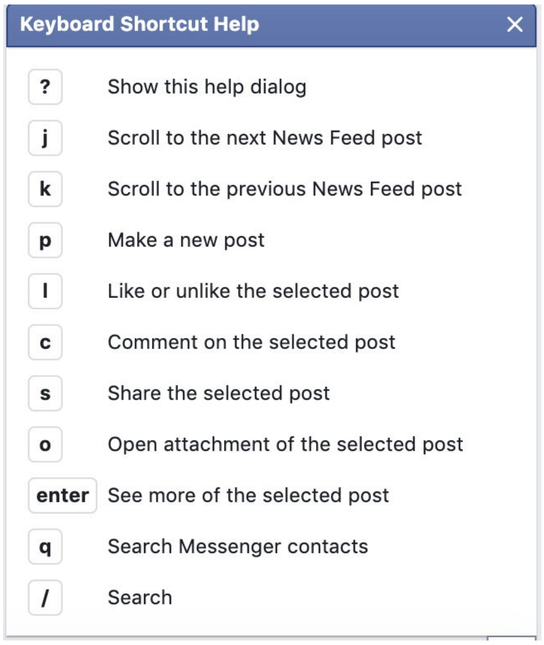
- Chrome for PC: Alt + #
- Chrome for Mac: control + option + #

Navigation Shortcuts

- 0 - Get Started.
- 1 - Home
- 2 - Library
- 3 - Create

Footnotes

If they ever press ? on the library or exploration pages, they will also bring up a set of keyboard shortcuts for that page, that will look something like this facebook popup

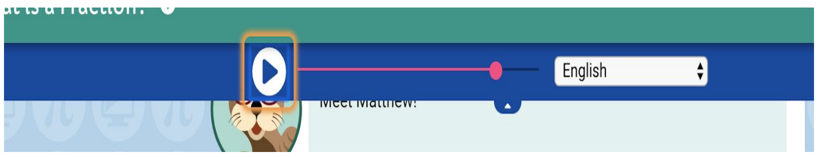


Keyboard Shortcut Help

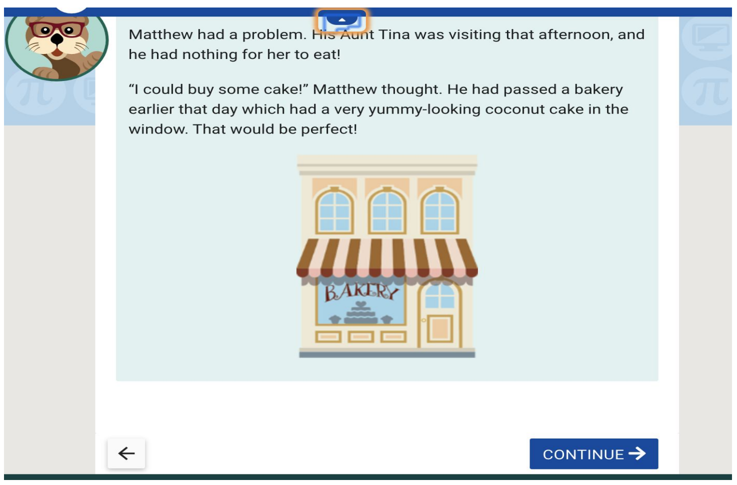
?	Show this help dialog
j	Scroll to the next News Feed post
k	Scroll to the previous News Feed post
p	Make a new post
l	Like or unlike the selected post
c	Comment on the selected post
s	Share the selected post
o	Open attachment of the selected post
enter	See more of the selected post
q	Search Messenger contacts
/	Search

Current User Journey

- To better understand the difficulties of using a screen reader on Oppia I used the screen reader ChromeVox Classic and only used the keyboard to navigate the website.
- I primarily used tab to move to next element, shift + tab to move backwards, and return/enter to select current element
- I sought to complete two tasks:
  - Start from oppia home and complete the fraction exploration
  - Start from oppia home and create a basic exploration
- Issues found in (Start from oppia home and complete the fraction exploration)
  - Library page:
    - No major issues found dealing with vague information or strange information
    - It is annoying that once you press the Skip to Main Content button and go to the explorations, the only way to access the search bar and categories bar again is to shift tabs backwards many times.
  - Explorer view page:
    - The play audio button is only announced as a button main



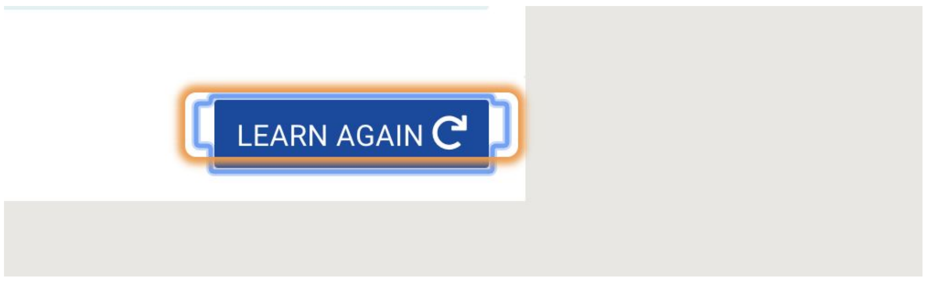
        - The back button and toggle button are only announced as button



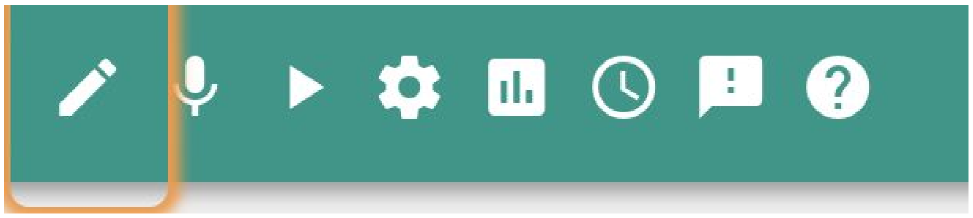
Matthew had a problem. His Aunt Tina was visiting that afternoon, and he had nothing for her to eat!

“I could buy some cake!” Matthew thought. He had passed a bakery earlier that day which had a very yummy-looking coconut cake in the window. That would be perfect!

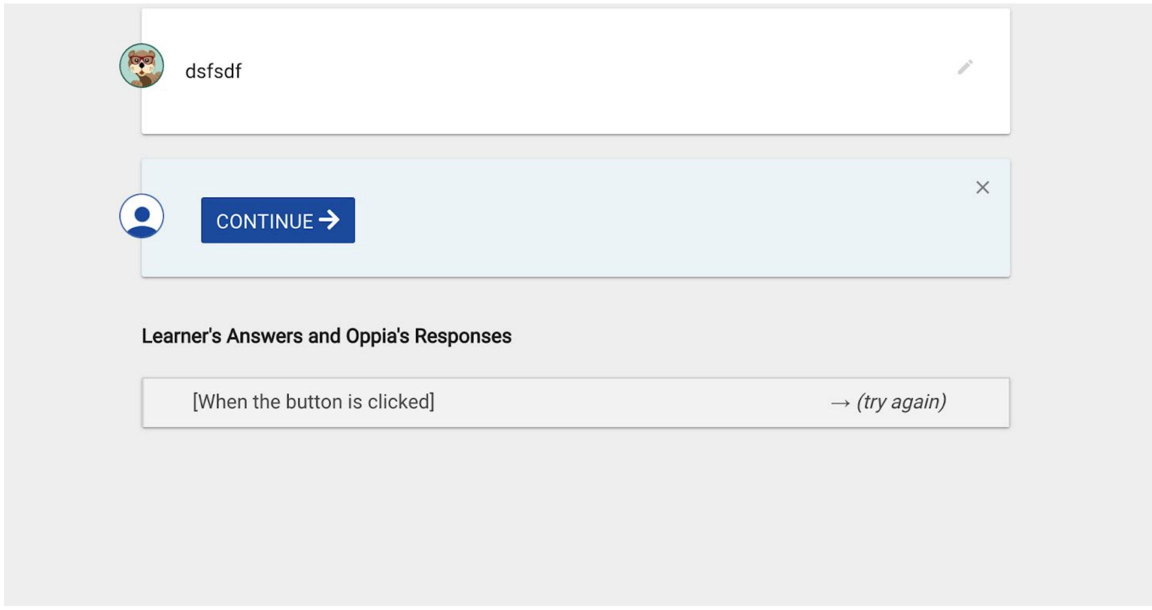
        - Learn again button is announced as button main



        - Sometimes my focus is in a weird place and I have to shift tab to get back to the interaction/lesson
        - Every time I want to continue or access the interaction I have to tab through the toggle button
- Issues found in (Start from oppia home and create a basic exploration)
    - Creator Dashboard page:
        - For a published exploration, when it tabs to the sharing section, the screen reader reads out the css information for some reason. like margin, color, etc.
        - exploration information is not very well read
    - Exploration editor page:
        - buttons on the nav bar are described to be “internal links”



        - Interaction section is described as a “button”. No information provided like “interaction button”
        - Cannot focus on button below *Learner’s Answers and Oppia’s Responses* and therefore cannot change responses to


dsfsdf
CONTINUE
Learner's Answers and Oppia's Responses
[When the button is clicked]
→ (try again)

Final Intended Screen Reader Journey

- The completed version of the project should fix all the major issues of using a screen reader, and mitigate some annoying aspects
- For example, the ambiguous button issue is one that google lighthouse caught in the exploration editor page (Figure 2) and can be fixed by providing the button with an accessible name.
- Manual testing will catch unreachable elements like the button below *Learner's Answers and Oppia's Responses*
- Keyboard shortcuts will expedite accessing important elements of a page like the search bars in the library or interactions in the Explorer View.

- The first journey (Start from oppia home and complete the fraction exploration):

  Accessing the Exploration:
  - Tab through the elements until you hit library and press enter
  - Press shortcut key - / to move to the search bar and type "fraction" and hit *enter* to search
  - If focus begins on Skip to Main Content button press enter/return, else press shortcut key - *S* to move to the Skip to Main Content button and then press *enter/return*
  - Press *tab* until you find desired exploration and press enter to begin the exploration

Completing the Exploration

- Press *tab* until you reach the start audio button which will announce itself as the “play audio button” and then press *enter/return* to begin the lesson
- Once finished listening to the audio, press the shortcut key - *J* to move the focus to the continue button or the interaction.
- Complete the interaction or press enter on the continue button to move to the next page
- If you want to go back press shortcut key - *K* to move focus to the back button if it exists and press enter to go back
- Continue like this until the end of the exploration

- The second journey(Start from oppia home and create a basic exploration):

  Accessing Exploration:

  - *tab* through elements until you hear “create button” and hit *enter* to go into the Exploration Editor page
  - press *enter* on the Skip to Main Content button to begin creating the exploration
  - *tab* to the Introduction element of the exploration and press *enter* to go into the input form and type your Introduction
  - Then *tab* to the first card of the exploration and press *enter* to go into the input form and type your topic
  - *tab* to the interaction button which announces it is creating an interaction and press *enter* to go into the interaction selection.
  - Pick the interaction you desire and press enter to use it
  - Focus on [when the button is clicked] element and change what you want the response to be
  - Repeat until satisfied
  - Shift tab back to the save draft button and save the exploration

Technical Design

This is the most important section of your proposal. Strong proposals will demonstrate familiarity with the codebase, a realistic implementation plan, and attention to detail.

Architectural Overview

Google LightHouse and Manual Audit

- Most issues that will fixed in the project will be located in the html and ts sections of core/templates/pages
  - eg. ARIA ([role]s are not contained by their required parent element)

- There are a few issues that need to be dealt with in Oppia.css as well, such as the contrast colors for certain elements.

Google LightHouse CI and Automated tests

- Use LightHouse CI to run scripts to look and audit the html urls of all of oppia’s pages

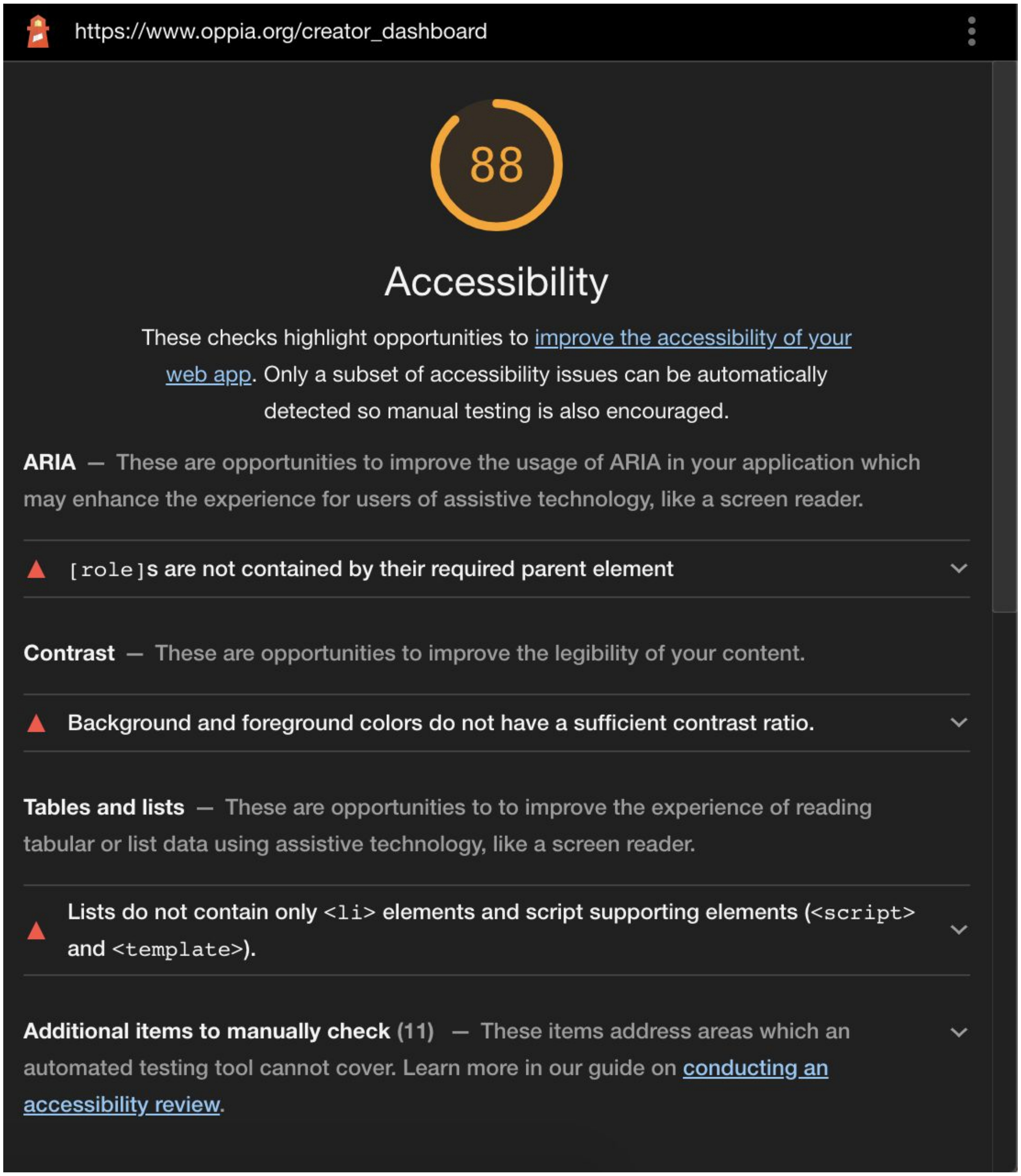
Keyboard Navigation

- Use Mousetrap to bind certain keys combinations to redirection or moving focus behaviors in library-page (Library) and exploration-player-page (Exploration Player).

Implementation Approach

Google LightHouse

1.) Run lighthouse audit to discover the baseline score and generate the issue list


https://www.oppia.org/creator_dashboard

Accessibility

These checks highlight opportunities to improve the accessibility of your web app. Only a subset of accessibility issues can be automatically detected so manual testing is also encouraged.

ARIA — These are opportunities to improve the usage of ARIA in your application which may enhance the experience for users of assistive technology, like a screen reader.

▲ `[role]`s are not contained by their required parent element

Contrast — These are opportunities to improve the legibility of your content.

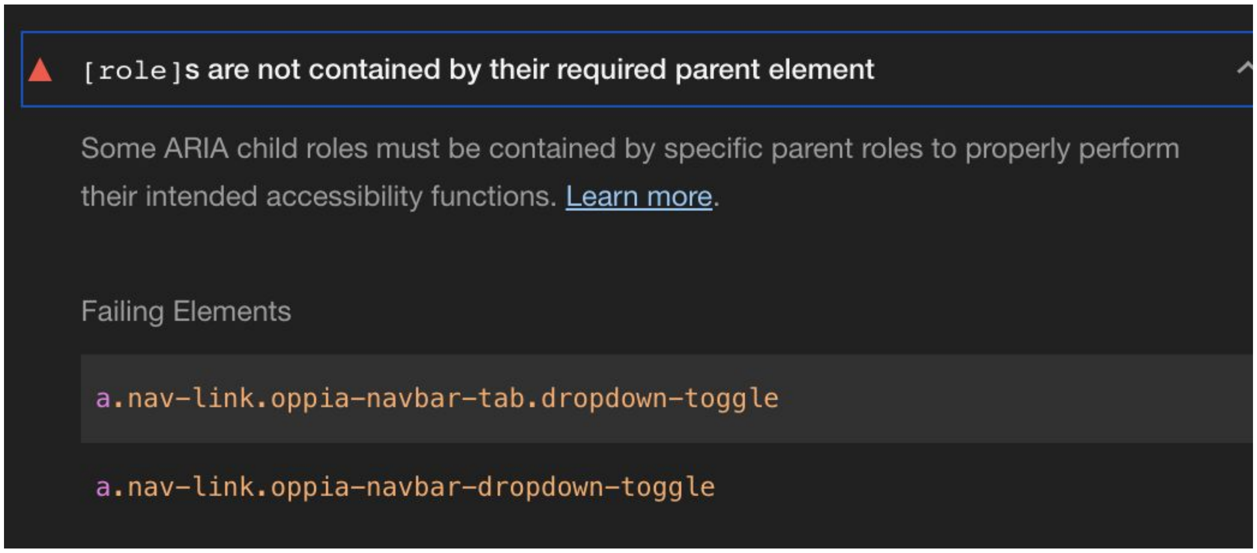
▲ Background and foreground colors do not have a sufficient contrast ratio.

Tables and lists — These are opportunities to to improve the experience of reading tabular or list data using assistive technology, like a screen reader.

▲ Lists do not contain only `<li>` elements and script supporting elements (`<script>` and `<template>`).

Additional items to manually check (11) — These items address areas which an automated testing tool cannot cover. Learn more in our guide on conducting an accessibility review.

2.) Record the failing elements and their code locations into an appropriate table eg. (Figure 2)



[role]s are not contained by their required parent element

Some ARIA child roles must be contained by specific parent roles to properly perform their intended accessibility functions. Learn more.

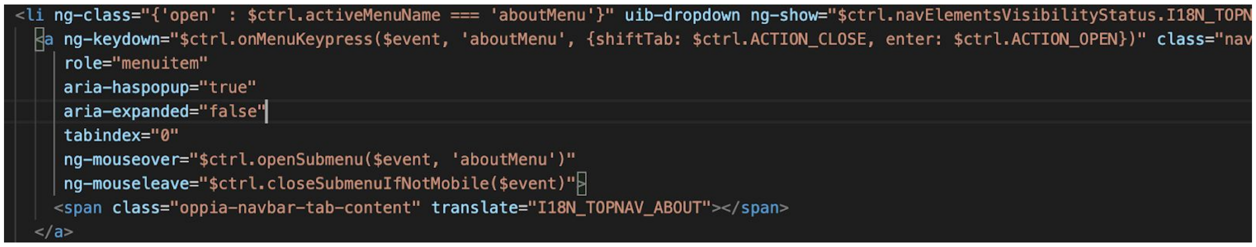
Failing Elements

```
a.nav-link.oppia-navbar-tab.dropdown-toggle
a.nav-link.oppia-navbar-dropdown-toggle
```

3.) Look into the codebase and try to find a fix based on the issue

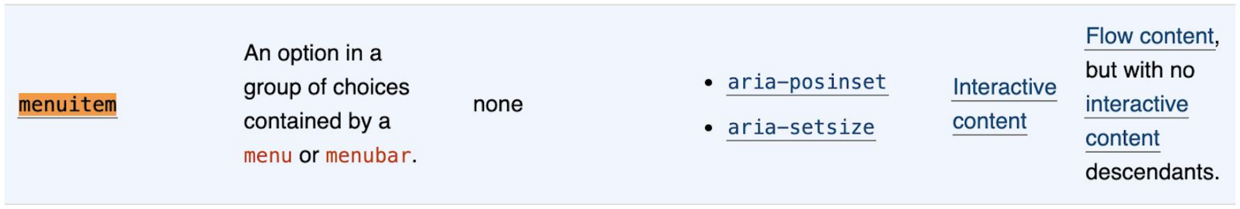
Examples:

1. ARIA ([role]s are not contained by their required parent element)



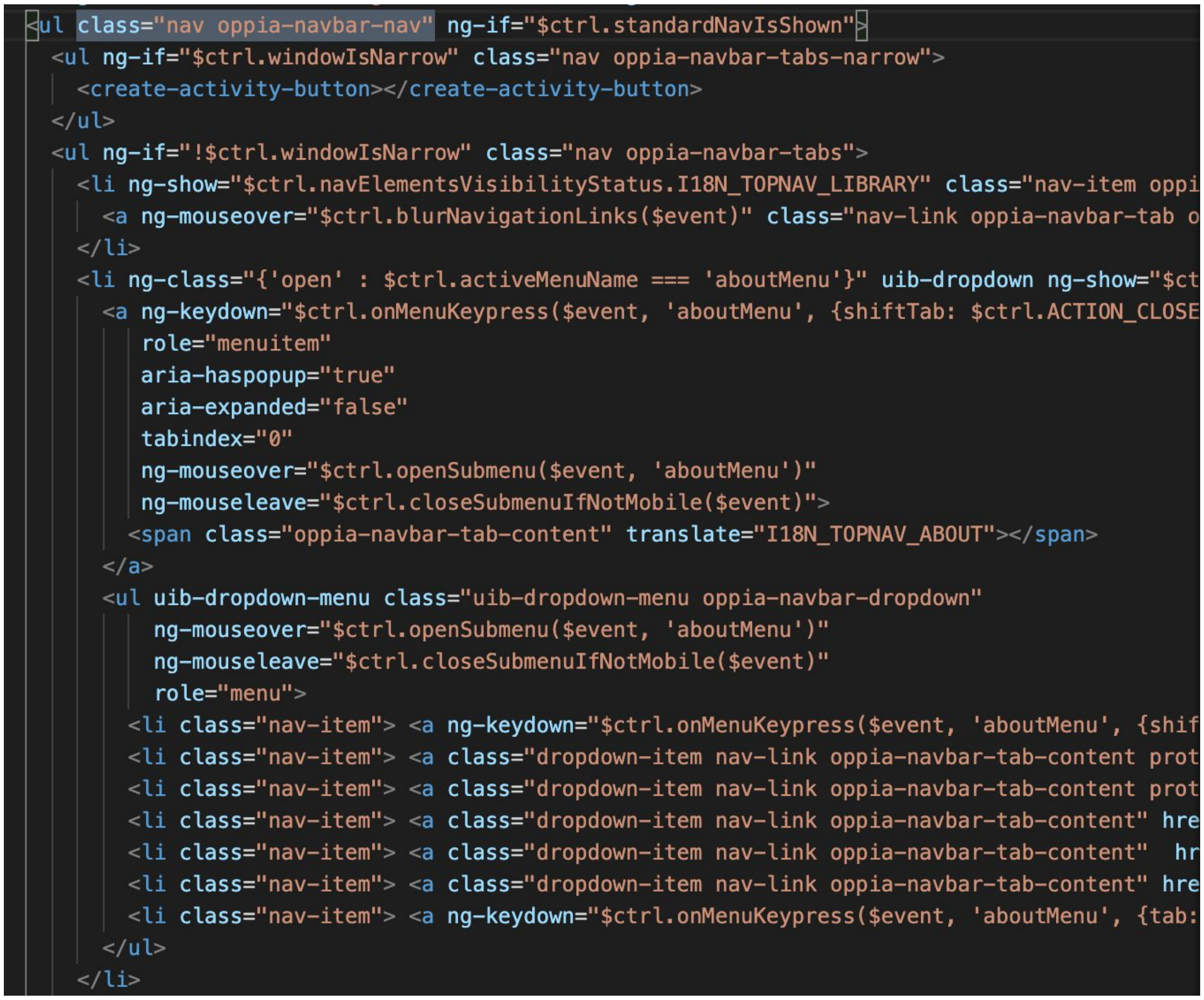
```
<li ng-class="{'open' : $ctrl.activeMenuName === 'aboutMenu'}" uib-dropdown ng-show="$ctrl.navElementsVisibilityStatus.I18N_TOPN
  <a ng-keydown="$ctrl.onMenuKeypress($event, 'aboutMenu', {shiftTab: $ctrl.ACTION_CLOSE, enter: $ctrl.ACTION_OPEN})" class="nav
     role="menuitem"
     aria-haspopup="true"
     aria-expanded="false"
     tabindex="0"
     ng-mouseover="$ctrl.openSubmenu($event, 'aboutMenu')"
     ng-mouseleave="$ctrl.closeSubmenuIfNotMobile($event)">
    <span class="oppia-navbar-tab-content" translate="I18N_TOPNAV_ABOUT"></span>
  </a>
```

The role, “menuitem”, of the child element is not contained by any a menu or a menubar (Reference: https://www.w3.org/TR/html-aria/)



`menuitem`	An option in a group of choices contained by a `menu` or `menubar`.	none	• `aria-posinset` • `aria-setsize`	Interactive content	Flow content, but with no interactive content descendants.

2. Tables and Lists



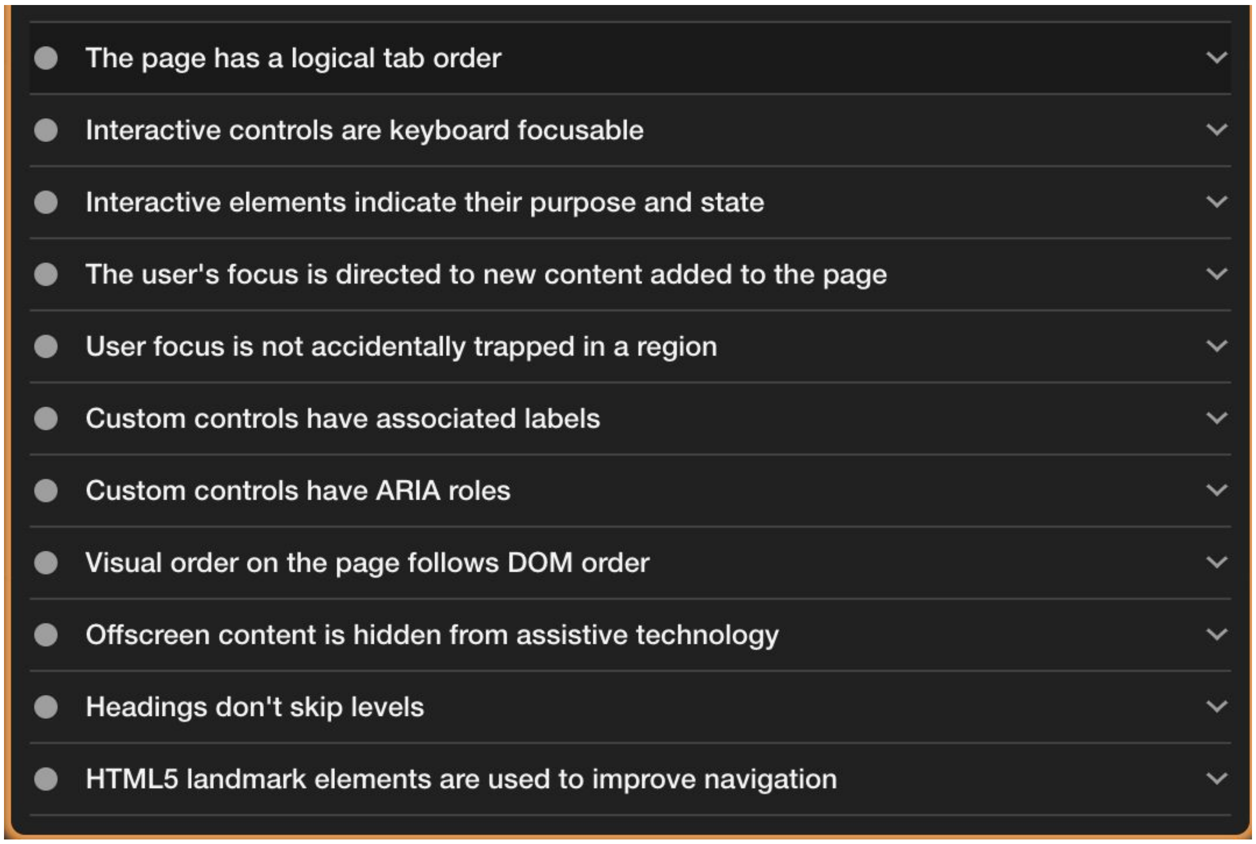
```
<ul class="nav oppia-navbar-nav" ng-if="$ctrl.standardNavIsShown">
  <ul ng-if="$ctrl.windowIsNarrow" class="nav oppia-navbar-tabs-narrow">
    <create-activity-button></create-activity-button>
  </ul>
  <ul ng-if="!$ctrl.windowIsNarrow" class="nav oppia-navbar-tabs">
    <li ng-show="$ctrl.navElementsVisibilityStatus.I18N_TOPNAV_LIBRARY" class="nav-item oppi
      <a ng-mouseover="$ctrl.blurNavigationLinks($event)" class="nav-link oppia-navbar-tab o
    </li>
    <li ng-class="{'open' : $ctrl.activeMenuName === 'aboutMenu'}" uib-dropdown ng-show="$ct
      <a ng-keydown="$ctrl.onMenuKeypress($event, 'aboutMenu', {shiftTab: $ctrl.ACTION_CLOSE
         role="menuitem"
         aria-haspopup="true"
         aria-expanded="false"
         tabindex="0"
         ng-mouseover="$ctrl.openSubmenu($event, 'aboutMenu')"
         ng-mouseleave="$ctrl.closeSubmenuIfNotMobile($event)">
        <span class="oppia-navbar-tab-content" translate="I18N_TOPNAV_ABOUT"></span>
      </a>
      <ul uib-dropdown-menu class="uib-dropdown-menu oppia-navbar-dropdown"
          ng-mouseover="$ctrl.openSubmenu($event, 'aboutMenu')"
          ng-mouseleave="$ctrl.closeSubmenuIfNotMobile($event)"
          role="menu">
        <li class="nav-item"> <a ng-keydown="$ctrl.onMenuKeypress($event, 'aboutMenu', {shif
        <li class="nav-item"> <a class="dropdown-item nav-link oppia-navbar-tab-content prot
        <li class="nav-item"> <a class="dropdown-item nav-link oppia-navbar-tab-content prot
        <li class="nav-item"> <a class="dropdown-item nav-link oppia-navbar-tab-content" hre
        <li class="nav-item"> <a class="dropdown-item nav-link oppia-navbar-tab-content"  hr
        <li class="nav-item"> <a class="dropdown-item nav-link oppia-navbar-tab-content" hre
        <li class="nav-item"> <a ng-keydown="$ctrl.onMenuKeypress($event, 'aboutMenu', {tab:
      </ul>
    </li>
```

The parent table (<ul class="nav oppia-navbar-nav") not only contains <li> elements, but also contains other elements like <ul>.

Manual Audit

After achieving a 100% accessibility score on lighthouse further manual testing is needed to catch other errors.

Eleven manual tests recommended from lighthouse will be conducted:


The page has a logical tab order

Interactive controls are keyboard focusable

Interactive elements indicate their purpose and state

The user's focus is directed to new content added to the page

User focus is not accidentally trapped in a region

Custom controls have associated labels

Custom controls have ARIA roles

Visual order on the page follows DOM order

Offscreen content is hidden from assistive technology

Headings don't skip levels

HTML5 landmark elements are used to improve navigation

with the methodologies based on the Google Accessibility Review Guide .

A few manual testing methodologies are described below:

1.) The page has a logical tab order (reference)

- Tab through the page, the order in which elements are focused should aim follow the DOM Order
    - If focus order seems wrong, rearrange DOM order or change their tabindex to make tab order more natural
    - Focus should generally be left - right, top to bottom on the page
- Check if all interactive controls are keyboard focusable
    - any control that a user can interact with should be focusable
    - If not reachable, a common fix is to replace custom controls with built-in HTML elements

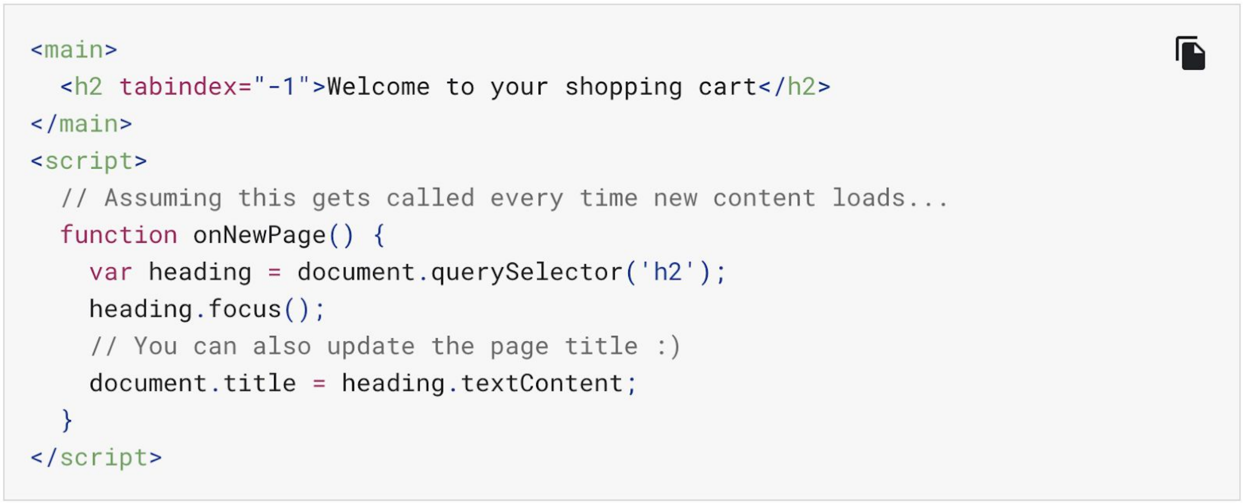
2.) Interactive elements indicate their purpose and state (reference)

- Interactive elements should indicate their state and be distinguishable from non-interactive elements.
- A visual and screen reader test should be conducted
- Visual Test:

    - Tab through the page and make sure each interactive element
        - is reachable
        - has a visual cue that shows it is interactable
- Screen Reader Test:
    - Navigate the page and check that screen reader is able to
        - announce the name of each interactive control
        - announce the role of that control
        - announce the current interactive role
        - If it can't do these things appropriate ARIA roles need to be added

3.) The user's focus is directed to new content added to the page (reference)

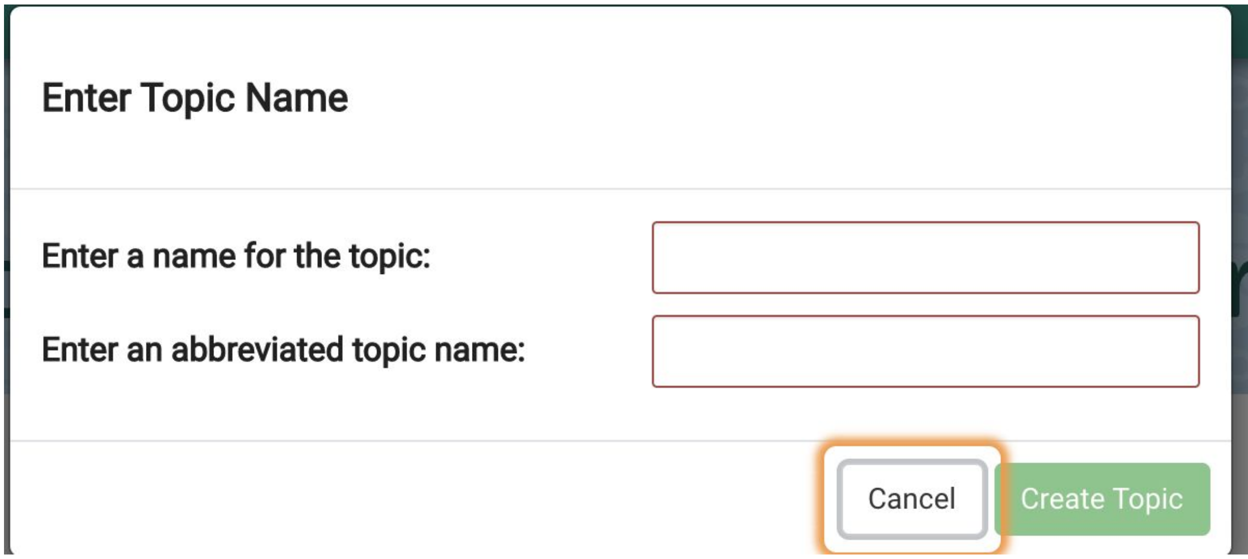
- Whenever new content is added to a page, directing the user to it is good practice because it will enable them to take action on it.
- Sighted users easily see any changes to the webpage, but users navigating with a screen reader may not know that new content has been added.
    - One potential way to fix this is find a good heading for newly loaded content and direct focus to it. An example is shown below by making the a tabindex = -1 and a calling the focus method



```
<main>
  <h2 tabindex="-1">Welcome to your shopping cart</h2>
</main>
<script>
  // Assuming this gets called every time new content loads...
  function onNewPage() {
    var heading = document.querySelector('h2');
    heading.focus();
    // You can also update the page title :)
    document.title = heading.textContent;
  }
</script>
```

4.) User focus is not accidentally trapped in a region (reference)

- If a keyboard user gets trapped on a particular element, they have no way of interacting with the page
- Test it by attempting to navigate through the page with only the keyboard. If you can't tab through all of the features you fail the test
- Common focus traps are pages that have modals, autocomplete widgets, or other widgets.
    - For modals these users should not be able to escape it by design, however having a way to escape a modal without refreshing the page should be made. Eg. having a cancel button


Enter Topic Name

Enter a name for the topic:

Enter an abbreviated topic name:

Google Chrome Automated tests

- Introduce Lighthouse CI library to the Oppia repository to set up automated accessibility tests
- After setting up Lighthouse CI, Oppia's build system should run Lighthouse on Oppias's url on every commit, asserting audits that pass, and uploading the reports for manual inspection.

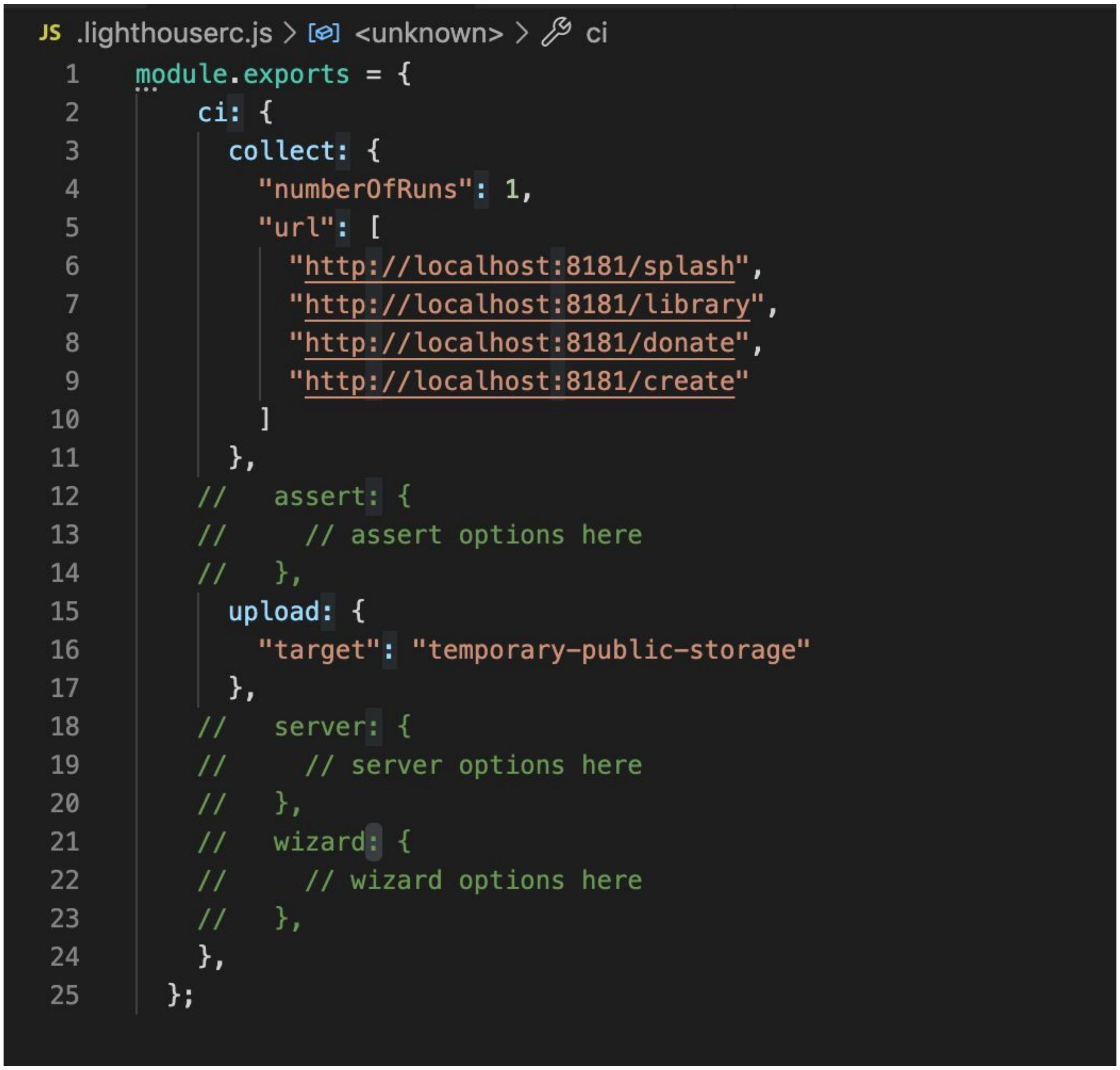
Prerequisites:

- install lighthouse-ci with command `npm install -g @lhci/cli@0.3.x`

Create lighthouse Config

Make it in the Oppia directory and name it lighthouse serc.ts.

In my example config, we have the config testing the splash, library, and donate page.



```
JS .lighthouserc.js > [@] <unknown> > ci
module.exports = {
    ci: {
      collect: {
        "numberOfRuns": 1,
        "url": [
          "http://localhost:8181/splash",
          "http://localhost:8181/library",
          "http://localhost:8181/donate",
          "http://localhost:8181/create"
        ]
      },
    //   assert: {
    //     // assert options here
    //   },
      upload: {
        "target": "temporary-public-storage"
      },
    //   server: {
    //     // server options here
    //   },
    //   wizard: {
    //     // wizard options here
    //   },
    },
  };
```

Explanation of Lighthouse Config

Collect: defines the URLS to be tested and other audit configurations like number of runs and connecting to puppeteer scripts. (Collect documentation)

- Once we start fixing the accessibility issues we will increase numberOfRuns to avoid flakey tests, but during the creation and testing of the config file we only have one run
- Note: We can also collect URLS from the proxy server that will deal with the performance compression issue that the performance team is working on.

Assert: The results of any audit can be asserted and can be used to prevent regressions (Assertion documentation)

- There are three levels of assertions
    - `off` - The audit result will not be checked. If an audit is not found in the `assertions` object, it is assumed to be `off`.
    - `warn` - The audit result will be checked, and the result will be printed to stderr, but failure will not result in a non-zero exit code.
    - `error` - The audit result will be checked, the result will be printed to stderr, and failure will result in a non-zero exit code.

```
{
  "assertions": {
    "audit-id-1": ["warn", {"maxNumericValue": 4000}],
    "audit-id-2": ["error", {"minScore": 0.8}],
    "audit-id-3": ["warn", {"maxLength": 0}]
  }
}
```

  - There are also presets that can be used [lighthouse:all, lighthouse:recommended, lighthouse:no-pwa]
    - All audits will be asserted in lighthouse:all
    - A subset of industry recommended audits will be asserted in lighthouse:recommended
- The assertions will be in the warn state during the development, and slowly be switched to error as each audit is fixed on all Oppia webpages.
- Alternative 1: We can use a budget.json file instead of the lighthouse-ci assert functionality (documentation)

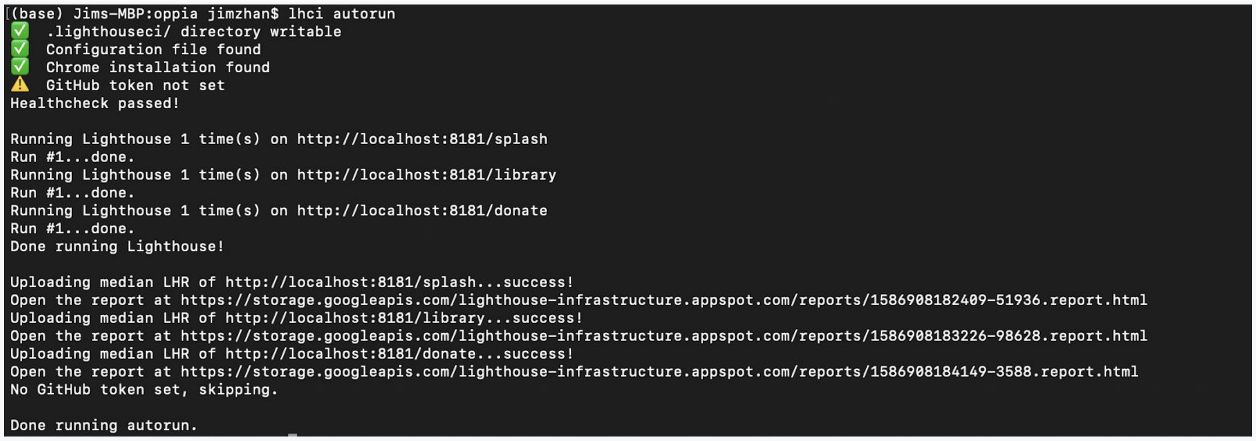
Upload: saves the runs in `.lighthouseci/` folder to desired target (Upload documentation)

- Currently config will store the results in a temporary public url on the google cloud, which expires after 7 days.
- Alternative 1: If storing on public url is not acceptable, we can also choose to not upload it at any location and just read the issues on the command line or the local .lighthouseci/ folder
- Alternative 2: We could set up an LHCI server to store the results instead

Wizard and Server: are not needed at the moment

Commands to run lighthouse audits manually

1.) Run oppia scripts: python -m scripts.start
2.) Run lighthouse: `lhci autorun`

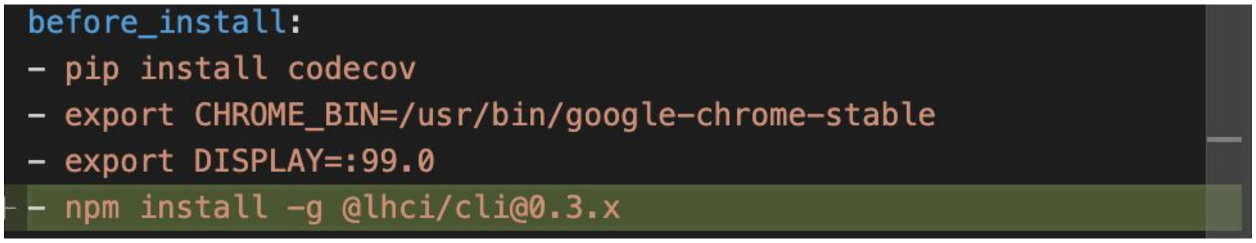


```
(base) Jims-MBP:oppia jimzhan$ lhci autorun
✅  .lighthouseci/ directory writable
✅  Configuration file found
✅  Chrome installation found
⚠️  GitHub token not set
Healthcheck passed!

Running Lighthouse 1 time(s) on http://localhost:8181/splash
Run #1...done.
Running Lighthouse 1 time(s) on http://localhost:8181/library
Run #1...done.
Running Lighthouse 1 time(s) on http://localhost:8181/donate
Run #1...done.
Done running Lighthouse!

Uploading median LHR of http://localhost:8181/splash...success!
Open the report at https://storage.googleapis.com/lighthouse-infrastructure.appspot.com/reports/1586908182409-51936.report.html
Uploading median LHR of http://localhost:8181/library...success!
Open the report at https://storage.googleapis.com/lighthouse-infrastructure.appspot.com/reports/1586908183226-98628.report.html
Uploading median LHR of http://localhost:8181/donate...success!
Open the report at https://storage.googleapis.com/lighthouse-infrastructure.appspot.com/reports/1586908184149-3588.report.html
No GitHub token set, skipping.

Done running autorun.
```

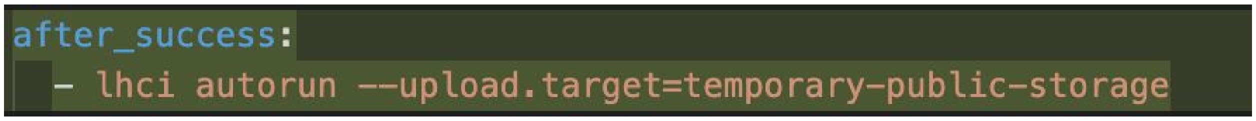
Create a Github Action

Github action will be used to integrate the lighthouse check into the oppia CI process. Where every pull request will have a lighthouse check on each webpage

1.) Modify .travis.yml

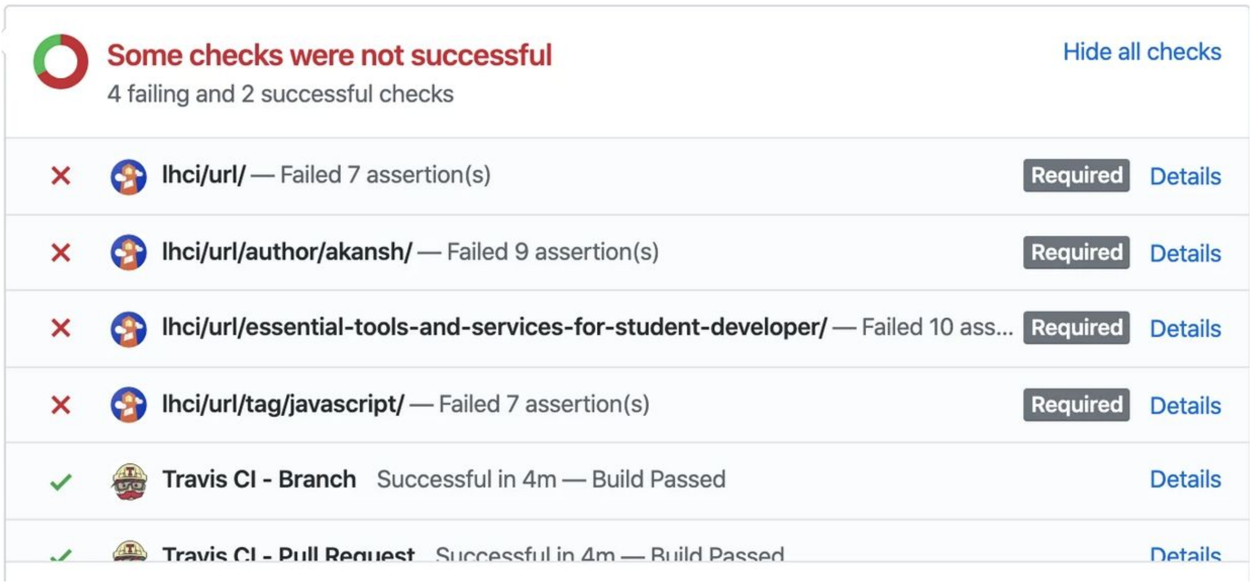


```
before_install:
- pip install codecov
- export CHROME_BIN=/usr/bin/google-chrome-stable
- export DISPLAY=:99.0
- npm install -g @lhci/cli@0.3.x
```



```
after_success:
  - lhci autorun --upload.target=temporary-public-storage
```

2.) Go to [Lighthouse CI APP LINK]()
3.) Authorize
4.) Copy the LHCI_GITHUB_APP_TOKEN token on Authorisation page
5.) Add the LHCI_GITHUB_APP_TOKEN to Oppia repository settings



Some checks were not successful
4 failing and 2 successful checks

Hide all checks

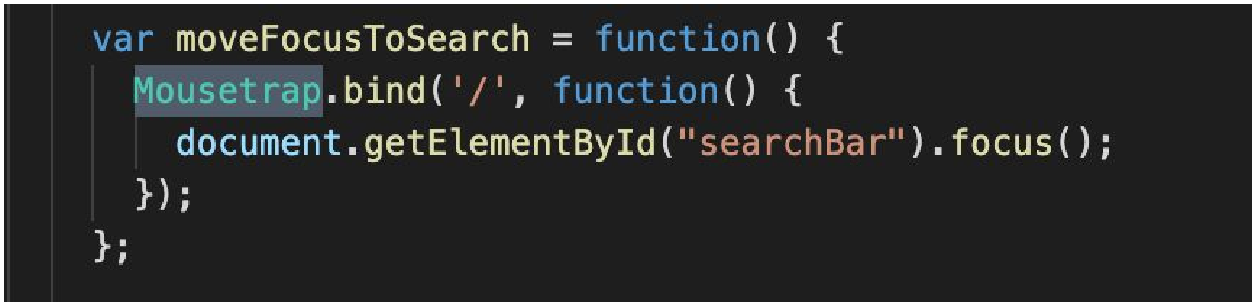
✗	lhci/url/ — Failed 7 assertion(s)	Required	Details
✗	lhci/url/author/akansh/ — Failed 9 assertion(s)	Required	Details
✗	lhci/url/essential-tools-and-services-for-student-developer/ — Failed 10 ass...	Required	Details
✗	lhci/url/tag/javascript/ — Failed 7 assertion(s)	Required	Details
✓	Travis CI - Branch Successful in 4m — Build Passed		Details
✓	Travis CI - Pull Request Successful in 4m — Build Passed		Details

Keyboard Shortcuts

- **Action shortcuts** are linked to specific pages, In this case they will be bound to the Library and Exploration Player page

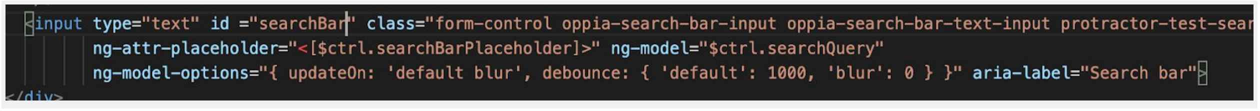
Example Workflow: Creating Library keyboard shortcut backslash “/”

- Create a function to shift the focus to the search bar in the library-page.directive.ts



```
var moveFocusToSearch = function() {
  Mousetrap.bind('/', function() {
    document.getElementById("searchBar").focus();
  });
};
```

- Go to search-bar.directive.html
- Make sure the SearchBar has an id, in this case it did not and we added one

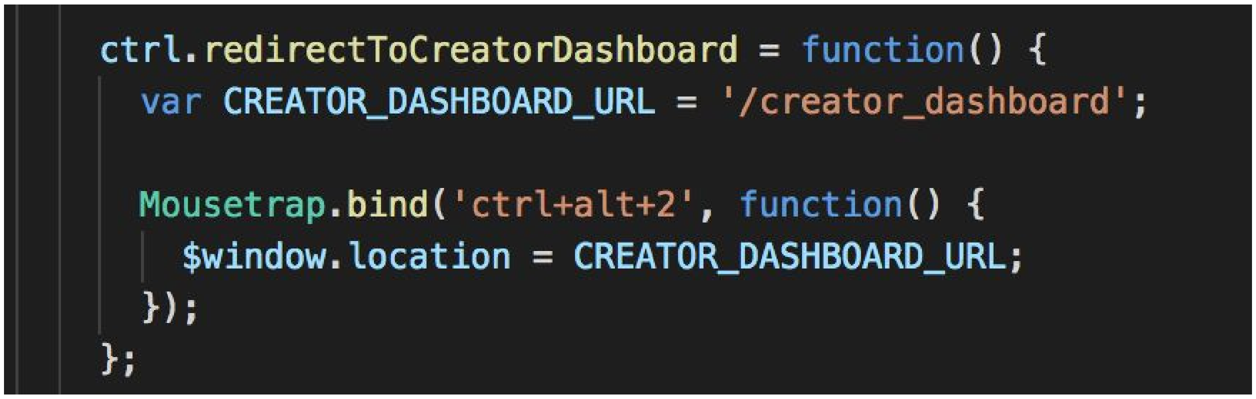


```
<input type="text" id ="searchBar" class="form-control oppia-search-bar-input oppia-search-bar-text-input protractor-test-sear
        ng-attr-placeholder="<[$ctrl.searchBarPlaceholder]>" ng-model="$ctrl.searchQuery"
        ng-model-options="{ updateOn: 'default blur', debounce: { 'default': 1000, 'blur': 0 } }" aria-label="Search bar">
</div>
```

- **Navigation shortcuts** will be available on all pages

Example Workflow: Creating navigation shortcut to creator_dashboard

- Use mouse trap to go to the creator_dashboard



```
ctrl.redirectToCreatorDashboard = function() {
  var CREATOR_DASHBOARD_URL = '/creator_dashboard';

  Mousetrap.bind('ctrl+alt+2', function() {
    $window.location = CREATOR_DASHBOARD_URL;
  });
};
```

- I'm unsure of where to put these shortcuts and wondering if there is a way to have every page have access to the navigation shortcuts without putting this code on every single page?

Shortcut discovery
- Getting_started page and user documentation will need to add a new section about these shortcuts and provide support

Testing Approach

E2E tests for keyboard shortcuts

Add multiple end-to-end test cases for ExplorationPlayerPage.js and LibraryPage.js. The test cases should ensure the following:
- Users can use keyboard shortcuts to redirect to other pages successfully
- Users can use keyboard shortcuts to move focus on some elements as expected

By adding these test cases, it will be ensured that newly added features won't break the workflow in the future.

LibraryPage.js.

1.) E2E test for keyboard shortcut **S**
- Press S on Library page
- Check if focus is on skip to main content button
- Move the focus away with tab
- Check if focus is not on skip to main content button
- Press S on Library page
- Check if focus is on skip to main content button

2.) E2E test for keyboard shortcut **/**
- Press / on Library page
- Check if focus is on search bar
- Move the focus away with tab

- Check if focus is not on search bar
- Press / on Library page
- Check if focus is on search bar

3.) E2E test for keyboard shortcut **C**
- Press C on Library page
- Check if focus is on category bar
- Move the focus away with tab
- Check if focus is not on category bar
- Press C on Library page
- Check if focus is on category bar

4.) E2E test for keyboard shortcut **?**
- Press ? on Library page
- Check if there is a keyboard shortcut box

ExplorationPlayerPage.js

5.) E2E test for keyboard shortcut **S**
- Go to an exploration
- Press S on Library page
- Check if focus is on skip to main content button
- Move the focus away with tab
- Check if focus is not on skip to main content button
- Press S on Library page
- Check if focus is on skip to main content button

6.) E2E test for keyboard shortcut **J**
- Go to an exploration
- Press J in the Exploration player page
- Check if focus is on a continue button or an interaction
- Move the focus away with tab
- Press J in the Exploration player page
- Check if focus is on a continue button or an interaction

7.) E2E test for keyboard shortcut **K**
- Go to an exploration
- Press K in the Exploration player page
- Check if focus is on a back button or an interaction
- Move the focus away with tab
- Press K in the Exploration player page
- Check if focus is on a back button or an interaction

8.) E2E test for keyboard shortcut **?**
- Press ? on Exploration player page
- Check if there is a keyboard shortcut box

All Pages (Windows navigation)

9.) E2E test for keyboard shortcut **alt + 0**

- Press alt + 0
- Check if you load into the get started page

10.) E2E test for keyboard shortcut **alt + 1**

- Press alt + 1
- Check if you load into the library page

11.) E2E test for keyboard shortcut **alt + 2**

- Press alt + 2
- Check if you load into the creator dashboard page

12.) E2E test for keyboard shortcut **alt + 3**

- Press alt + 3
- Check if you load into the learner dashboard page

13.) E2E test for keyboard shortcut **alt + 4**

- Press alt + 4
- Check if you load into the profile page

14.) E2E test for keyboard shortcut **alt + 5**

- Press alt + 5
- Check if you load into the notification page

15.) E2E test for keyboard shortcut **alt + 6**

- Press alt + 6
- Check if you load into the preferences page

All Pages (Mac navigation)

15.) E2E test for keyboard shortcut **ctrl + options + 0**

- Press ctrl + options + 0
- Check if you load into the get started page

16.) E2E test for keyboard shortcut **ctrl + options + 1**

- Press ctrl + options + 1
- Check if you load into the library page

17.) E2E test for keyboard shortcut **ctrl + options + 2**

- Press ctrl + options + 2
- Check if you load into the creator dashboard page

18.) E2E test for keyboard shortcut **ctrl + options + 3**

- Press ctrl + options + 3
- Check if you load into the learner dashboard page

19.) E2E test for keyboard shortcut **ctrl + options + 4**

- Press ctrl + options + 4
- Check if you load into the profile page

20.) E2E test for keyboard shortcut **ctrl + options + 5**

- Press ctrl + options + 5
- Check if you load into the notification page

21.) E2E test for keyboard shortcut **ctrl + options + 6**

- Press ctrl + options + 6
- Check if you load into the preferences page

Milestones

Each big milestone is 4-week long, following the GSoC 2020 timeline. Each of them is divided into 6 sub-milestones. Each sub-milestone contains one major objective, and it is expected to take five days to complete. At the end of each milestone, there will be a buffer period to make sure there is extra time for testing and objectives can be completed before the official deadline.

Milestone 1 (6/1 - 6/29)

Key Objective:

- Set up LightHouse CI for automated testing
- For the following web pages: Admin, Classroom, About, Collection_editor, Collection_player, Contact, Console_errors, Creator_dashboard, Community_dashboard, Delete_account, Donate, Email_dashboard, Email_dashboard_Result
    - Achieve Google Chrome Lighthouse audit report scores of 100
    - Complete manual accessibility review (Review Checklist)
- Develop and refine a robust manual testing methodology to catch accessibility issues that cannot be checked automatically by Lighthouse, and create documentation for this, so that the QA process includes a full suite of manual tests for accessibility starting from July onwards.

No.	Description of PR	Prereq PR numbers	Target date for PR submission	Target date for PR to be merged
1.1	Connect LightHouse CI to Oppia		6/5	6/8
1.2	Score 100% in both Chrome and manual accessibility audits on **Admin, Classroom, About, Collection_Editor pages**		6/10	6/13
1.3	Score 100% in both Chrome and manual accessibility audits on **Collection Player, Contact, Console errors pages**		6/15	6/18
1.4	Score 100% in both Chrome and manual accessibility audits on the **Creator Dashboard, Community, Delete_Account pages**		6/20	6/23

1.5	Score 100% in both Chrome and manual accessibility audits on the **Donate, Email_Dashboard, Email_Dashboard_Result pages** Create documentation for manual tests for monthly release testers		6/25	6/28
1.6	Buffer period. Do manual testing to look for bugs and fix them.		6/28	6/30

Milestone 2 (6/29 - 7-27)

Key Objective:

- For all web pages listed above in M1, as well as: Error, Exploration_Editor, Exploration_Player, Get_Started, Landing, Learner_Dashboard, Library, Maintenance, Moderator, Notifications_Dashboard, Pending_Account_Deletion, Practice_session, Privacy.
    - Achieve Google Chrome Lighthouse audit report scores of 100
    - Complete manual accessibility review (Review Checklist)
- Users can use keyboard shortcuts to navigate to all critical pages (Get Started, Library, Creator Dashboard, Learner Dashboard, Profile, Notification, Preferences)
- Users can use keyboard shortcuts for critical actions on the Library page and Exploration Viewer. Users also have a way of discovering these keyboard shortcuts

No.	**Description of PR**	**Prereq PR numbers**	**Target date for PR submission**	**Target date for PR to be merged**
2.1	Got 100% score in both Chrome and manual accessibility audits on the **Error, Exploration_Editor, Exploration_Player, Get_Started, Landing pages**		7/3	7/6
2.2	Score 100% in both Chrome and manual accessibility audits on **Learner_Dashboard, Library, Maintenance, Moderator pages**.		7/8	7/11
2.3	Score 100% in both Chrome and manual accessibility audits on **Notifications_Dashboard,**		7/13	7/16

	Pending_Account_Deletion, Practice_Session, Privacy pages.			
2.4	All action keyboard shortcuts for the **Library Page** and **Exploration Player** and their e2e test are added.		7/18	7/21
2.5	All navigation keyboard shortcuts and their e2e test are added. Implement keyboard shortcut discovery.		7/23	7/26
2/6	Buffer period. Do manual testing to look for bugs and fix them.		7/27	7/30

Milestone 3 (7/28 - 8/24)

Key Objective:

- For all web pages in the Oppia web application, including all webpages listed above in M1 and M2, as well as: Preferences, Profile, Review_Test, Signup, Skill_Editor, Splash, Stewards, Story_Editor, Story_Viewer, Subtopic_Viewer, Teach, Terms, Thanks, Topic_Editor, Topics_and_Skills_Dashboard, Topic_Viewer, and any other pages.
    - Achieve Google Chrome Lighthouse audit report scores of 100
    - Complete manual accessibility review (Review Checklist)

No.	**Description of PR**	**Prereq PR numbers**	**Target date for PR submission**	**Target date for PR to be merged**
3.1	Score 100% in both Chrome and manual accessibility audits on **Preferences, Profile, Review_Test, Signup pages**.		8/1	8/4
3.2	Score 100% in both Chrome and manual accessibility audits on **Skill_Editor, Splash, Stewards, Story_Editor pages**.		8/6	8/9
3.3	Score 100% in both Chrome and manual accessibility audits on **Story_Viewer, Subtopic_Viewer, Teach, Terms pages**.		8/11	8/14
3.4	Score 100% in both Chrome and manual accessibility audits on the **Thanks, Topic_Editor, Topics and Skills**		8/16	8/19

	dashboard, Topic_Viewer Pages			
3.5	Buffer period. Do manual testing to look for bugs and fix them.		8/21	8/24
3.6	Additional Buffer period. Do manual testing to look for bugs and fix them.		8/24	8/26

Evaluation Period (8/24 - 8/31)

1. Final wrap up. Go through the entire project and fix any bugs or conflictions.
2. Complete final report and turn in the final product.

Optional Sections

Future Work

List here any optional future work that isn't in the scope of the GSoC project, but that could be taken up after the current project is complete.

Additional Project-Specific Considerations

Security

Does this feature provide any new opportunities for users to gain unauthorized access to user data or otherwise impact other users' experience on the site in a negative way?

Accessibility (if user-facing)

List the ways in which your feature is accessible for a wide audience of users, including users who have partial or full blindness, deafness, or any other disorder which may inhibit their ability to use the feature or platform.

Documentation Changes

List the changes (if any) that should be made to the Oppia wiki to help support this feature or provide additional guidance to maintainers of Oppia instances. Types of projects that should include documentation changes:

- *Complex migrations that require custom steps*
- *Infrastructural changes that require updates to the developer-facing instructions (e.g. new APIs added to our Protractor testing library that can help improve the reliability of e2e tests)*