

Google Summer of Code 2022

Migrate Away From Protractor

Shivam Jha

About You

My name is Shivam Jha. I am a second-year undergraduate student at the Indian Institute of Information Technology, Design and Manufacturing Jabalpur pursuing Computer Science Engineering as my major.

What project are you applying for?

Migrate Away From Protractor.

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

I started contributing to Oppia in September 2021. The reason why I started was to explore the open-source community and interact with amazing people around the globe. Contributing to Oppia helped me in learning Angular, AngularJS introduced me to the unit and end-to-end testing. All the amazing mentors present here were readily available for help whenever I needed it. The noble motto of Oppia always inspires and motivates me to push my limits.

While working with the migration team over the past months, I was able to learn a lot about Angular and Frontend unit testing, End to End Testing and have become very much comfortable with the codebase now. It would be a great learning experience to complete this project.

Prior Experience

I have been contributing to Oppia for the last 5-6 months as a part of the migration team. I have been doing web development for the past 2 years and made some projects listed below:

1. [Food Ordering Website](#) using handlebars, sass as Frontend and node, express as backend, MySQL as database.
2. [Shopping List Website](#) using Angular 12+ as frontend and firebase as a backend.
3. Contributed to Public Lab, fixes a couple of frontend bugs, PRs [link](#).

4. As a member of the Migration team in Oppia, I have been regularly contributing to migration projects since November 2021.
5. At present I have **30 PRs** merged.
6. I am also a member of the web welfare team where we help and welcome new contributors.

Links to previous PRs.

- Migrates Select Skill Difficulty Modal and other few components ([#14565](#))
- Migrates Question misconception editor ([#14672](#))
- Migrates Concept Card Editor and Skill Preview Modal ([#14939](#))
- Migrates Add or Update Solution Modal ([#14771](#))
- Scroll to the top while navigating to the static pages ([#15010](#))

Complete list of my PRs can be found [here](#).

I did volunteer work in writing frontend tests and resolving the E2E Test in [Schema-based-editor PR](#), Srijan Reddy Vasa my PRs can be found [here](#)

I have also opened 8 issues regarding frontend bugs, which can be found [here](#)

Project Size

I am applying for a large project (~350 hours).

Project Timeframe

13 June 2022 to 12 September 2022

Contact info and timezone(s)

Primary Email and Hangout: 20bcs206@iiitdmj.ac.in

Secondary Email: sssvjha@gmail.com

Contact Number: +91 7011082573

GitHub: @ShivamJhaa

Time Zone: Kolkata, India (GMT+5:30)

Preferred mode of Communication: Hangout, Discord, Email.

Time Commitment

- I would be working on the GSoC project for 15 weeks, from 13th June to 8th October 2022.
- I am having my summer vacation from 5th May to 17th July so I would be able to devote **~30 hr/week** during this time and after that, I would be able to devote **~20 hr/week** which may increase if the need arises.

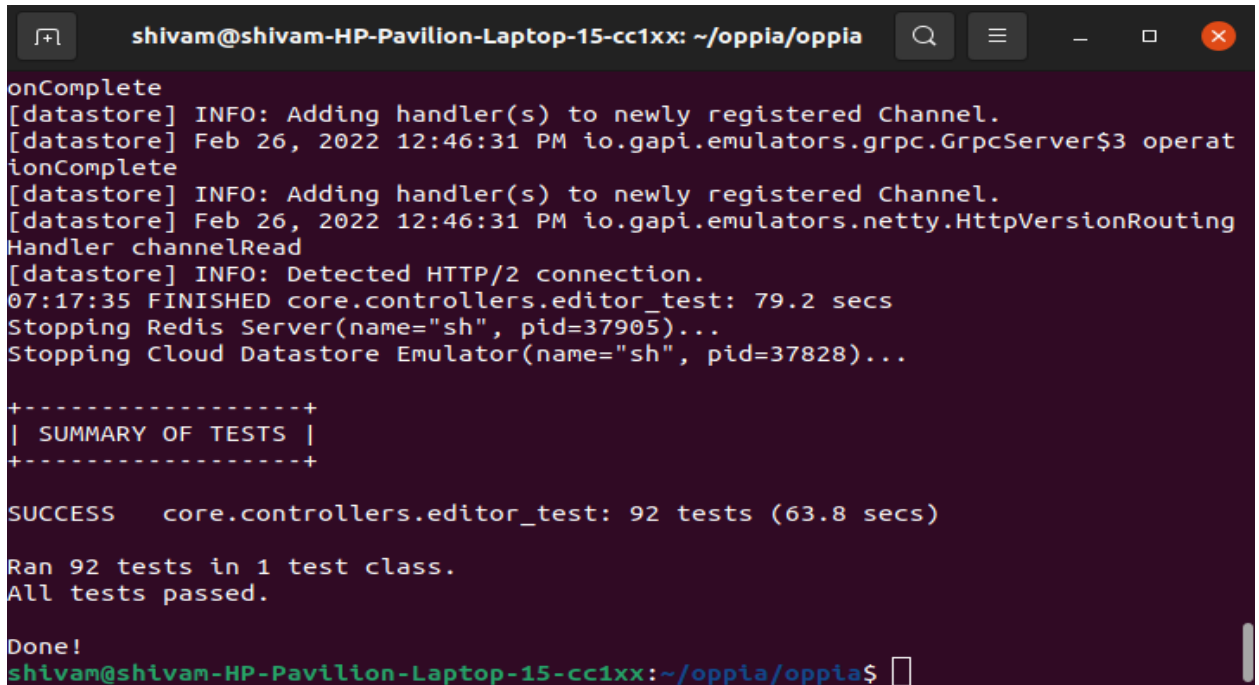
S. No	Dates	Days (Total)	Time Commitment
1.	13th June - 19th June	Mon-Sat (6)	5 hr/day (Mon-Sat)
2.	20th June - 26th June	Mon-Sat (6)	5 hr/day (Mon-Sat)
3.	27th June - 3rd July	Mon-Sat (6)	5 hr/day (Mon-Sat)
4.	4th July - 10th July	Mon-Sat (6)	5 hr/day (Mon-Sat)
5.	11th July - 17th July	Mon-Sat (6)	5 hr/day (Mon-Sat)
6.	18th July - 24th July	Mon-Sat (6)	3 hr/day (Mon-Sat)
7.	25th July - 31st July	Mon-Sat (6)	3 hr/day (Mon-Sat)
8.	1st Aug - 7th Aug	Mon-Sat (6)	3 hr/day (Mon-Sat)
9.	8th Aug - 14th Aug	Mon-Sat (6)	3 hr/day (Mon-Sat)
10.	15th Aug - 21st Aug	Mon-Sat (6)	3 hr/day (Mon-Sat)
11.	22nd Aug - 28th Aug	Mon-Sat (6)	3 hr/day (Mon-Sat)
12.	29th Aug - 3rd Sept	Mon-Sat (6)	4 hr/day (Mon-Sat)
13.	4th Sept - 12th Sept	Mon-Sat (6)	4 hr/day (Mon-Sat)
14.	13th Sept - 19th Sept	Mon-Sat (6)	4 hr/day (Mon-Sat)
15.	20th Sept - 26th Sept	Mon-Sat (6)	4 hr/day (Mon-Sat)
16.	27th Sept - 3rd Oct	Mon-Sat (6)	4 hr/day (Mon-Sat)

Estimated Total Working Days: 90

Estimated Hours: ~350 hours (This may change as per requirements)

Essential Prerequisites

- I am able to run a single backend test target on my machine.

A terminal window screenshot showing the execution of a test. The window title is 'shivam@shivam-HP-Pavilion-Laptop-15-cc1xx: ~/oppia/oppia'. The output shows log messages from the datastore, a summary of tests, and a final success message. The test 'core.controllers.editor_test' passed 92 tests in 63.8 seconds. The terminal prompt is 'shivam@shivam-HP-Pavilion-Laptop-15-cc1xx:~/oppia/oppia\$'.

```
onComplete
[datastore] INFO: Adding handler(s) to newly registered Channel.
[datastore] Feb 26, 2022 12:46:31 PM io.gapi.emulators.grpc.GrpcServer$3 operat
ionComplete
[datastore] INFO: Adding handler(s) to newly registered Channel.
[datastore] Feb 26, 2022 12:46:31 PM io.gapi.emulators.netty.HttpVersionRouting
Handler channelRead
[datastore] INFO: Detected HTTP/2 connection.
07:17:35 FINISHED core.controllers.editor_test: 79.2 secs
Stopping Redis Server(name="sh", pid=37905)...
Stopping Cloud Datastore Emulator(name="sh", pid=37828)...

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

SUCCESS   core.controllers.editor_test: 92 tests (63.8 secs)

Ran 92 tests in 1 test class.
All tests passed.

Done!
shivam@shivam-HP-Pavilion-Laptop-15-cc1xx:~/oppia/oppia$
```

- I am able to run all the frontend tests at once on my machine.

```
shivam@shivam-HP-Pavilion-Laptop-15-cc1xx: ~/oppia/oppia
Chrome Headless 98.0.4758.80 (Linux x86_64): Executed 7306 of 7321 SUCCESS (0 se
Chrome Headless 98.0.4758.80 (Linux x86_64): Executed 7307 of 7321 SUCCESS (0 se
Chrome Headless 98.0.4758.80 (Linux x86_64): Executed 7308 of 7321 SUCCESS (0 se
Chrome Headless 98.0.4758.80 (Linux x86_64): Executed 7309 of 7321 SUCCESS (0 se
Chrome Headless 98.0.4758.80 (Linux x86_64): Executed 7310 of 7321 SUCCESS (0 se
Chrome Headless 98.0.4758.80 (Linux x86_64): Executed 7311 of 7321 SUCCESS (0 se
Chrome Headless 98.0.4758.80 (Linux x86_64): Executed 7312 of 7321 SUCCESS (0 se
Chrome Headless 98.0.4758.80 (Linux x86_64): Executed 7313 of 7321 SUCCESS (0 se
Chrome Headless 98.0.4758.80 (Linux x86_64): Executed 7314 of 7321 SUCCESS (0 se
Chrome Headless 98.0.4758.80 (Linux x86_64): Executed 7315 of 7321 SUCCESS (0 se
Chrome Headless 98.0.4758.80 (Linux x86_64): Executed 7316 of 7321 SUCCESS (0 se
Chrome Headless 98.0.4758.80 (Linux x86_64): Executed 7317 of 7321 SUCCESS (0 se
Chrome Headless 98.0.4758.80 (Linux x86_64): Executed 7318 of 7321 SUCCESS (0 se
Chrome Headless 98.0.4758.80 (Linux x86_64): Executed 7319 of 7321 SUCCESS (0 se
Chrome Headless 98.0.4758.80 (Linux x86_64): Executed 7320 of 7321 SUCCESS (0 se
Chrome Headless 98.0.4758.80 (Linux x86_64): Executed 7321 of 7321 SUCCESS (0 se
Chrome Headless 98.0.4758.80 (Linux x86_64): Executed 7321 of 7321 SUCCESS (2 mi
ns 39.214 secs / 2 mins 19.624 secs)
TOTAL: 7321 SUCCESS
TOTAL: 7321 SUCCESS
26 02 2022 12:27:24.218:WARN [launcher]: ChromeHeadless was not killed in 2000 m
s, sending SIGKILL.
Done!
shivam@shivam-HP-Pavilion-Laptop-15-cc1xx:~/oppia/oppia$
```

- I am able to run one suite of e2e tests on my machine.

```
shivam@shivam-HP-Pavilion-Laptop-15-cc1xx: ~/oppia/oppia
.   ♦♦♦ should create a question for the skill
.   ♦♦♦ should create and delete misconceptions

5 specs, 0 failures
Finished in 277.194 seconds

Executed 5 of 5 specs SUCCESS in 4 mins 37 secs.
[12:42:18] I/launcher - 0 instance(s) of WebDriver still running
[12:42:18] I/launcher - chrome #01 passed
Stopping Protractor Server(pid=35756)...
Stopping Webdriver manager(name="sh", pid=35714)...
Stopping GAE Development Server(name="sh", pid=35489)...
[2022-02-26 12:42:19 +0530] [35689] [INFO] Handling signal: term
[2022-02-26 12:42:19 +0530] [35926] [INFO] Handling signal: term
[2022-02-26 12:42:19 +0530] [35935] [INFO] Handling signal: term
[2022-02-26 12:42:19 +0530] [35693] [INFO] Worker exiting (pid: 35693)
[2022-02-26 12:42:19 +0530] [35969] [INFO] Handling signal: term
[2022-02-26 12:42:19 +0530] [35931] [INFO] Worker exiting (pid: 35931)
[2022-02-26 12:42:19 +0530] [36053] [INFO] Handling signal: term
[2022-02-26 12:42:19 +0530] [35967] [INFO] Worker exiting (pid: 35967)
```

Other Summer Obligations

Currently I do not have any summer obligations.

Communication Channels

I plan on communicating with my mentor weekly for progress reports and as-needed during the project.

Channels:

- Google Meet
- Zoom
- MS Teams
- Hangout

or any other platform according to the mentor's preference.

Section 2: Proposal Details

Problem Statement

Link to PRD (or N/A if there isn't one)	N/A
Target Audience	Developers of the Oppia Team
Core User Need	The project aims to completely migrate all the end-to-end testing files from protractor to Webdriverlo and set up a new script file to run the new version of the tests both on CI as well as locally. This will help in maintaining the tests for the long run as the Angular team is depreciating the protractor that we are currently using for end-to-end testing.
What goals do we want the solution to achieve?	All the test suites and their dependencies must be migrated to Webdriverlo. The migrated tests should not hamper the flakiness, and memory usage and also support Oppia's functionality. They should be running smoothly on both CI as well as locally. Additionally, document the new tool we are going to use.

Section 2.1: WHAT

Key User Stories and Tasks

#	Title	User Story Description (role, goal, motivation) <i>"As a ..., I need ..., so that ..."</i>	Priority	List of tasks needed to achieve the goal (this is the "User Journey")	Links to mocks/prototypes, and/or PRD sections that spec out additional requirements.
1	Change config file	As an Oppia developer, I might need to enable auto-capture screenshots for failed tests or change my browser for running e2e tests, and enable recording videos of tests while running them locally so that I can run and debug tests efficiently.	Must-Have	We will create a new config file related to the testing framework (i.e wdio.conf.js for Webdriverlo)	N/A
2	Setup a new script for running tests.	As an Oppia developer, I need a script file for the smooth running of tests both locally as well as on CI so that the code can be tested properly and regressions can be avoided easily.	Must-Have	We will write a new script to run the migrated test suites both locally and on CI.	N/A
3	Migrates e2e tests.	As an Oppia developer, I want the e2e tests to be fully functional and up to date so that it allows developers to push code without worrying about breaking things. It enables releases with extra confidence and it catches errors that are missed during manual regression testing.	Must Have	Migrates all the test suites/utils and their dependencies to the Webdriverlo. Folders that contain protractor test suites: <ul style="list-style-type: none"> • protractor • protractor_desktop • protractor_utils • protractor_mobile • objects/protractor.js • interactions/protractor.js • TextInput/protractor.js • RatioExpressionInput/protractor.js • NumericInput/protractor.js • NumericExpressionInput/protractor.js • NumberWithUnits/protractor.js • MultipleChoiceInput/protractor.js • MathEquationInput/protractor.js 	N/A

				<ul style="list-style-type: none"> • ItemSelectionInput/protractor.js • GraphInput/protractor.js • FractionInput/protractor.js • EndExploration/protractor.js • Continue/protractor.js • CodeRepl/protractor.js • AlgebraicExpressionInput/protractor.js 	
4	Document the new tool.	As an Oppia developer, I want to quickly fix any issues that arise in my e2e tests, so that I can get my PR submitted and not be blocked on those.	Must-Have	Update the wiki page for the new tool we are going to use.	N/A

Technical Requirements

Additions/Changes to Web Server Endpoint Contracts

We are not adding/changing any endpoint contracts to Web Server.

Calls to Web Server Endpoints

We are not making any new calls to Web Server Endpoints.

UI Screens/Components

To reduce developers' efforts in debugging we are using a combination of reporters for generating test reports.

1. [Spec reporter](#) (Mostly the same as we have for Protractor)

```

1) DEV MODE Test should not show Dev Mode label in prod
   Expected true to be false.
     actual expected
     true

Error: Expected true to be false.
   at <Jasmine>
   at <Jasmine>
   at UserContext.<anonymous> (/home/shivam/oppia/oppia/core/tests/webdriverio/navigation.js:162:8)
   at processTicksAndRejections (node:internal/process/task_queues:96:5)

Spec Files:      0 passed, 1 failed, 1 total (100% completed) in 00:03:17

```


2. [HTML Reporter](#) Features

- Loud Summary of the Test Results.
- Test Results filtering. Great for focusing on failed tests
- Error stack trace attached to test.

Cons:

- Screenshots cannot be attached to the report.
- Sometimes style file for the report is not generated.

Test Passed:

The screenshot shows a web browser displaying a test report for 'Oppia landing pages tour'. The report is titled 'Oppia landing pages tour' and shows a suite start time of 2022-06-01T07:07:51.772Z and a suite end time of 2022-06-01T07:08:10.519Z. The suite duration is 00:00:18.747. There are 5 tests listed, all of which passed:

- ✓ should visit the Fractions landing page
- ✓ should visit the Partners landing page
- ✓ should visit the Nonprofits landing page
- ✓ should visit the Parents landing page
- ✓ should visit the Teachers landing page

The report also shows a 'Donation flow' section below the tests.

Tests Failed:

The screenshot shows a web browser displaying a test report for 'Static Pages Tour'. The report is titled 'Static Pages Tour' and shows a suite start time of 2022-06-01T07:08:28.887Z and a suite end time of 2022-06-01T07:09:03.174Z. The suite duration is 00:00:34.287. There are 14 tests listed, with one test failing:

- ✗ should visit the Home page

The failed test has an assertion error: 'Assertion: Expected false to be true.' The error stack trace is as follows:

```
Error: Expected false to be true.  
    at <Jasmine>  
    at <Jasmine>  
    at UserContext.<anonymous> (/home/shivam/oppia/oppia/core/tests/webdriverio/navigation.js:91:53)  
    at processTicksAndRejections (node:internal/process/task_queues:96:5)
```

3. [Timeline Reporter](#) Features

- Loud Summary of the Test Results.
- Detail of each test run including all screenshots captured during test execution.
- Test Results filtering. Great for focusing on failed tests
- Error stack trace attached to test.
- Ability to add additional information to test at runtime.

Cons:

- Screenshots attached don't necessarily are of the failed test case.
- Videos cannot be attached to the report.
- Error in frontend tests while using this reporter. [link](#)

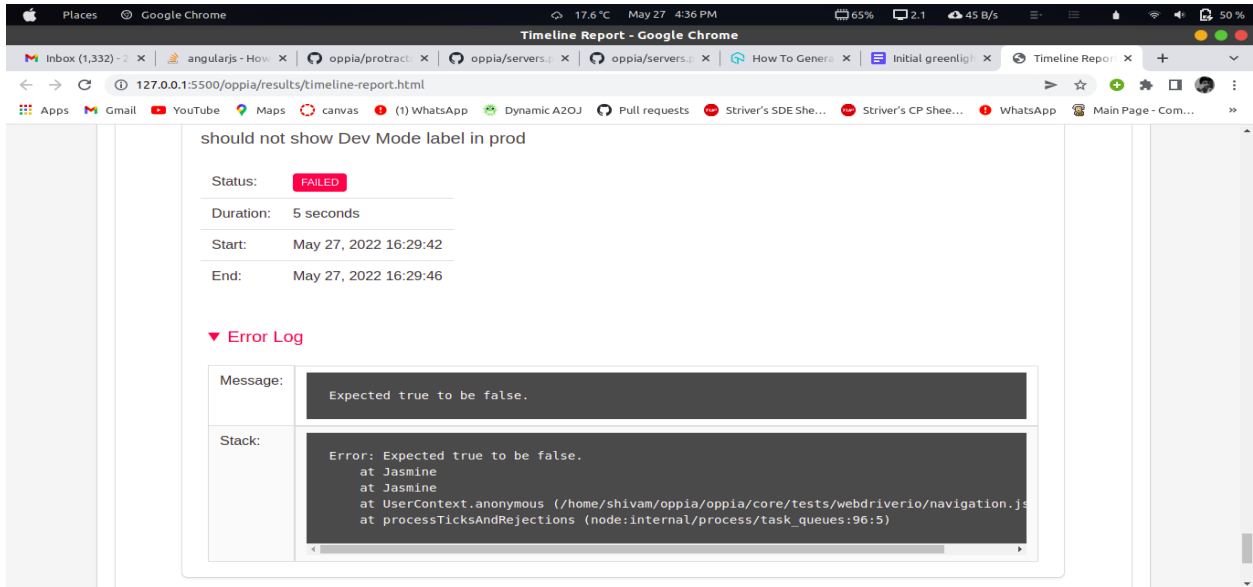
Total Duration: 1 minute, 45 seconds

22 Total	20 Passed	2 Failed	0 Skipped
--------------------	---------------------	--------------------	---------------------

Search for specs containing specific text e.g browser version

Spec: /home/shivam/oppia/oppia/core/tests/webdriverio/navigation.js
Duration: 1 minute, 25 seconds
Browser: chrome 101.0.4951.64

Oppia landing pages tour



3. [Allure Reporter](#)

Features:

- This is a very detailed report where we can see every request and response for each action in the test is captured
- Videos, and screenshots are attached with every test if set to true.
- There is also a section called Graphs – where we can find more details such as the Status of the entire suite, severity, duration, duration trend, and many more details as we explore.
- The TREND graph is handy to monitor the stability of the tests/suites.

Cons:

- The reports can only be viewed using an IDE(using a live server). Directly opening the html file with the browser will not show stats.

ALLURE REPORT 6/7/2022
1:41:27 - 1:43:40 (2m 13s)

22 test cases

95.45%

TREND

There is nothing to show

SUITES 4 items total

Static Pages Tour	1	13
Oppia landing pages tour		5
Donation flow		2
DEV MODE Test		1

ENVIRONMENT

There are no environment variables

CATEGORIES 1 item total

Product defects	1
-----------------	---

Static Pages Tour.should visit the Home page

Failed should visit the Home page

Overview History Retries

Expected false to be true.
actual expected
false[]

Categories: Product defects

Severity: normal

Duration: 1s 044ms

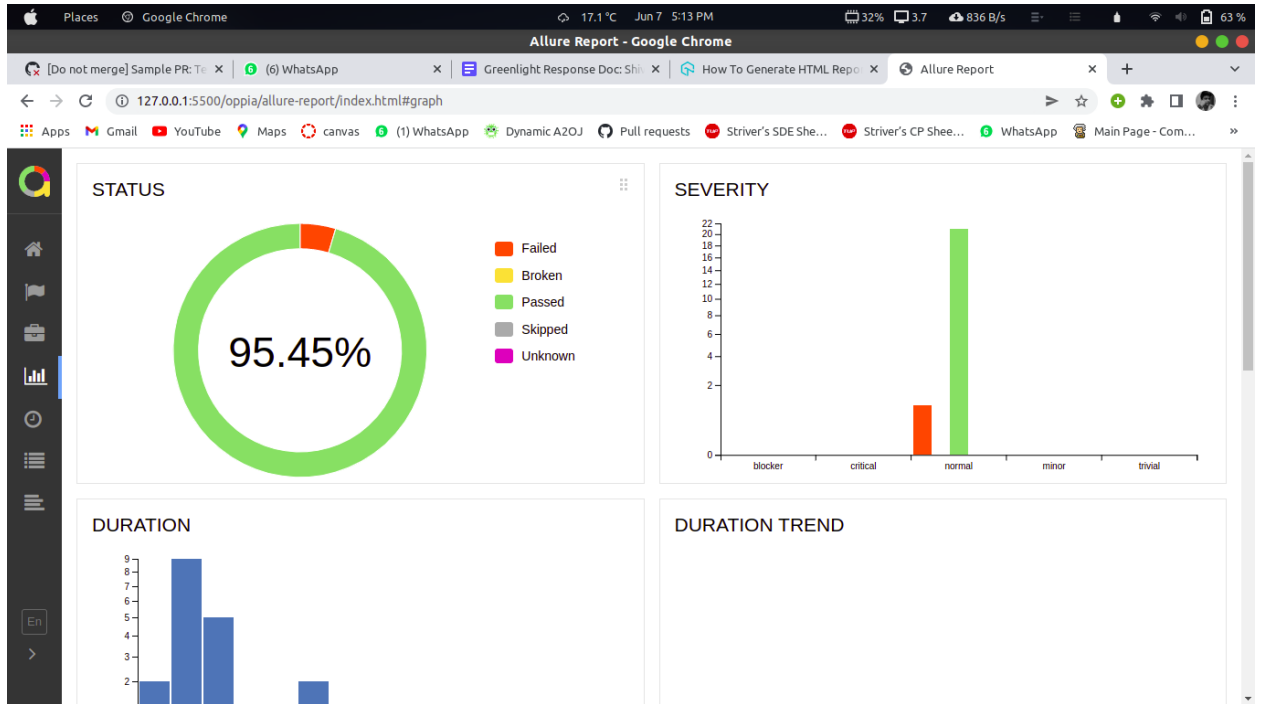
Parameters
browser: chrome-101.0.4951.64

Execution

Test body

Screenshot 9.9 KB

Test Case	Status	Duration
DEV MODE Test	1	
Donation flow	2	
Oppia landing pages tour	5	
Static Pages Tour	13	
#14 should show the error page when an incorrect url is given	Pass	3s 579ms
#5 should visit the About page	Pass	4s 113ms
#9 should visit the About the Oppia Foundation page	Pass	5s 219ms
#6 should visit the Contact page	Pass	10s 739ms
#7 should visit the Donate page	Pass	3s 760ms
#1 should visit the Get started page	Pass	4s 167ms
#4 should visit the Home page	Failed	1s 044ms
#2 should visit the Login page	Pass	4s 456ms
#8 should visit the Partnerships page	Pass	2s 528ms
#10 should visit the Privacy page	Pass	3s 441ms
#3 should visit the Teach page	Pass	2s 737ms
#11 should visit the Terms page	Pass	2s 396ms
#12 should visit the Thanks page	Pass	3s 315ms
#13 should visit the Volunteer page	Pass	2s 788ms



All other reporters are XML and JSON based which is very confusing and not beneficial. So after comparing all these I am going to use allure-reporter + spec reporter for webdriverIO as allure-reporter provides all the features needed for Oppia. I have used this reporter in my sample PR and uploaded the reports as Artifacts.

1. Report with a screenshot of failing test: [Link](#)
2. Report with videos and screenshots of failing test: [Link](#)

Data Handling and Privacy

No new data is stored.

Other Requirements

Recording Videos:

- In webdriverIO we have a third party package named [wdio-video-reporter](#) which handles recording videos of failed test suites.

To configure this we just need to add the following lines in the wdio.conf.js

```
reporters: [
  [video, {
    saveAllVideos: false, // If true, also saves videos for successful test cases
    videoSlowdownMultiplier: 3, // Higher to get slower videos, lower for faster
    videos [Value 1-100]
  }], ],
```

- The command [browser.saveRecordingScreen\(filepath\)](#) can be used to start video recording of any test case while running it locally.

Note: To avoid the flakiness of tests due to memory issues that arise while recording the videos we will only record the videos of failing tests only when we will set the environment variable [VIDEO_RECORDING_IS_ENABLED](#) to 1 (it will be 0 by default) in Github Actions like [this](#). This is configurable per suite as we are already having for the protractor and developers can use this to record videos on the CI in order to debug e2e tests.

Taking Screenshots:

- The command `browser.saveScreenshot(filepath)` can be used to save a screenshot of any particular instance on running the tests locally.
For saving a screenshot on any failed test case we need to add the `afterTest` hook in the `wdio.conf.js` file.

First create a folder in which screenshots will be saved.

```
var dirPath = path.resolve('__dirname', '..', '..', 'webdriverio-screenshots/');
try {
  fs.mkdirSync(dirPath, { recursive: true });
  var screenshotPath = '../webdriverio-screenshots';
} catch (err) {}
```

Then save the screenshots of failed tests in that particular folder

```
afterTest: async function (test, context, { error, result, duration, passed,
retries }) {
  if (error) {
```

```
var testName = encodeURIComponent(test.fullName.replace(/\s+/g, '-'));
var fileName = testName + '.png';
var filePath = path.join(screenshotPath, fileName);
// save screenshot
await browser.saveScreenshot(filePath);
}
```

Handling Offline Mode:

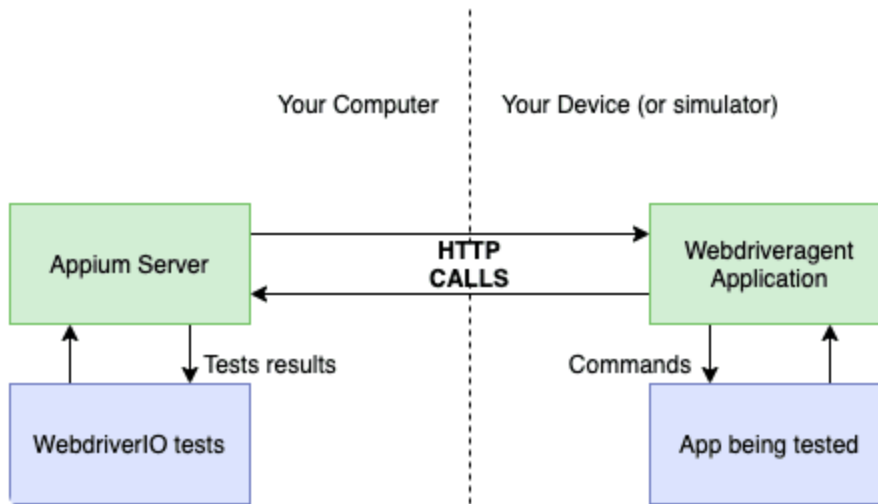
WebdriverIO provides a network command that allows modifying the network throughput of the browser allowing to test under different network conditions, e.g. Regular 3G or even Offline mode:

```
// throttle to Regular 3G
browser.throttle('Regular 3G')
// disable network completely
browser.throttle('Offline')
// set custom network throughput
browser.throttle({
  'offline': false,
  'downloadThroughput': 200 * 1024 / 8,
  'uploadThroughput': 200 * 1024 / 8,
  'latency': 20
})
```

Running tests on Mobile Devices:

We will use **Appium** for running tests on mobile devices with **WebdriverIO**. Appium is basically a tool (like chromedriver) that connects a testing process with a device. It installs an application in the device, called Webdriveragent, and then communicates with the device using HTTP calls. So Appium is the service that allows communication in the first place.

WebdriverIO is the framework itself that we use to write tests for the different platforms. We write code for WebdriverIO and then Appium executes those tests in the devices specified in the configuration. It is free of cost.



Section 2.2: HOW

Existing Status Quo

At Oppia, we have end-to-end (E2E) tests to test our features from the user's perspective.

These tests interact with pages just like a user would, for example by clicking buttons and typing into text boxes, and they check that pages respond appropriately from the user's perspective, for example by checking that the correct text appears in response to the user's actions.

Oppia currently uses protractor for end-to-end testing but as the Angular team plans to end the development of Protractor at the end of 2022 we need to migrate all these tests to some other framework that supports most of the Oppia's functionality. Apart from this, the framework should also have the following features:

1. Supports taking screenshots and videos. This doesn't have to be supported natively (e.g. with protractor we use *ffmpeg* for screen recordings), but it should be possible to do.
2. Supports simulating mobile devices, slow connections, and offline mode.
3. Is easy to write tests.
4. Is easy for us to migrate to.
5. Is fast and has low memory usage.

There are many excellent alternatives available to the open-source community, such as

1. [Cypress](#)
2. [Playwright](#)
3. [Puppeteer](#)
4. [Selenium-webdriver](#)
5. [TestCafe](#)
6. [WebdriverIO](#)

Note: As Cypress has limited support for iframes and no support for new tab creation and Puppeteer does not have a proper migration guide we are eliminating this for use.

So as per the need requirements, there are only four which we can consider:

1. Playwright
2. Selenium-webdriver
3. TestCafe
4. WebdriverIO

Now we will look at the pros and cons of each of the selected frameworks one by one.

Playwright:

Playwright is a web test automation library that tests against the underlying engine for the most popular browsers. Playwright leverages the DevTools protocol to write powerful, stable automated tests.

Advantages:

- Test across all modern browsers with a single API to automate Chromium, Firefox, and WebKit.
- The API can be used in JavaScript & TypeScript, Python, C#, and Java.
- It's simple to set up.
- Stable features.
- Bidirectional (events) – automating things like console logs is easy.
- Auto-wait for elements to be ready before executing actions (like click, fill).
- Intercept network activity for stubbing and mocking network requests.
- Seamless integration with Jest.

Disadvantages:

- It is very new so the APIs are evolving.

- Has no support for IE11 or non-browser platforms.
- Documentations and community are not as good as the other framework yet.

Credits: browserstack.com

WebdriverIO:

WebdriverIO is a test automation framework that allows you to run tests based on the WebDriver protocol and Appium automation technology.

WebdriverIO is written in JavaScript and uses Selenium under the hood. It also comes with its own inbuilt test runner and supports other testing frameworks like Jasmine, Cucumber, and Mocha.

Advantages:

- **Easy to Set up:** WebdriverIO follows a simple setup process. Just install node packages using npm and start testing
- **Customization:** WebdriverIO is highly extendable so users can customize the framework as they need
- **Cross Browser Testing:** WebdriverIO supports multiple browsers such as Chrome, Edge, Firefox, Internet Explorer, and Safari.
- **Native Mobile Application Testing:** WebdriverIO framework can be extended to test native mobile applications.
- **Cross-Origin Support:** WebdriverIO doesn't restrict origins. Origin doesn't matter much as testers can automate them unconditionally.
- **Multiple Tab/Window Support:** WebdriverIO Supports switching to and from multiple windows and tabs.
- **iFrame Support:** WebdriverIO doesn't restrict in terms of iFrame. Testers can automate iframe-based scenarios using simple web driver commands.
- **Reporters:** WebdriverIO supports more than dozens of reporters.
- **Testing Framework/Assertions:** WebdriverIO supports Mocha, Jasmine, and [Cucumber test frameworks](#).
- **Parallel Testing:** Testers can configure WebdriverIO to launch multiple instances and execute tests parallelly.
- **Screenshots:** WebdriverIO can be configured to take screenshots of tests.
- **Video:** Though WebdriverIO doesn't support video recording out of the box it can be configured to do so.

- **Pipeline Integration:** WebdriverIO tests can be integrated into CI Systems like Jenkins, Azure, etc.
- **Selectors:** It supports various types of selectors including CSS and Xpath.
- **Page Object Pattern:** WebdriverIO Framework can be easily configured to Page Object Model.
- **File Upload and Download:** WebdriverIO supports File Upload and Download features.

Disadvantages:

- Much slower compared to frameworks like Playwright and Puppeteer.

Credits: [browserstack.com](https://www.browserstack.com)

Note: In order to verify that speed is not an issue for webdriverIO, I have made a comparison between protractor and webdriverIO based on running the same tests ([navigaion.js](#)) in both of them and the result was as follows:

There were some major time gaps in each test execution on my local system (maybe due to RAM availability difference during each test), I tried to make a comparison on Github Action instead.

Protractor:

S.No	Link	Time
1	Link 1	90 seconds
2	Link 2	68 seconds
3	Link 3	75 seconds

Avg time = 77 seconds

WebdriverIO:

S.No	Link	Time
1	Link 1	65 seconds
2	Link 2	53 seconds
3	Link 3	73 seconds

Avg time = 64 seconds

Conclusion: Though webdriverIO is slower as compared to playwright and cypress the average time taken by webdriverIO is less the time is taken by protractor to run the same suite. So speed is not an issue for webdriverIO.

TestCafe:

It is a pure node.js end-to-end solution for testing web apps.

Advantages:

- **Super Easy setup:** TestCafe is easy and quick to set up. Anyone who knows the basics can do it on their own.
- **No third-party dependency:** TestCafe doesn't depend on any third-party libraries like webdriver, external jars, etc.
- **Easy Test Scriptwriting:** TestCafe command chaining techniques make teams more productive. 20 lines of code in other frameworks can be just written in 10 to 12 lines using TestCafe syntax.
- **Fast and Stable:** Because a test is executed inside a browser, the tests are faster compared to other frameworks. Tests are also more stable as events are simulated internally using JavaScript.
- **Mock Requests:** TestCafe helps to emulate HTTP responses to feed sample data to an app, troubleshoot connectivity errors, and cheat downtime.
- **Multiple Tab Support:** Unlike Cypress, Testcafe provides functionalities like switching between windows and multiple tab support.
- **iframe Support:** Testcafe supports iframes and one can switch to and from iframes in their tests.
- **Parallel Testing:** With concurrency mode enabled, TestCafe tests can be run in parallel.
- **Automated Waiting:** TestCafe waits automatically for elements to appear. There's no need to insert External Waits.
- **Cross Browser Testing:** Testcafe supports all major browsers like old and new Edge, Firefox, IE, and all Chrome family browsers.
- **Debuggability:** Testcafe provides Live Mode which helps to visualize individual actions on the browser for easier debugging.
- **Screenshots:** TestCafe supports taking screenshots for tests using built-in screenshot commands.

Disadvantages:

- **Assertion Libraries:** TestCafe supports built-in assertion libraries only.
- **Selector Support:** By default, TestCafe supports only CSS selectors.
- **Execution of Tests:** Browsers are not aware that they are running in test mode. So, in some edge cases, automation control can be disrupted. It's also quite hard to debug possible issues.

Credits: dzone.com

Selenium-Webdriver:

Selenium comes in mostly two variations: Selenium WebDriver and Selenium IDE. The WebDriver version is a robust framework you can interact with programmatically using a variety of programming languages.

Advantages:

- **Parallel Test Execution:** Selenium supports the parallel execution of tests on multiple machines. Through the use of Selenium Grid, users can perform tests across a variety of browsers and platforms, managing different browsers and their configurations in a centralized way.
- **Third-Party Integrations:** Here's another area where Selenium shines: integrations. Since third-party plugins can extend Selenium's functionality, you can use that to your advantage. You can use one of the many supported or unsupported plugins that already exist or create your own.
- **Community Support:** Selenium is an open-source tool that's been around for quite a while. It has great community support you can count on, not only with regular updates and upgrades but also with comprehensive documentation and many other learning resources.

Disadvantages:

- **High Test Maintenance:** One of the main disadvantages of working with Selenium is that it often leads to fragile tests. Selenium tests will rely on a single, rigid element identifier. Changes to the application, especially those in elements' identifiers will break Selenium tests. This stops releases in their tracks as teams try to diagnose failures, fix tests, and rerun them.

- No built-in image comparison: Selenium does not have a built-in image comparison it is important to validate that images that should be displayed in the application are there, and are correctly shown

Credits: testim.io

The following table sum up the comparisons for each of them:

Key Factors	Playwright	WebdriverIO	TestCafe	Selenium-Webdriver
Performance	<ul style="list-style-type: none"> • Fast • Stable • Reliable 	<ul style="list-style-type: none"> • Slower • Stable • Reliable 	<ul style="list-style-type: none"> • Fast • Stable • Reliable 	<ul style="list-style-type: none"> • Fast • Stable • Reliable
Developer Experience	<ul style="list-style-type: none"> • Simple Setup • Javascript-based 	<ul style="list-style-type: none"> • No additional browser driver • Javascript-based 	<ul style="list-style-type: none"> • No browser control • Only CSS selectors by default 	<ul style="list-style-type: none"> • High Test Maintenance • Supports multiple languages
Documentation	<ul style="list-style-type: none"> • Great Documentation • Migration Guide Present 	<ul style="list-style-type: none"> • Good Documentation • Migration Guide Present 	<ul style="list-style-type: none"> • Good Documentation • Migration Guide Present 	<ul style="list-style-type: none"> • Good Documentatio n • No migration Guide
Community	<ul style="list-style-type: none"> • Smaller Community • Few Maintainers 	<ul style="list-style-type: none"> • Larger Community • Many maintainers 	<ul style="list-style-type: none"> • Less community support than Webdriverlo 	<ul style="list-style-type: none"> • Huge community support

So after analyzing the pros and cons of each of the above-mentioned frameworks, I am able to conclude that WebdriverIO is best suited for Oppia.

Solution Overview

After the selection of the framework, I need to achieve the following tasks to successfully complete this project:

1. Setup WebdriverIO for running e2e tests.
2. Migrating the test suites from protractor to WebdriverIO
3. Documenting the use of WebdriverIO for other developers.

Task 1: Setup WebdriverIO for running E2E tests.

The setup can be done by following the given steps:

Step 1: Installation of WebdriverIO.

The installation can be done by simply adding the following lines in the devDependencies section of the [package.json](#) file.

```
"@wdio/cli": "^7.16.0",  
"@wdio/jasmine-framework": "^7.16.0",  
"@wdio/local-runner": "^7.16.0",  
"wdio-chromedriver-service": "^7.2.8",  
"webdriverio": "^7.19.7",
```

Then we need to run the following command to install all these packages:

```
../oppia_tools/yarn-<yarn version>/bin/yarn install
```

Note: Developers do not need to type this command, this step is just for me to install the webdriverIO on my local system for the first time, once the setup PR will be merged, then the developer will just have to pull the changes and they can directly run the command to run the tests (this is being handled by Oppia's existing dependency installation code, not by new code that we need to write) in webdriverIO

Now we are done with the installation process, we need to create a config file for running the tests in webdriverIO. We will make this file in the same directory where we have our protractor config file i.e in the core/tests directory.

The basic structure of this config file will look something like this. ([wdio.config.js](#))

Step 2: Configuring the script [run_e2e_tests.py](#) and migrating the `protractor.conf.js`

To run the e2e tests on the local environment we need to run the following script. The command structure to run the e2e tests is as follows:

```
python -m scripts.run_e2e_tests --suite="suiteName"
```

Here 'suiteName' specifies which suite we want to run or if we want to run all the tests then we need to provide 'full' as an argument for suiteName. When this command is executed the following lines in the [run_e2e_tests.py](#) run the tests for the protractor after starting the server.

```
proc = stack.enter_context(servers.managed_protractor_server(
    suite_name=args.suite,
    dev_mode=dev_mode,
    debug_mode=args.debug_mode,
    sharding_instances=args.sharding_instances,
    stdout=subprocess.PIPE))
```

Now, there will be two stages for which we need to modify these script files. The stages will be:

- **Hybrid State:** In this stage, we will have some tests written in protractor while some of them are migrated to WebdriverIO.
- **Migration completed state:** This stage will come after we have completed the migration of all test suites.

Hybrid State

As we will be having test suites written in both protractor as well as WebdriverIO we need to first check whether the suite requested by the user is migrated or not. If the tests suite is migrated in WebdriverIO then we will run the tests using the wdio.conf.js file else we will run the tests using the protractor.conf.js file. To do this, we will create a list in this file where we will have the name of all the test suites migrated to WebdriverIO like this and a list where we will list all the test suites still in Protractor.

```
SUITES_MIGRATED_TO_WEBDRIVERIO = [
    'abc.js'
]
SUITES_STILL_IN_PROTRACTOR = [
    ...
]
```

Now the check to whether run the webdriverIO or protractor server will be like this.

```
if args.suite in SUITES_MIGRATED_TO_WEBDRIVERIO:
    proc = stack.enter_context(servers.managed_webdriverIO_server(
        suite_name=args.suite,
        stdout=subprocess.PIPE))
```



```

else:
    stack.enter_context(servers.managed_webdriver_server(
        chrome_version=args.chrome_driver_version))

    proc = stack.enter_context(servers.managed_protractor_server(
        suite_name=args.suite,
        dev_mode=dev_mode,
        debug_mode=args.debug_mode,
        sharding_instances=args.sharding_instances,
        stdout=subprocess.PIPE))

```

Note: After a test suite is migrated from protractor to webdriverIO we will delete the following suite name from the [protractor.conf.js](#), SUITES_STILL_IN_PROTRACTOR and add them in the [wdio.conf.js](#) and SUITES_MIGRATED_TO_WEBDRIVERIO lists.

Now when the user wants to run all the test suites then we will simply run both webdriverIO and protractor tests one by one with the suiteName value being passed as the list of files in that particular framework.

```

if args.suite == 'full':
    proc = stack.enter_context(servers.managed_protractor_server(
        suite_name=args.suite,
        dev_mode=dev_mode,
        debug_mode=args.debug_mode,
        sharding_instances=args.sharding_instances,
        stdout=subprocess.PIPE))

    proc = stack.enter_context(servers.managed_webdriverIO_server(
        suite_name=args.suite,
        debug_mode=args.debug_mode,
        chrome_version=args.chrome_driver_version,
        stdout=subprocess.PIPE))

elif args.suite in SUITES_MIGRATED_TO_WEBDRIVERIO:
    proc = stack.enter_context(servers.managed_webdriverIO_server(
        suite_name=args.suite,
        debug_mode=args.debug_mode,
        chrome_version=args.chrome_driver_version,
        stdout=subprocess.PIPE))

print(
    'Servers have come up.\n'
    'Note: You can find a detailed report of running tests '
    'in ../webdriverIO-tests-report/')

```

```

elif args.suite in SUITES_STILL_IN_PROTRACTOR:
    proc = stack.enter_context(servers.managed_protractor_server(
        suite_name=args.suite,
        dev_mode=dev_mode,
        debug_mode=args.debug_mode,
        sharding_instances=args.sharding_instances,
        stdout=subprocess.PIPE))

    print(
        'Servers have come up.\n'
        'Note: If ADD_SCREENSHOT_REPORTER is set to true in '
        'core/tests/protractor.conf.js, you can view screenshots of the '
        'failed tests in ../protractor-screenshots/')
else:
    print(
        'The suite requested to run does not exists'
        'Please provide a valid suite name')
    sys.exit(1)

```

Here, managed_webdriverIO_server function looks like this:

```

@contextlib.contextmanager
def managed_webdriverIO_server(
    suite_name='full', debug_mode=False, sharding_instances=1,
    chrome_version=None, **kwargs):
    """Returns context manager to start/stop the WebdriverIO server gracefully.

    Args:
        suite_name: str. The suite name whose tests should be run. If the value
            is `full`, all tests will run.
        dev_mode: bool. Whether the test is running on dev_mode.
        **kwargs: dict(str: *). Keyword arguments passed to psutil.Popen.

    Yields:
        psutil.Process. The webdriverio process.

    Raises:
        ValueError. Number of sharding instances are less than 0.
    """
    if sharding_instances <= 0:
        raise ValueError('Sharding instance should be larger than 0')

    if chrome_version is None:
        chrome_version = get_chrome_verison()

    webdriverIO_args = [

```

```

common.NODE_BIN_PATH2,
common.NODEMODULES_BIN_PATH, common.WEBDRIVERIO_CONFIG_FILE_PATH,
'--suite', suite_name, chrome_version,
]

if debug_mode:
    # NOTE: This is a flag for Node.js, not Protractor, so we insert it
    # immediately after NODE_BIN_PATH.
    webdriverIO_args.insert(0, 'DEBUG=true')

# OK to use shell=True here because we are passing string literals and
# constants, so there is no risk of a shell-injection attack.
managed_webdriverIO_proc = managed_process(
    webdriverIO_args, human_readable_name='WebdriverIO Server', shell=True,
    **kwargs)

with managed_webdriverIO_proc as proc:
    yield proc

```

Note: We will need a lint check in `python_linter.py` file in order to ensure the union of both `SUITES_*` constants is the full list. The check will look like this:

```

def check_all_e2e_suites(self):
    """This function is used to check that whether all
    the e2e suites are present in run_e2e_tests.py file.

    Returns:
        TaskResult. A TaskResult object representing the result of the lint
        check.
    """
    name = 'All E2E Suites'
    error_messages = []
    failed = False

    ALL_SUITES_PRESENT = (
        SUITES_MIGRATED_TO_WEBDRIVERIO + SUITES_STILL_IN_PROTRACTOR)

    if set(ALL_SUITES_PRESENT) != set(TOTAL_E2E_SUITES):
        failed = True;

    if failed:
        error_message = 'E2E suites missing from list in run_e2e_tests.py'
        error_messages.append(error_message)

```

```
return concurrent_task_utils.TaskResult(
    name, failed, error_messages, error_messages)
```

Migration completed state

After the complete migration of all the test suites we will delete the `managed_protractor_server` function and also the `protractor.conf.js` file from the codebase.

Also, we will not need any checks mentioned in the above section while running the test suites, so we will undo those changes as well. We will only have a single config file (`wdio.conf.js`) as well.

Note: To run the tests in debug mode currently I am using the following command which will be configured in the same way (order) as it's written below in the script file at the start of the migration:

```
python -m scripts.run_e2e_tests --debug_mode --suite="suiteName"
```

All the flags will follow the same configuration as it was earlier for [protractor](#).

[wdio.conf.js](#)

The configuration file contains all the necessary information to run your test suite. It's just a NodeJS module that exports a JSON. The basic structure of this file will be something like this:

```
var suites = {
  ...
};
exports.config = {
  ...
};
```

Here `suites` object will be providing the path of test suites that we will be running for a particular suite and the `config` will contain all the information about the running of tests like the timeout for which the tests will be running, sharding instances, etc. The config file after migration will look something like [this](#).

Task 2: Migrating the test suites from protractor to WebDriverIO.

Hybrid Mode

When I will start migrating the E2E suites till the time it gets completed we will be in hybrid mode i.e the tests will be present in both webdriverIO and protractor. There are a few important points about the hybrid mode-

- We will have two tools to run the E2E tests i.e protractor and webdriverIO
- Each test suite will be either present in webdriverIO or protractor.
- Same utils files can be present in both webdriverIO and protractor.
- No file will contain the code of both webdriverIO and protractor.
- Protractor suite can only work on protractor utils and vice versa for webdriverIO.

After having a lot of discussions with mentors, we have decided to have a tracker sheet that I will update on a regular basis which will have all the information like **suites migrated to webdriverIO, suites still in protractor and when I am going to migrate them, dependencies migrated to webdriverIO, dependencies still in protractor and common dependencies which are present in both protractor and webdriverIO**. The developers can take references before making any changes to e2e files in order to know which files are present in which version, what files I am going to migrate in future so that they can avoid any unnecessary clashes. I will share the link to this tracker with all Oppia developers (oppia-dev@googlegroups.com) and also update it on the wiki page before I will start migrating the e2e files.

Now there are some potential issues that need to be addressed:

1. **How exactly are common files migrated, and how will the interdependencies work?**

Res: Let's understand this situation with a simple test case. I have two test files **AdditionalPlayer** (having dependencies **ExplorationEditorPage, ExplorationPlayerPage, LibraryPage**), and **AdditionalEditorFeaturesModals** (having dependencies **ExplorationEditorPage**). We can see that the **ExplorationEditorPage** is common in both the test suites. So now I need to migrate the **AdditionalPlayer** then I will also have to migrate its dependencies, there will be no issue with **ExplorationPlayerPage, LibraryPage** as they do not have any suites dependent on them, we will just migrate them and delete their protractor version. But for **ExplorationEditorPage** we still have one dependent suite i.e **AdditionalEditorFeatures** so we will migrate it to successfully migrate out the **AdditionalPlayer**

suite but we will not delete the protractor version of it (as a protractor suite cannot work on webdriverIO util files). So now the **ExplorationEditorPage** file will be a common file that presents in both webdriverIO as well as protractor version. To reduce code duplicacy to some extent I will migrate only that portion of **ExplorationEditorPage** that will be used in the **AdditonalPlayer** suite, rest can be migrated when we will migrate **AdditonalEditorFeaturesModals**.

2. What if some user made some changes in the common files (like **ExplorationEditorPage** here) during hybrid mode?

Res: The tests might break if the changes are only applied in one version of the common file. So we need to make sure to keep the two versions of the common file (**ExplorationEditorPage here**) in sync. We will not merge the PR of the developer if it's not synced. I will be helping in case the developer is having any difficulty in making changes for other versions.

Note: I have to depend upon the QA-reviewers for letting me know about any such case, or else I can be one of the code owners of utility files of both versions (as common files will be present here only) so that I will be aware of any such change is being made.

3. How to handle a situation where a protractor suite is accessing a WebdriverIO dependency?

Res: We have already seen that the protractor suite will not access any WebdriverIO dependency, if any dependency is common to them then it will be present in both webdriverio and protractor util folder.

4. How to distinguish the files so that developers aren't confused about which e2e file to modify?

Res: There are two major scenarios for them.

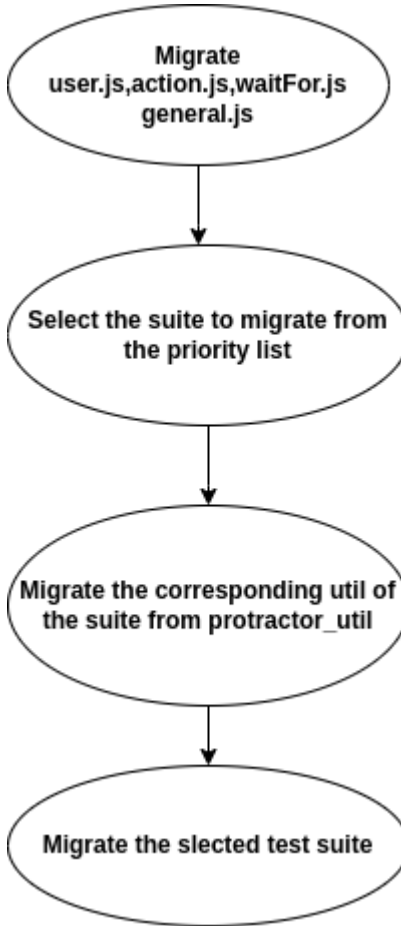
1. If they want to add a new suite:
 - Add the new e2e test suite in webdriverio.
2. Add the tests in the already present suite:
 - If the suite is migrated to webdriverIO add the tests in webdriverIO
 - If the suite is still in protractor add the tests in protractor.

Edge Case: If I already have a migration PR open for a particular suite and at the same time someone also made a PR adding the tests in the same suite, in this case, if my PR will be merged first then the contributor needs to write the tests in webdriverIO (I will coordinate

and help him) otherwise I will update my PR with the changes.

So now we can move on to the migration part:

The basic flow of migration will be something like this:



To make a priority list we need to first look at the dependencies of each of the following suites.

Note: The priority list is made in order to minimize the number of common dependencies at any point of time (dependency present in both WebdriverIO and Protractor). There are 5 dependencies that are present in more than 10 suites, 3 dependencies that are present in 5-10 suites, and the rest all are present in less than 5 suites. So what I did is, I tried to make a combination of suites that shares common dependencies in order to finish the complete migration of dependencies in the minimum number of time.

S.No	Suite Name	Dependencies
1.	Accessibility	LibraryPage
2.	AdditonalEditorFeatures	ExplorationEditorPage, ExplorationPlayerPage, LibraryPage
3.	AdditonalEditorFeaturesModals	ExplorationEditorPage
4.	AdditionalPlayerFeatures	ExplorationEditorPage, ExplorationPlayerPage, LibraryPage
5.	AdminPage	AdminPage
6.	BlogDashBoard	BlogDashboardPage
7.	ClassroomPage	ClassroomPage, LibraryPage
8.	ClassroomPageFileUploadFeatures	TopicEditorPage, TopicsAndSkillsDashboardPage, LibraryPage, ClassroomPage, AdminPage
9.	Collections	CreatorDashboardPage, CollectionEditorPage, LibraryPage
10.	ContributorDashboard	ContributorDashboardPage, ContributorDashboardAdminPage, ExplorationEditorPage, SkillEditorPage, TopicsAndSkillsDashboardPage
11.	CoreEditorAndPlayerFeatures	CreatorDashboardPage, ExplorationEditorPage, ExplorationPlayerPage, LibraryPage
12.	CreatorDashboard	CreatorDashboardPage, ExplorationPlayerPage, SubscriptionDashboardPage
13.	Embedding	ExplorationEditorPage, ExplorationPlayerPage
14.	ExplorationFeedbackTab	CreatorDashboardPage, ExplorationEditorPage, ExplorationPlayerPage, LibraryPage
15.	ExplorationHistoryTab	ExplorationEditorPage, ExplorationPlayerPage
16.	ExplorationImprovementsTab	AdminPage, ExplorationEditorPage
17.	ExplorationStaticticsTab	CreatorDashboardPage,

		ExplorationEditorPage, ExplorationPlayerPage, LibraryPage
18.	ExplorationTranslationTab	CreatorDashboardPage, ExplorationEditorPage
19.	Extensions	ExplorationEditorPage, ExplorationPlayerPage, LibraryPage, Interactions
20.	FeatureGating	AdminPage
21.	FileUploadFeatures	ExplorationEditorPage, CreatorDashboardPage
22.	FileUploadExtensions	ExplorationEditorPage, ExplorationPlayerPage
23.	Library	AdminPage, ExplorationEditorPage, LibraryPage
24.	LearnerDashboard	TopicsAndSkillsDashboardPage, ExplorationPlayerPage, LearnerDashboardPage, AdminPage, TopicEditorPage, StoryEditorPage, LibraryPage, SubscriptionDashboardPage, TopicAndStoryViewerPage, ExplorationEditorPage, SkillEditorPage
25.	Learner	AdminPage, CreatorDashboardPage, CollectionEditorPage, ExplorationEditorPage, ExplorationPlayerPage, LearnerDashboardPage, LibraryPage
26.	Navigation	GetStartedPage
27.	PlayVoiceover	ExplorationEditorPage, ExplorationPlayerPage, LibraryPage
28.	Preferences	PreferencesPage
29.	ProfileFeatures	CreatorDashboardPage, ExplorationPlayerPage, LibraryPage, PreferencesPage, ProfilePage
30.	ProfileMenu	LearnerDashboardPage
31.	Publication	AdminPage, ExplorationEditorPage, ExplorationPlayerPage, LibraryPage

32.	SkillEditor	ExplorationEditorPage, TopicsAndSkillsDashboardPage, SkillEditorPage
33.	Subscription	CreatorDashboard, PreferencesPage, SubscriptionDashboardPage
34.	TopicAndStoryEditor	TopicsAndSkillsDashboardPage, TopicEditorPage, StoryEditorPage, SkillEditorPage, ExplorationEditorPage, ExplorationPlayerPage
35.	TopicAndStoryEditorFileUploadFeatures	TopicsAndSkillsDashboardPage, TopicEditorPage, StoryEditorPage, SkillEditorPage, ExplorationEditorPage
36.	TopicAndStoryViewer	AdminPage, TopicsAndSkillsDashboardPage, TopicAndStoryViewerPage, TopicViewerPage, TopicEditorPage, StoryEditorPage, SubTopicViewerPage, ExplorationEditorPage, ExplorationPlayerPage, SkillEditorPage
37.	TopicAndSkillDashboard	ExplorationEditorPage, TopicsAndSkillsDashboardPage, SkillEditorPage, TopicEditorPage
38.	Users	CollectionEditorPage, CreatorDashboardPage, ExplorationEditorPage, LibraryPage, ModeratorPage, PreferencesPage
39.	Wipeout	DeleteAccountPage, ExplorationEditorPage

Note: While migrating a suite with its dependencies, then it might happen that there can be more than one test suite with the same dependency. So, in that case, we will be having two dependencies, one for protractor and the other for webdriverio. We can only delete the protractor version if all the suites with that dependency are migrated to webdriverIO.

After carefully analyzing the above that I am able to create the following order in which we are going to select the tests suite for migration:

- Collections
- Learner
- User
- ProfileFeatures

- Subscription
- Preferences
- CreatorDashboard
- LearnerDashboard
- ProfileMenu
- TopicAndStoryViewer
- TopicAndStoryEditor
- TopicAndStoryEditorFileUploadFeatures
- BlogDashBoard
- ContributorDashboard
- ClassroomPage
- ClassroomPageFileUploadFeatures
- TopicAndSkillDashboard
- SkillEditor
- Wipeout
- Navigation
- AdditonalEditorFeatures
- AdditonalEditorFeaturesModals
- AdditionalPlayerFeatures
- Accessibility
- AdminPage
- CoreEditorAndPlayerFeatures
- ExplorationFeedbackTab
- ExplorationHistoryTab
- ExplorationImprovementsTab
- ExplorationStaticticsTab
- ExplorationTranslationTab
- Embedding
- Extensions
- FeatureGating
- FileUploadFeatures
- FileUploadExtensions
- Library
- PlayVoiceover

- Publication

Now after we select a file to migrate we need to do the actual migration part. So, we will see how I am going to migrate the file:

Selectors:

The \$ command is a short way to call the [findElement](#) command in order to fetch a single element on the page

The [WebDriver Protocol](#) provides several selector strategies to query an element. WebdriverIO simplifies them to keep selecting elements simple.

Please note that even though the command to query elements in webdriverio is called \$ and \$\$, they have nothing to do with jQuery.

Name	Protractor	WebdriverIO
By class name	element(by.css('.className'))	\$('.className')
Element containing certain string	element(by.cssContainingText(tag, text))	\$('tag=text')
By ID	element(by.css('#idName'))	\$('#idName')
Multiple Elements	element.all(by.css('.className'))	\$\$('.className')
Chain Selectors	element(by.css('some-css')).element(by.css('tag-within-css'))	\$('some-css').\$('tag-within-css')
Selecting element one by one in loop	element.get(i) Here i refers to the index of element to be selected	\$(selector).\$\$('selector')[i]

We are going to use mostly these selectors only while doing migration, so I am not mentioning other query selectors.

Note:

1. There is no proper substitute available for the command `element.getWebElement()` in webdriverIO but we can use [browser.findElement\(\)](#) as a substitute for this.

2. For selecting elements one by one in a loop we can use \$\$ operator and find the element with a given index, like [\\$\(selector\).\\$\\$\\$\(selector\)\[i\]](#).

After selecting any element we can do a lot of operations on them. Now we will see some important operations that we need.

Actions	WebdriverIO	Protractor
Click on the element	\$(selector).click()	element.click()
Get the text content from a DOM-element	\$(selector).getText()	element.getText()
Get the source code of the specified DOM element by the selector.	\$(selector).getHTML()	element.getInnerHTML() element.getOuterHtml()
Navigate to the given destination	browser.url(url)	browser.get(url)
Get the url of the currently opened website.	browser.getUrl();	browser.getCurrentUrl()
Get an attribute from a DOM-element based on the attribute name.	\$(selector).getAttribute(attributeName)	element.getAttribute(attributeName)
Wait until a given condition is fulfilled	browser.waitUntil(condition, { timeout, timeoutMsg, interval })	browser.wait(condition, time, optional_msg)
Send a sequence of keystrokes to the active element	browser.keys(value)	element.sendKeys(keys)
Uploads a file to the browser driver (e.g. Chromedriver)	browser.uploadFile(localPath)	N/A
Drag an item to a destination element or position.	\$(selector).dragAndDrop(target, { duration })	N/A
Selected DOM-Element is clickable, visible, and exists.	\$(selector).isClickable()	until.elementToBeClickable(element)
Get the value of a <textarea>, <select> or text <input> found by given selector	\$(selector).getValue()	element.getText()
Clear a <textarea> or text <input> element's value	\$(selector).clearValue()	element.clear()

Timeouts:

Each command in WebDriverIO is an asynchronous operation. Therefore, time is a crucial component in the whole testing process. When a certain action depends on the state of different action, you need to make sure that they get executed in the right order. Timeouts play an important role when dealing with these issues.

- WaitFor* timeout

WebDriverIO provides multiple commands to wait on elements to reach a certain state (e.g. enabled, visible, existing). These commands take a selector argument and a timeout number, which determines how long the instance should wait for that element to reach the state. The `waitForTimeout` option allows you to set the global timeout for all `waitFor*` commands, so you don't need to set the same timeout over and over again.

```
// wdio.conf.js
exports.config = {
  // ...
  waitForTimeout: 5000,
  // ...
}
```

Expected Conditions:

It's a library of canned expected conditions that are very much useful while writing e2e tests. We are using a lot of expected conditions in the test suites I am going to migrate. So, we need a proper replacement for protractor Expected Conditions.

In webdriverIO we have a third-party library called [wdio-wait-for](#) which is already installed with **wdio-test-runner** which takes care of this. Now, we will see the proper replacement of each of these commands in webdriverIO.

Action	Protractor	WebDriverIO
A condition for checking an element contains a specific text	<code>until.textToBePresentInElement((selector, text), time);</code>	browser.waitUntil(textToBePresentInElement('selector', expectedText));
A condition for checking an element is visible and clickable	<code>until.elementToBeClickable(element)</code>	elementToBeClickable(selector)
A condition for checking the element to be invisible	<code>until.invisibilityOf(element)</code>	invisibilityOf(selector)
A condition for checking that an element is present on the DOM	<code>until.presenceOf(element)</code>	presenceOf('.header')

of a page		
A condition for checking the element to be visible	until.visibilityOf(element)	visibilityOf('.header')
A condition for checking an alert on the page	until.alertIsPresent()	alertIsPresent()

* until = protractor.ExpectedConditions

Debugging:

In many cases, we can use [browser.debug\(\)](#) to pause your test and inspect the browser.

Our command-line interface will also switch into REPL mode. This mode allows you to fiddle around with commands and elements on the page. In REPL mode, you can access the browser object—or \$ and \$\$ functions—just like you can in your tests.

When using browser.debug(), you will likely need to increase the timeout of the test runner to prevent the test runner from failing the test for taking too long. For example:

In wdio.conf.js:

```
jasmineOpts: {
  defaultTimeoutInterval: (24 * 60 * 60 * 1000)
}
```

Auto-Waiting

One of the most common reasons for flaky tests is interactions with elements that don't exist in your application at the time you want to interact with it. Modern web applications are very dynamic, elements show up and disappear. As a human, we are waiting unconsciously for elements but in an automated script we don't consider this as an action. There are two ways to wait on an element to show up.

Implicit vs. Explicit

The WebDriver protocol offers implicit timeouts that allow specifying how long the driver is supposed to wait for an element to show up. By default, this timeout is set to 0 and therefore makes the driver return with a no such element error immediately if an element could not be found on the page. Increasing this timeout using the setTimeout would make the driver wait and increases the chances that the element shows up eventually.

A different approach is to use explicit waiting which is built into the WebdriverIO framework in commands such as `waitForExist`. With this technique, the framework polls for the element by calling multiple `findElements` commands until the timeout is reached.

Built-in Waiting

Both waiting mechanisms are incompatible with each other and can cause longer wait times. As implicit waits are a global setting it is applied to all elements which is sometimes not the desired behavior. Therefore WebdriverIO provides a built-in wait mechanism that automatically explicitly waits on the element before interacting with it.

Credits: [WebdriverIO Official Documentation](#).

In the protractor, we are using `ExpectedCondition` i.e we specify for how much time the driver will wait for the element to show up. In webdriverIO we will use `wdio-wait-for` which is the replacement for `ExpectedCondition`.

Task 3: Document the usage of WebdriverIO.

We need to completely change the following sections of Oppia's wiki page after the migration is completed in order to document the usage of the new testing framework i.e WebdriverIO.

- [End to End Tests](#)
- [Debug end to end tests](#)

Third-Party Libraries

No.	Third-party library name and version	Link to third-party library	Why it is needed	License ¹ (if third-party library)	[Android only] Min / target / max SDK version that the library supports
1	webdriverio	webdriverIO	To run the e2e tests in webdriverio	MIT	
2	@wdio/jasmine-framework	@wdio/jasmine-framework	To use jasmine for writing the test suites of webdriverio	MIT	
3	@wdio/cli	@wdio/cli	To run the webdriverio tests using wdio.conf.js file	MIT	
4	wdio-chromedriver-service	wdio-chromedriver-service	To run chromedriver seamlessly while running	MIT	

			tests using wdio-testrunner		
5	@wdio/local-runner	@wdio/local-runner	To run the tests locally	MIT	
6	wdio-video-reporter	wdio-video-reporter	To record videos of tests cases	MIT	
7	wdio-html-nice-reporter	wdio-html-nice-reporter	To make a proper html report of tests results.	MIT	

Impact on Other Oppia Teams

Conflicts with other project's e2e tests:

I talked with all the fellow GSoC students in order to know who all have to add new e2e tests in their GSoC projects.

Now there are two major scenarios for them.

1. If they want to add a new suite:
 - Add the new e2e test suite in webdriverio.

Note: As the first new suite will be added in late August, we will have most of the files already migrated to WebdriverIO like action.js, forms.js, user.js, general.js waitFor.js which are mostly needed for writing a new test.

2. Add the tests in the already present suite:
 - If the suite is migrated to webdriverIO add the tests in webdriverIO
 - If the suite is still in protractor add the tests in protractor

After comparing their PR expected date with my expected date of migration of that suite I have mentioned the framework in which they have to add the tests.

Project (Student)	New Suite/Changes in Existing suite	PR Expected Date	Need to write test in
Adding a contributor dashboard stats page (Ayush Jain)	New Suite	1st Sept	WebdriverIO
Learner Group MVP (Pankaj Prajapati)	New Suite	20th Aug	WebdriverIO

Helping Learners when they are stuck (Manan Rathi)	Changes in the existing suite (eplorationStatisticsTab.js)	10 July	WebdriverIO
Improving the lesson creation experience (Soumyajyoti Dey)	Changes in the existing suite (coreEditorAndPlayer.js)	13 July and 25 Aug	Protractor -13 July WebdriverIO- 25 Aug
Making the Contributor Dashboard UI Responsive (Harshvardhan Singh)	New test (Mobile test)	Not yet decided	WebdriverIO
Celebrating learners' accomplishments (Vishnu)	Changes in the existing suite (topicAndStoryViewer.js)	30th Aug	WebdriverIO
Blog Integration (Rijuta Singh)	Changes in the existing suite (blogDashboard.js)	End Aug	WebdriverIO

Risks and mitigations

Potential Risk	Mitigation
Introduction of e2e flakes due to migration of test suites	After the migration of each test suite I will provide the video recording for the protractor version of test suites as well as the webdriverIO version of test suites. Also due to the features like auto-waiting that we get in webdriverIO, the introduction of flakes will be having a minimal chance. If by any means there is some introduction of flakes in the codebase, I will refactor the tests in a way to remove the flake from the codebase.
There may be some PRs that might contain changes in the e2e files written in the protractor and at the same time, it might happen, that I also have an open PR for the migration of that particular suite. This may result in failing tests on the develop branch.	In this particular case, I need to add the label Post-merge sync to my PR so that the contributor needs to change the tests to webdriver instead of the protractor.

Implementation Approach

Storage Model Layer Changes

The storage model layer will not be changed.

Domain Objects

The domain objects will not be changed.

User Flows (Controllers and Services)

The user flow will not be changed.

Web frontend changes

There will be no change in the web frontend.

Documentation changes

We need to change the following sections of Oppia's wiki page in order to document the usage of WebdriverIO.

[End to End Tests](#) :

The basic structure for this section will be like this:

End to End Tests

- **WebdriverIO**
- **Protractor**
 - This will be same as we currently have for protractor.
- **Migration**
 - **Hybrid State**
 - **Guide to Migrate E2E tests/utils**

The protractor and hybrid page will be removed after the migration will be completed.

WebdriverIO Section

1. Introduction

Same as now

2. Flaky tests

Explain what to do if flakes occur.

- **If the end-to-end tests are failing on your PR**

Same content as we have [here](#)

3. Layout of E2E tests file.

The layout will be the same as we have for protractor suites.

Suite Files:

- **core/tests/webdriverio_desktop**
This directory will contain all test suites that are exclusive to desktop interfaces.
- **core/tests/webdriverio_mobile**
This directory will contain all test suites which are exclusive to mobile interfaces.

Utilities:

- **core/tests/webdriverio_utils**
This directory will contain utilities for performing actions using elements from the core components of Oppia.
- **extensions/**/webdriverio.js**
The extensions include a webdriverio.js file that provides functions for customizing an interaction and checking that the created interaction matches the expected criteria.

Note: For the migration period we will be having folders for both webdriverIO and protractor.

4. Run E2E tests

I tried to keep the same commands for running the e2e tests in webdriverio as well. So it will mostly same as we have for the protractor.

Need to update all sections after this for webdriverIO.

Migration:

- **Hybrid State**
 1. **Overview**
Explain about the hybrid mode.
 2. **E2E Migration Tracker**

Link to the e2e tracker will be present here. This tracker contains information like **suites migrated to webdriverIO, suites still in protractor and when I am going to migrate them, dependencies migrated to webdriverIO, dependencies still in protractor, and common dependencies which are present in both protractor and webdriverIO**. Contributors can use this to get the idea about when I am going to migrate a particular suite and hence plan their PR accordingly in order to avoid clashes such as if I opened a PR to migrate a particular suite and the contributor also opens a PR making changes in the same suite at the same time.

3. Add/Modify E2E tests

If any contributor wants to add/modify e2e tests then they will have the following option:

- If they want to add a new suite add it in webdriverIO
- If they want to modify any existing suite then do it according to the version of test suite which is present.

The table prepared for GSoC students will also be present here. Also, this section will explain what the user needs to do if they are modifying any of the common files which are present in both versions i.e **they need to sync up the files in both versions**

They can take help from the migration guide, or else they can contact me as well. My approach to migrating dependencies files is that I will only migrate a portion of the file that is needed for the suite in order to minimize the duplicity.

4. Contact

My contact info shared for any queries that developers might have.

- **Guide to Migrate E2E tests/utils**

This page will cover how we can write/ migrate tests to webdriverIO. This is needed because if the user plan to make any changes to the common files then they need to write the tests in both protractor and webdriverIO. Also I am believing they will be aware of how to write tests in protractor, so they can use this guide even if they plan to add new suite in WebdriverIO (majorly GSoC students).

[Debug end to end tests](#)

The basic structure for this section will be like this:

Debug End to End tests:

- **WebdriverIO**
- **Protractor**
 - Same content that we currently have in Debug end-to-end tests.

WebdriverIO

1. Introduction

Same as now

2. Using the debugger

In webdriverio for using a debugger we need to add the line [browser.debug\(\)](#), also need to update the wiki page regarding how to change the timeout interval in wdio.conf.js to prevent timeout errors for running tests in debugging mode.

3. Downloading screenshots

As webdriverio provides a feature to take a screenshot at an instance using the command [browser.saveScreenshot\(filepath\)](#). This will help contributors to take the screenshots locally instead of downloading the screenshots from the Github CI. Also will update this section on how they can also run the tests on CI and takes screenshots as well.

4. Downloading screen recordings

WebdriverIO also provides the feature to record the screen while running the tests locally using the following command [browser.saveRecordingScreen\(filepath\)](#). Also, I will update the wiki on how to use [wdio-video-reporter](#) to get the recording of failed tests cases locally as well as on Github CI.

Testing Plan

E2E Testing Plan

Local Testing

To check whether the migrated tests are working the same as it was working before, I will add a screen recording of tests running on both protractor and webdriverIO test suites. I will be running the tests on webdriverIO a minimum of 5 times and will provide the recording of each time. We can compare in the video itself whether the tests are working the same way or not.

CI Testing

To check whether the tests are also passing on CI, I will run the tests on CI for a minimum of 5 successful times and will provide the successful runs links with each test.

Lint Checks

The lint checks in [e2e-action.js](#) can be used with webdriverIO as well with a small change in it (adding 'keys' in this [array](#) because in webdriverIO we use browser.keys() instead of browser.sendKeys).

The lint checks in the [protractor-practices.js](#), and [check-element-selector-at-top.js](#) will not work with the webdriverIO test suites, so we need to have new lint checks for webdriverIO. I will create new eslint checks files with the following rules:

Rules for webdriverio-practices.js

- Expect calls must be prefixed with an await (in webdriverio-practices.js)
- Do not allow browser.debug() statements (in webdriverio-practices.js)
- Do not allow browser.pause() statements (in webdriverio-practices.js)
- Constant name in all caps (in webdriverio-practices.js)
- Do not use forEach (in webdriverio-practices.js)
- Class name should start with "webdriverio-test-*" (in webdriverio-practices.js)
- All element selector at the top (in check-element-selector-at-top-wdio.js)

I am planning to add these checks at the start of migration, with the PR that will configure webdriverIO for the codebase.

Feature testing

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

NO

Implementation Plan

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

Milestone 1: Set up WebdriverIO with Github Actions. Document how we use it, including adding a step-by-step guide to the developer wiki on “how to debug e2e tests” (which should be kept up-to-date and address any issues that devs face). Add eslint rules to ensure the code quality. Add a lint check to ensure that the union of suites present in both versions of the e2e tests is the complete list of e2e suites. Fully migrate 20 test suites to WebdriverIO.

No.	Description of PR / action	Prereq PR numbers	Target date for PR creation	Target date for PR to be merged
1.1	Update the documentation for End-To-End tests and Debug End-To-End wiki page.	N/A	13 June 2022	18 June 2022
1.2	Setup webdriverIO for running e2e tests, write eslint rules for webdriverIO suites to make sure the code quality is maintained, write a lint check to ensure union of suites present in both version is complete list of e2e suites, and migrate the following tests: Suites Covered: <ul style="list-style-type: none"> ● Collections ● Learner ● User ● ProfileFeatures Utils Covered: <ul style="list-style-type: none"> ● CreatorDashboardPage (Partial) ● CollectionEditorPage (Complete) ● LibraryPage (Partial) ● AdminPage (Partial) ● ExplorationEditorPage (Partial) ● ExplorationPlayerPage (Partial) ● LearnerDashboardPage (Partial) ● ModeratorPage (Complete) ● PreferencesPage (Partial) ● ProfilePage (Complete) <i>After this PR is merged there will be 7 common dependencies</i>	1.1	22 June 2022	2 July 2022
1.3	Migrate the following test suite from protractor to webdriverIO. Suites Covered: <ul style="list-style-type: none"> ● Subscription ● Preferences ● CreatorDashboard ● LearnerDashboard Utils Covered:	1.1,1.2	28 June 2022	10 July 2022

	<ul style="list-style-type: none"> ● CreatorDashboard (Partial) ● PreferencesPage (Complete) ● SubscriptionDashboardPage (Complete) ● ExplorationEditorPage (Partial) ● ExplorationPlayerPage (Partial) ● LearnerDashboardPage (Partial) ● AdminPage (Partial) ● TopicEditorPage (Partial) ● StoryEditorPage (Partial) ● SkillEditorPage (Partial) ● LibraryPage (Partial) ● TopicAndStoryViewerPage (Partial) ● TopicsAndSkillsDashboardPage (Partial) <p><i>After this PR is merged there will be 11 common dependencies</i></p>			
1.4	<p>Migrate the following test suite from protractor to webdriverIO.</p> <p>Suites Covered:</p> <ul style="list-style-type: none"> ● ProfileMenu ● TopicAndStoryViewer ● TopicAndStoryEditor <p>Utils Covered:</p> <ul style="list-style-type: none"> ● LearnerDashboardPage (Complete) ● AdminPage (Partial) ● TopicsAndSkillsDashboardPage (Partial) ● TopicAndStoryViewerPage (Complete) ● TopicViewerPage (Complete) ● TopicEditorPage (Partial) ● StoryEditorPage (Partial) ● SubTopicViewerPage (Complete) ● ExplorationEditorPage (Partial) ● ExplorationPlayerPage (Partial) ● SkillEditorPage (Partial) <p><i>After this PR is merged there will be 9 common dependencies</i></p>	1.1,1.2. 1.3	8 July 2022	20 July 2022
1.5	<p>Migrate the following test suite from protractor to webdriverIO.</p> <p>Suites Covered:</p> <ul style="list-style-type: none"> ● TopicAndStoryEditorFileUploadFeatures ● BlogDashBoard ● ContributorDashboard <p>Utils Covered:</p> <ul style="list-style-type: none"> ● TopicsAndSkillsDashboardPage (Partial) ● TopicEditorPage (Partial) ● StoryEditorPage (Complete) ● SkillEditorPage (Partial) ● BlogDashboardPage (Complete) ● ExplorationEditorPage (Partial) 	1.1,1.2, 1.3, 1.4	16 July 2022	26 July 2022

	<ul style="list-style-type: none"> ContributorDashboardPage (Complete) ContributorDashboardAdminPage (Complete) ExplorationEditorPage (Partial) <p><i>After this PR is merged there will be 8 common dependencies</i></p>			
1.6	<p>Migrate the following test suite from protractor to webdriverIO.</p> <p>Suites Covered:</p> <ul style="list-style-type: none"> ClassroomPage ClassroomPageFileUploadFeatures TopicAndSkillDashboard <p>Utils Covered:</p> <ul style="list-style-type: none"> ClassroomPage (Complete) LibraryPage (Partial) TopicEditorPage (Complete) TopicsAndSkillsDashboardPage (Partial) AdminPage (Partial) ExplorationEditorPage (Partial) SkillEditorPage (Partial) <p><i>After this PR is merged there will be 7 common dependencies</i></p>	1.1,1.2, 1.3,1.4, 1.5	22 July 2022	30 July 2022
1.7	<p>Migrate the following test suite from protractor to webdriverIO.</p> <p>Suites Covered:</p> <ul style="list-style-type: none"> SkillEditor Wipeout Navigation <p>Utils Covered:</p> <ul style="list-style-type: none"> ExplorationEditorPage (Partial) TopicsAndSkillsDashboardPage (Complete) SkillEditorPage (Complete) DeleteAccountPage (Complete) GetStartedPage (Complete) <p><i>After this PR is merged there will be 5 common dependencies</i></p>	1.1,1.2, 1.3,1.4, 1.5,1.6	28 July 2022	8 Aug 2022

Total Suites Migrated: 20

Milestone 2: Fully migrate all remaining e2e test suites to WebdriverIO, and remove all references to Protractor from the codebase and the developer wiki.

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

2.1	<p>Migrate the following test suite from protractor to webdriverIO.</p> <p>Suites Covered:</p> <ul style="list-style-type: none"> • AdditonalEditorFeatures • AdditonalEditorFeaturesModals • AdditionalPlayerFeatures <p>Utils Covered:</p> <ul style="list-style-type: none"> • ExplorationEditorPage (Partial) • ExplorationPlayerPage (Partial) • LibraryPage (Partial) <p><i>After this PR is merged there will be 5 common dependencies</i></p>	M-1 All PRs,	12 Aug 2022	20 Aug 2022
2.2	<p>Migrate the following test suite from protractor to webdriverIO.</p> <p>Suites Covered:</p> <ul style="list-style-type: none"> • Accessibility • AdminPage • CoreEditorAndPlayerFeatures <p>Utils Covered:</p> <ul style="list-style-type: none"> • LibraryPage (Partial) • AdminPage (Partial) • CreatorDashboardPage (Partial) • ExplorationEditorPage (Partial) • ExplorationPlayerPage (Partial) <p><i>After this PR is merged there will be 5 common dependencies</i></p>	M-1 All PRs	20 Aug 2022	28 Aug 2022
2.3	<p>Migrate the following test suite from protractor to webdriverIO.</p> <p>Suites Covered:</p> <ul style="list-style-type: none"> • ExplorationFeedbackTab • ExplorationHistoryTab • ExplorationImprovementsTab <p>Utils Covered:</p> <ul style="list-style-type: none"> • CreatorDashboardPage (Partial) • ExplorationEditorPage (Partial) • ExplorationPlayerPage (Partial) • LibraryPage (Partial) • AdminPage (Partial) <p><i>After this PR is merged there will be 5 common dependencies</i></p>	M-1 All PRs	28 Aug 2022	6 Sep 2022
2.4	<p>Migrate the following test suite from protractor to webdriverIO.</p> <p>Suites Covered:</p> <ul style="list-style-type: none"> • ExplorationStatisticTab • ExplorationTranslationTab • Embedding 	M-1 All PRs	8 Sep 2022	16 Sep 2022

	<ul style="list-style-type: none"> • Extensions Utils Covered: <ul style="list-style-type: none"> • CreatorDashboardPage (Partial) • ExplorationEditorPage (Partial) • ExplorationPlayerPage (Partial) • LibraryPage (Partial) • Interactions (Complete) <p><i>After this PR is merged there will be 5 common dependencies</i></p>			
2.5	Migrate the following test suite from protractor to webdriverIO. Suites Covered: <ul style="list-style-type: none"> • FeatureGating • FileUploadFeatures • FileUploadExtensions Utils Covered: <ul style="list-style-type: none"> • AdminPage (Partial) • ExplorationEditorPage (Partial) • CreatorDashboardPage (Complete) • ExplorationPlayerPage (Partial) <p><i>After this PR is merged there will be 4 common dependencies</i></p>	M-1 All PRs	18 Sep 2022	25 Sep 2022
2.6	Migrate the following test suite from protractor to webdriverIO. Suites Covered: <ul style="list-style-type: none"> • Library • PlayVoiceover • Publication Utils Covered: <ul style="list-style-type: none"> • AdminPage (Complete) • ExplorationEditorPage (Complete) • LibraryPage (Complete) • ExplorationPlayerPage (Complete) 	M-1 All PRs	25 Sep 2022	1 Oct 2022
2.7	Remove all references to Protractor from the codebase and the developer wiki.	M-1 M-2 All Prs	1 Oct 2022	3 Oct 2022

Total Suites Migrated: 19

Future Work

- As webdriverIO provides regular updates for better performance and improved functionality recently there was an update from version 6 to version 7. So, we need to keep our tests updated with these updates.

- We can use impressive services that webdriverIO provides like ['Docker Service'](#), as we are planning to shift to docker in the future.