

Google Summer of Code 2021

Angular Migration

Ashutosh Chauhan

About You

My name is Ashutosh Chauhan. I am a first-year undergraduate at Indian Institute of Technology, Jammu, pursuing Electrical Engineering as my major. I am a core member of the coding club at our institute and also maintains the club's website which is built with angular.

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

I've started to work on Oppia in Feb 2021. The reason I started contributing to Oppia was that I wanted to improve my angular and angularjs skills. But along the way, I started to admire the motto of Oppia. The amount of selfless work developers put here for the betterment of free education is unparalleled and inspiring.

Most of the Oppia's users belong to developing nations where internet speeds are not good. I also belong to a remote village in my country and daily experience internet fluctuations and slow speeds. I want to improve Oppia's performance by migrating the project to the angular router. By doing this project, I want to play my part to help Oppia achieve its mission. I also believe that the best way to learn is by practice. This project will help me to get proficient with angular and angularjs.

Prior experience

I have been doing web development for the past four and half years. I am a full stack developer and work primarily with Node/Flask backend and Angular/React frontend frameworks. During my high school, I stood first in an inter-school competitive programming competition in the north india region. I also stood first in a National Level Programming Olympiad for high school students.

1. I have contributed to the fossasia organisation from the end of November 2020 to Feb 2021. During my time with **fossasia**, I mainly worked on [fossasia/susi.ai](https://github.com/fossasia/susi.ai) (A web-client built with react for a Artificial Intelligence system called SUSI). I have 7 PRs merged there, most of them were fixing issues labeled bugs, high priority and 3 are still open for review. You can find a list of my contributions on [susi.ai](https://github.com/fossasia/susi.ai) [here](#). I have also contributed to [fossasia/open-event-server](https://github.com/fossasia/open-event-server) (An API built with flask). I made PR [#7705](#) on that project.
2. I am a member of the angular migration team in Oppia. I have been regularly contributing from Feb 2021 to the migration project. My contributions involve creating PRs, reviewing other contributor's

PRs and helping new developers with Oppia installation. At present I have 14 PRs merged and 4 PRs open. I have also participated in Oppia release testing (March 2021).

3. I maintain a [website](#) for The Coding Club of IIT Jammu (my college) built with angular. We used the angular router and built an API to power it. We have open sourced it on github [coding-club-iit-jammu/coding-club-iit-jammu.github.io](https://github.com/coding-club-iit-jammu/coding-club-iit-jammu.github.io) and are open to contributions from developers across the globe.
4. I have built a music library application. Users can upload songs to Django powered backend and play music on their browsers. I used JWT tokens to build a stateless authentication system and deployed it on Amazon Web Services. This project taught me the concepts of token based authentication, SQL databases, WSGI specification and nginx reverse proxy system.
5. I have also worked with backends in python, nodejs and php. I have created a realtime chat application with nodejs and e-commerce application with php as personal projects.

Links to my PRs for Oppia:

From the beginning of my contributions to Oppia, I had a keen interest in migration and the build process. Most of my PRs are related to migration.

1. [#12027](#) Migrates collection-editor-state.service.ts along with spec to angular.
2. [#12076](#) Migrates admin-jobs-tab.component to angular.
3. [#12191](#) Migrates translation opportunity modal and its dependencies.
4. [#12410](#) Migrates topics list directive to angular.
5. [#12371](#) Migrates base-content directive and its dependencies to angular.

The complete list of PRs created by me is [here](#).

I have opened 4 issues regarding UI bugs, those can be found [here](#).

My contributions also involve helping others gitter and discord.

I have also reviewed PRs from other contributors, most notably [#11791](#), [#12071](#) and [#12110](#)

Contact info and timezone(s)

Email and Hangout: ashutoshc8101@gmail.com

Contact Number: +91 9816092935

Gitter: [@ashutoshc8101](#)

Discord: [ashutoshchauhan81](#)

TimeZone: Kolkata, India (GMT+5:30)

I am active on all these platforms. My preferred methods of communication are email, hangout, and discord.

Time commitment

I would be able to devote **20 hr/ week** to the GSoC project, which can be extended upto **40 hr/week** if the need arises. From **12 June to 16 June**, I will have my end semester examination and the time devoted will be **1 hr /day**. After that, I will get a vacation from college till **4 July**. I can easily make up by devoting (**6 hr/day**) to The GSoC project.

Essential Prerequisites

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

```
ashutosh@omen:~/Desktop/oppia_org/oppia
Done in 0.62s.

Installing pre-commit hook for git
Symlink already exists
Making pre-commit hook file executable ...
pre-commit hook file is now executable!
Installing pre-push hook for git
Symlink already exists
Making pre-push hook file executable ...
pre-push hook file is now executable!
-----
Tasks still running:
  core.controllers.admin_test.AdminIntegrationTest (started 21:28:34)
-----
15:58:51 FINISHED core.controllers.admin_test.AdminIntegrationTest: 17.1 secs

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

SUCCESS core.controllers.admin_test.AdminIntegrationTest: 32 tests (14.7 secs)

Ran 32 tests in 1 test class.
All tests passed.

Done!
oppia (develop) |
```

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

```
ashutosh@omen:~/Desktop/oppia_org/oppia
Chrome Headless 88.0.4324.96 (Linux x86_64): Executed 4213 of 4216 SUCCESS (0 se
cs / 38.293 secs)
LOG: 'Spec: ImageLocalStorageService should call helper service function correct
Chrome Headless 88.0.4324.96 (Linux x86_64): Executed 4214 of 4216 SUCCESS (0 se
LOG: 'Spec: ImageLocalStorageService should show error message if number of stor
ed images crosses limit has passed'
Chrome Headless 88.0.4324.96 (Linux x86_64): Executed 4214 of 4216 SUCCESS (0 se
cs / 38.3 secs)
LOG: 'Spec: ImageLocalStorageService should show error message if number of stor
Chrome Headless 88.0.4324.96 (Linux x86_64): Executed 4215 of 4216 SUCCESS (0 se
LOG: 'Spec: ImageLocalStorageService should set and clear the thumbnail backgrou
nd color has passed'
Chrome Headless 88.0.4324.96 (Linux x86_64): Executed 4215 of 4216 SUCCESS (0 se
cs / 38.308 secs)
LOG: 'Spec: ImageLocalStorageService should set and clear the thumbnail backgrou
Chrome Headless 88.0.4324.96 (Linux x86_64): Executed 4216 of 4216 SUCCESS (0 se
Chrome Headless 88.0.4324.96 (Linux x86_64): Executed 4216 of 4216 SUCCESS (44.9
77 secs / 38.315 secs)
TOTAL: 4216 SUCCESS
TOTAL: 4216 SUCCESS
07 03 2021 19:29:40.720:WARN [launcher]: ChromeHeadless was not killed in 2000 m
s, sending SIGKILL.
Done!
oppia (ImageUploader) X |
```

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

```
ashutosh@omen:~/Desktop/oppia_org/oppia
[21:37:38] I/hosted - Using the selenium server at http://localhost:4444/wd/hub
Started
Jasmine started
ERROR    2021-03-08 16:08:22,680 email_manager.py:48] This app cannot send emails to users.
.
  Learner dashboard functionality
    ? should visit the exploration player and play the correct exploration
ERROR    2021-03-08 16:09:19,876 email_manager.py:48] This app cannot send emails to users.
.    ? should visit the collection player and play the correct collection
ERROR    2021-03-08 16:10:54,464 email_manager.py:48] This app cannot send emails to users.
ERROR    2021-03-08 16:11:18,890 email_manager.py:48] This app cannot send emails to users.
.    ? should display incomplete and completed explorations
ERROR    2021-03-08 16:13:45,683 email_manager.py:48] This app cannot send emails to users.
ERROR    2021-03-08 16:14:09,668 email_manager.py:48] This app cannot send emails to users.
.    ? should display incomplete and completed collections

4 specs, 0 failures
Finished in 458.783 seconds

Executed 4 of 4 specs SUCCESS in 7 mins 39 secs.
[21:45:18] I/launcher - 0 instance(s) of WebDriver still running
[21:45:18] I/launcher - chrome #01 passed

i emulators: Received SIGTERM for the first time. Starting a clean shutdown.
```

Other summer obligations

I have end semester examinations from **12 June to 16 June** in my college. After that I will be completely free till August 2021. I don't have other obligations.

Communication channels

I am active on **Gitter, hangout and discord**. I can be flexible with any platform my mentor likes. Meetings can be held bi-weekly to discuss progress and workflow to be followed ahead.

Application to multiple orgs

I'm only applying to Oppia organization this year

I'm applying for the **Improve Frontend Type System** project other than this project.

My preferences are:

1. Angular Migration
2. Improve Frontend Type System

Project Details

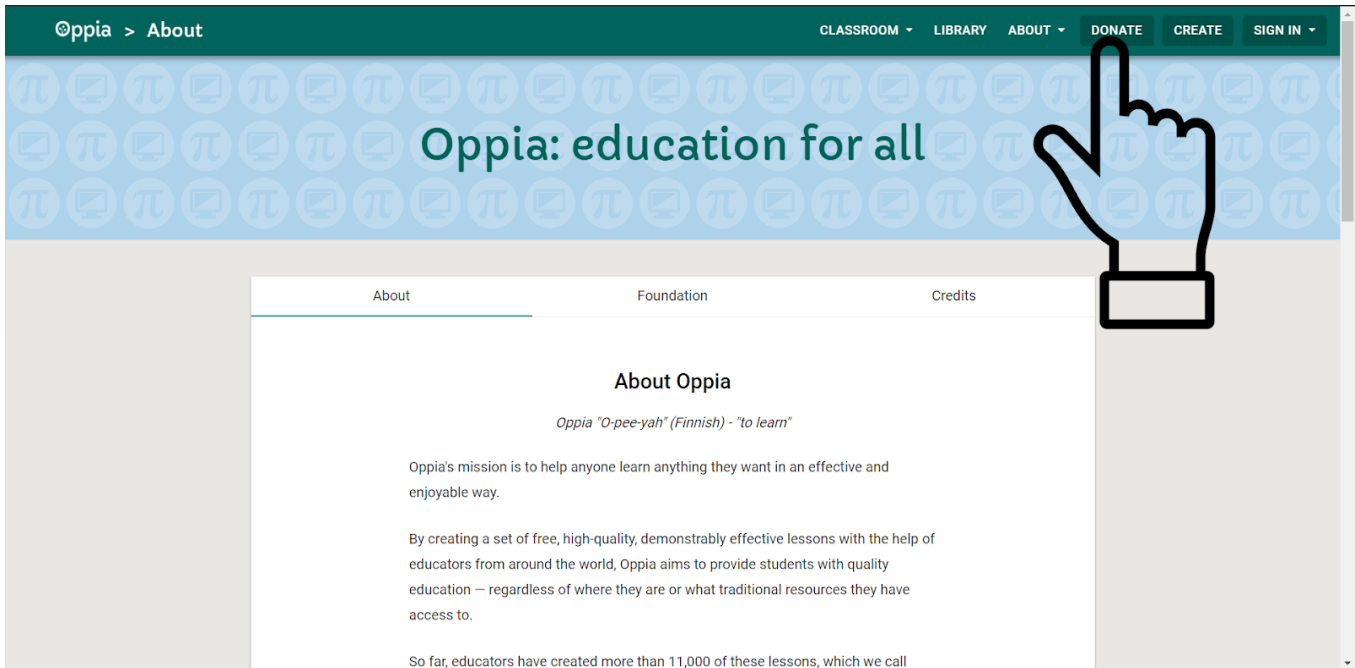
Product Design

For Users:

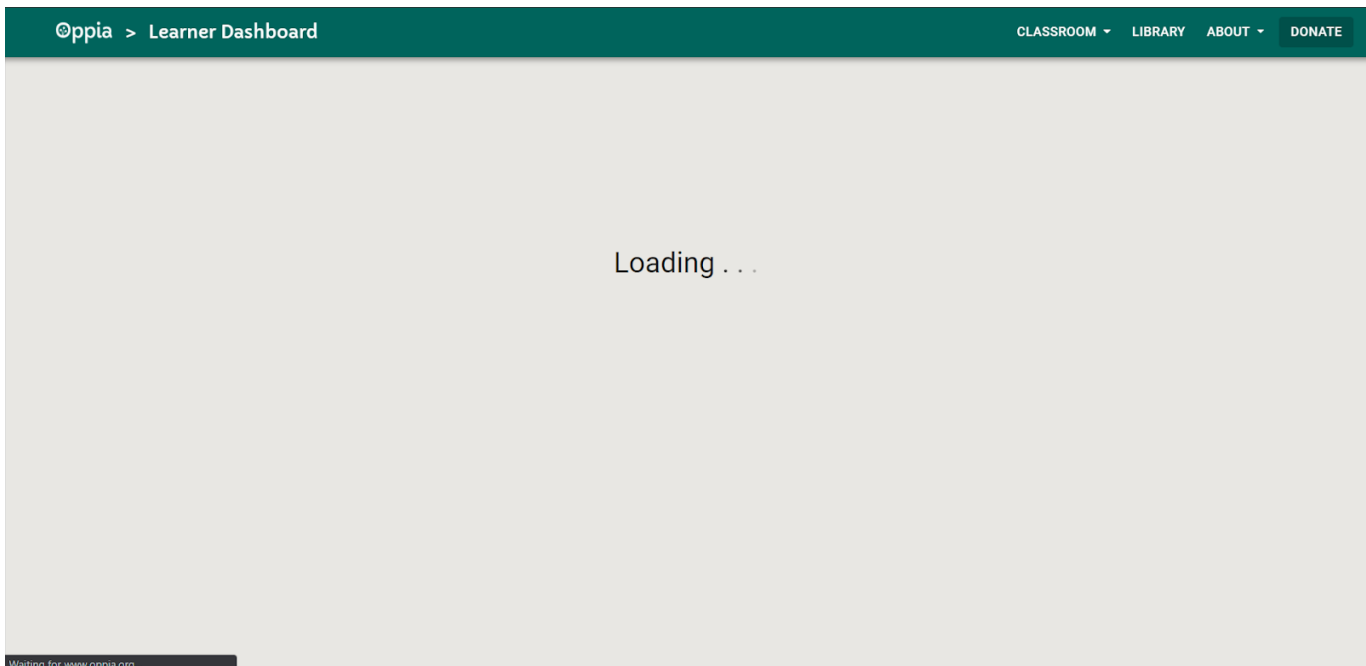
After the completion of this project, the end user will be able to change the view of the application, without reloading and rebuilding of the angular application on a page change.

Example:

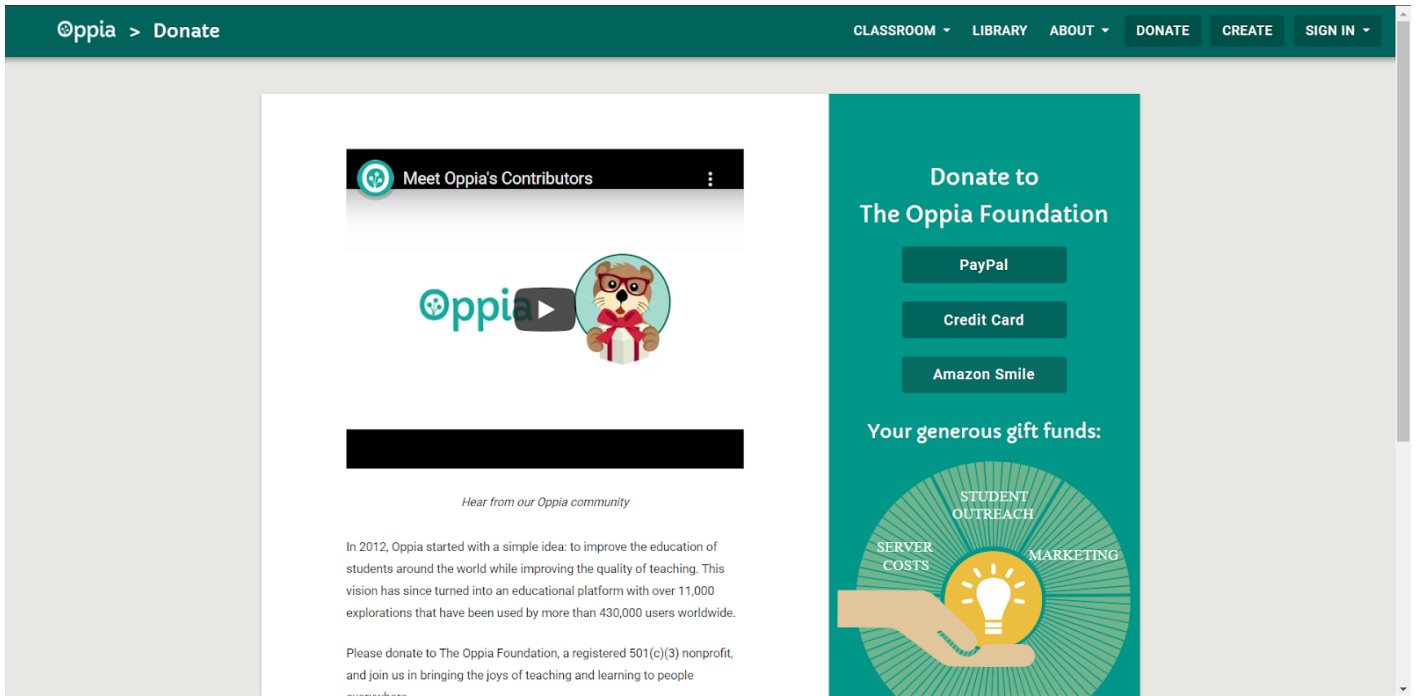
1. User clicks on the donate button in the navigation bar.



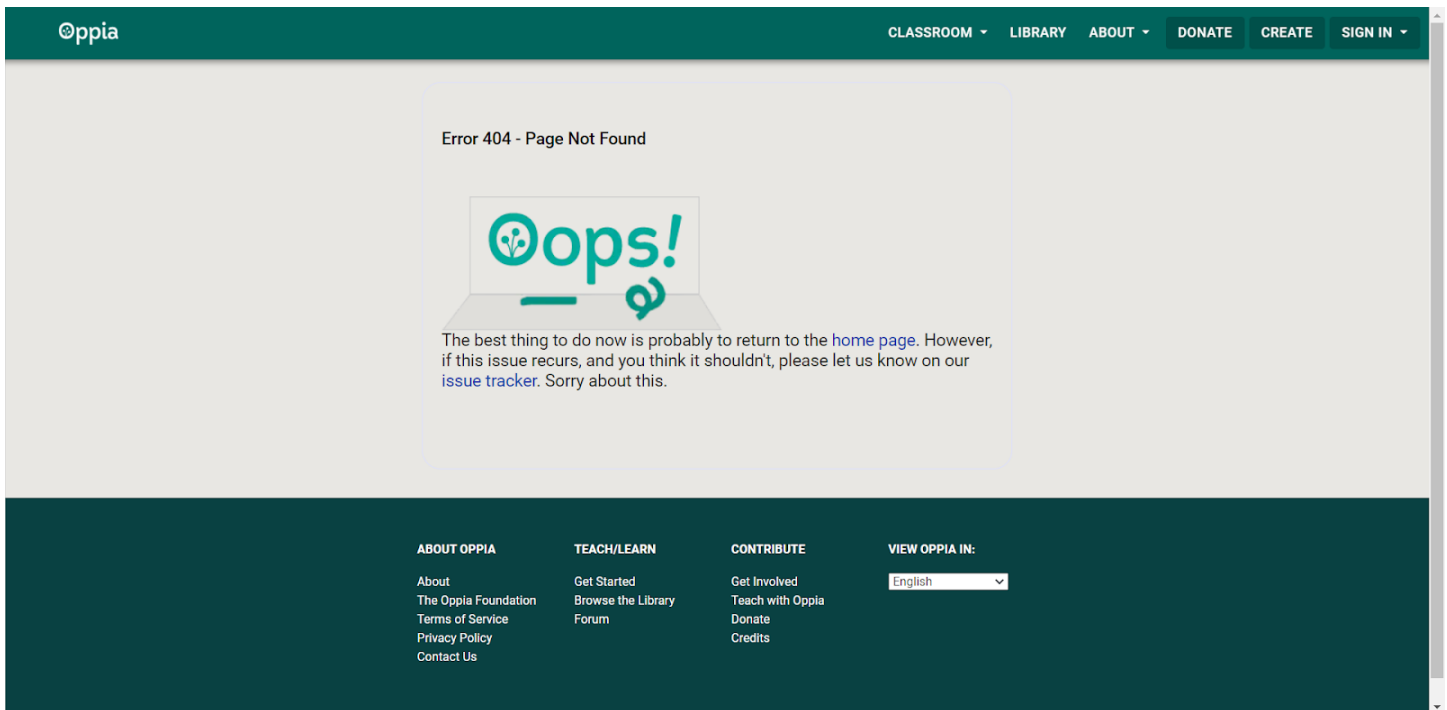
2. Oppia does not reload, only a section of the webpage is changing.



3. Users would reach the donate page, without reloading and rebooting of a new app.



When a user tries to visit a unauthorised page, he will be redirected to the not found page.

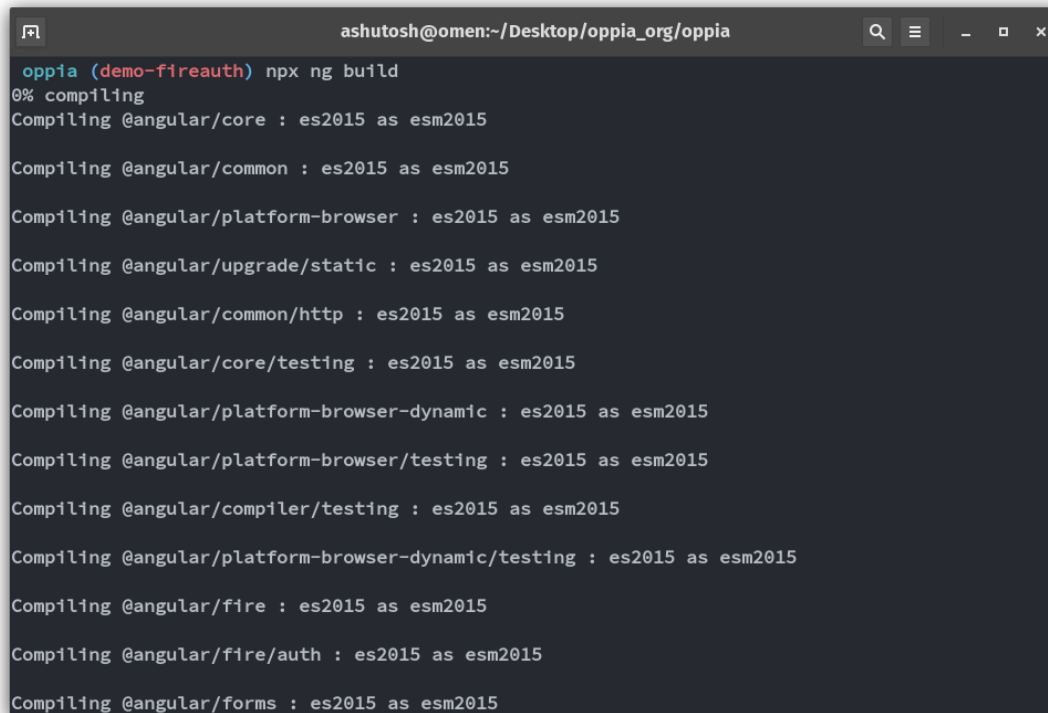


If a user tries to access a page like profile which requires him to be logged in, he will be redirected to the login page.

For developers:

The frontend build process will be simplified for the developers.

- They will be able to build the frontend using **ng build** command and can add **--prod** flag to build in production mode.



```
ashutosh@omen:~/Desktop/oppia_org/oppia
oppia (demo-fireauth) npx ng build
0% compiling
Compiling @angular/core : es2015 as esm2015

Compiling @angular/common : es2015 as esm2015

Compiling @angular/platform-browser : es2015 as esm2015

Compiling @angular/upgrade/static : es2015 as esm2015

Compiling @angular/common/http : es2015 as esm2015

Compiling @angular/core/testing : es2015 as esm2015

Compiling @angular/platform-browser-dynamic : es2015 as esm2015

Compiling @angular/platform-browser/testing : es2015 as esm2015

Compiling @angular/compiler/testing : es2015 as esm2015

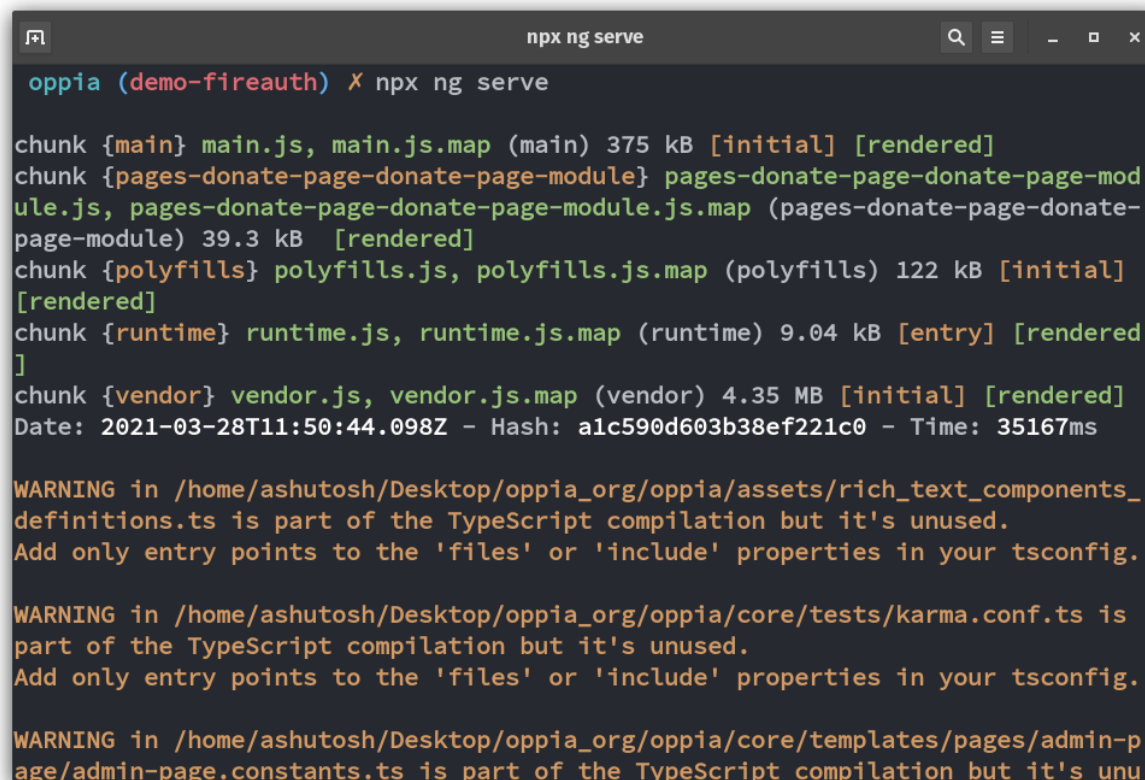
Compiling @angular/platform-browser-dynamic/testing : es2015 as esm2015

Compiling @angular/fire : es2015 as esm2015

Compiling @angular/fire/auth : es2015 as esm2015

Compiling @angular/forms : es2015 as esm2015
```

- They will be able spin up a development server for frontend using **ng serve** and can add **--prod** flag to build and serve in **production** mode.



```
npx ng serve
oppia (demo-fireauth) X npx ng serve

chunk {main} main.js, main.js.map (main) 375 kB [initial] [rendered]
chunk {pages-donate-page-donate-page-module} pages-donate-page-donate-page-mod
ule.js, pages-donate-page-donate-page-module.js.map (pages-donate-page-donate-
page-module) 39.3 kB [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 122 kB [initial]
[rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 9.04 kB [entry] [rendered]
]
chunk {vendor} vendor.js, vendor.js.map (vendor) 4.35 MB [initial] [rendered]
Date: 2021-03-28T11:50:44.098Z - Hash: a1c590d603b38ef221c0 - Time: 35167ms

WARNING in /home/ashutosh/Desktop/oppia_org/oppia/assets/rich_text_components_
definitions.ts is part of the TypeScript compilation but it's unused.
Add only entry points to the 'files' or 'include' properties in your tsconfig.

WARNING in /home/ashutosh/Desktop/oppia_org/oppia/core/tests/karma.conf.ts is
part of the TypeScript compilation but it's unused.
Add only entry points to the 'files' or 'include' properties in your tsconfig.

WARNING in /home/ashutosh/Desktop/oppia_org/oppia/core/templates/pages/admin-p
age/admin-page.constants.ts is part of the TypeScript compilation but it's unu
```

- They will be able to run **eslint** with **ng lint** command

```

ashutosh@omen:~/Desktop/oppia_org/oppia
oppia (demo-firebase) X npx ng lint

Linting "oppia"...

/home/ashutosh/Desktop/oppia_org/oppia/core/templates/components/common-layout-directives/navigation-bars/top-navigation-bar.directive.ts
 64:1  error  Expected indentation of 12 spaces but found 14  indent

/home/ashutosh/Desktop/oppia_org/oppia/core/templates/guards/auth.guard.ts
 33:7  error  'MockUserService' is defined but never used  @typescript-eslint/no-unused-vars

/home/ashutosh/Desktop/oppia_org/oppia/core/templates/pages/about-page/about-page.import.ts
 22:1  error  Comments should not begin with a lowercase character  capitalized-comments
 31:1  error  Comments should not begin with a lowercase character  capitalized-comments

/home/ashutosh/Desktop/oppia_org/oppia/core/templates/pages/about-page/about-page.module.ts
 19:27  error  'ChangeDetectorRef' is defined but never used  unused-imports/no-unused-imports-ts
 19:27  error  'ChangeDetectorRef' is defined but never used  @typescript-eslint/no-unused-vars
 25:1  error  Comments should not begin with a lowercase character  capitalized-comments
 77:4  error  .catch is a syntax error  dot-notation
 77:17  error  Unexpected console statement  no-console

/home/ashutosh/Desktop/oppia_org/oppia/core/templates/pages/app-page/app.routing.module.ts
 27:1  error  Trailing spaces not allowed  no-trailing-spaces

```

- They will be able to run end to end tests using **ng e2e**

```

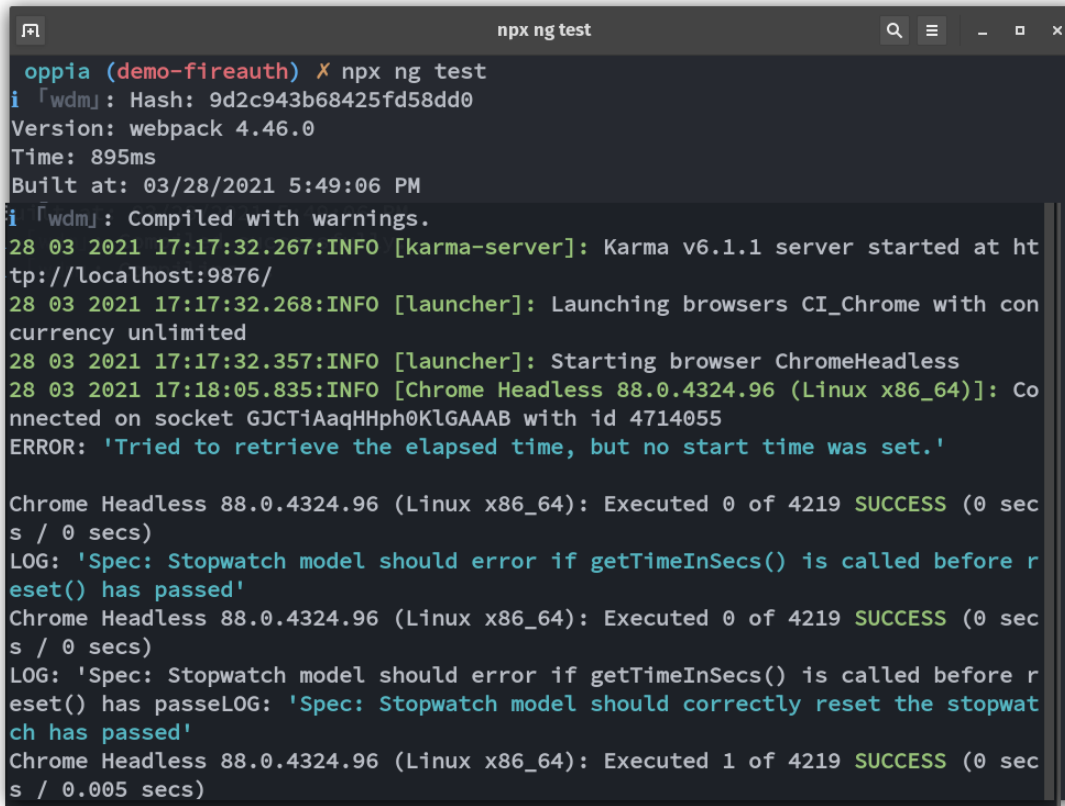
ashutosh@omen:~/Desktop/oppia_org/oppia
oppia (demo-firebase) X npx ng e2e
[17:22:35] I/update - chromedriver: file exists /home/ashutosh/Desktop/oppia_org/oppia/node_modules/webdriver-manager/selenium/chromedriver_89.0.4389.23.zip
[17:22:35] I/update - chromedriver: unzipping chromedriver_89.0.4389.23.zip
[17:22:35] I/update - chromedriver: setting permissions to 0755 for /home/ashutosh/Desktop/oppia_org/oppia/node_modules/webdriver-manager/selenium/chromedriver_89.0.4389.23
[17:22:35] I/update - chromedriver: chromedriver_89.0.4389.23 up to date

chunk {main} main.js, main.js.map (main) 375 kB [initial] [rendered]
chunk {pages-donate-page-donate-page-module} pages-donate-page-donate-page-module.js, pages-donate-page-donate-page-module.js.map (pages-donate-page-donate-page-module) 39.3 kB [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 122 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 9.04 kB [entry] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 4.35 MB [initial] [rendered]
Date: 2021-03-28T11:53:10.972Z - Hash: a1c590d603b38ef221c0 - Time: 34001ms

WARNING in /home/ashutosh/Desktop/oppia_org/oppia/assets/rich_text_components_definitions.ts is part of the TypeScript compilation but it's unused.
Add only entry points to the 'files' or 'include' properties in your tsconfig.

```


- They will be able to run frontend tests with **ng test** command



```
npx ng test
oppla (demo-fireauth) X npx ng test
i [wdm]: Hash: 9d2c943b68425fd58dd0
Version: webpack 4.46.0
Time: 895ms
Built at: 03/28/2021 5:49:06 PM
i [wdm]: Compiled with warnings.
28 03 2021 17:17:32.267:INFO [karma-server]: Karma v6.1.1 server started at http://localhost:9876/
28 03 2021 17:17:32.268:INFO [launcher]: Launching browsers CI_Chrome with concurrency unlimited
28 03 2021 17:17:32.357:INFO [launcher]: Starting browser ChromeHeadless
28 03 2021 17:18:05.835:INFO [Chrome Headless 88.0.4324.96 (Linux x86_64)]: Connected on socket GJCTiAaqHHph0KLGAAAB with id 4714055
ERROR: 'Tried to retrieve the elapsed time, but no start time was set.'

Chrome Headless 88.0.4324.96 (Linux x86_64): Executed 0 of 4219 SUCCESS (0 secs / 0 secs)
LOG: 'Spec: Stopwatch model should error if getTimeInSecs() is called before reset() has passed'
Chrome Headless 88.0.4324.96 (Linux x86_64): Executed 0 of 4219 SUCCESS (0 secs / 0 secs)
LOG: 'Spec: Stopwatch model should error if getTimeInSecs() is called before reset() has passed'
LOG: 'Spec: Stopwatch model should correctly reset the stopwatch has passed'
Chrome Headless 88.0.4324.96 (Linux x86_64): Executed 1 of 4219 SUCCESS (0 secs / 0.005 secs)
```

Following commands will also be integrated with present commands.

Following changes will be made to already present commands:

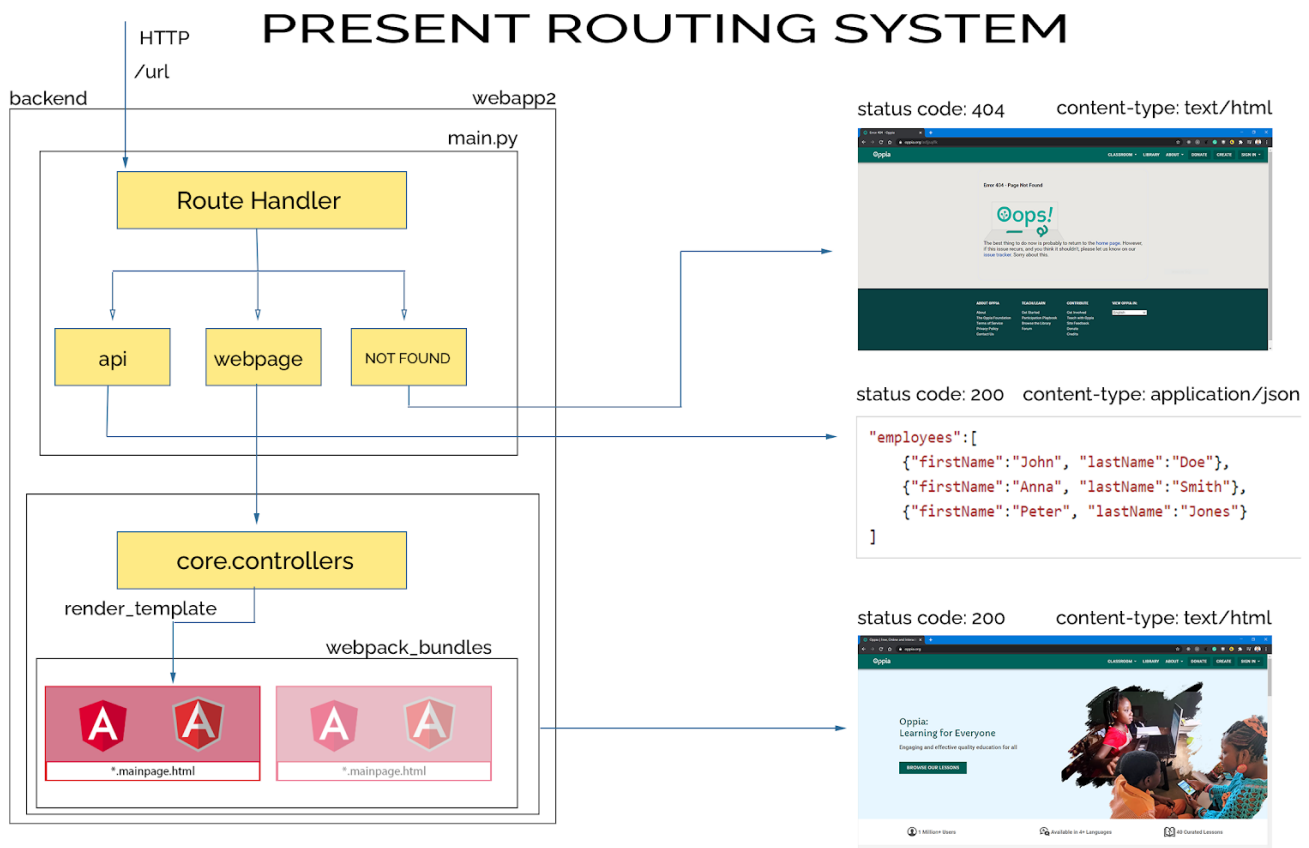
1. **python -m scripts.start:**
Instead of using webpack to build the project, this command will use **ng cli** to build the application for both **development** and **production**.
2. **python -m scripts.run_frontend_tests:**
Instead of directly using karma cli, this command will use the '**ng test**' command to run the frontend tests.
3. **python -m scripts.run_e2e_tests:**
Instead of directly using protractor, this command will use '**ng e2e**' command to run end to end tests.

Technical Design

I am dividing my project into two subprojects

SUB-PROJECT 1: Migration to angular router:

Let us first understand how the present routing system works. I will be explaining it with the help of the following illustration.



- Client sends a HTTP GET request to the backend.
- Backend searches the request url in the **URLS** list present in **main.py** and routes defined in **app.yaml** (production) and **app_dev.yaml** (development) files.
- Now there are three possibilities :
 1. **/url** is API route, in this case, backend calls the associated controller with the route and sends a **JSON response** back to the browser with an **appropriate status code**.
 2. **/url** is a route for a static web page, in this case, the backend calls the associated controller, which finds the required template in the **webpack_bundles** folder and uses **render_template** to send a webpage back to the client.
 3. **/url** does not match any of routes present in the URLs list, it sends the user not found error page by rendering 'error-page-404.mainpage.html' with status code: **404**.

Relevant Code:

```
base.py  
413
```

```
self.render_template('maintenance-page.mainpage.html')
```

```

414         else:
415             self.render_template(
416                 'error-page-%s.mainpage.html' % values['status_code'])
417     else:
418         if return_type != feconf.HANDLER_TYPE_JSON and (

```

Pros of present system:

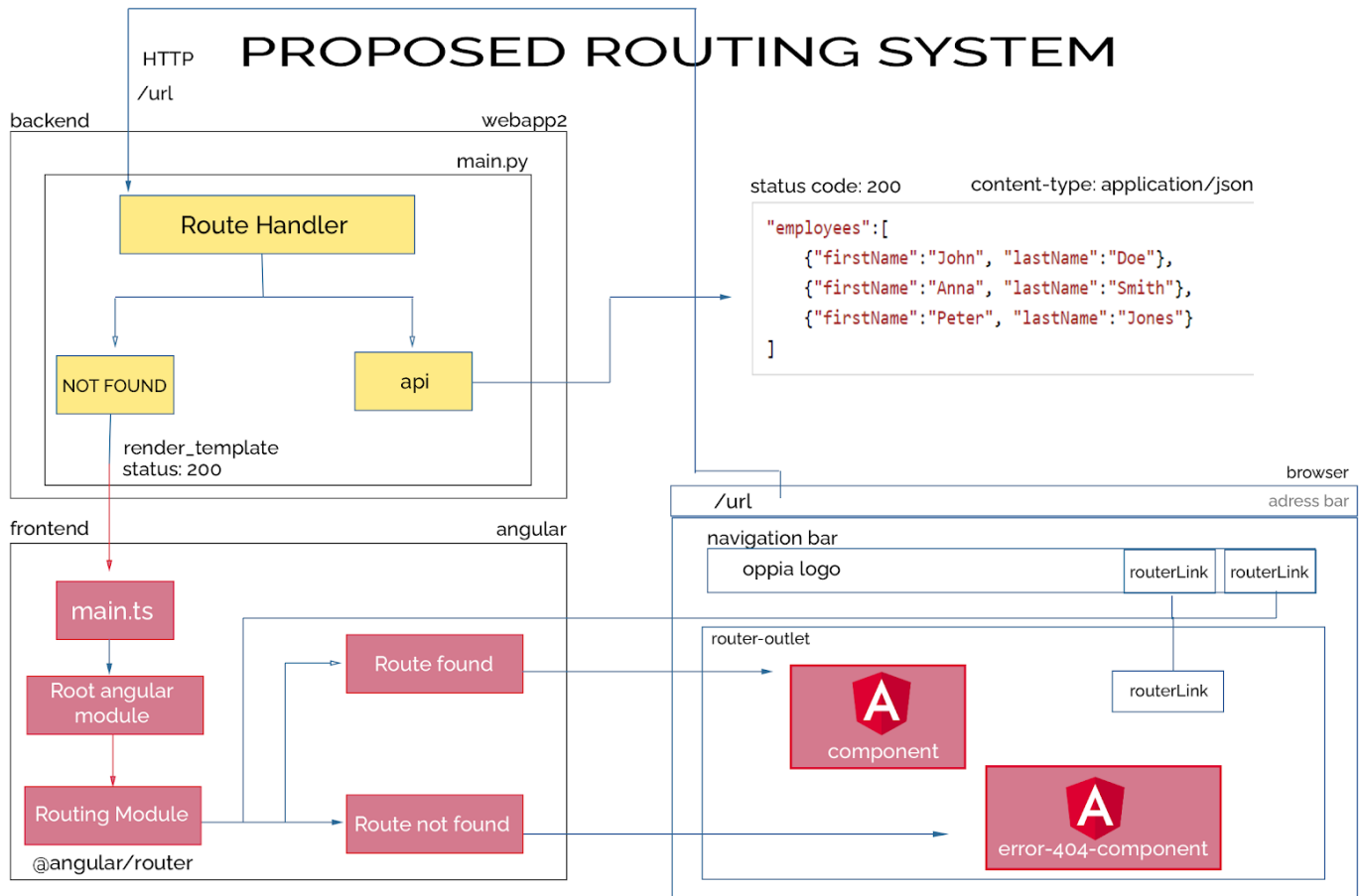
1. This system only sends the data required for a single page, nothing less or more.
2. As this system was the standard for a long time, search engine are optimized for the web pages that are coming from a server

Cons of present system:

1. This model leads to slow navigation between pages for the users. On every request a new angular application is bootstrapped by the angular’s runtime compiler which leads to a slower user experience.
2. This puts more load on the client's cpu and increases the overall size of the application. This affects mobile devices more as they have generally less powerful cpus.

Now let us understand the proposed routing system.

I will be explaining it with the help of the following illustration



- In this modal, only API routes are handled by the backend. The request which does not match any backend route, is then handed over to the frontend router.
- Then the frontend router matches the request with its list of defined routes, if the request is matched, it displays the associated component to the user, if it does not match with any route, it shows the not found component.
- This model uses routerLinks for navigation between pages
- When the user clicks a routerLink the URL changes but request to the server is prevented.
- The change of URL results in a change of state of application. This could be rendering of a new component and even requesting the api to fetch data.
- The whole page won't refresh, the libraries and components which were available in the previous view are reused again instead of being fetched again from the server.
- Only data which is required and is not available on the client is requested from the server through AJAX requests.
- Uses lazy loading to fetch angular modules only when they are required.

Pros:

1. Less data is processed, navigation between views is faster.
2. CSS and Javascript libraries are only fetched on first load.
3. Reduced load on the server.
4. Smooth animations can be added for view changes.

Cons:

1. If lazy loading is not implemented, the whole application needs to be loaded on the first request which usually takes longer.
2. Search engines are not nearly as efficient as they are for the present routing system.
This can be solved by using Interaction Observer API and Server Side Rendering. I will add SSR after GSoC as a future project.

Why lazy loading?

By default modules are eagerly loaded, which means as soon as app loads, all NgModules are loaded too, irrespective of whether they are required or not. For larger apps, lazy loading provides better loading times for apps, as it makes **initial bundle size smaller**.

These are the conceptual benefits, lets see, how well do they practically?

To find this, I conducted a survey.

The main objective of my survey is to find which approach provides better user experience for users when they hop from one page to another in a website.

The participants of this survey are students of my university, who are regular users of the web.

I used these two famous websites as examples:

1. **Wikipedia** : Uses backend routing, does full refresh on page changes.
2. **Angular.io** : Uses angular router and uses animations during page changes.

Total **27 students** participated in the survey.

Results:

1. **85.2%** participants like partial view change of application more than full page refresh.
2. **74.1%** participants liked animations during view change of application.

Conclusion:

The proposed strategy does not only provide better performance but also better user experience. [Here](#) is the link to the survey form and [here](#) is the link to the survey data.

How will Oppia benefit?

When a user first visits Oppia, bundles required for that page are downloaded on his browser. Then angular compiles the templates and modules to build an application. This process of Just In Time compilation takes some time and uses cpu for processing. When the user jumps to another page, again a new application is built and cpu is utilized. When we shift to the angular router, this **recreation of application will no longer be required**. This will reduce user's cpu usage and the change of the view of application will be **quick**. This will result in **faster user experience**.

Implementation Strategy:

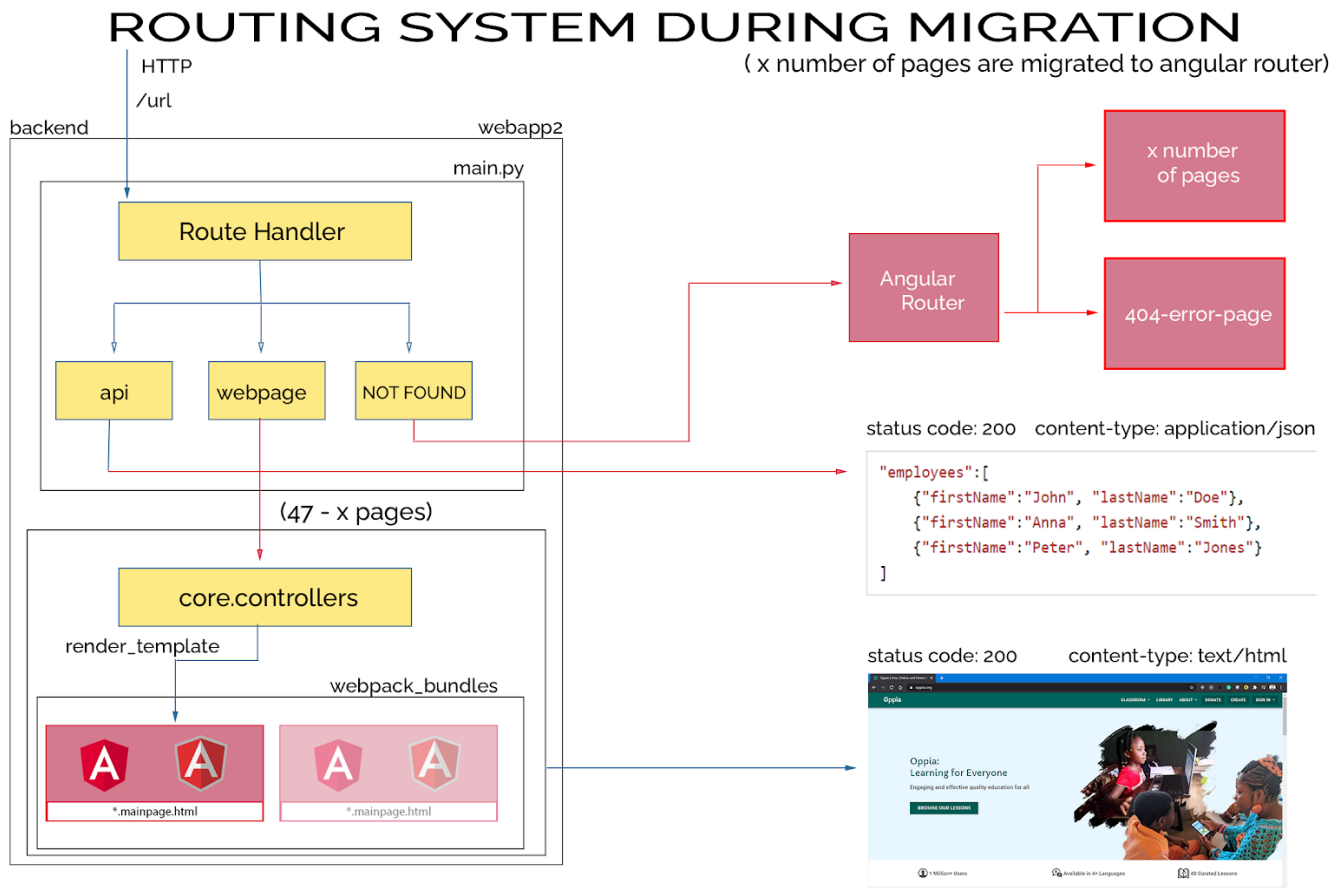
I plan to migrate to the angular router in three phases.

1. Dual router setup
2. Preparation of modules
3. Actual migration and cleanup

Phase I : Dual Router Setup

I will migrate to angular router incrementally (one page at a time), so that my PRs remain small which will result in easier PR creation and review.

To achieve that, I have planned a migration flow. I will be explaining that using the following illustration.

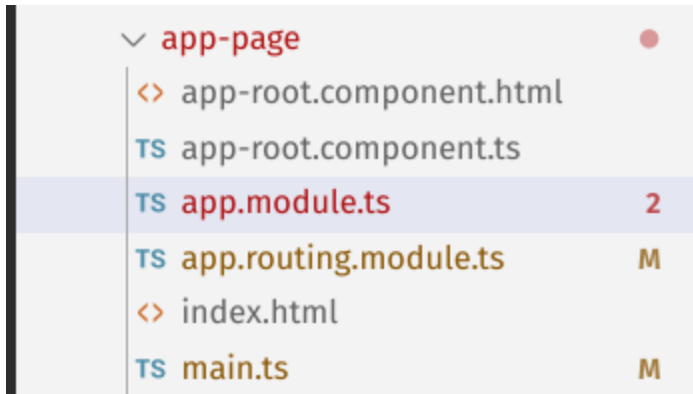


- During migration, some of the pages will be handled through angular router (let say, x) and rest will be handled through the previous model (total - x).
- To achieve that, I will create a new page (this page will become the unified entry to the application after the router migration), a root module and a root routing module. I will add this page to the webpack entry list.

webpack.common.config.ts

```
59  entry: {
60    about: commonPrefix + '/pages/about-page/about-page.import.ts',
61    admin: commonPrefix + '/pages/admin-page/admin-page.import.ts',
62    app: commonPrefix + '/pages/app-page/main.ts',
63    classroom:
64      commonPrefix + '/pages/classroom-page/classroom-page.import.ts',
65    collection_editor:
66      commonPrefix + '/pages/collection-editor-page/' +
67        'collection-editor-page.import.ts',
68    collection_player:
69      commonPrefix + '/pages/collection-player-page/' +
70
71      ...
72
73  plugins: [
74    new HtmlWebpackPlugin({
75      chunks: ['app'],
76      filename: 'index.mainpage.html',
77      meta: {
78        name: defaultMeta.name,
79        description: 'With Oppia, you can access free lessons on ' +
80          'math, physics, statistics, chemistry, music, history and ' +
81          'more from anywhere in the world. Oppia is a nonprofit ' +
82          'with the mission of providing high-quality ' +
83          'education to those who lack access to it.'
84      },
85      template: commonPrefix + '/pages/app-page/index.html',
86      minify: htmlMinifyConfig,
87      inject: false
88    }),
89  ],
90
```

I will add following files to create this new page:



- I will use the existing error module for all not found routes.
- I will register error module to the root routing module of our new unified entry app with lazy loading for wildcard route '**'

```
1 import { APP_BASE_HREF } from '@angular/common';
2 import { NgModule } from '@angular/core';
3 import { Route, RouterModule } from '@angular/router';
4
5 const routes: Route[] = [
6   {
7     path: '**',
8     loadChildren: () => import('pages/error-pages/error-page.module')
9       .then(m => m.ErrorPageModule)
10  }
11 ];
12
13 @NgModule({
14   imports: [
15     RouterModule.forRoot(routes)
16   ],
17   exports: [
18     RouterModule
19   ],
20   providers: [
21     {
22       provide: APP_BASE_HREF,
23       useValue: '/'
24     }
25 ]
26 })
27 export class AppRoutingModule {}
```

With present typescript configuration, we cannot use import syntax for lazy loading.

This can be fixed using two methods:

1. Adding “module”: “esnext” in tsconfig.json
2. Adding “module”: “es2020” in tsconfig.json

As for the typescript ^4.0.2, which Oppia is using, both configurations work the same.

But the “esnext” can evolve in backward incompatible ways, which can cause issues in the future.

So, I will use “es2020” during this project.

To ensure that this updation of tsconfig.json will not cause any side effects:

1. I ran frontend tests with this configuration. All were successful.
 2. I ran a webpack build, it was successful without any issues
 3. I ran e2e suites, they were successful.
- Then I will set up the backend to serve this newly created page when no requests are matching with available routes.
This will be done by rendering **index.mainpage.html** for all **404** routes at the **backend**.
To implement this I will add a condition in `_render_exception_json_or_html` function present in `base.py` file.

base.py

```
397.     def _render_exception_json_or_html(self, return_type, values):
404.
405.         method = self.request.environ['REQUEST_METHOD']
406.
407.         if return_type == feconf.HANDLER_TYPE_HTML and method == 'GET':
408.             self.values.update(values)
409.             if self.iframeed:
410.                 self.render_template(
411.                     'error-iframeed.mainpage.html', iframe_restriction=None)
412.             elif values['status_code'] == 503:
413.                 self.render_template('maintenance-page.mainpage.html')
414. +             elif values['status_code'] == 404:
415. +                 self.render_template('index.mainpage.html')
416.             else:
417.                 self.render_template(
418.                     'error-page-%s.mainpage.html' % values['status_code'])
419.         else:
420.             if return_type != feconf.HANDLER_TYPE_JSON and (
421.                 return_type != feconf.HANDLER_TYPE_DOWNLOADABLE):
422.                 logging.warning(
```

Now the dual router setup is complete, 404 routes will be redirected to the frontend angular router.

Phase 2 Preparation of Modules:

2.1 Creation of Route guards.

Why do we need them?

Some pages like admin-page are not supposed to be accessible by everyone. They should have restricted access. If we don't restrict access, users can open restricted pages however they can't do anything on that page because api requests are secured at the backend. This will lead to bad user experience.

At present blocking of unauthorized users from accessing a certain page is handled at the backend, but after migration this should be handled in the frontend.

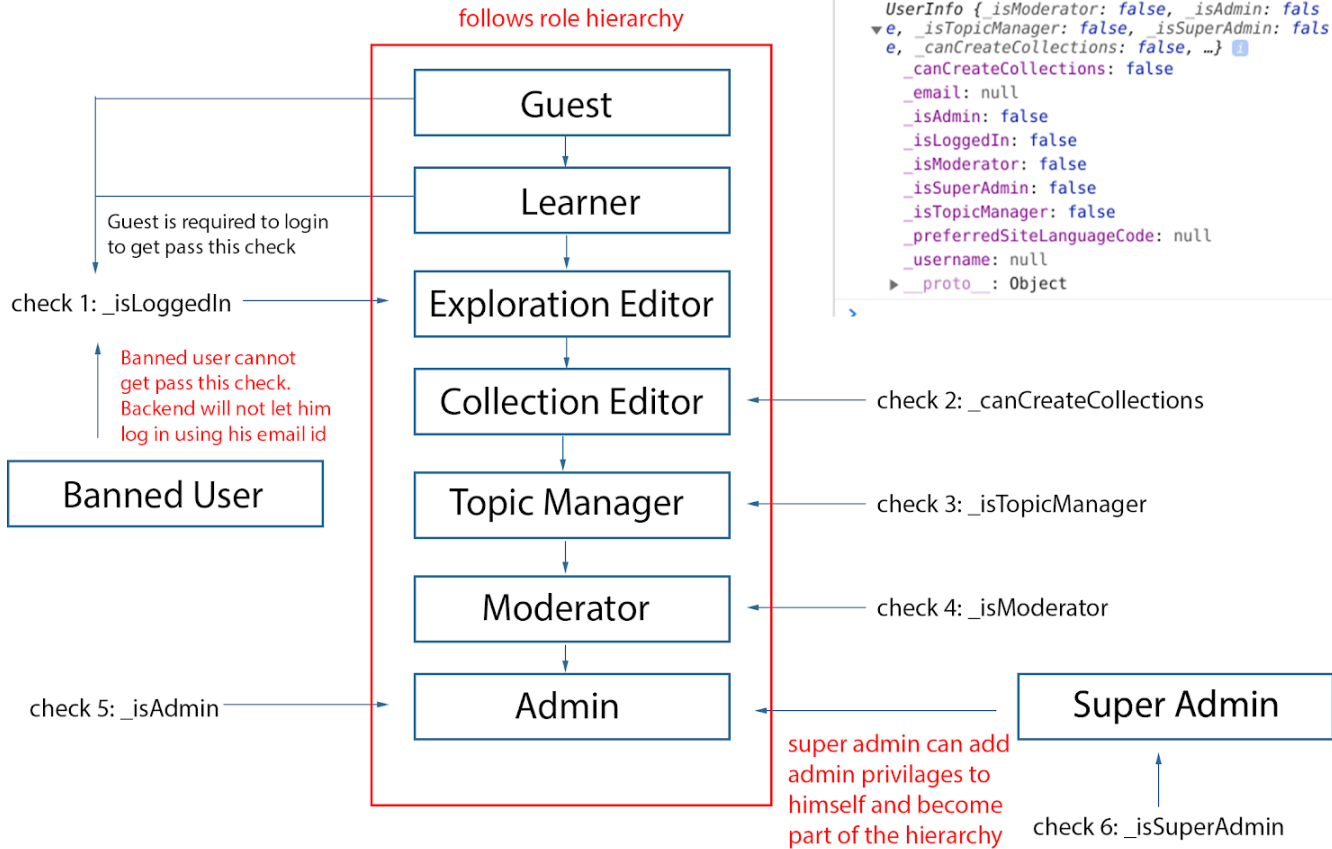
I will implement frontend route protection using this route guard:

```
const ROLE_PERMISSION_LEVELS: ROLE_PERMISSION_LEVELS_DICT = {
  USER: 0,
  COLLECTION_EDITOR: 1,
  TOPIC_MANAGER: 2,
  MODERATOR: 3,
  ADMIN: 4,
  SUPERADMIN: 5
};
```

```
@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanLoad {
  constructor(
    private userService: UserService,
    private router: Router,
    private windowRef: WindowRef
  ) {}
```

```
canLoad(route: Route, segments: UrlSegment[]): boolean | UrlTree |
Promise<boolean | UrlTree> | Observable<boolean | UrlTree> {
  return this.userService.getUserInfoAsync().then((user) => {
    if (route.data.minRole <= ROLE_PERMISSION_LEVELS.USER) {
      if (user.isLoggedIn()) { # Check 1
        return true;
      } else {
        this.windowRef.nativeWindow.location.assign('/login?return_url=%2F');
        return false;
      }
    }
  });
}
```

```
    }  
  }  
  
  if (route.data.minRole <= ROLE_PERMISSION_LEVELS.COLLECTION_EDITOR &&  
      user.canCreateCollections()) { # Check 2  
    return true;  
  }  
  
  if (route.data.minRole <= ROLE_PERMISSION_LEVELS.TOPIC_MANAGER &&  
      user.isTopicManager()) { # Check 3  
    return true;  
  }  
  
  if (route.data.minRole <= ROLE_PERMISSION_LEVELS.MODERATOR &&  
      user.isModerator()) { # Check 4  
    return true;  
  }  
  if (route.data.minRole <= ROLE_PERMISSION_LEVELS.ADMIN &&  
      user.isAdmin()) { # Check 5  
    return true;  
  }  
  if (route.data.minRole <= ROLE_PERMISSION_LEVELS.SUPERADMIN &&  
      route.data.minRole >= ROLE_PERMISSION_LEVELS.ADMIN &&  
      user.isSuperAdmin()) { # Check 6  
    return true;  
  }  
  
  this.router.navigate(['not-found']);  
  return false;  
}, (error) => {  
  return false;  
});  
}
```



Following are the characteristics of the route guard:

- Auth guard follows the role hierarchy.
- Auth guard redirects the user to the login page, if the route requires the user to be logged in. Currently it is using location api to redirect but it will be moved to router.navigate(['/login']), once the login page is migrated to the angular router.
- Auth guard redirects the user to not found page (404) page, if the user does not have sufficient rights.

Usage:

```

const routes: Route[] = [
  {
    path: 'donate',
    loadChildren: () => import('pages/donate-page/donate-page.module')
      .then(m => m.DonatePageModule)
  },
  {
    path: 'profile',
    loadChildren: () => import('pages/profile-page/profile-page.module')
      .then(m => m.ProfilePageModule),
    canActivate: [AuthGuard],
    data: { minRole : ROLE_PERMISSION_LEVELS.USER }
  },
]
  
```

```

{
  path: 'moderator',
  loadChildren: () => import('pages/moderator-page/moderator-page.module')
    .then(m => m.ModeratorPageModule),
  canLoad: [AuthGuard],
  data: { minRole: ROLE_PERMISSION_LEVELS.MODERATOR }
},
{
  path: 'admin',
  loadChildren: () => import('pages/admin-page/admin-page.module')
    .then(m => m.AdminPageModule),
  canLoad: [AuthGuard],
  data: { minRole: ROLE_PERMISSION_LEVELS.ADMIN }
},

```

- **/profile** route can be accessed if the user is logged in.
- **/admin** route can be accessed by admin and super admin.
- **/moderator** route can be accessed by moderator and admin (**superadmin cannot access this page, which exactly replicates the current behaviour**).

Note: Superadmin is a special role, which does not follow the role hierarchy. Superadmin cannot access topic manager and collection creator pages. But he can add these roles to himself by accessing the admin page.

Following table shows what role is required for what page.

PAGE	MINIMUM ROLE REQUIRED
about	No role required
admin	Admin
classroom	No role required
collection_editor	Collection Editor
collection_player	No role required
contact	No role required
creator_dashboard	User
contributor_dashboard	User
delete_account	User
donate	No role required
email_dashboard	Admin

Error page	No role required
exploration_editor	Exploration Editor
exploration_player	No role required
get_started	No role required
learner-dashboard	User
library	No role required

maintenance	No role required
moderator	Moderator
notifications_dashboard	User
pending_account_deletion	User
privacy	No role required
preferences	User
profile	User
signup	User
skill-editor	Topic Editor
splash	No role required
subtopic_viewer	Topic Editor
teach	No role required
terms	No role required
thanks	No role required

topic_editor	Topics Editor
topics_and_skills_dashboard	Topics Editor
topic_viewer	Topics Editor

There is a **“Refactor role structure & add new roles”** project proposition by Sandeep Dubey under consideration.

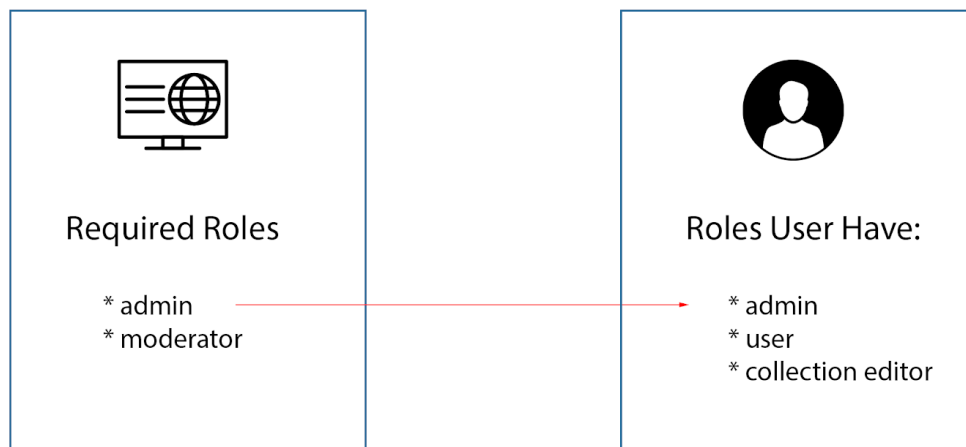
If that project is completed before/during the GSoC period, I will change the architecture of the route guard from hierarchical based to multi roles based route protection to support his changes.

The role refactor project will add three new roles to oppia:

1. **Release Coordinator:** These grants access to run jobs on the MR job dashboard and perform Memcache operations.
Associated pages to be added with this role:
 - MR job dashboard
2. **Voiceovers admin:** This grants the ability to manage the assignment of voiceover artists to lessons.
3. **Contributor Dashboard admin:** This grants the ability to see statistics for the contributor dashboard, and provides access to various administrative controls to tweak its operation.

Multi roles based route protection architecture:

MULTI ROLE BASED ROUTE PROTECTION



How will multi role route protection work?

- Every page will have a predefined set of roles which have access.
- Every user will have a set of assigned roles.
- The route guard will try to match roles required by a page with the roles of the user who is trying to access the page.
- If at least a single role of the user meets the requirement of the page, the page can be accessed.

I have prepared this route guard for implementing multi role route protection.

Auth-multi-role.guard.ts

```
export const ROLES = {  
  USER: 'user',  
  COLLECTION_EDITOR: 'collection_editor',
```

```
TOPIC_MANAGER: 'topic_manager',
MODERATOR: 'moderator',
ADMIN: 'admin',
SUPERADMIN: 'superadmin',
RELEASE_COORDINATOR: 'release_coordinator',
VOICEOVERS_ADMIN: 'voiceovers_admin',
CONTRIBUTOR_DASHBOARD_ADMIN: 'contributor_dashboard_admin'
};
```

```
export class AuthGuard implements CanLoad, CanActivate {
  constructor(
    private userService: UserService,
    private router: Router,
    private windowRef: WindowRef
  ) {}
```

```
  canLoad(route: Route, segments: UrlSegment[]): boolean | UrlTree |
  Promise<boolean | UrlTree> | Observable<boolean | UrlTree> {
    return this.userService.getUserInfoAsync().then((user) => {
      if (route.data.roles.indexOf('user') > -1 &&
        route.data.roles.length === 1) {
        if (user.isLoggedIn()) {
          return true;
        } else {
          this.windowRef.nativeWindow.location.assign('/login?return_url=%2F');
          return false;
        }
      }
    })
  }
```

```
  for (let i = 0; i < route.data.roles.length; i++) {
    if (route.data.roles[i] === ROLES.COLLECTION_EDITOR &&
      user.canCreateCollections()) {
      return true;
    }
    if (route.data.roles[i] === ROLES.TOPIC_MANAGER &&
      user.isTopicManager()) {
      return true;
    }
    if (route.data.roles[i] === ROLES.VOICEOVERS_ADMIN &&
      user.isVoiceoversAdmin()) {
      return true;
    }
  }
```

```

    }
    if (route.data.roles[i] === ROLES.CONTRIBUTOR_DASHBOARD_ADMIN &&
        user.isContributorDashboardAdmin()) {
        return true;
    }
    if (route.data.roles[i] === ROLES.MODERATOR &&
        user.isModerator()) {
        return true;
    }
    if (route.data.roles[i] === ROLES.RELEASE_COORDINATOR &&
        user.isReleaseCoordinator()) {
        return true;
    }
    if (route.data.roles[i] === ROLES.ADMIN &&
        user.isAdmin()) {
        return true;
    }
    if (route.data.roles[i] === ROLES.SUPERADMIN &&
        user.isSuperAdmin()) {
        return true;
    }
}

this.router.navigate(['not-found']);
return false;
}, (error) => {
    return false;
});
}

```

Usage:

```

const routes: Route[] = [
  {
    path: 'donate',
    loadChildren: () => import('pages/donate-page/donate-page.module')
      .then(m => m.DonatePageModule)
  },
  {
    path: 'profile',
    loadChildren: () => import('pages/profile-page/profile-page.module')
      .then(m => m.ProfilePageModule),
  }
]

```



```

canLoad: [AuthGuard],
data: { roles : [ROLES.USER] }
},
{
  path: 'moderator',
  loadChildren: () => import('pages/moderator-page/moderator-page.module')
    .then(m => m.ModeratorPageModule),
  canLoad: [AuthGuard],
  data: { roles: [ROLES.MODERATOR, ROLES.ADMIN] }
},
{
  path: 'admin',
  loadChildren: () => import('pages/admin-page/admin-page.module')
    .then(m => m.AdminPageModule),
  canLoad: [AuthGuard],
  data: { roles: [ROLES.ADMIN, ROLES.SUPERADMIN] }
},
},

```

2.2 Preparation of modules:

The way that the modules (*-page.module.ts files) are structured now, they can't be used for lazy loading. We need to perform some changes.

Issues with present structure:

1. Lazy loading demands a routing module to be added to lazy loaded modules.
- 2.

```

26
27 <body>
28   <oppia-root>
29     <div ng-controller="Base">
30       <base-content>
31         <navbar-breadcrumb>
32           <ul class="nav navbar-nav oppia-navbar-breadcrumb">
33             <li>
34               <span class="oppia-navbar-breadcrumb-separator"></span>
35               Donate
36             </li>
37           </ul>
38         </navbar-breadcrumb>
39
40         <content>
41           <donate-page></donate-page>
42         </content>
43       </base-content>
44     </div>
45   </oppia-root>
46   @load('pages/footer_js_libs_ngx.html')
47 </body>
48 </html>
49

```

Second issue:

There are two sections of code in each *.mainpage.html files.

- One is static, all *.mainpage.html files share this section, so I will place this part in index.html of our newly created unified app's folder
- The other section which is present inside the red box in the above illustration is different for every *.mainpage.html file.

Solution to these issues:

- I will create a routing module for every page and register it with corresponding *-page.module.ts files.

```
core > templates > pages > donate-page > donate-page.routing.module.ts

1  import { NgModule } from '@angular/core';
2  import { Route, RouterModule } from '@angular/router';
3  import { DonatePageRootComponent } from './donate-page-root.component';
4
5  const routes: Route[] = [
6    {
7      path: '',
8      component: DonatePageRootComponent
9    }
10 ];
11
12 @NgModule({
13   imports: [
14     RouterModule.forChild(routes)
15   ],
16   exports: [
17     RouterModule
18   ]
19 })
20
21 export class DonatePageRoutingModule {}
```

- I will move the common section of *.mainpage.html to the index.html and will create a root component (top level) for every page containing the dynamic part. Body of index.html will look like this.

```
27 <body>
28   <oppia-root>
```

```

29     <div ng-controller="Base"> <!-- ng-controller will not be required after
        angularjs is removed from the codebase -->
30     <base-content>
31     <router-outlet></router-outlet>
32     </base-content>
33 </div>
34 </oppia-root>
    <!-- script tag for libraries -->
35 </body>

```

Root component for every page (eg demo page) will look like this -

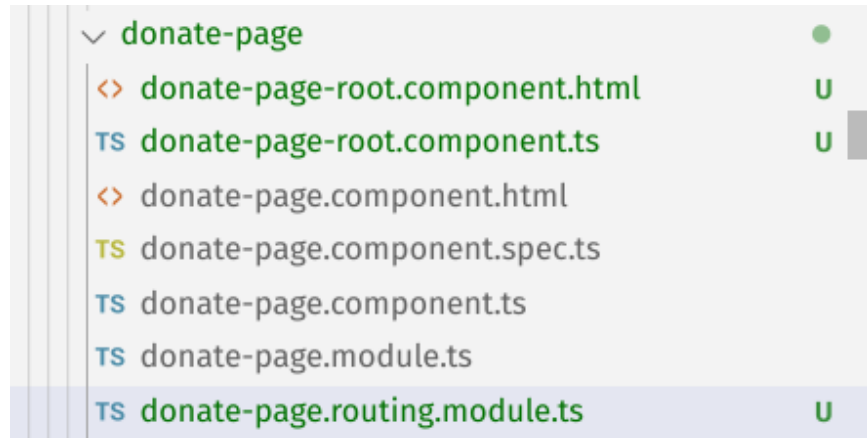
```
core > templates > pages > donate-page > donate-page-root.component.html
```

```

1 <navbar-breadcrumb>
2   <ul class="nav navbar-nav oppia-navbar-breadcrumb">
3     <li>
4       <span class="oppia-navbar-breadcrumb-seperator"></span>
5       Donate
6     </li>
7   </ul>
8 </navbar-breadcrumb>
9
10 <content>
11   <router-outlet></router-outlet>
12 </content>

```

Final folder structure for every page (eg. donate page) would look like this.



Phase 3: Actual migration and clean up

- This will be done one page at a time
- Usage of **window.location** will be replaced with **angular router**

Location Api	Using angular router
<code>this.windowRef.nativeWindow.location.href = this.classroomUrl</code>	<code>this.router.navigateByUrl(this.classroomUrl);</code>
<code>this.windowRef.nativeWindow.location.assign('/signup');</code>	<code>this.router.navigateByUrl('/signup');</code>

- Modules will be registered in root routing module using lazy loading
- I will use the already present **PageTitleService** to set titles for pages.
I will use PageTitleService in the root component of every page to set titles.

eg. Donate Page:

```
export class DonatePageRootComponent {
  constructor(
    private pageTitleService: PageTitleService
  ) {}
  ngOnInit(): void {
    this.pageTitleService.setPageTitle('Donate - Oppia');
  }
}
```

- Migrated Routes, corresponding controllers along with their tests will be removed from the backend (**related files:** main.py, base.py, app.yaml, app_dev.yaml etc)

Example 1: How will remove unused code in the backend for the donate page.

app_dev.yaml, app.yaml

```
165.     secure: always
166.     expiration: "0"
167. - - url: /donate
168. -   static_files: webpack_bundles/donate-page.mainpage.html
169. -   upload: webpack_bundles/donate-page.mainpage.html
170. -   http_headers:
171. -     Pragma: no-cache
172. -     Strict-Transport-Security: "max-age=31536000; includeSubDomains"
173. -     X-Content-Type-Options: "nosniff"
174. -     X-Frame-Options: "DENY"
175. -     X-Xss-Protection: "1; mode=block"
176. -   secure: always
```

```
178. - expiration: "0"
179. - url: /get-started
```

Example 2: How will I remove unused code for the learners dashboard page.

main.py

```
377.         r'%s' % feconf.UPLOAD_EXPLORATION_URL,
378.         creator_dashboard.UploadExplorationHandler),
379. - get_redirect_route(
380. -     '/learner_dashboard',
381. -     learner_dashboard.OldLearnerDashboardRedirectPage),
382.     get_redirect_route(
383.         r'%s' % feconf.LEARNER_DASHBOARD_URL,
```

Learner_dashboard.py

```
39.         """Handles GET requests."""
40.         self.redirect(feconf.LEARNER_DASHBOARD_URL, permanent=True)
...
43. - class LearnerDashboardPage(base.BaseHandler):
44. -     """Page showing the user's learner dashboard."""
45. -
46. -     @acl_decorators.can_access_learner_dashboard
47. -     def get(self):
48. -         """Handles GET requests."""
49. -         self.render_template('learner-dashboard-page.mainpage.html')
...
52. class LearnerDashboardHandler(base.BaseHandler):
53.     """Provides data for the user's learner dashboard page."""
```

Learner_dashboard_test.py

```
297.         self.logout()
298.
299. - def test_learner_dashboard_page(self):
300. -     self.login(self.OWNER_EMAIL)
301. -
302. -     response = self.get_html_response(feconf.LEARNER_DASHBOARD_URL)
303. -     self.assertIn('{"title": "Learner Dashboard | Oppia"}', response.body)
304. -
305. -     self.logout()
```

- Migrated page will be removed from the webpack entry list along with its HtmlWebpackPlugin entry.

HANDLING OF ROUTING SERVICES:

At present, oppia codebase uses routing services (eg router.service.ts, url.service.ts, skill-editor-routing.service.ts and topics-editor-routing.service.ts) to implement a frontend router like functionality.

After the introduction of angular router, some functions of these services will become redundant. It is best to remove redundant code.

One of the most important services here is url service:

Following table shows alternate interfaces provided by the angular router to achieve the same functionality without having to maintain that code.

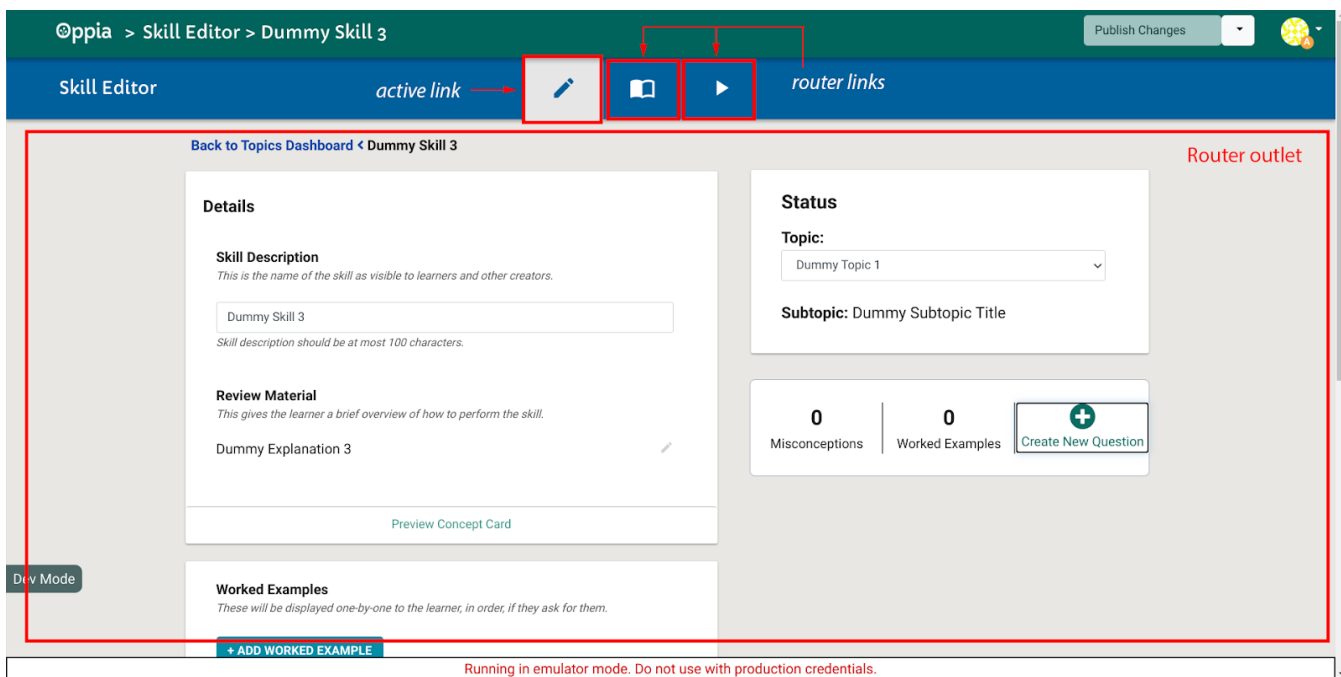
Url service	Angular router
getCurrentLocation()	Router.url
getCurrentQueryString()	ActivatedRoute.queryParams
getUrlParams()	ActivatedRoute.params
<pre>isIframed(): boolean { let pathname = this.getPathname(); let urlParts = pathname.split('/'); return urlParts[1] === 'embed'; }</pre>	<pre>isIframed(): boolean { let pathname = this.router.url; let urlParts = pathname.split('/'); return urlParts[1] === 'embed'; }</pre>
getHash()	ActivatedRoute.fragment

1. Skill editor routing service

This service act as a mini frontend router for skill editor page

The usage of this service can be replaced with the angular router.

I will set up nested routing using child routes for this page.



Implementation Details:

Following changes will be made to skill-editor-page.component.html

```

- <div ng-if="$ctrl.getActiveTabName() === 'main'">
-   <skill-editor-main-tab></skill-editor-main-tab>
- </div>

- <div ng-if="$ctrl.getActiveTabName() === 'questions'">
-   <questions-tab></questions-tab>
- </div>

- <div ng-if="$ctrl.getActiveTabName() === 'preview'">
-   <skill-preview-tab></skill-preview-tab>
- </div>
+ <router-outlet></router-outlet>

```

Following routes will be added to lazy loaded module skill-editor.routing.module.ts

```

const routes: Route[] = [
  {
    path: '',
    component: SkillEditorPageComponent,

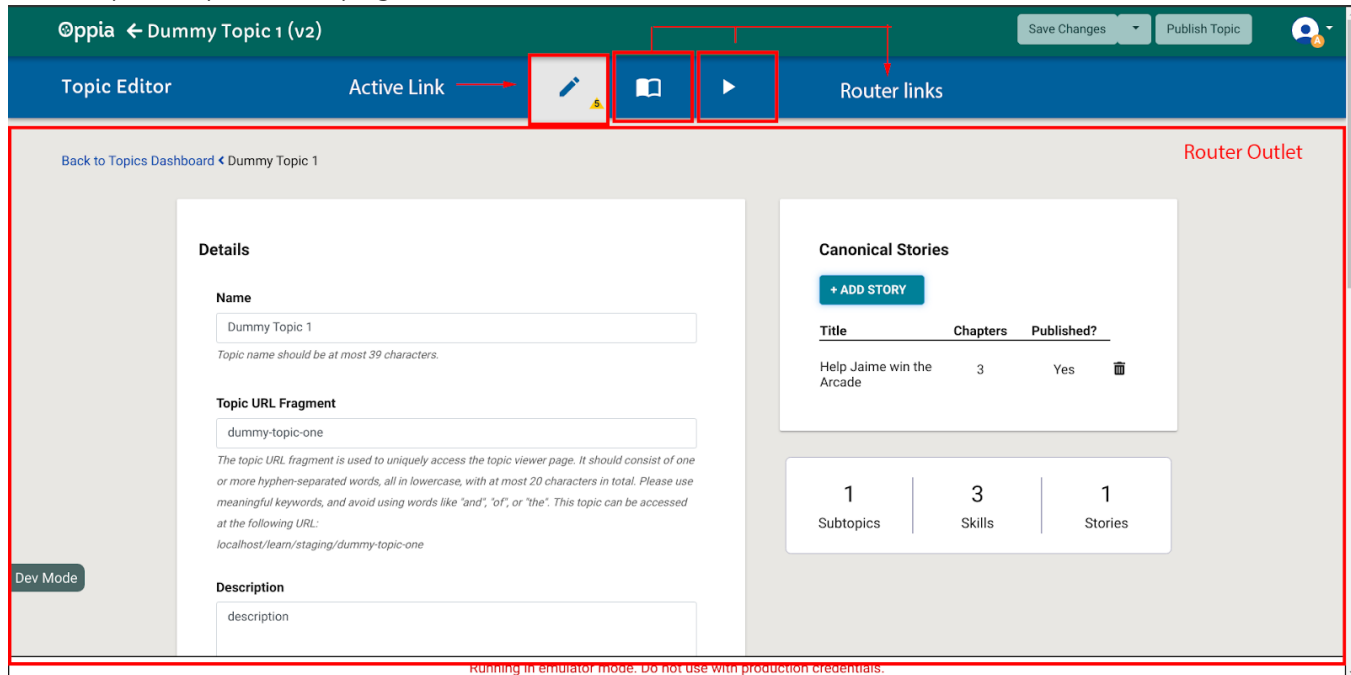
```

```

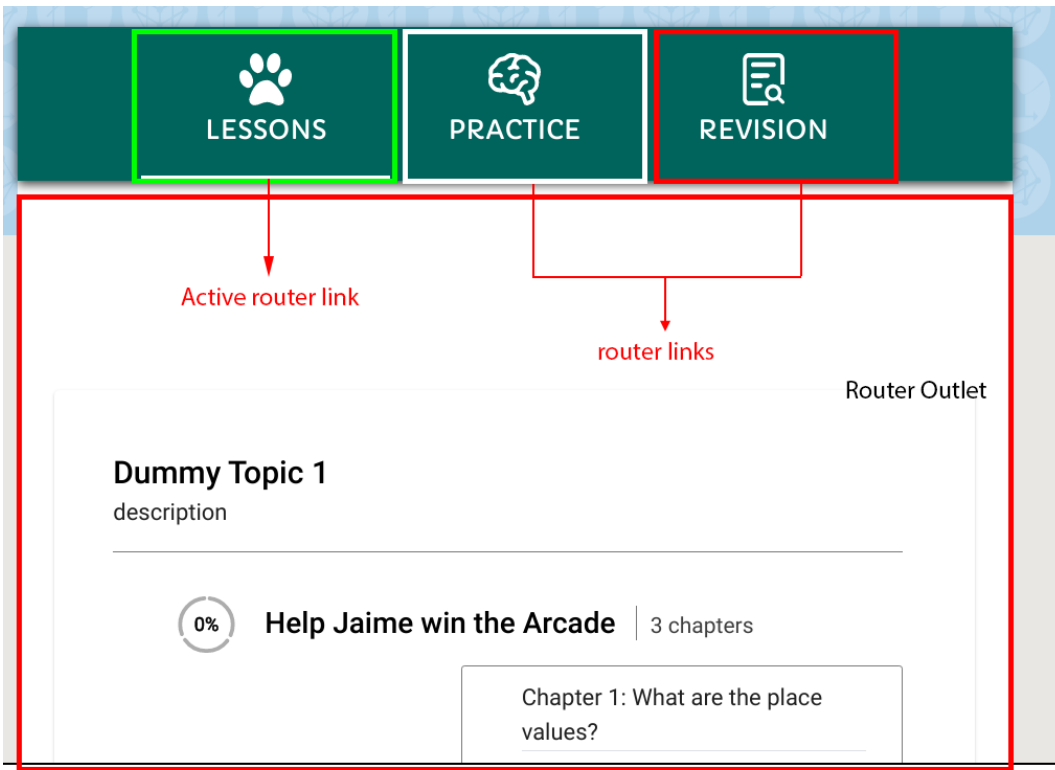
children: [
  {
    path: 'questions',
    component: SkillsQuestionTabComponent
  },
  {
    path: 'editor',
    component: SkillEditorTabComponent
  },
  {
    path: 'preview',
    component: SkillPreviewTabComponent
  }
]
}
];

```

Similarly for topic editor page:

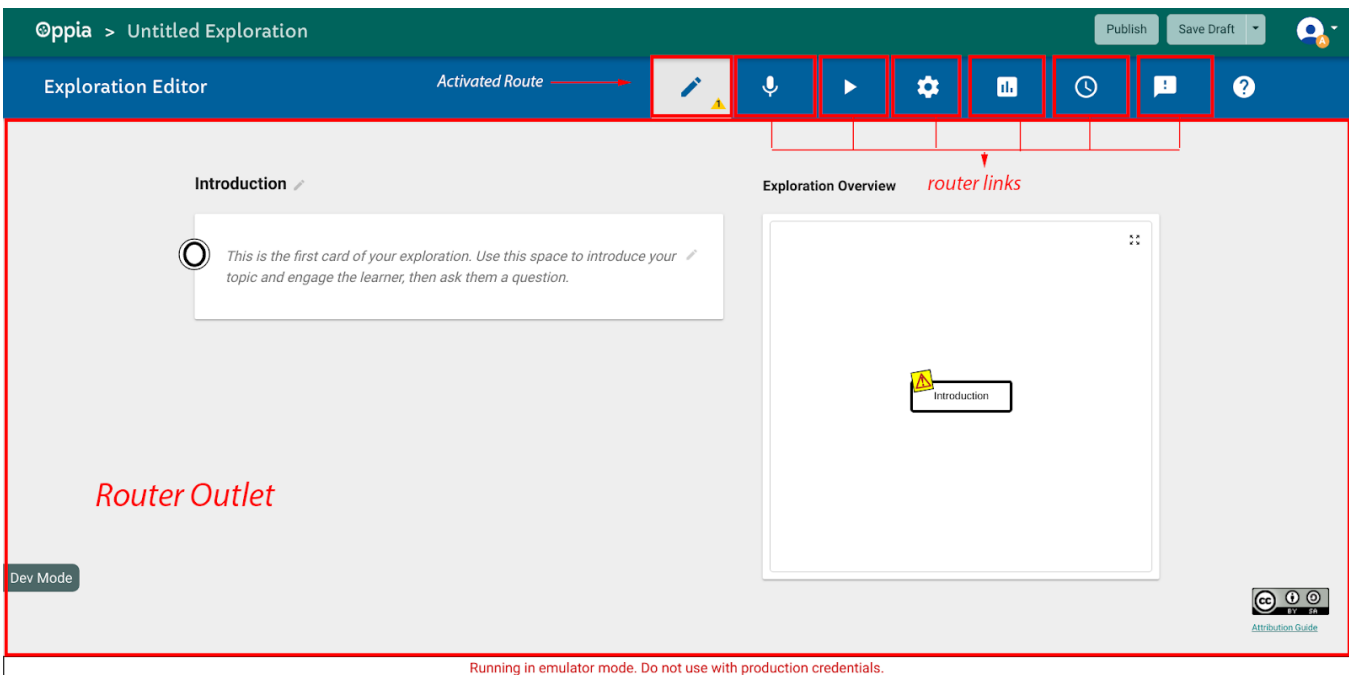


Implementation will remain similar to the skill editor page.
 Preview tab on topic editor page:



Running in emulator mode. Do not use with production credentials.

Exploration editor page:



SEO CONCERNS:

Will we see an SEO drop?

Yes,

Why?

Currently title and meta description are both statically added to *-mainpage.html files.

After the migration, the code for title and meta description tags will be placed in the root.component.ts files of every page. These components are lazy loaded. So, client side rendering is required for setting the title and meta description.

So, there will be a drop in SEO.

Nowadays, most of the crawlers are getting better and better with client side rendering. So, It will depend on crawler to crawler, on how much SEO drop we will get.

Measures taken to minimize the drop?

Statically adding a generalized (that works for all pages) title and meta description in the root index.html file and then replacing those meta tags and title when the root component loads lazily.

How will this fix work?

1. Crawlers usually take two passes while indexing a web page.
2. First they crawl without rendering javascript and look for title and meta description tags
3. In the second pass, they render javascript and index.

It is very important for crawlers to be able to read crucial tags like title and meta description in the first pass to get good SEO.

So, by using the above fix, during the first pass the crawlers will get a generalized title and meta description. So that at least, the most important search engine result will not get affected.

Search results for about and fraction pages will definitely get a hit, as they will get indexed after the second pass of the crawler.

I don't think there will be any related regressions.

PRESERVING STATUS CODES:

Why to preserve?

HTTP Status codes are important for SEO. Only backends can set status code in response headers. If we handle other error pages in the frontend too then, we won't be able to add appropriate status code to them. The search engine crawlers might identify them as regular pages instead of error pages.

How to preserve?

Error pages are usually assigned status codes other than 200 (OK). Most of these error pages are handled at the backend. So, they will have appropriate status code assigned to them.

One exception to this is error 404 page.

Possible fix for the status code:

The fix for this is to define a new **/not-found** route in the backend with status code 404. When a route is not found by the frontend router, it should redirect to this **/not-found** route in the backend.

Example:

```
export class HardRedirectToNotFoundErrorResponseComponent implements OnInit {
  constructor(
    private windowRef: WindowRef
  ) {}

  ngOnInit(): void {
    this.windowRef.nativeWindow.location.href = '/not-found';
  }
}
```

Router configuration:

```
{
  path: '**',
  component: HardRedirectToNotFoundErrorResponseComponent
}
```

HOW TO KEEP ROLES IN SYNC WITH BACKEND?

The frontend roles which i plan to introduce correspond directly to `UPDATABLE_ROLES` in `role_services.py`.

`role_services.py`

```
91. # NOTE: LEARNER role should not be updated to any other role, hence do
not
92. # add it to the following list.
93. UPDATABLE_ROLES = [
94.     feconf.ROLE_ID_ADMIN,
95.     feconf.ROLE_ID_BANNED_USER,
96.     feconf.ROLE_ID_COLLECTION_EDITOR,
97.     feconf.ROLE_ID_EXPLORATION_EDITOR,
98.     feconf.ROLE_ID_MODERATOR,
99.     feconf.ROLE_ID_TOPIC_MANAGER
100. ]
```

To keep the roles in sync, I will write a linter to ensure that the roles present constants.ts files in the frontend will be the same as the `UPDATABLE_ROLES` list in the backend.

This linter will be added to pre push hooks and circle ci lint checks.

Why is CSRF not a security concern?

CSRF stands for **Cross Site Request Forgery**.

Every website which uses a cookie is vulnerable to this attack. Forms are most vulnerable to this attack and forms usually send POST / PUT / DELETE requests.

How Oppia prevents CSRF?

Oppia uses a token based prevention system.

- For every POST / PUT / DELETE request to be a valid request, the request should provide a valid **csrf_token** in its body.
- This csrf token is obtained in the frontend by using the `csrf-token.service.ts` which provides a `getTokenAsync()` function.
- This service sends a GET request to `/csrfhandler` route at the backend which creates the token and sends it back to the frontend.

Why does this prevention system work?

Attackers can trick the browser into sending the cookie from a cross site origin but sending the `csrf_token` from cross site origin is not possible.

The Page Handlers which I propose to remove are only for GET requests.

The GET requests are not vulnerable to CSRF attacks.

The Api endpoints will not be altered during this project by any means.

So, I think that this project does not cause a CSRF vulnerability

META TAGS:

At present, meta tags are placed in `header.template.html` file and this file is imported to every `*-mainpage.html` file (This is done using `underscore-template-loader`).

This is done to avoid repeated writing of the same meta tags in all the mainpage html files individually.

Pros of current approach:

1. Easier updation, any change made to `header.template.html` file will reflect in all the pages.
2. Duplication of html code is avoided here.

After migration to the router, all the entry points will be unified and there will only be one `mainpage.html` file that is `index.html` (unified entry point). As we will have only one entry file now, there is no need to separate meta tags in a separate file.

The issue which is solved in the present approach will not be there after migration to router.

As there is no problem, then fix is also not required.

So, I will add the meta tags (static) directly to the unified entry point `mainpage.html` file.

During Migration:

The `*-mainpage.html` files which are not migrated will use `header.template.html` file using `underscore-template-loader`. The new unified entry point will also contain the meta tags.

When all pages are migrated to angular. The `header.template.html` file will be finally deleted.

This will be done for static meta tags.

Currently we also interpolate some meta tags using underscore-template-loader. This wouldn't be possible after migration to angular cli.

To resolve this, I will use the **Meta** service provided by angular to set dynamic meta tags.

Example: (donate page)

```
@Component({
  selector: 'donate-page-root',
  template: '<donate-page></donate-page>'
})

export class DonatePageRootComponent {
  constructor(
    private pageTitleService: PageTitleService,
    private meta: Meta
  ) {}

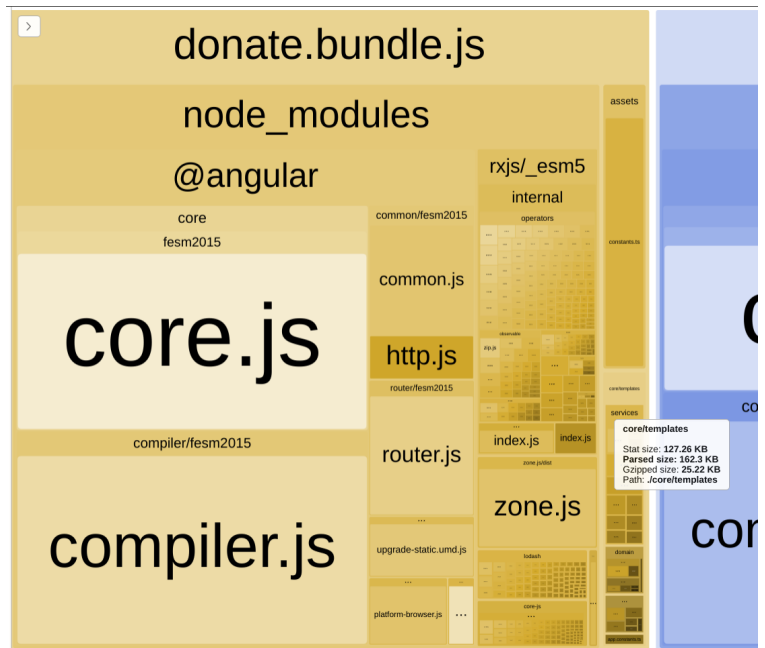
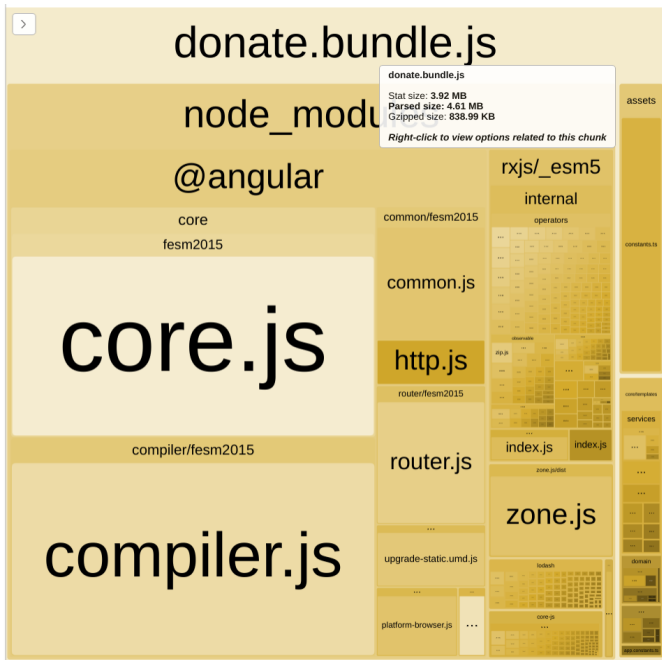
  ngOnInit(): void {
    this.pageTitleService.setPageTitle('Donate - Oppia');
    this.meta.addTag(
      { itemprop: 'description',
        content: 'Donate to The Oppia Foundation to enable more ' +
          'students to receive the quality education they deserve.' });
  }
}
```

PERFORMANCE GAIN:

The biggest issue that affects our bundle size is **duplication of libraries** across pages.

I have analyzed our current bundles using the webpack-bundle-analyzer plugin.

1. Let us start with donate bundle:



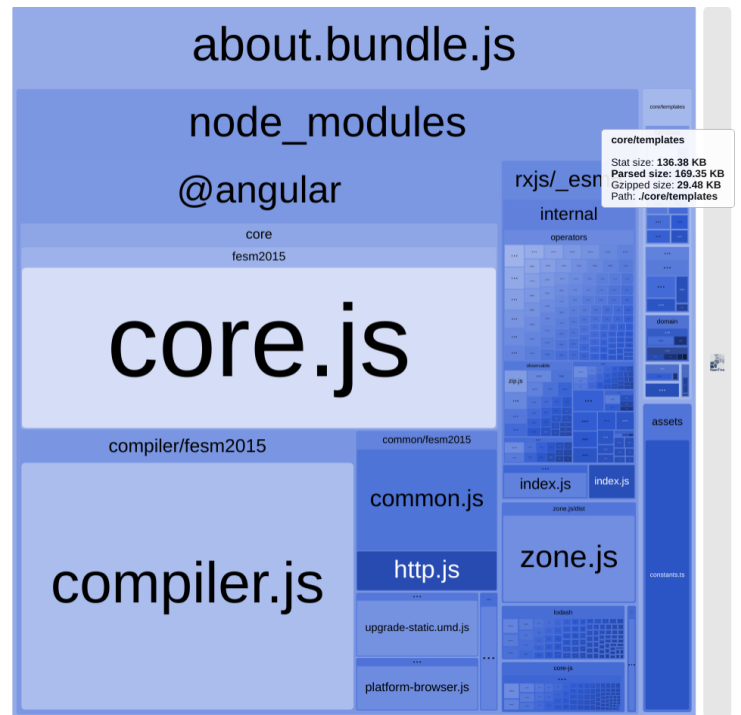
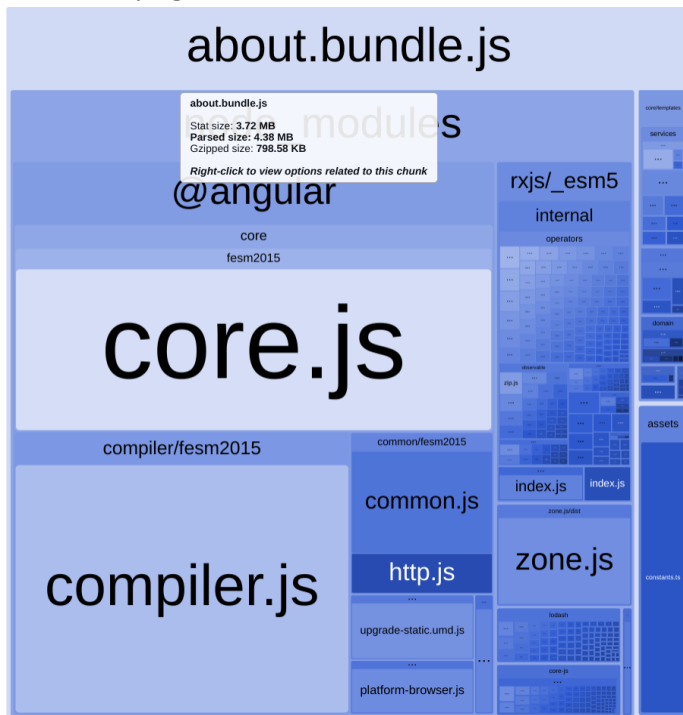
As you can see about **90%** of the total size of the bundle consist of libraries imported from **node_modules** and **constants.ts**

Gzipped sizes:

Total size of donate bundle: **838.99 KB**
 Size of common libraries and code: **813.77 KB**
 Size of unique code for donate page: **25.22 KB**

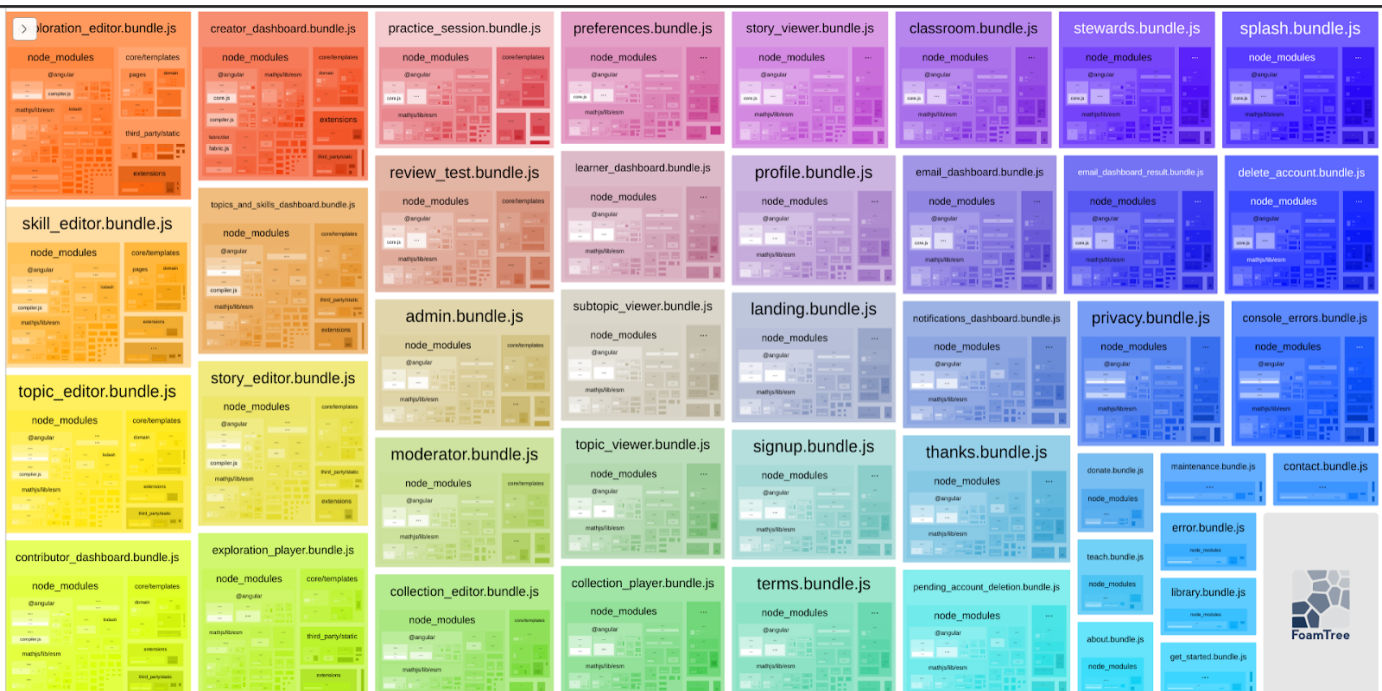
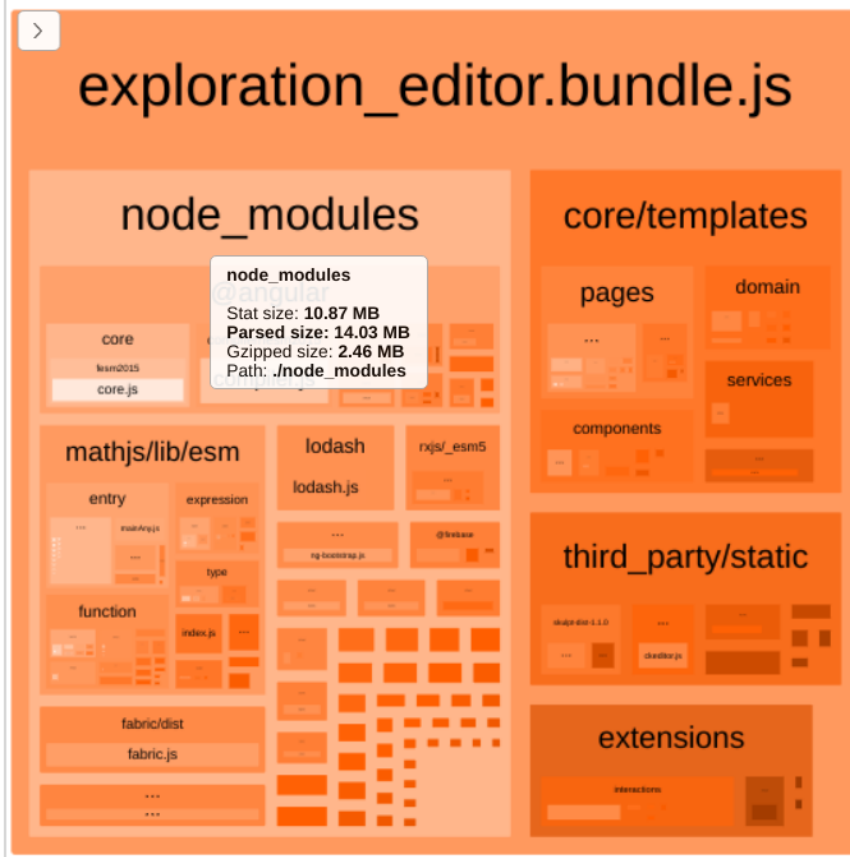
97% code inside this bundle consists of common libraries.

For about page:



We are seeing 75.74% reduction in bundle size for these two pages.

For large pages like the exploration editor page, node_modules alone takes 2.46 MB (gzipped).



After unifying all the entry points,

The reduction in total build size of oppia will be: **70.206 MB**

The present total bundle size (gzipped) for all the pages is : **97 MB**

So with unification only we are getting a **72 % reduction** in total bundle size of oppia.

With **lazy loading**, we will have an initial bundle under **400 KB** (gzipped).

With **AOT enabled**, we can get the initial bundle size under **80 KB** (gzipped).

After migration angularjs libraries will no longer be required. Which led to a significant reduction in size of **third_party.js** file.

Current size: **531 KB (gzipped)**

After removal of angularjs libraries: **81 KB (gzipped)**

Here we are seeing **84.74 %** reduction in size.

Speed Changes:

Considering the user's internet speed to be constant, decrease in bundle size will directly proportional to decrease in loading time.

For example if there is 80 % decrease in bundle size, then loading time will also decrease by 80 %.

SUB-PROJECT 2: MIGRATION FROM WEBPACK TO ANGULAR - CLI

Why?

Currently, Oppia uses webpack to bundle and compile frontend code.

Pros:

1. Fully customisable, can be configured for every requirement.
2. Provides more control over the build process.

Cons of using custom webpack configurations:

1. Steeper learning curve than angular cli
2. Lots of knowledge is required to configure it the right way
3. Not future proof, plugins and loaders can get deprecated, developers have to handle this themselves

I propose to migrate to **angular cli**:

Pros:

1. Easy to learn as compared to webpack
2. Does not require much configuration and boilerplate code.
3. It is future proof, underlying functionality will adapt to keep us updated.
4. Provides code generation, developers can create components, services, modules etc using ng generate command. This will help in increasing productivity of developers.
5. Supports Server Side Rendering.

Cons:

1. Less customizable than webpack, but this can be overcome using @angular-builders/custom-webpack plugin

Performance Comparison:

Note:

Our present webpack configuration, only transpiles typescript, doesn't perform type checking.

Whereas, angular cli builds also performs type checking along with transpilation.

We cannot disable type checking in angular cli build, So, to have a level ground, I have enabled type checking while compiling with webpack configuration to produce these results.

Configuration	webpack	angular cli
Development mode without source maps	17203ms Command Used: node node_modules/webpack/bin/webpack.js --config webpack.dev.config.ts	36269ms Command used: npx ng build
Development mode with source maps enabled	88003ms Command Used: node --max-old-space-size=8192 node_modules/webpack/bin/webpack	51233ms Command Used: npx ng build --source-map

	ack.js --config webpack.dev.sourcemap.config.ts	
Production mode without sourcemaps	65262ms Command Used: node --max-old-space-size=8192 node_modules/webpack/bin/webp ack.js --config webpack.prod.config.ts	32811ms Command Used: npx ng build --prod
Production mode with sourcemaps	155791ms Command Used: node --max-old-space-size=8192 node_modules/webpack/bin/webp ack.js --config webpack.prod.sourcemap.config.ts	33073ms Command Used: npx ng build --prod --source-map
Rebuilding time for minor changes	732ms	539ms

Note:

These builds were tested on a 8 core 16 thread CPU, 16 GB DDR4 RAM and PCIe SSD.

These build times are highly hardware dependent but relative performance will remain the same for most systems.

Conclusion:

ng cli provides better build speeds than webpack.

It is slower while using in development mode without source maps but it also performs build optimisation, differential loading and caching out of the box.

What do developers prefer?

To find this, I conducted a survey with a small pool of participants. The main objective of the survey was to find what tool is preferred more and why.

The participants of the survey were angular developers.

Results:

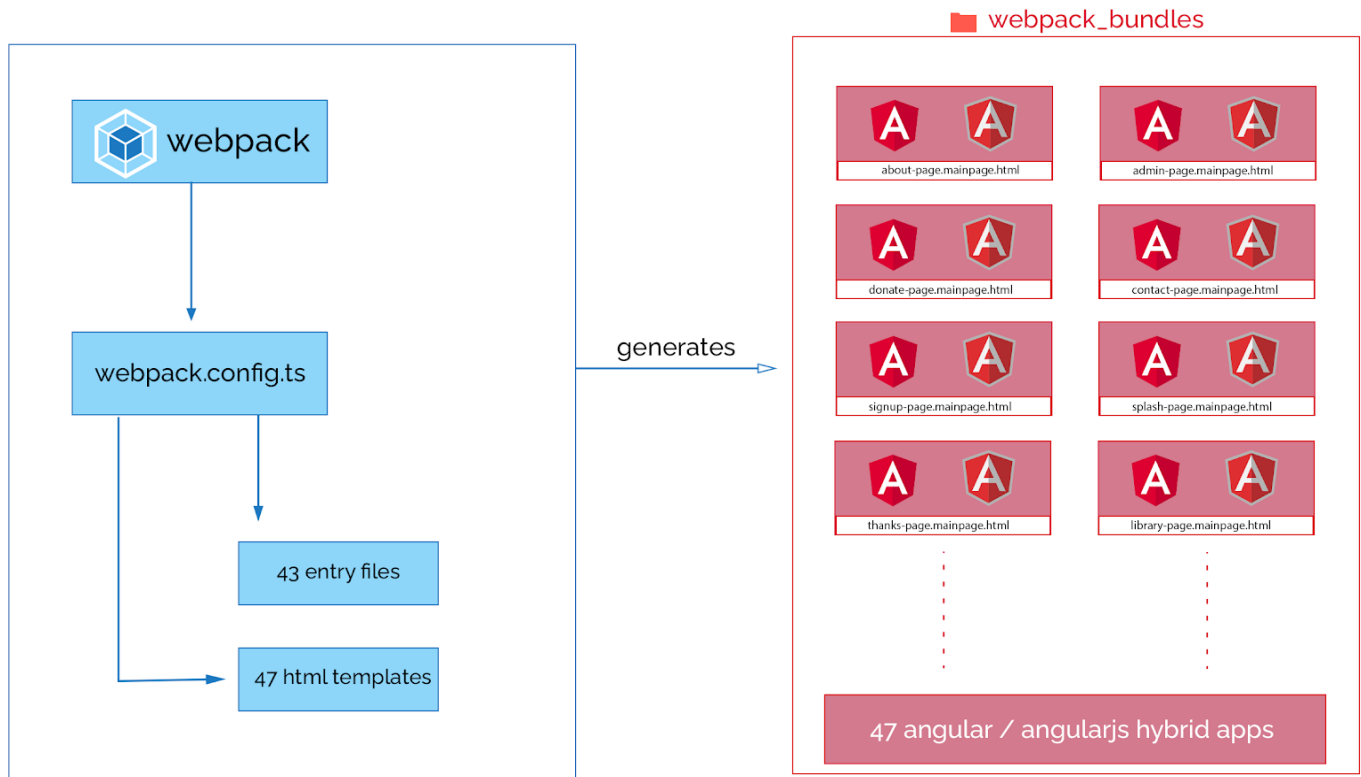
More than 75% of angular developers prefer angular cli over webpack because it provides simplicity, code generation and is easier to learn.

[Here](#) is the link to survey form.

Build system at present:

- At present webpack builds an application for each page.

PRESENT FRONTEND BUILD SYSTEM



- After migration and integration of the angular router all the entry points will be unified.
- Now I will introduce angular cli in the build process, when it is fully integrated with the build system, I will start removing the webpack from the project.
- This will be done incrementally, so PRs remain small and the Oppia is working at every stage.
- Add angular.json at project root and will optimize the production bundle.

Implementation Strategy:

I will implement this in 6 steps:

1. Introduction of angular cli using angular.json

```
angular.json
```

```
...
16   "architect": {
17     "build": {
18       "builder": "@angular-devkit/build-angular:browser",
19       "options": {
20         "outputPath": "dist/oppia",
21         "index": "core/templates/pages/app-page/index.html",
22         "main": "core/templates/pages/app-page/main.ts",
23         "tsConfig": "tsconfig.json",
24         "aot": false,
25         "vendorChunk": true,
```

```

26     "assets": [
27       "core/templates/pages/app-page/assets/ashes.json",
28       "core/templates/pages/app-page/assets"
29     ],
30     "sourceMap": true,
31     "poll": 1000
32   },
33   "configurations": {
34     "production": {
35       "fileReplacements": [
36         {
37           "replace":
"core/templates/pages/app-page/environments/environment.ts",
38           "with":
"core/templates/pages/app-page/environments/environment.prod.ts"
39         }
40       ],
41       "optimization": {
42         "scripts": true,
43         "styles": true
44       },
45       "outputHashing": "all",
46       "sourceMap": false,
47       "namedChunks": false,
48       "extractLicenses": true,
49       "vendorChunk": false,
50       "styles" : [
51         "third_party/generated/css/third_party.css",
52         "core/templates/css/oppia.css",
53         "core/templates/css/oppia-material.css"
54       ],
55       "scripts": ["third_party/static/jquery-3.5.1/jquery.min.js",
        "third_party/static/jqueryui-1.12.1/jquery-ui.min.js",
        "third_party/static/jquery-ui-touch-punch-0.3.1/jquery.ui.touch-punch-improved.js",
        "third_party/ngx_generated/js/third_party.js"
56     ]
57   }

```

Specifications of proposed angular.json file:

1. Should we use a separate vendor bundle?:

Pros:

The vendor bundle contains the libraries which usually are big in size and are not changed frequently. So, by using a separate vendor file, libraries can be cached which will result in faster rebuilding of code for developers.

Cons:

Using a vendor chunk can lead to large build size, which will increase Oppia loading time for users

So, I came to the conclusion, to only use vendor chunk in development and not in production, so both developers and users can be benefitted.

2. There are two approaches to handle third party libraries. These are the following:

2.1 Use <link> and <script> tag.

2.2 Bundle third party libraries using ng cli.

Pros of not bundling libraries:

1. By not including third party libraries in our bundle, we can reduce build time significantly.

Cons:

1. This approach leads to slower loading of the website for the users.

2. Browser fetches libraries from server one by one. Latency, browser and server response time can lead to significant increase in Oppia's loading time.

So, I came to a conclusion, to bundle libraries in production mode, so Oppia's users will get better loading times and not to bundle libraries in development mode, so Oppia's developers can enjoy faster builds. Page loading time is generally very low for developers as they have server running on the same machine unlike users

3. Production build is optimized with **minification, tree shaking and dead code elimination.**

2. Integration of ng build with `python -m scripts.start`:

I will replace call to spawn webpack with call to spawn ng build process:

I will replace following code in start.py:

```
135 background_processes = []
136 if not parsed_args.prod_env:
137     # In prod mode webpack is launched through scripts/build.py
138     python_utils.PRINT('Compiling webpack...')
138     webpack_config_file = (
140         build.WEBPACK_DEV_SOURCE_MAPS_CONFIG if parsed_args.source_maps
141         else build.WEBPACK_DEV_CONFIG)
142     background_processes.append(subprocess.Popen([
143         common.NODE_BIN_PATH,
144         os.path.join(
145             common.NODE_MODULES_PATH, 'webpack', 'bin', 'webpack.js'),
146         '--config', webpack_config_file, '--watch']))
147
148     # Give webpack few seconds to do the initial compilation.
149     time.sleep(10)
```

with

```
135 ANGULAR_CLI_FILE = os.path.join('node_modules', '@angular', 'cli', 'bin', 'ng')
136 if not parsed_args.prod_env:
```

```

137     # In prod mode ng build is launched through scripts/build.py
138     background_processes.append(subprocess.Popen([
139         ANGULAR_CLI_FILE,
140         'build', '--deploy-url=/dist/oppia', '--watch'
141     ]))
142     # Give ng build a few seconds to do the initial compilation.
143     time.sleep(10)

```

I will perform the following changes to build.py.

Replace:

```

657 def build_using_webpack(config_path):
658     """Execute webpack build process. This takes all TypeScript files we have
in
659     /templates and generates JS bundles according the require() imports
660     and also compiles HTML pages into the /backend_prod_files/webpack_bundles
661     folder. The files are later copied into /build/webpack_bundles.
662     Args:
663         config_path: str. Webpack config to be used for building.
664     """
665
666     python_utils.PRINT('Building webpack')
667
668     cmd = '%s %s --config %s' % (
669         common.NODE_BIN_PATH, WEBPACK_FILE, config_path)
670     subprocess.check_call(cmd, shell=True)

```

With:

```

def build_using_ng_cli(enable_sourcemap=False):
    """Execute angular-cli build process.
    """

    python_utils.PRINT('Building with angular-cli')

    cmd = '%s build --deploy-url=/dist/oppia/ --prod' % ANGULAR_CLI_FILE
    if enable_sourcemap:
        cmd = cmd + ' --source-map'
    subprocess.check_call(cmd, shell=True)

```

Ng cli will build and store the bundle in the dist/oppia folder. To make files present in this folder available to the users. I will add **dist/oppia** as a **static folder** in both **app.yaml** and **app_dev.yaml**.

app.yaml, app_dev.yaml

```
53.     expiration: "0"
54.
55. + - url: /dist/oppia
56. +   static_dir: dist/oppia
57. +   secure: always
58. +   application_readable: true
59. +   expiration: "0"
60.
61. - url: /asset
```

- Build using webpack will be replaced by build_using_ng_cli
- webpack configs, webpack calls in start.py and build.py will be removed.

3. Linting

```
"lint": {
  "builder": "@angular-eslint/builder:lint",
  "options": {
    "eslintConfig": ".eslintrc",
    "cache": true,
    "force": true,
    "lintFilePatterns": [
      "core/**/*.ts"
    ],
    "ignorePath": ".eslintignore"
  }
},
```

Oppia uses eslint as preferred linter for js/ts files. Angular Cli, out of the box, only provides Tslint.

So, I plan to use a third party library **@angular-eslint** to use eslint with ng cli.

I will add the above configuration to angular.json, so developers run **eslint** using **'ng lint'** command.

4. Karma tests

I will add following configuration to angular.json, so developers can run frontend tests using 'ng test'

```
"test": {
  "builder": "@angular-devkit/build-angular:karma",
```



```

    "options": {
      "main": "core/templates/pages/app-page/main.ts",
      "tsConfig": "tsconfig.spec.json",
      "karmaConfig": "core/tests/karma.conf.ts"
    }
  }
}

```

Karma test runner uses webpack internally with karma-webpack plugin:

I plan to replace the **karma-webpack** plugin with **@angular-devkit/build-angular/plugins/karma** plugin.

To implement this, I will modify the current **angular.json** file:

- I will add **@angular-dev/build-angular** to the frameworks array.
- I will replace karma-webpack with **@angular-devkit/build-angular/plugins/karma** plugin.
- I will remove the webpack test configuration added in the **karma.conf.ts** file.

core > tests > karma.conf.ts

```

...
8.  module.exports = function(config) {
9.  config.set({
10.   basePath: '../..',
11.   frameworks: ['jasmine', '@angular-devkit/build-angular'],
12.   files: [
13.     // Constants must be loaded before everything else.
14.     // Since jquery, angular-mocks and math-expressions
15.     // are not bundled, they will be treated separately.
16.     'third_party/static/jquery-3.5.1/jquery.min.js',
17.     'third_party/static/angularjs-1.8.2/angular.js',
18.     'core/templates/karma.module.ts',
19.     'third_party/static/angularjs-1
...
116.  plugins: [
117.    'karma-coverage-istanbul-reporter',
118.    'karma-jasmine',
119.    'karma-chrome-launcher',
120.    'karma-ng-html2js-preprocessor',
121.    'karma-json-fixtures-preprocessor',
122.    'karma-coverage',
123. - 'karma-webpack',
123. + '@angular-devkit/build-angular/plugins/karma'
124.  ],

```

```

...
140. -   webpack: {
141. -     mode: 'development',
142. -     resolve: { ... },
160. -     devtool: 'inline-cheap-source-map',
161. -     module: {
162. -       rules: [...]
217. -     }
218. -   }

```

To integrate ng test with `python -m run_frontend_tests` command, I will perform these changes to the `run_frontend_tests.py` file.

Replace

```

123     cmd = [
124         os.path.join(common.NODE_MODULES_PATH, 'karma', 'bin', 'karma'),
125         'start', os.path.join('core', 'tests', 'karma.conf.ts')]

```

with

```

37  ANGULAR_CLI_FILE = os.path.join('node_modules', '@angular', 'cli', 'bin', 'ng')
...
123  cmd = [
124      ANGULAR_CLI_FILE, 'test'
125  ]

```

- combined-tests.spec.ts will remain the same.

5. e2e tests:

angular.json

```

"e2e": {
  "builder": "@angular-devkit/build-angular:protractor",
  "options": {
    "protractorConfig": "core/tests/protractor.conf.js",
    "devServerTarget": "oppia:serve"
  },
  "configurations": {
    "production": {
      "devServerTarget": "oppia:serve:production"
    }
  }
}

```

6. ng serve:

angular.json

```
"serve": {
  "builder": "@angular-devkit/build-angular:dev-server",
  "options": {
    "browserTarget": "oppia:build",
    "proxyConfig": "proxy.conf.json"
  },
  "configurations": {
    "production": {
      "browserTarget": "oppia:build:production"
    }
  }
},
```

proxy.conf.json:

```
{
  "/": {
    "target": "http://localhost:8181",
    "secure": false
  }
}
```

Specification of the above configuration:

- Can spin up a local development server.
- Proxy api requests to the backend.

My Progress on the project before submitting the proposal:

I have been successful in setting up and migrating 3 pages to the angular router with route protection using auth guards. I have also been successful adding angular cli (ng build, ng serve, ng lint, ng test, ng e2e) to the project.

You can check out the work on my fork [here](#).

Third-party Libraries

[@angular-eslint](#) (MIT License)

Oppia project uses eslint as its preferred linter for javascript and typescript files. Tslint is deprecated now. But angular cli out of box only supports Tslint. So, to use eslint with angular cli, I will use this third party library. For implementation details, please refer to step 3 of implementation detail of sub project 2 [here](#).

Testing Approach

- Unit tests will be written along the route guard, components and services (if any) to be added.

Milestones

Community Bonding Period:

As I am already contributing to Oppia from Feb 2021, I have become quite familiar with the codebase. So, I will start working on the project right away. This will provide me a strong head start which will help me to cover up unexpected delays in the future.

As I am part of the angular migration team, I along with other developers will finish migration of project to angular and will remove angularjs during this period.

Key Objective: Remove angularjs from the project

Starts: 17 May 2021

Ends: 6 June 2021

Tasks:

1. Complete migration of angularjs directives, components and left services.
2. Remove angularjs libraries from the project.
 - 2.1 Following libraries will be removed:
 1. angular
 2. Angular-cookies
 3. Angular-route
 4. Angular-ui-sortable
 5. Angular-ui-validate
 6. ng-infinite-scroll
3. Remove unused code which was written to dual boot angular and angularjs.

Milestone 1:

Key Objective: Migration to angular router

Starts: 7 June 2021

Ends: 12 July 2021

Tasks:

1. Introduce angular router in the project.
2. Migrate pages to angular router.
3. Clean up unused code.

No.	Description of PR	Prereq PR numbers	Target date for PR submission	Target date for PR to be merged
-----	-------------------	-------------------	-------------------------------	---------------------------------

1.1	Angular router will be ready to be used.		June 8	June 10
1.2	About page will be migrated to angular router		June 9	June 11
1.3	Classroom, contact page will be migrated to angular router		June 10	June 12
1.4	Delete account and donate pages will be migrated to angular router		June 11	June 13
1.5	Email dashboard and error pages will be migrated to angular router		June 12	June 14
1.6	Get started page and landing pages will be migrated to angular router		June 13	June 15
1.7	Maintenance and moderator pages will be migrated to angular router		June 14	June 16
1.8	Collection player and exploration pages will be migrated to angular router		June 15	June 17
1.9	Learner dashboard and library pages will be migrated		June 16	June 18
1.10	Preferences and privacy pages will be migrated		June 17	June 19
1.11	Review test and signup pages will be migrated		June 18	June 20
1.12	Story viewer and subtopic viewer pages will be migrated		June 19	June 21
1.13	Admin page will be migrated to angular router		June 20	June 22

1.14	Collection editor page will be migrated to angular router.		June 21	June 23
1.15	Creator dashboard page will be migrated to angular router		June 22	June 24
1.16	Exploration editor page will be migrated		June 24	June 26
1.17	Sign up page will be migrated		June 25	June 27
1.18	Story editor page will be migrated		June 26	June 28
1.19	Topic editor page will be migrated		June 27	June 29
1.20	Topics and skills dashboard page will be migrated		June 28	June 30
1.21	Topic viewer page will be migrated		June 29	July 1
1.22	Review and sign up page will be migrated		June 30	July 2
1.23	subtopic viewer and story viewer pages will be migrated		July 1	July 3
1.24	Not found and notification dashboard pages will be migrated		July 2	July 4
1.25	Pending account deletion and practice session page will be migrated		July 3	July 5
1.26	Privacy and splash page will be migrated		July 4	July 6
1.27	Teach and terms pages will be migrated		July 5	July 7

1.28	Thanks and login pages will be migrated		July 6	July 8
1.29	Buffer time for milestone 1			

Milestone 2

Key Objective: Migration to angular cli

Tasks:

- Introduce angular cli.
- Introduce ng build to build oppia.
- Introduce ng lint for linting of project with eslint.
- Introduce ng test to run karma test runner.
- Introduce ng e2e to run end to end tests.
- Introduce ng serve to run local development server.

No.	Description of PR	Prereq PR numbers	Target date for PR submission	Target date for PR to be merged
2.1	Angular cli will be introduced in the project.		19 July	23 July
2.2	Developers will be able to build the Oppia's frontend with angular cli.		24 July	28 July
2.3	Developers will be able to lint the project using the ng lint command.		26 July	30 July
2.4	Developers will be able to run frontend tests with ng test command.		1 August	4 August
2.5	Developers will be able to run end to end workflows using ng e2e command.		6 August	10 August
2.6	Developers will be able to run a development server for frontend using ng serve command.		11 August	14 August
2.7	Buffer time for milestone 2			

Additional Project-Specific Considerations

Privacy

No, this project does not add any new feature which collects user data.

Security

No, this feature does not have any security considerations. Most of the changes are only related to frontend. If by mistake any wrong change is made, it will in the worst case only affect frontend security. If any user gets through the frontend security layer, the api endpoints are secured at the backend. So, he will not be able to gain any unauthorized access.

Accessibility

Only subproject 1 (Migration to angular router) is a user facing feature.

For the feature to be accessible, I will ensure following:

- All the routerLinks will be reachable through tabs.
- The title of every page can be announced using screen readers.

Documentation Changes

Oppia's wiki has a section for webpack.

That section will no longer be required after the migration to angular cli.

I will replace that section with a section explaining the usage of ng cli and its configuration.

Future Work

1. Implementation of preloading strategy for frontend routing:

Default strategy for routing provides fast interpage navigation but results in large initial bundle size.

On the other hand, Lazy loading provides a smaller initial bundle but interpage navigation is not fast as default strategy.

Preloading provides the best of both worlds, with preloading we can have a smaller initial bundle without the cost of relatively slower interpage navigation.

2. Server Side Rendering for Oppia using angular universal:

With Server Side Rendering, static pages are generated on the server. As servers have generally better resources, they can render web pages more quickly as compared to a user's browser. It also provides better Search Engine Optimization.