GOOGLE SUMMER OF CODE

STATIC SERVING

James, James John

March 2019

Name of the project:

Static Serving

Why am I interested in working with Oppia?

I discovered Oppia while going through the organization list published by Google on the 26th day of February. I immediately got attracted to Oppia because it used technologies that I was interested in.

While digging in and finding more about Oppia, I discovered it's the mission of providing quality education to those who lack access to it.

I've been an educator for a large part of my life. And finding an organization that believes in the same principle as me and uses similar technologies as I'm used to just seemed like the perfect fit.

What interests me about the project?

While going through the suggested starter issues, I stumbled upon a set of related issues and this had to do with removing a *global* variable from the base.html. Working on this issue made me understand the codebase. I also noticed the predominant use of jinja templates for controlling the frontend flow, creating sections to be filled by other pages that required them, including other templates needed for the page.

I find the need to move from python injected frontend code to javascript controlled code as exciting. I have done something similar in a PHP project and I feel it would be a good way to contribute to the Oppia foundation.

Prior experience:

I have been building web applications for almost 2 years.

I have worked with Javascript frameworks such as EmberJs, Angular and AngularJs, React and Vue. In recent times, I have been working more with Vue. Here's a link to a javascript starter-kit which I built to aid rapid frontend development.

I also use PHP in building web backends using the Laravel framework.

My python knowledge has mostly been used for building CLI (Command Line Interface) tools for my personal use and I have not used it in a large application before. Which is also one of the reasons I decided to apply to Oppia. To use my python knowledge in building advanced web applications.

From August 2017 to December 2017, I interned with Hotels.ng.
Hotels.ng is the biggest online hotels booking company in Nigeria. I worked on the front end of a new platform that was to manage hotels booking across Africa Timbu.com. I interned with this company again from May 2018 to August 2018 where I got to work on the backend of Spendtrim.com - a platform that helps manage companies expenses, quotes and vendors, and Roomhub - a property management platform. The projects were open source during the internship and made private after the internship program. Working on these large projects gave me the experience of working on very large software. Although the technologies used are quite different, I believe the experience gained will be helpful in contributing to the project.

I am very proficient with cloud tools especially the Google Cloud Platform. I recently took a course on coursera offered by Google about the Google Cloud Platform. I got to deploy some applications using the Google *App Engine* and also set up the *Compute Engine* and the *Kubernetes Engine*.

Since most of my web development experience has not been with python, I decided to make up for this by taking the python course in codecademy and also the python web development training from realpython.com. I also completed the angular.js course on codecademy to keep me abreast with the right angular coding patterns. These training courses also helped in my understanding of the codebase, the project and in working on some starter issues whose PRs will be given below.

Open Source Contribution

Contributing to Oppia is my first time contributing to a major open source project.
I started contributing to Oppia shortly after the organization list was announced by Google and I have successfully made 2 pull requests that got merged into the develop branch and raised one issue which I am working will be working on.
I focussed on working on issues that had a direct relationship with this project so that I could get more understanding of the problem and how it should be solved.

Here are links to the pull requests:
https://github.com/oppia/oppia/pull/6365
https://github.com/oppia/oppia/pull/6410

Link to issue:
https://github.com/oppia/oppia/issues/6565

Apart from my contribution to Oppia, I have also had some personal projects which I've hosted on GitHub. They are can be found here

Project Plan and Implementation Strategy

This project aims at removing the jinja templates used in serving of pages to a more static method.
Below are problems that are currently faced as a result of using jinja templates in the pages and problems that will be faced when jinja templates are removed from all the pages.

Problems

1. All pages are currently served using jinja templating engine and this poses some issues such as not being able to cache the pages. Since the jinja templates will have to be converted every time before rendering the page, a cached version of each page cannot be kept in the browser and this means that for every time a user visits a page, a freshly baked version of the page is sent to the user with all its dependencies.

2. Static templates are injected to each page using jinja constructs `{% include %}`. This also leads to the pages being compiled before being rendered and hence pose a similar issue as 1 above.
3. Data required by each page is passed to the page using jinja constructs and as such, removing jinja from the pages will lead to loss of data required by the pages.
4. Template inheritance is currently done using jinja constructs using base.html as the skeleton template. Removing the jinja templates will mean losing the template inheritance offered by jinja.

Having listed the problems associated with the use of jinja constructs and the problems that will arise as a result of removing jinja templates, it is pertinent to use an approach that not only solves the problems currently faced as a result of the use of jinja templates but also to proffer solutions to possible problems that will arise as a result of the removal of jinja templates.

Implementation Approach

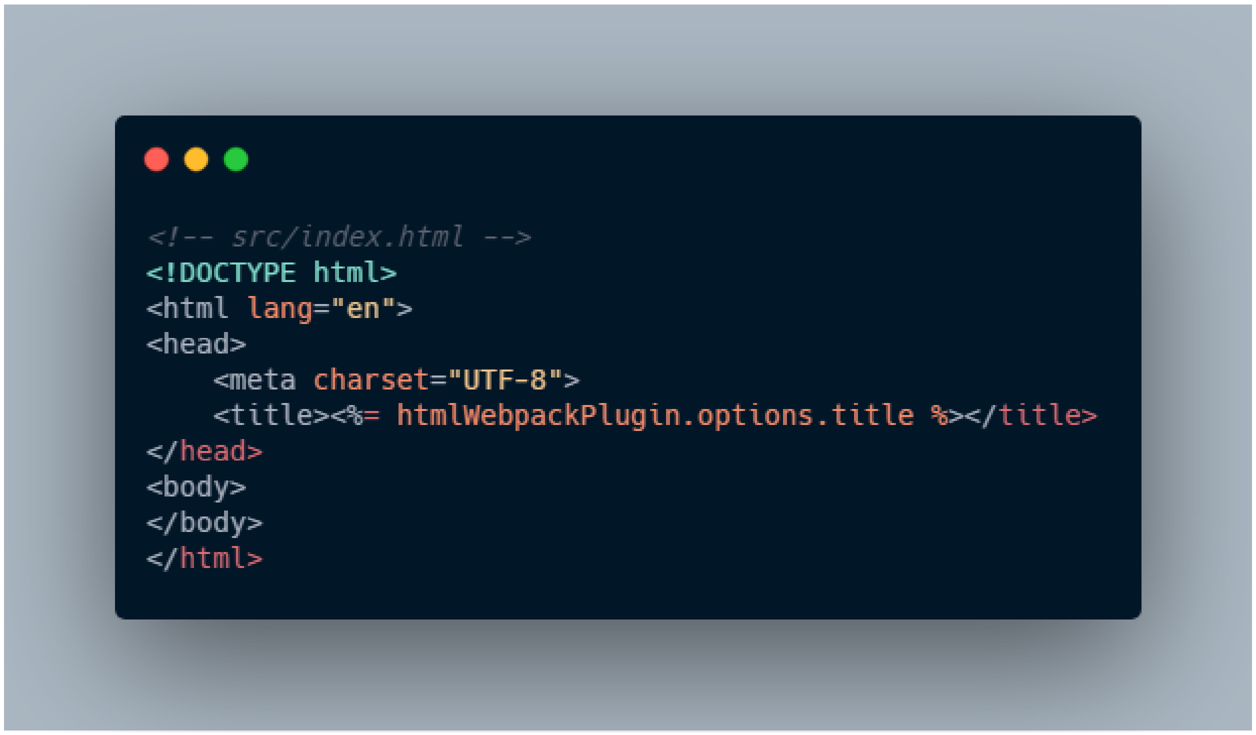
1. Getting rid of base.html:

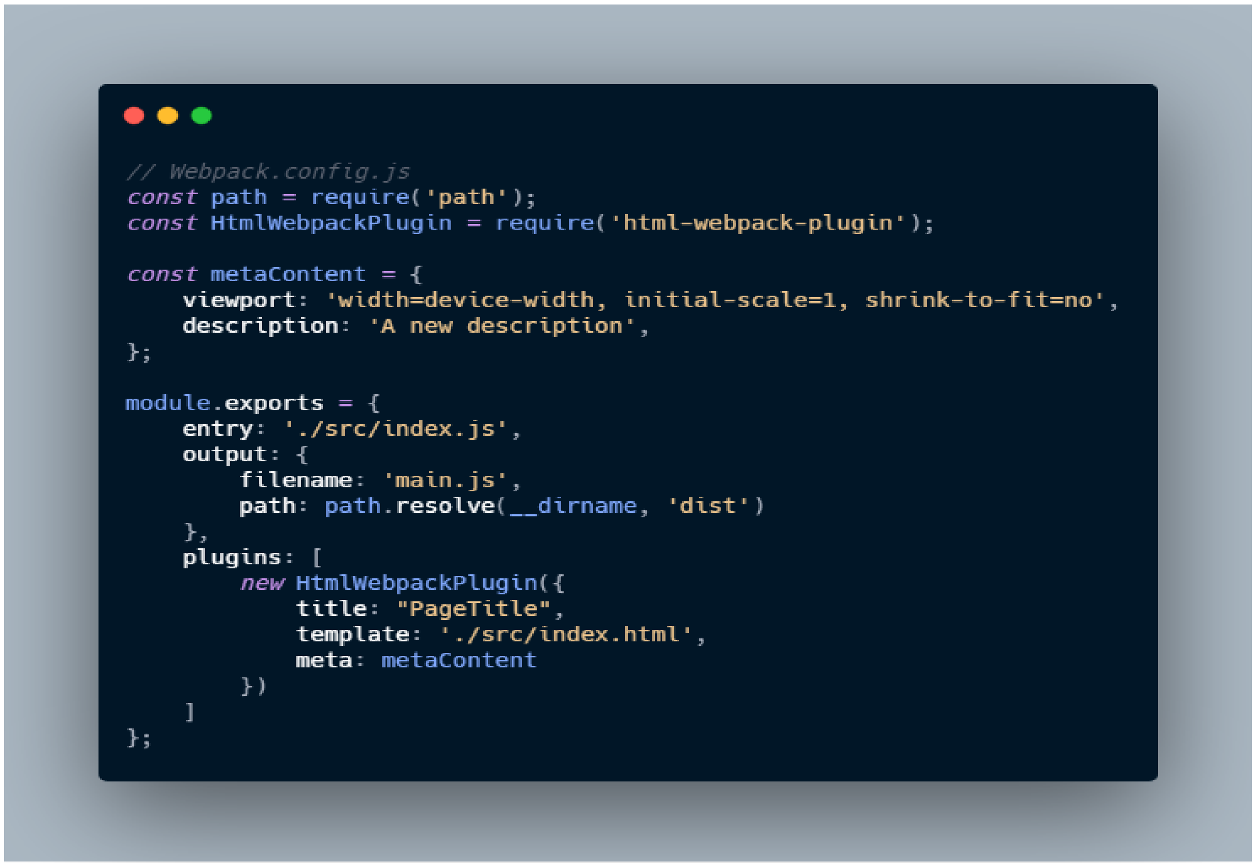
   Currently, base.html is used for the following:
   - Meta tag inclusion
   - Specifying sections of the page using jinja blocks
   - Warnings and loader macro
   - Scripts inclusion

   I. Meta tag inclusion:

   A study of the meta tags for each page shows that some meta elements are the same for all pages while others are different per page.
   Using the htmlWebpackPlugin, meta elements can be injected into each page during the build process.

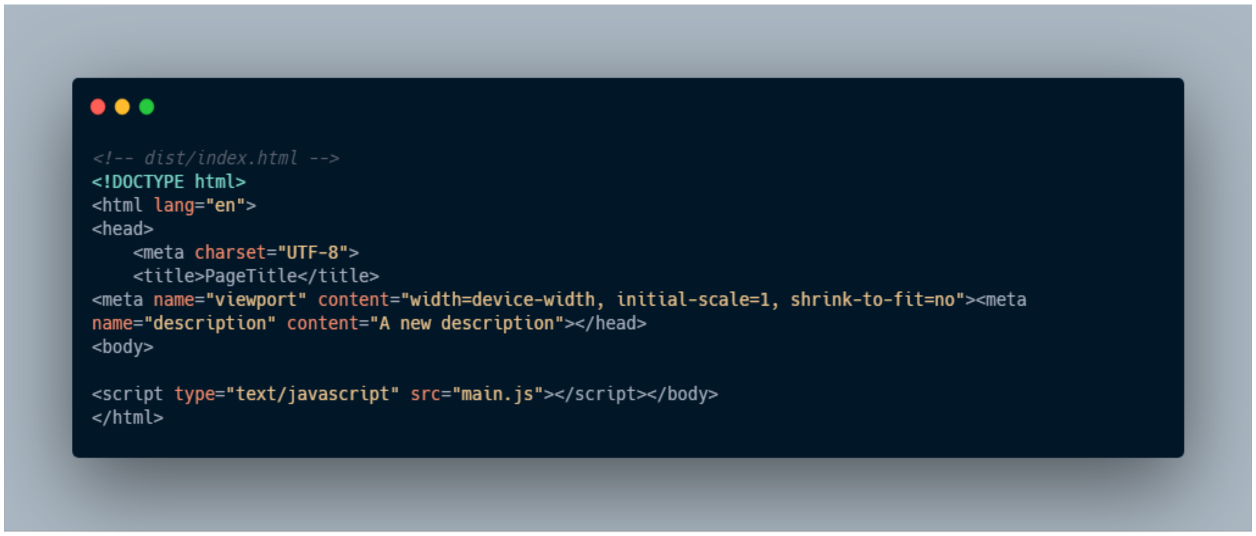
   Below is a sample webpack configuration file, template and built HTML generated by webpack.


```html
<!-- src/index.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title><%= htmlWebpackPlugin.options.title %></title>
</head>
<body>
</body>
</html>
```


```js
// Webpack.config.js
const path = require('path');
const HtmlWebpackPlugin = require('html-webpack-plugin');

const metaContent = {
    viewport: 'width=device-width, initial-scale=1, shrink-to-fit=no',
    description: 'A new description',
};

module.exports = {
    entry: './src/index.js',
    output: {
        filename: 'main.js',
        path: path.resolve(__dirname, 'dist')
    },
    plugins: [
        new HtmlWebpackPlugin({
            title: "PageTitle",
            template: './src/index.html',
            meta: metaContent
        })
    ]
};
```



```
<!-- dist/index.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>PageTitle</title>
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no"><meta
name="description" content="A new description"></head>
<body>

<script type="text/javascript" src="main.js"></script></body>
</html>
```

As seen above, the meta tags needed per page can be added into the webpack configuration and included into each page. The general meta properties like application_name, title, type etc will be included using the above-shown mechanism, while the page specific meta properties will be added in the individual templates which will be kept in the built files.

II. Removal of jinja blocks:

- **Prerender block**: This block is currently only used in one page, splash.html and only contains a link

  ```
  {% block prerender %}
   <link rel="prerender" href="/library">
  {% endblock prerender %}
  ```

  This will be added directly to the page and hence, removed from the base.html
- **Title block**: This is used in almost all pages, but as seen i the webpack example, it can be added injected into the built template. Work is currently ongoing to remove the use of the title block in non-static pages here.
- **Base_url block**: This is also only used in one page, library.html and will be added to the html-webpack-plugin configuration for the library page to be automatically injected into the built page.
- **Header_css block**: This block is used to load stylesheets to all pages by including a file which calls the stylesheets. A CSS loader will be used with webpack to bundle the style sheets. Since some

sheets are only used on development while the minified variant is used in the production build, following the ongoing work on webpack introduction, two webpack configs are made, webpack.dev.config.js and webpack.prod.config.js the unminified CSS will be loaded in webpack.dev.config.js while the minified form of it will be at webpack.prod.config.js.

- **Before end head tag hook:** This construct is currently used to include the google analytics snippet. This snippet will be kept in a script file of its own and the needed parameters (ANALYTICS_ID and SITE_NAME_FOR_ANALYTICS) will be fetched via an AJAX call then they will be loaded into the analytics script. The script will be injected into each page using webpack.

- **Navbar_breadcrumb**: This will be explained in the transclusion section.
- **Local_top_nav_options**: This will be explained in the transclusion section.
- **Content**: This will be explained in the transclusion section.
- **Footer**: This will be explained in the transclusion section.

III. Removal of warnings and loader macro:

The warnings and loader macro will be replaced by the transclusion component which will be created. This component will house the content block and the footer block.

IV. Script inclusion:

This is currently done using webpack here and should be completed before GSOC begins.

2. Removal of remaining jinja templates:

Some jinja constructs which were not covered above include:

- {{ interaction_templates}}
- {{ dependencies_html}}
- {{visualizations_html}}
- {{value_generators_js}}
- {% include <active tab in exploration_editor page> %}

1. **Interaction_templates**

   The interaction_templates are a set of HTML pages which house scripts that are to be injected on the page based on the type of interaction that is expected in the page. Some pages like creators dashboard load all while others load specific interaction templates by specifying the ID of the interaction as defined in constants.js.
   In order to remove {{ interaction_templates }}, the scripts needed for those pages need to be injected into the page using webpack. The process to determine the interactions needed for each page will be by specifying the IDs to be loaded.
   Each interaction template such as Continue.html, DragAndDropSortInput.html, etc loads not one script but multiple scripts. These scripts will be bundled into different chunks. For example, the scripts in Continue.html will be bundled into 1 script which can then be referenced from the dist folder as /dist/continue.js.
   The scripts will be concatenated using webpack-concat-plugin and then injected into the template using html-webpack-plugin.

2. **Dependencies_html:**
   Each interaction comes with an array of dependencies which are required for the interaction. The codemirror and ui_leaflet dependencies also require additional angular modules to be registered during the initialization of the app.
   These dependencies are HTML files containing scripts and in one case (guppy.html), also contain styles.

   As in the case of interactions_html, the scripts needed for each dependency will be bundled into one script file.
   Then all the dependencies for the page will be gotten from an ajax call to the server to get the dependencies.

   However, the dependencies come with some additional angular modules to be added to the page when the app is first initialized. Currently, there are only two possible additional modules (ui.codemirror, ui-leaflet) and they are only required in five pages (creator_dashboard, topic_editor, exploration_editor, skill_editor, exploration_player). The additional modules can be added in the script needed for these pages since they are not required in other pages.

3. **Visualizations_html:**
   This is only used in the editor pages and it returns some directives used for visualizations. They are needed for the editor pages and will be compiled and added to the bundled script for the pages.

4. **Value_generators_js:**
   It is used in the admin page and the exploration editor. It loads two extra directives (Copier Directive and Random Selector). Since they are only required in those two pages, they will be bundled together as a part of the script needed for the pages.

5. **Including the active tab in the exploration editor page:**
   Currently, in exploration_editor.html a check for the active tab is done with angularjs but the HTML for the active tab is included using jinja for most.
   A new directive activeTab directive will be created to manage the logic of showing the currently active tab.
   It will receive the currently active tab as an attribute. And in its controller, an array of objects containing *tab name*, *isComponent* (to denote if it is a component) and *pageLink* (a relative path to the html for the section if it is not a component) will be provided. So when the active tab attribute is set, the corresponding HTML will be loaded and included in the page using ngInclude. If they are components, the components will be loaded instead.

   Also, in the editor_tab.html, the {% include %} construct is used to add other HTML, this will be replaced with ngInclude.

3. A PageData Handler

As earlier stated in the problem statement, the total removal of jinja constructs in pages will lead to data not being injected into the HTML using jinja. Hence a stable method of fetching the data required by the pages should be made.

Essentially, data required by most pages are loaded into the `GLOBALS` variable in `core\templates\dev\head\pages\base.html`.

Other pages like `core\templates\dev\head\pages\collection_editor\collection_editor.html`, `core\templates\dev\head\pages\exploration_editor\exploration_editor.html`, `core\templates\dev\head\pages\creator_dashboard\creator_dashboard.html`, etc have data required by them loaded into the `GLOBALS` variable also as seen here:

```
{% block header_js %}
 {{ super() }}
 <script type="text/javascript">
   GLOBALS.DEFAULT_TWITTER_SHARE_MESSAGE_DASHBOARD =
JSON.parse(

'{{DEFAULT_TWITTER_SHARE_MESSAGE_DASHBOARD|js_string}}');
   GLOBALS.INTERACTION_SPECS =
JSON.parse('{{INTERACTION_SPECS|js_string}}');
   GLOBALS.ALLOWED_INTERACTION_CATEGORIES = JSON.parse(
     '{{ALLOWED_INTERACTION_CATEGORIES|js_string}}');
   GLOBALS.DEFAULT_OBJECT_VALUES = JSON.parse(
     '{{DEFAULT_OBJECT_VALUES|js_string}}');
 </script>
{% endblock header_js %}
```

The above code snippet is from `core\templates\dev\head\pages\creator_dashboard\creator_dashboard.html`, and this is the same approach used by several other pages to load in data from the backend.

Since global variables can typically cause some issues especially with third-party libraries, several measures have to be taken in order to minimise their use. Currently, work is ongoing in the removal of these global variables here.

A good way to send these variables to their respective pages would be by using an AJAX request to send them to the pages.
Following the work that was done in the signup handler, that can be replicated across all other pages. This will help each page controller to manage its data independently.

Note: This is not the only way that can be used to remove these global variables. Other ways currently being worked on include moving them to `\assets\constants.js`.

4. Multi-slot Transclusion

As stated in the problem set, removing the jinja constructs will affect template inheritance thereby requiring a javascript solution for this.

Here is how the template inheritance is currently done using jinja:
All pages extend `core\templates\dev\head\pages\base.html`, using the `{% extends 'pages/base.html' %}` jinja construct and add specific markup to jinja blocks already specified in `core\templates\dev\head\pages\base.html` thus:

```
{% block maintitle %}
 About us - Oppia
{% endblock maintitle %}

{% block header_js %}
  {{ super() }}
{% endblock %}
```

One way angular helps fix the issue is through the use of components with support for multi-slot transclusion.
Transclusion is a mechanism provided for components which allow the component to specify a named slot for allowing additional markup to be added to the component.

Here is how the transclusion works:

A new directive (component) is created. This directive specifies slots which are capable of accepting HTML as content. And when no content is provided, falls back to the default content which was provided during the creation of the component.

In this case, two transclusion components will be created. One which creates two slots to accept *content currently inserted in the* ***content*** *block using jinja*, and *content inserted in the* ***footer*** *block*. This first transclusion component will replace the warnings and loader macro.
The second transclusion component will be much larger than the first and will provide slots for *navbar_breadcrumb, local_top_nav_options,* and for the first component to be used.
Two different transclusion components are needed for the case of embedded explorations. They will not contain breadcrumb, local_top_nav_options and some other content that will be in the second transclusion component, hence a different component for it.

In addition to providing slots for HTML to be inserted, all the current HTML content of the base.html will be available in the transclusion components. For the first, only the HTML in the macro will be available.

Proposed Directory Structure

A new directory called base_components will be created at `core\templates\dev\head\base_components`.
The directory will house the two transclusion components and their templates.

Here is a link to a demo on github.

Transclusion in Angular2

As with many improvements, angular2 came with a better way to do transclusion, using `<ng-content>`.
In angularjs, the transclude slots are defined in the directive script, while in angular2, the slots are defined in the templates. Using a select attribute to name slots, multi slot transclusion is achieved. And it works in the same way as that of angularjs, except that instead of just HTML tags, the slot names can also be used as an attribute.

Testing Plan

- **Frontend tests**:
  Frontend tests will be written to test the individual components like the breadcrumb component.
- **Backend tests:**
  Backend tests will be written to test and ensure the functionality of the page handler which sends back data needed for each page.
- **End to end tests**:
  End to end tests will be written to test the transclusion functionality. The tests will be written to assert the following:
  1. That the content entered into the slots is found in the page.
  2. That the fallback content is used when no content is entered into the slot.
  3. That the content has its scope independent of the transclusion component. This can be tested by passing a property with the content to be inserted in the slot and check for its availability.

Milestones

1. Removing Jinja Constructs from Static Pages

   In order to remove jinja constructs from the static pages like `core\templates\dev\head\pages\splash\splash.html`, `core\templates\dev\head\pages\about\about.html` and `core\templates\dev\head\pages\landing\topic_landing_page.html`, total dependence on `core\templates\dev\head\pages\base.html` will have to be eliminated and a new way to reuse components has to be implemented.

Some methods to approach this has been discussed in the ***implementation approach*** section. These methods will be used to remove the jinja constructs from the static pages.

After studying the pages and their contents, it can be noticed that they have a similar structure :

1. Extends `core\templates\dev\head\pages\base.html`.
2. A `maintitle` section which adds the specific title of that page to the head element of the page.
3. A `header_js` section where they merely call super() which uses the content of the header section in the base template.
4. A `navbar_breadcrumb` section which passes a set of markup to the template to be rendered as the breadcrumb for the page.
5. A `content` section which contains the main content of the page and is passed to the macro of the base template.
6. A `footer` section which renders the footer template.
7. A `footer_js` section which adds some page specific scripts to be loaded.

From the above, since we are going to be removing the base.html dependency, the head section of the HTML will be manually inserted in each page. Therefore, each page will contain its own title tags, add the needed scripts in the header section, link the stylesheets that are required etc.

Points 4 to 6 are actually markup that is added to these pages in the body section of the HTML. Meaning that they can be totally controlled by **angularjs**. Since these sections are required by all static pages with different contents, they can be extracted to a global component.
Using multi-slot transclusion which was described above, a global `static-content` directive can be made which acts as a template which will contain a breadcrumb slot, content slot and footer slot.
Since most of these pages actually use the same footer content, a default footer will be made in the template so the pages won't have to include it again except they actually need a different footer content.

A new page title service has been created which automatically sets the title for the page. So work will be done to refactor all pages to use the new page title service.

Following the ongoing work on webpack introduction into the codebase, the footer_js and the header_js section will automatically be removed as webpack will generate a single bundle per page and this bundle can be included in just one script tag, thereby removing the need for these sections in the base.html

- Week 1: (May 27, 2019 – June 01, 2019)

  Before all of the above can be done, the necessary global components need to be created. So, this first week will be to create the necessary global components: `breadcrumb` component, a `static-content` component which will compose the necessary components in order to be used by the different pages.
  They will be tested accordingly, following the testing plan described above.

  These components will be pushed in separate PRs so that they can be reviewed by my mentor individually, and can be reworked on if there is any problem with any of them.

  The first PR will contain the breadcrumb component. It will be fully tested to ensure that it works as expected.
  The second PR will be on the static-content component. It will use multi-slot transclusion to create slots for breadcrumb, content and footer. This component, using ngInclude will add the default footer to the component and sets it as the default component on the footer view for future uses of this component.
  This component will essentially have markup similar to:

```
//static-content.js
app.directive('static-content', function () {
    return {
        restrict: 'E',
        transclude: {
            'breadcrumb': '?breadcrumb',
            'topNavOptions': '?topNavOptions',
            'content': '?content',
            'footer': '?footer'
        },
        templateUrl: 'static-content.html'
    };
});

<!-- Static-content.html -->
<div style="border: 1px solid black;">
   <div class="title"
ng-transclude="breadcrumb">breadcrumb to be displayed
here</div>
   <div ng-transclude="topNavOptions"></div>
   <div ng-transclude="content"></div>
   <div class="footer" ng-transclude="footer">
       <ngInclude src="footer.html"></ngInclude>
   </div>
</div>
```

Note: This is just a short implementation of how it is supposed to behave basically. Some needed classes, etc are not added here for simplicity. These components will be the base upon which the project is dependent on.

1 PR will be made which implements the components in the `core\templates\dev\head\pages\about\about.html` page. This will also help test the compatibility of the components with the codebase. This will be reviewed and all compatibility issues if any fixed and sent in another commit to the same PR.

- Week 2 (June 03, 2019 – June 08, 2019)

After work on that, another PR will be made which further tests the compatibility of the components: using them in the `core\templates\dev\head\pages\splash\splash.html`, `core\templates\dev\head\pages\signup\signup.html`, `core\templates\dev\head\pages\privacy\privacy.html`, `core\templates\dev\head\pages\teach\teach.html`, `core\templates\dev\head\pages\terms\terms.html`, `core\templates\dev\head\pages\thanks\thanks.html`. After these are done, we can be sure that the components are ready to be implemented in other pages.

- Week 3 (June 10, 2019 – June 15, 2019)

  Make
  `core\templates\dev\head\pages\landing\topic_landing_page.html`,
  `core\templates\dev\head\pages\creator_dashboard\creator_dashboard.html`,
  `core\templates\dev\head\pages\email_dashboard\email_dashboard_result.html`,
  `core\templates\dev\head\pages\get_started\get_started.html`,
  `core\templates\dev\head\pages\landing\stewards\landing_page_stewards.html`,
  to use the components developed earlier.
  One PR will be made to reflect this change but each page will be done in an individual commit ensuring that they are reviewed individually and problems will be fixed appropriately per page.

- Week 4 (June 17, 2019 – June 22, 2019)
  Continue work of week 3 by making the following pages use the components developed earlier:
  `core\templates\dev\head\pages\learner_dashboard\learner_dashboard.html`,
  `core\templates\dev\head\pages\notifications_dashboard\notifications_dashboard.html`,
  `core\templates\dev\head\pages\preferences\preferences.html`,
  `core\templates\dev\head\pages\email_dashboard\email_dashboard.html`,
  `core\templates\dev\head\pages\moderator\moderator.html`.

  One PR will be made to reflect this change but each page will be done in an individual commit to ensure that they are reviewed individually and problems will be fixed appropriately per page.

Proposed Pull Requests For the First Milestone

S/N	PR Description	Proposed creation date	Expected merge date
1.	Breadcrumb component, static-content component, about.html implementation of the components.	May 30th, 2019	June 1st, 2019
2.	Make `signup.html`, `splash.html`, `privacy.html`, `teach.html`, `terms.html`, `thanks.html` use the already developed components.	June 6th, 2019	June 8th, 2019
3.	Change `topic_landing_page.html`, `creator_dashboard.html`, `email_dashboard_result.html`, `get_started.html`, `landing_page_stewards.html` to use the components developed earlier	June 13th, 2019	June 15th, 2019
4.	Make `learner_dashboard.html`, `notifications_dashboard.html`, `email_dashboard.html`, `moderator.html`, `preferences.html`, to use the components developed earlier	June 20th, 2019	June 22nd, 2019

2. Serve more pages statically.

The success of the first milestone will be due to the use of components to aid code reusability and aid template inheritance.

This approach will be continued and be implemented in the other pages.

While maintaining the use of the base template for the non-static pages, most fractions will be changed to use the components that were created earlier and some other methods in order to reduce the dependence on jinja in the non-static pages.

- Week 5 (June 24th - June 29th)

  Move other variants of the splash page (`core\templates\dev\head\pages\splash\splash_av0.html`, `core\templates\dev\head\pages\splash\splash_av1.html`, `core\templates\dev\head\pages\splash\splash_ah0.html`, `core\templates\dev\head\pages\splash\splash_ah1.html`, `core\templates\dev\head\pages\splash\splash_ai0.html`) to use a similar pattern to that which was implemented in the first milestone.

  Submit PR containing the above-mentioned changes to be reviewed and merged.

- Week 6 (July 1st - July 6th)

  At this point, all static pages have been served without any static content and it would be time to move to serve non-static pages using the same mechanism as done for the static pages.
  The major difference in the structure of the non-static pages is the dynamic setting of the page title which is a part of the issue I created here and will be working on before the GSOC period begins, and a large amount of data injected into the GLOBALS which the technique for fixing was discussed above and will be implemented in coming weeks.

  Aside from the above, the non-static pages essentially the same pattern and as such, the static-content component can be used to serve them.

  1 PR will be made this week which uses the static-content component in the `core\templates\dev\head\pages\exploration_player\exploration_player.html`, `core\templates\dev\head\pages\exploration_editor\exploration_editor.html` and `core\templates\dev\head\pages\admin\admin.html`.

Note: The `exploration_editor.html` page loads some scripts using jinja:

```
{{ interaction_templates }}
{{ visualizations_html }}
{{ dependencies_html }}
```

They will be kept as they are to be removed in the third milestone which will use webpack bundle the scripts and inject them into the page.

- Week 7 (July 8th - July 13th)

Following the success of work on week 6, other non-static pages can be implemented with relative ease.
The following pages will be changed to use the static-content component:
`core\templates\dev\head\pages\library\library.html`, `core\templates\dev\head\pages\practice_session\practice_session.html`, `core\templates\dev\head\pages\profile\profile.html`, `core\templates\dev\head\pages\skill_editor\skill_editor.html`, `core\templates\dev\head\pages\story_editor\story_editor.html`, `core\templates\dev\head\pages\topic_viewer\topic_viewer.html`, `core\templates\dev\head\pages\topic_editor\topic_editor.html`, and a PR will be made showing this change with one commit per page.

Note: For library.html, a base URL is added to the title section. This base URL block will be kept as is using jinja to be replaced by webpack injecting the value into the webpack generated library page.
This will be done in the third milestone.

After this, base.html will essentially be performing only the action of adding meta tags and `GLOBALS`. Other blocks will be removed since the individual pages are now using the static-content component.
A pull request will be made to reflect this change at the end of the week.

- Week 8 (July 15th - July 20th)

  This week, the `page_data.py` Module which will be responsible for sending data required by any page as JSON will be created. This module will house several classes as required for each page.

  A corresponding `page_data_test.py` module will be created to test the functionality of the `page_data.py` module.

  Starting with the class to send data to the admin page here: (`core\templates\dev\head\pages\admin\admin.html`), which is currently added to the page using Jinja construct injected into the `GLOBALS` global variable.
  This page will act as a medium to test work on the `page_data.py` module and a corresponding class to test its functionality will be included in the `page_data_test.py` module.
  A single PR with different commits will be made to merge the changes.

Proposed Pull Requests For The Second Milestone

S/N	PR Description	Proposed creation date	Expected merge date
1.	Change other variants of splash page to use the new static-content component.	June 27th, 2019	June 29th, 2019
2.	Using the static-content component in non-static pages: `exploration_player.html`, `exploration_editor.html`, `admin.html.`	July 4th, 2019	July 6th, 2019
3.	Further work on other non-static pages: `library.html`, `practice_session.html`, `profile.html`, `skill_editor.html`, `story_editor.html`, `topic_viewer.html`,	July 11th, 2019	July 13th, 2019

	topic_editor.html to use the components developed earlier		
4.	Remove jinja constructs from base.html that are not needed in the pages.	July 13th, 2019	July 14th, 2019
5.	Create page_data.py module, page_data_test.py module and *PageData* service to retrieve data needed for admin.html .	July 18th, 2019	July 20th, 2019

3. Investigate use of GLOBALS in the pages and use other techniques (including page_data.py module) to make data available to the pages.

At this point, most of the static pages will be rid of their use of jinja constructs and all static pages will not have any use for jinja constructs.
Two major tasks will be remaining: to remove the use of GLOBALS from the pages and to remove dependence on
core\templates\dev\head\pages\base.html to totally get rid of jinja constructs .

Before going into the weekly deliverables for the third milestone, it's important to review the implication of either removing
core\templates\dev\head\pages\base.html or keeping it.

1. SEO Concerns:

   While Google crawler is capable of parsing javascript, not all crawlers are capable of doing so, so to optimize for other it would be good to have most of the page content, especially the meta tags available on the server.
   This is available by default with the base.html template which other pages inherit from.
   So to achieve the same or similar, the head section of each page will have to be coded with pure HTML markup and the meta tags be injected using webpack and the scripts needed for the page will be injected using webpack.

2. Performance:
   Removing dependence on `base.html` will greatly improve page load time, also reducing the load on the server and more work will be done by the user's browser. Consequently, for people with a bad internet connection, downloading the scripts needed for the pages will be slow and can affect performance. However, leveraging on browser caching we can ensure that the scripts are cached and a new script is requested only when there is a new build.

- Week 9 (July 22nd - July 27th)

  Create handlers to send data required for the following pages: `core\templates\dev\head\pages\collection_player\collection_player.html`, `core\templates\dev\head\pages\collection_editor\collection_editor.html`, `core\templates\dev\head\pages\creator_dashboard\creator_dashboard.html`, and create corresponding test classes for each of them.
  A pull request will be made per page to allow for them to be reviewed independently.

- Week 10 (July 29th - August 3rd)

  Extend work on the previous week to include: `core\templates\dev\head\pages\skill_editor\skill_editor.html`, `core\templates\dev\head\pages\topic_editor\topic_editor.html`.
  A pull request will be made for work on each page.

  Investigate the `GLOBALS` variable in base.html.
  Since the `base.html` was preserved for use with the non-static pages (which are not so much), a more general approach to getting the data needed to replace the `GLOBALS` variable in the `base.html` will be used. Methods to be considered include: adding them to `assets\constants.js`, using the `page_data.py` module.
  However, the approach taken will be such that the data will be available to all pages.

At the end of this week, a single PR will be made that will apply the methods to remove the remaining GLOBALS form base.html.

Note: Work on this is also ongoing at the moment and may be completed before the GSOC project begins.

- Week 11 (August 5th - August 10th)

  Investigate GLOBALS in `core\templates\dev\head\pages\profile\profile.html`, `core\templates\dev\head\pages\exploration_player\exploration_player.html`, `core\templates\dev\head\pages\exploration_editor\exploration_editor.html,.`
  It would be to create a class for it in page_data.py module and use it to fetch the needed data.

  1 PR will be made to reflect the changes at the end of the week.

- Week 12 (August 12th - August 17th)

  About search engine optimization for the pages, the meta tags for each page will be added to each page since the *description* and *title* meta properties are usually different for each page.
  Other meta properties for the different pages are the same and can be injected during the build process using the html-webpack-plugin described above.
  A pull request will be made showing the above changes.

  Some scripts which are needed by `exploration_editor.html` page are currently loaded using jinja. They will be bundled with webpack into individual bundles which will then be injected into the built pages using the html-webpack-plugin.
  A pull request will be made to reflect this change

  The library page requires a base URL and in the previous milestone was allowed to be injected using jinja. This will now be injected using webpack during the build process and the value for the URL will be kept in `assets\constants.js`, so it can easily be changed if necessary.

After this is done, there will be no need for the base.html template and will be deleted.
A pull request will be made showing the above changes will be made.

Proposed Pull Requests For the Third Milestone

S/N	PR Description	Proposed creation date	Expected merge date
1.	Create a class in page_data.py and to get data using PageDataService for `collection_player.html`. Also, create a test class to test the newly created class in page_data_test.py.	July 22nd, 2019	July 23rd, 2019
2.	Create a class in page_data.py and to get data using PageDataService for `collection_editor.html`. Also, create a test class to test the newly created class in page_data_test.py.	July 24th, 2019	July 25th, 2019
3.	Create a class in page_data.py and to get data using PageDataService for `creator_dashboard.html`. Also, create a test class to test the newly created class in page_data_test.py.	July 26th, 2019	July 27th, 2019
4.	Create a class in page_data.py and to get data using PageDataService for `skill_editor.html`. Also, create a test class to test the newly created class in page_data_test.py.	July 29th, 2019	July 30th, 2019
5.	Create a class in page_data.py and to get data using PageDataService for `topic_editor.html`. Also, create a test class to test the newly created class in page_data_test.py.	July 31st, 2019	August 1st, 2019
6.	Investigate GLOBALS in base.html and remove them using the	August 2nd, 2019	August 3rd, 2019

	aforementioned techniques.		
7.	Create a class in page_data.py and to get data using PageDataService for `exploration_player.html`. Also, create a test class to test the newly created class in page_data_test.py.	August 5th, 2019	August 6th, 2019
8.	Create a class in page_data.py and to get data using PageDataService for `profile.html`. Also, create a test class to test the newly created class in page_data_test.py.	August 7th, 2019	August 8th, 2019
9.	Create a class in page_data.py and to get data using PageDataService for `exploration_editor.html`. Also, create a test class to test the newly created class in page_data_test.py.	August 9th, 2019	August 10th, 2019
10.	Inject meta tags into the built template for all pages using webpack.	August 13th, 2019	August 14th, 2019
11.	Bundle and inject extra scripts needed into `exploration_editor.html page` using webpack.	August 15th, 2019	August 16th, 2019
12.	Add base_url for library page to constants.js and inject it using webpack to the built template, And deletion of base.html	August 16th, 2019	August 17th, 2019

End of GSOC 2019 !!!

Summer Plans

Timezone:

I will be working from Uyo, Nigeria (GMT + 1) throughout the duration of the Google summer of code.

Time to be Dedicated for GSOC

My Semester examinations end on the last week of May, therefore, from June till August, I will be able to devote 8 - 9 hours daily to work on my GSoC project.

The next semester is the period which I am to go to my industrial training and it doesn't start until about September.

Other Commitments

I am a part of the DSC (Developer student club) University of Uyo, Uyo core team. And we have meetings at most once per month and mostly during the weekend.
Apart from that, I am free to work on the project even during the weekends to make sure the project gets delivered before the deadline.

Communication

Contact Information

Name:	James, James John
Email:	Jamesjay4199@gmail.com
Mobile:	+243-8168272063
Twitter:	@jjaycodes
School:	University of Uyo, Uyo
Course:	B.Eng, Mechanical Engineering

Channels

I typically use Whatsapp, Gmail, and Twitter for communication. I have used slack for teamwork several other times. I don't mind using another medium of communication for communicating with my mentor.

Also, I love using tools like Wrike, Trello Boards and Pivotal Tracker to track progress in each milestone and the project as a whole. The board can be made publicly available and it can be used by my mentor to track my progress and give reviews on the work done so far.

Future Plans for the Project

I intend contributing to Oppia even after GSoC, so I will definitely pay particular attention to this project after GSoC and help in maintaining the project by offering by constantly adding new ways to improve the project and offering reviews to necessary pull requests. I will also be working with the speed team in other ways in improving the speed of Oppia.