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:

   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.

   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.

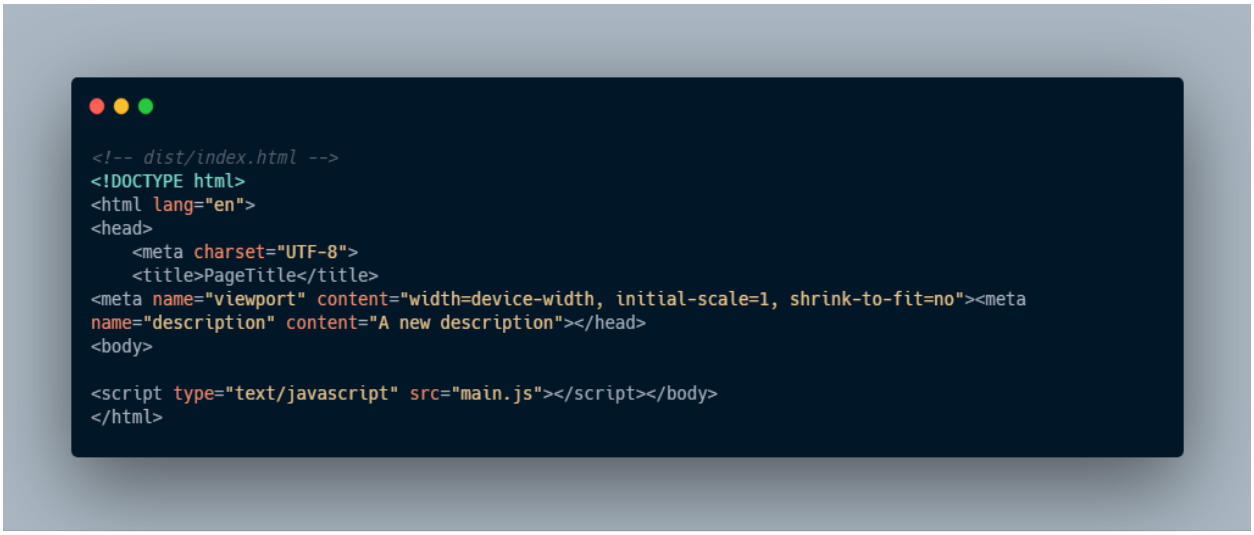
Below is a sample webpack configuration file, template and built HTML generated by webpack.

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

```
// 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 in 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.
- **Header_js block:** This block is used to load all scripts required in the page before the page is rendered like angularjs, etc.
  This will be removed the scripts will be included using webpack.
- **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* 'components/rich_text_components.html' %}
- {% 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.

6. **Rich Text Components:**
   Currently, in some pages, {% *include* 'components/rich_text_components.html' %} is used to include some additional scripts. These scripts will be bundled into a chunk and will be included into the pages where they are required.

3. Sending data to pages

The total removal of jinja from each page will lead to a problem of the pages not having all the data required for complete functionality.
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 creating a data handler for each page that requires additional data. This data handler will return all the data required for the page.

Following the work that was done in the signup handler, that can be replicated across all other pages. This will make the handler to be page specific and handle all things required by the page.

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.

Multi-slot 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.

The first transclusion component will essentially have markup similar to:

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

<!-- base-template.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>
```

The second transclusion component will be similar to the first but will receive the main content of the page and the footer content if any.

```
//page-content.js
    return {
        restrict: 'E',
        transclude: {
            'content': '?content',
            'footer': '?footer'
        },
        templateUrl: 'page-content.html'
```

```
    };
});
```

```
<!-- page-content.html -->
<div style="border: 1px solid black;">
    <div ng-transclude="content"></div>
    <div class="footer" ng-transclude="footer">
        <ngInclude src="footer.html"></ngInclude>
    </div>
</div>
```

Here is a link to a demo on github.

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.

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. Create transclusion component and use it in all static pages thereby removing all jinja constructs from the static pages

   Static pages as used here include:
   - ➔ **about.html**
   - ➔ **splash.html**
   - ➔ **splash_at0.html**
   - ➔ **splash_at1.html**
   - ➔ **signup.html**
   - ➔ **privacy.html**
   - ➔ **teach.html,**
   - ➔ **terms.html**
   - ➔ **topic_landing_page.html**
   - ➔ **email_dashboard_result.html**
   - ➔ **get_started.html**
   - ➔ **landing_page_stewards.html**
   - ➔ **thanks.html**
   - ➔ **learner_dashboard.html**
   - ➔ **notifications_dashboard.html**
   - ➔ **preferences.html**
   - ➔ **practice_session.html**
   - ➔ **email_dashboard.html**
   - ➔ **moderator.html.**

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

  This week will focus on creating the needed transclusion components and using it on the about page.

  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.

Another PR will be made which uses webpack to set the meta tags needed in the about page

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

  Use the new components in the following pages
  `Core\templates\dev\head\pages\splash\splash.html`,
  `core\templates\dev\head\pages\splash\splash_at0.html`,
  `core\templates\dev\head\pages\splash\splash_at1.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`.
  Use webpack to add meta tags to the above pages.
  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\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`,
  `core\templates\dev\head\pages\thanks\thanks.html`
  to use the components developed earlier.
  Use webpack to add meta tags to the above pages.
  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\practice_session\practice_session.html`,
  `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`.
  Use webpack to add meta tags to the above pages.
  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.

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 4th, 2019
2.	Make `signup.html`, `splash.html`, `splash_at0.html`, `splash_at1.html`,`privacy.html`, `teach.html`, `terms.html`, `thanks.html` use	June 6th, 2019	June 11th, 2019

	the already developed components.		
3.	Change `topic_landing_page.html`,`email_dashboard_result.html`, `get_started.html`, `landing_page_stewards.html` to use the components developed earlier	June 13th, 2019	June 17th, 2019
4.	Make `learner_dashboard.html`, `notifications_dashboard.html`,`email_dashboard.html`, `moderator.html`, `preferences.html`, to use the components developed earlier	June 19th, 2019	June 22rd, 2019

2. Use transclusion components in dynamic pages and create handlers for pages where needed.

   Continue work on the first month to include the dynamic pages.
   The following pages are considered as dynamic here:

   - ➔ **creator_dashboard.html**
   - ➔ **exploration_player.html**
   - ➔ **exploration_editor.html**
   - ➔ **profile.html**
   - ➔ **admin.html**
   - ➔ **library.html**
   - ➔ **skill_editor.html**
   - ➔ **story_editor.html**
   - ➔ **topic_viewer.html**
   - ➔ **topic_editor.html**
   - ➔ **collection_editor.html**
   - ➔ **collection_player.html**.

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

  Make the following pages use the transclusion components `core\templates\dev\head\pages\library\library.html`, `core\templates\dev\head\pages\topics_and_skills_dashboard\topics_and_skills_dashboard.html`, `Core\templates\dev\head\pages\creator_dashboard\creator_dashboard.html`, `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.`
  Use webpack to add meta tags to the above pages.

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

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

  Extend work done the previous week to include: `core\templates\dev\head\pages\profile\profile.html`, `Core\templates\dev\head\pages\collection_editor\collection_editor.html`, `core\templates\dev\head\pages\collection_player\collection_player.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`.
  Use webpack to add meta tags to the above pages.

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

  Create handlers for the following pages:
  `Core\templates\dev\head\pages\admin\admin.html`,
  `core\templates\dev\head\pages\collection_editor\collection_editor.html`,
  `core\templates\dev\head\pages\collection_player\collection_player.html`,
  `core\templates\dev\head\pages\topic_editor\topic_editor.html`,

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

  Create handlers for the following pages:
  `core\templates\dev\head\pages\skill_editor\skill_editor.html`,
  `core\templates\dev\head\pages\exploration_editor\exploration_editor.html`,
  `core\templates\dev\head\pages\exploration_player\exploration_player.html`,
  `core\templates\dev\head\pages\profile\profile.html`,

Proposed Pull Requests For The Second Milestone

S/N	PR Description	Proposed creation date	Expected merge date
1.	Use transclusion components in: `library.html`, `topics_and_skills_dashboard.html`, `creator_dashboard.html`, `exploration_player.html`, `exploration_editor.html` and `admin.html`	June 27th, 2019	July 2nd, 2019
2.	Use transclusion components in: `core\templates\dev\head\pa`	July 4th, 2019	July 8th, 2019

	ges\profile\profile.html, collection_editor.html, collection_player.html, skill_editor.html, story_editor.html, topic_viewer.html,topic_editor.html.		
3.	Create handlers for the following pages: admin.html, collection_editor.html, collection_player.html, topic_editor.html,	July 11th, 2019	July 17th, 2019
4.	Create handlers for the following pages: skill_editor.html, exploration_editor.html, exploration_player.html, profile.html,	July 18th, 2019	July 22nd, 2019

3. Use Webpack to inject other scripts needed by the dynamic pages thereby totally removing jinja from the codebase.

   At this point, all static pages have been served with the transclusion component and have no jinja use in them, while some dynamic pages like creator_dashboard.html, exploration_editor.html, exploration_player.html, skill_editor.html, topic_editor.html, admin.html, story_editor.html use some other jinja constructs to add other script and dependencies into the pages.

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

     Use webpack to get rid of {{ interaction_templates}} in the following pages:
     Core\templates\dev\head\pages\creator_dashboard\creator_dashboard.html,
     core\templates\dev\head\pages\exploration_editor\explo

ration_editor.html,
core\templates\dev\head\pages\exploration_player\exploration_player.html,
core\templates\dev\head\pages\skill_editor\skill_editor.html,
core\templates\dev\head\pages\topic_editor\topic_editor.html

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

  Use webpack to get rid of {{ dependenties_html }} in:
  Core\templates\dev\head\pages\creator_dashboard\creator_dashboard.html,
  core\templates\dev\head\pages\exploration_editor\exploration_editor.html,
  core\templates\dev\head\pages\exploration_player\exploration_player.html,
  core\templates\dev\head\pages\skill_editor\skill_editor.html,
  core\templates\dev\head\pages\topic_editor\topic_editor.html

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

  Use webpack to get rid of {{ visualizations_html }} in:
  Core\templates\dev\head\pages\creator_dashboard\creator_dashboard.html,
  core\templates\dev\head\pages\exploration_editor\exploration_editor.html,
  core\templates\dev\head\pages\skill_editor\skill_editor.html,
  core\templates\dev\head\pages\topic_editor\topic_editor.html

  Use webpack to get rid of {{ value_generators_js }} in:
  Core\templates\dev\head\pages\admin\admin.html,
  core\templates\dev\head\pages\exploration_editor\exploration_editor.html

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

  Create a new chunk for rich text components and add it to the bundle for the following pages:
  `core\templates\dev\head\pages\admin\admin.html`,
  `core\templates\dev\head\pages\creator_dashboard\creator_dashboard.html`,
  `core\templates\dev\head\pages\exploration_editor\exploration_editor.html`,
  `core\templates\dev\head\pages\exploration_player\exploration_player.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_editor\topic_editor.html`.
  Work on showing active tab without using jinja using the already described method in the exploration_editor page.

Proposed Pull Requests For the Third Milestone

S/N	PR Description	Proposed creation date	Expected merge date
1.	Get rid of {{interaction_templates}}	July 25th, 2019	July 30th, 2019
2.	Get rid of {{dependencies_html}}	August 1st, 2019	August 5th, 2019
3.	Get rid of {{visualizations_html}} and {{value_generators_js}}	August 9th, 2019	August 11th, 2019
4.	Get rid of {% *include* 'components/rich_text_components.html' %}	August 13th, 2019	August 16th, 2019
5	Work on showing active tab in exploration editor page.	August 16th, 2019	August 19th, 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.