Google Summer Of Code 2018
Rich Text Editor Upgrade

Nitish Bansal

Name of the Project

Rich Text Editor Upgrade

Why am I interested in working with Oppia?

The purpose which Oppia serves is the foremost reason which drives me towards contributing to Oppia. Being a student myself, I understand the importance of this organisation for millions of students who do not have access to classrooms due to various reasons. Oppia provides a platform to gain and share knowledge without any hurdles.

The work environment in Oppia also motivates to learn and contribute with the team of Oppia. All the members associated with Oppia are active and enthusiastic about their work. They are always available to help and review my work whenever I am stuck. Contributing to Oppia for the last two months has not only boosted my technical knowledge but also taught me the importance of working in a team and helping my fellow contributors.

I would love contributing to Oppia to provide a better experience to all the teachers and students using Oppia.

I would want to continue to contribute in Oppia even after the GSoC period ends.

What interests me about this project? Why is it worth doing?

The platform of Oppia is undoubtedly a great platform for students to learn and for teachers (authors) to share their knowledge. I would love to improve any thing that makes the work of a student or a teacher easier. Improving the current editor would definitely be a great help for the teachers to write their content.

In the Oppia's current editor, there are many problems in the preview mode which do make the work of a teacher difficult. Some of the issues which indicate the problem with the current RTE are:

1. Issue #1811
2. Issue #1933
3. Issue #1810

Due to these issues, a teacher has to save the content and then cross check about how the content is actually looking. So the current editor needs to be improved. More the number of teachers who love this platform, the better it would be.

Prior Experience

I have been working with Python since last two years. I have implemented the following in python:

1. Digit Classifier using neural networks by training the model on MNIST dataset.
2. Video chat app using WebRTC.
3. Cloud Music player using Django framework in which a user adds and listens to songs. These songs are uploaded by the user on per account basis.

I have also completed a course on Network Programming in python.

I am fluent and experienced in Javascript and HTML/CSS too. Most of my PR's on github are associated with Javascript. I had implemented a multiplayer game Armoured Aces where the objective of each player is to kill other players tank. I have also contributed in mozilla/brackets and processing/p5.js. Both of these organisations needed a good familiarity with Javascript and HTML/CSS.

I have not implemented anything based on AngularJS but when I started contributing in Oppia I learnt about AngularJS and it was not very new, knowing the basics of Javascript. It was difficult in the beginning but as time passed I understood the code structure of Oppia, went through tutorials on AngularJS, searched whatever I could not understand in Oppia codebase. Now I feel I have a good command over this language too.

I was not an active GitHub user till last year but for the last four to five months, I have been actively contributing to open source. I have made around 40PRs in this time trying to give back to the open source community. Link to my github page is bansalnitish.

I have collected sufficient information about content migration and testing. I have also gone through the documentation of CKEditor and TextAngular on their respective websites and hence find myself in a strong position for the project.

Some of my PR's and issues

Some of my PR's :

- mozilla/brackets #916 - Quick Edit UI for Padding
- mozilla/brackets#918 - Quick Edit UI for Margin
- oppia/oppia #4770 - Enhancement to Correctness Footer
- oppia/oppia #4740 - Added Speech Recognition Functionality

A total of my **14 PR's** in Oppia are merged till date.

The following issues were also created in Oppia:

- #4660
- #4737

Apart from these issues and PR's I have been following almost all the PR's after my first PR. I have tried to help my fellow contributors and reviewed PR's like #4787. This has increased my understanding of the Oppia source code for those files also which are not directly associated with my project.

Project Plan

This section include a list of topics that will be covered during GSoC. I have written a detailed explanation of each these along with mock implementation in the implementation strategy section.

The list includes these five major topics:

1. Content Migration

Problem: The current database stores html content in a form supported by TextAngular which is different from that used by CKEditor. Some content from the older editor jWysiwyg may also be present which too needs to be migrated finally to CKEditor.

I found out the differences in tags produced by TextAngular vs CKEditor. The differences are summarized in the table shown below.

Text Angular		CKEditor	
Tag	Allowed parent tags	Tag	Allowed parent tags
<b>	<i> <li> <p> <pre>	<strong>	<em> <li> <p> <div>*
<blockquote>	<blockquote> no parent	<blockquote>	<blockquote> no parent
<br />	<b> <i> <li> <p>	<br />	<strong> <em> <li> <p> <div>*
<div>	<blockquote>	<div>	<div> tag will not be produced by user
<i>	<b> <li> <p> <pre>	<em>	<strong> <li> <p> <div>*
<li>	<ol> <ul>	<li>	<ol> <ul>
<ol> / <ul>	<blockquote> <li> <pre> <div> no parent	<ol> / <ul>	<blockquote> <li> <div>* no parent
<p>	<blockquote> <div> <pre> no parent	<p>	<blockquote> no parent
<pre>	<blockquote> no parent	<div>*	<blockquote> No parent
<oppia-noninteractive-link>	<b> <i> <li> <p>	<oppia-noninteractive-link>	<strong> <em> <li> <p>

	<pre>		<div>*
<oppia-noninteractive-math>	<b> <i> <li> <p> <pre>	<oppia-noninteractive-math>	<strong> <em> <li> <p> <div>*
<oppia-noninteractive-image>	<li> <p> <pre>	<oppia-noninteractive-image>	<li> <p> <div>*
<oppia-noninteractive-collapsible>	<li> <p> <pre>	<oppia-noninteractive-collapsible>	<li> <p> <div>*
<oppia-noninteractive-tabs>	<li> <p> <pre>	<oppia-noninteractive-tabs>	<li> <p> <div>*
<oppia-noninteractive-video>	<li> <p> <pre>	<oppia-noninteractive-video>	<li> <p> <div>*

`<div>*` tag is replacement of `<pre>` tag, `<div>*` = `<div style="background:#eeeeee; border:1px solid #cccccc; padding:5px 10px">`

Migration of tags in detail

<b>

`<b>` tag will be migrated to `<strong>` and the parents for `<b>` tag in both the editors are same,, so no other migration is needed.

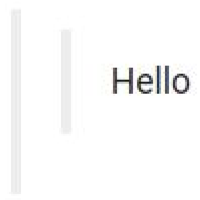
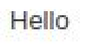
```
for bold in soup.findAll('b'):
    bold.name = 'strong'
```

<blockquote>

`<blockquote>` tag is used for indentation. The number of `<blockquote>` tags used is equivalent to the number of times user increases indentation. Indent plugin in CKEditor does not produce blockquotes on indenting. CKEditor has a blockquote plugin but it does not function in the same manner as the increase indent option in TextAngular. So, I would add a new plugin for indentation which adds blockquotes on indentation.

No migration is needed for this tag since the valid parents are same except `<pre>` which will be migrated separately.

Here are outputs from both the editors if we use increase indent button:

TextAngular	CKEditor
`<blockquote><blockquote><p>Hello</p></blockquote></blockquote>`	`<p style="margin-left:80px">Hello</p>`
 Hello	 Hello

`<blockquote>` tag is not nested inside any other tag i.e there can be no parent of blockquote tag can be produced by a user in CKEditor. Though `<blockquote>` tag can be nested inside `<blockquote>` tag in textAngular. However if this nested `<blockquote>` tag is provided as source html code in CKEditor it produces same visible output as TextAngular. Hence we don't need any migration here.

Indentation in TextAngular adds blockquotes to all the cases except when single list items are indented. The custom plugin which I will add will function in the same manner.

<br />

Use of `<br />` tag is allowed in both the editors. In CKEditor, if allowedContent is true `<br />` tag will not be replaced by `&nbsp;`. If `<br />` tag is not migrated, it will not create any problem since CKEditor will render it correctly as a blank line, so we can keep these tags as such. However if the author creates a blank line in CKEditor its HTML output would be stored as `<p>&nbsp;</p>` whereas in TextAngular it would be `<p><br /></p>`. If `<br />` is present along with some other content in parent tag, then it remains the same in CKEditor and is not replaced by `&nbsp;`. So, I will migrate only blank lines.

```
for linebreak in soup.findAll('br'):
    parent_tag = linebreak.parent
    if parent_tag.name == 'p' and parent_tag.get_text() == '':
        linebreak.replaceWith('&nbsp;')
```

<div>

In textAngular this tag is produced if we click on increase indent button first and then write any content. This tag is always present between `<blockquote>` and `<p>` tags.

```
<blockquote><div><p>Content</p></div></blockquote>
```

For CKEditor I will remove `<div>` tags html content since in CKEditor `<div>` cannot be produced by a user except in the special container used as a replacement for `<pre>` tag. This step will be performed before migrating `<pre>` tags to avoid migration of the newly added `<div>` tags as replacement for `<pre>`.

```
for div in soup.findAll('div'):
    div.unwrap()
```

<i>

`<i>` tag will be migrated to `<em>` and the parents for `<i>` tag in both the editors are same, so no other migration is needed.

```
for italic in soup.findAll('i'):
    italic.name = 'em'
```

<li>

Lists have the same valid html criteria except when they are indented.

Indenting the list as a whole will add blockquote for indentation. For indenting the list as a whole, use increase indent button first and then use the list button.

Indenting any list item say by two spaces produces different html output as follows:

TextAngular	CKEditor
`<ul>`	`<ul>`
`    <li>hello1</li>`	`    <li>hello1`
`    <ul>`	`    <ul style="margin-left:40px">`
`        <ul>`	`        <li>hello2</li>`
`            <li>hello2</li>`	`    </ul>`
`        </ul>`	`    </li>`
`    </ul>`	`    <li>hello3</li>`
`    <li>hello3</li>`	`</ul>`
`</ul>`	

Migration for any such case will require checking the number of consecutive `<ul>` or `<ol>` tags and if that is greater than one, replacing it with a single `<ul>` or `<ol>` respectively and adding margin styling to it.

```
# same procedure will be followed for ol
for ul in soup.findAll('ul'):
    cnt = 0
    while True:
        child = ul.findChildren()
        if child:
            first_child = child[0]
        if first_child.name == 'ul':
            cnt += 1
            first_child.unwrap()
        else:
            if cnt >= 1:
                ul['style'] = 'margin-left:%d' %(40 * cnt)
            break
```

The plugin created for indentation will take care of the difference in indentation of lists when single list items are indented i.e. blockquote will not be added for indentation if user indents single list items.

<ol> / <ul>

`<div>` tag cannot be produced as a parent of `<ol>` or `<ul>` tag in CKEditor. This will fixed by migration of `<div>` tag (unwrapping `<div>` tag).

`<p>`

`<div>` tag cannot be produced as a parent of `<p>` tag in CKEditor. This will fixed by migration of `<div>` tag (unwrapping `<div>` tag).

Migration is only needed when `<p>` tag is present inside `<pre>` tag (this form can be produced by using lists inside pre and then removing the lists). In this case, I will remove the `<p>` tag.

```
for p in soup.findAll('p'):
    if p.parent.name == 'pre':
        p.unwrap()
```

`<pre>`

In this case we will have tags of the form `<pre>Content</pre>`. We can also have `<pre><p>Content</p></pre>` (this form can be produced by using lists inside pre and then removing the lists). Both these forms will be replaced by the form mentioned in the table above. CKEditor does not have a pre option. Instead it has special container feature which produces same result as pre.

```
for pre in soup.findAll('pre'):
    pre.name = 'div'
    pre['style'] = 'background:#eeeeee; border:1px solid #cccccc;
padding:5px 10px'
```

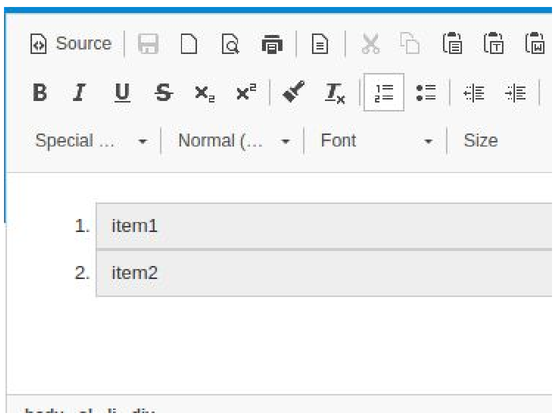
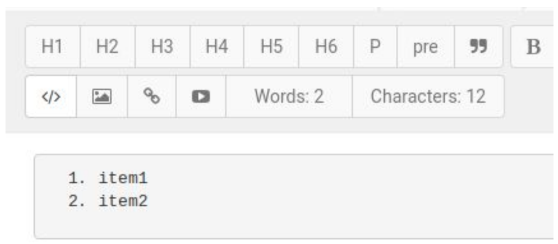
If we use pre with a list, the html output in textAngular would be

```
<pre><ul><li>item1</li><li>item2</li></ul></pre>
```

and that in CKEditor would be

```
<ul>
  <li>
  <div style="background:#eeeeee; border:1px solid #cccccc; padding:5px
10px">item1</div>
  </li>
  <li>
  <div style="background:#eeeeee; border:1px solid #cccccc; padding:5px
10px">item2</div>
  </li>
</ul>
```

However these two outputs will differ as shown here:

CKEditor	Text Angular
 1. item1 2. item2	 1. item1 2. item2

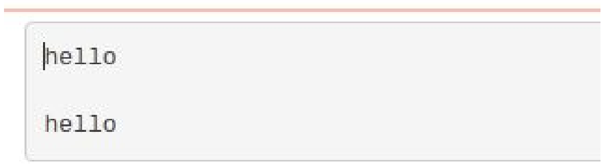
So I will create a custom plugin for pre which produces the same format as followed by TextAngular in all cases. So I need to convert the form

`<pre><ul><li>item1</li><li><item2></li></ul></pre>`
to a format compatible with CKEditor:

```
<div style="background:#eeeeee; border:1px solid #cccccc; padding:5px
10px;">
<ul>
    <li>item1</li>
    <li>item2</li>
</ul>
</div>
```

This will be done as well when migrating `<pre>` as in code above. No additional code is needed for this.

Also if user presses Shift + Enter inside pre container, blank lines are added in html output instead of `<br />` tag. Here is the output when user presses Shift + Enter inside pre container:

`<pre>hello`
 	
`hello</pre>`
	 hello
	
	hello

The output for the above case in CKEditor is rendered as:

```
<div style="background:#eeeeee; border:1px solid #cccccc; padding:5px
10px">hello<br />
hello</div>
```

So, any blank lines within `<pre>` tags need to be migrated to `<br />`.

```
def inject_tag(text, start, end, tagname):
    root = text
    while root.parent:
        root = root.parent

    before = root.new_string(text[:start])
    new_tag = root.new_tag(tagname)
    after = root.new_string(text[end:])

    text.replace_with(before)
    before.insert_after(new_tag)
    new_tag.insert_after(after)
    return after

soup = BeautifulSoup(html_data, 'html.parser')
for pre in soup.findAll('pre'):
    text = pre.string
    start = text.find('\n')
    while start >= 0:
        end = start + 1
        text = inject_tag(text, start, end, 'br')
        start = text.find('\n')
```

Rich Text Components

Rich text components are of the form `<oppia-noninteractive-x>` where x can be math, image, link, video, tabs, collapsible. I will use the same RTE components. Since the RTE components same, the html output will also be same and migration is not required. Only difference in valid parent tags is due to pre and div which will be solved by migration of pre tag.

Goal: Transform the current content into the format used by CKEditor. The above mentioned details will come in handy while carrying out this task.

2. Testing for Content Migration

Problem: Content migration can result in data format which is incompatible with CKEditor and hence will not function properly with CKEditor.

Goal: Test for all the cases and ensure that all content is migrated safely and works as expected on CKEditor.

3. Integrating CKEditor

Problem: Current RTE uses Text Angular which has many issues with preview format of rich text components. Using CKEditor instead of Text Angular will fix all these issues.

Goal: CKEditor integrated in RTE with new plugins.

4. Testing features of new editor and resolving bugs

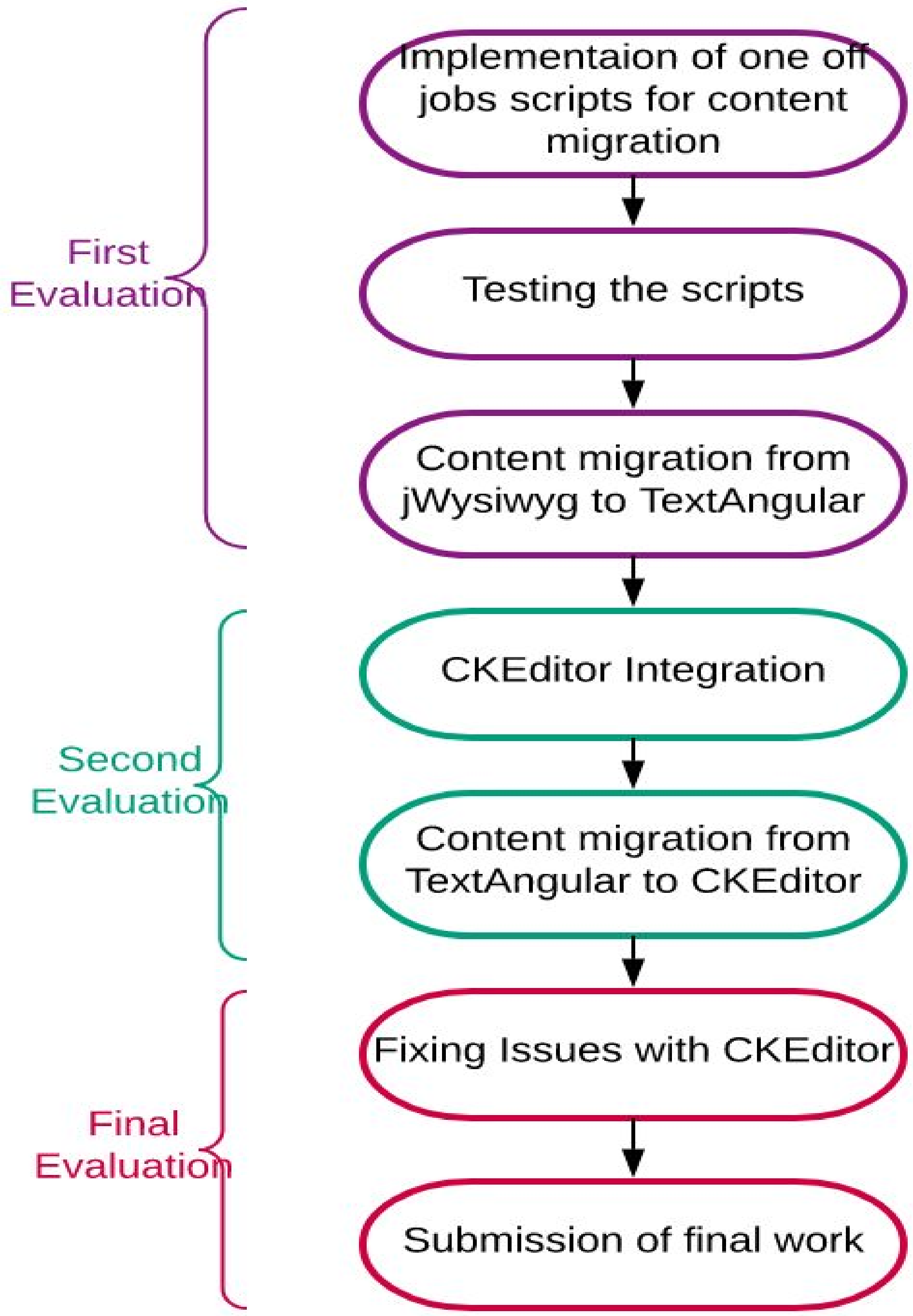
Problem: Integration of a new editor will produce bugs and issues.

Goal: Testing all the plugins and features to produce a bug free integration.

5. Documentation

Problem: The integration of a new editor will modify our RTE by introducing its associated plugins as well modifying the existing features. This needs to be documented for users as well as developers.
Goal: Document the complete work for users and developers of Oppia.

Project Workflow


First Evaluation
Implementaion of one off jobs scripts for content migration
Testing the scripts
Content migration from jWysiwyg to TextAngular
Second Evaluation
CKEditor Integration
Content migration from TextAngular to CKEditor
Final Evaluation
Fixing Issues with CKEditor
Submission of final work

I will firstly begin by writing the one off job scripts, one for content migration from jWysiwyg to TextAngular and one for TextAngular to CKEditor. I will perform testing of these scripts. Then I will use the first script and perform content migration from jWysiwyg to TextAngular.

After this I will replace the TextAngular editor with the CKEditor. Then I will perform content migration from TextAngular to CKEditor. I will be having a ready one off job script to perform the migration (script will be made ready in first phase).

Finally I will fix all the issues that come in way after the CKEditor integration and content migration.

Documentation will occur simultaneously along with all the three phases.

Implementation Strategy

One-off job for performing content migration

The goal of content migration is to convert the html content in the exploration data into a format compatible with CKEditor. This will take place in two phases:

1. Migrate html content from jWysiwyg to TextAngular format
2. Migrate html content from TextAngular to CKEditor format

I will describe a process for migration from TextAngular to CKEditor. The same will be used for migration from jWysiwyg to TextAngular.

Step 1: Find the fields containing html data in an exploration

The html content is present in the html fields in state content and in the feedback fields in interaction outcomes. It can also be nested within these fields. I will find all such fields which can contain html content and then perform the migration accordingly. I have demonstrated the working for html fields in state content. A similar procedure will be used to extract the html from other fields.

Step 2: Validate current html content

I have prepared a list (listed above) of all the supported tags in CKEditor and TextAngular. It is necessary to validate this list and confirm that any other tags are not found in the current html content. I will write a MR job that will check the current content and validate that tags of only the currently used editor are present in the html content. To be more precise, the MR job checks that:

- Only valid TextAngular tags and html is present before migrating to CKEditor. This will find any jWysiwyg tags or html format which are not present in TextAngular supported format.
- After migration from TextAngular to CKEditor on backup data, this job will find any TextAngular tags and html which are not present in CKEditor supported format.

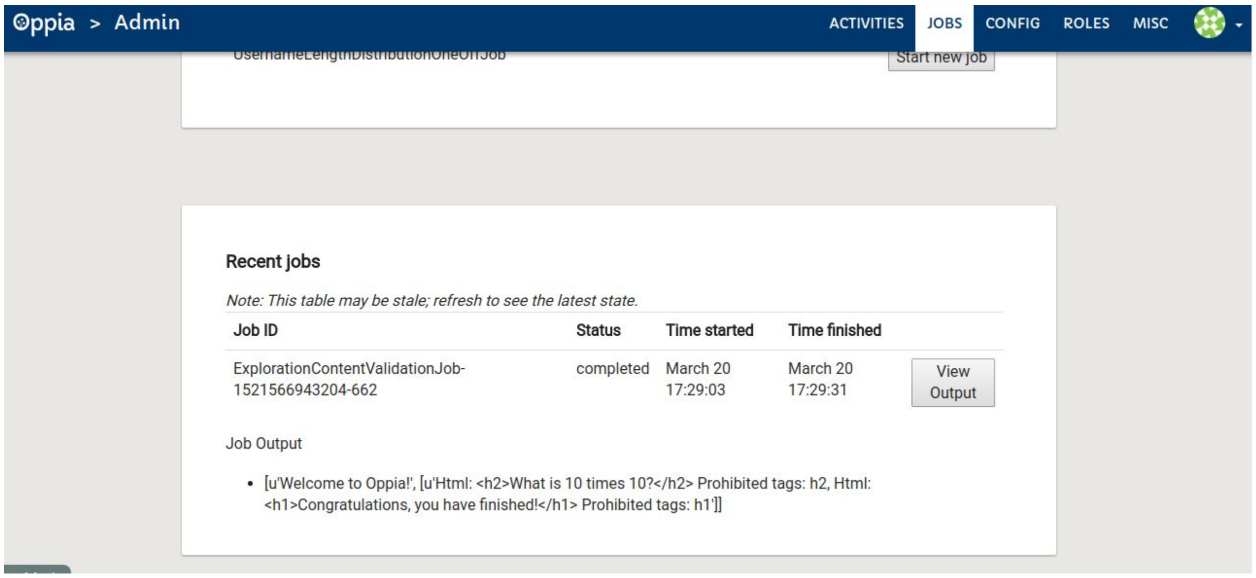
This MR job would also be used for testing phase.

Here is a mock which I have implemented:

```
class ExplorationContentValidationJob(jobs.BaseMapReduceOneOffJobManager):
    """Job that checks the html content of exploration and validates it.
    """
    @classmethod
    def entity_classes_to_map_over(cls):
        return [exp_models.ExplorationModel]

    @staticmethod
    def map(item):
        if item.deleted:
            return
        exploration = exp_services.get_exploration_from_model(item)
        tagerr_msg = []
        htmlerr_msg = []
        for state_name, state in exploration.states.iteritems():
            html_data = state.content.html.encode('utf-8')
            soup = BeautifulSoup(html_data, 'html.parser')
            ALLOWED_TAG_LIST = ['List_of_allowed_tags_in_current_editor']
            used_tag_list = soup.find_all()
            tagerr_list = []
            htmlerr_list = []
            for tag in used_tag_list:
                tag_name = str(tag.name)
                if tag_name not in ALLOWED_TAG_LIST:
                    tagerr_list += [tag_name]
            tagerr_list = list(set(err_list))
            tagerr_list = ', '.join(err_list)
            if len(tagerr_list):
                tagerr_msg += ['Html: %s Prohibited tags: %s' %(
                    html_data, err_list)]
            # This is a demonstration for p tag only. A similar method will
            # be used for all other tags.
            ALLOWED_PARENT_LIST =
['list_of_allowed_parents_in_current_editor']
            for tag in soup.findAll('tag_name'):
                parent = tag.parent.name
                if parent not in ALLOWED_PARENT_LIST:
                    htmlerr_list +=[html_data]
            htmlerr_list = list(set(htmlerr_list))
            htmlerr_list = ', '.join(htmlerr_list)
            if len(htmlerr_list):
                htmlerr_msg += ['Invalid html: %s' %(err_list)]

        tagerr_msg = ', '.join(tagerr_msg)
        htmlerr_msg = ', '.join(htmlerr_msg)
        Err_msg = tagerr_msg + ‘\n’ + htmlerr_msg
        if len(err_msg):
            yield(item.title, err_msg)

    @staticmethod
    def reduce(key, values):
        yield (key, values)
```

This job will be added to **exp_jobs_one_off.py**.

I added some prohibited tags to the welcome exploration, disabled html cleaning and tested this job on the admin page. Here is the output I obtained:



Recent jobs

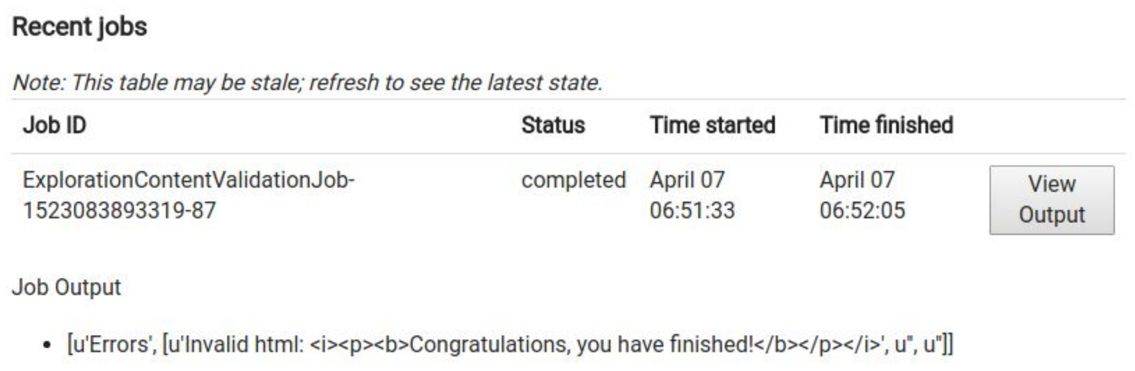
Note: This table may be stale; refresh to see the latest state.

Job ID	Status	Time started	Time finished	
ExplorationContentValidationJob-1521566943204-662	completed	March 20 17:29:03	March 20 17:29:31	View Output

Job Output

- [u'Welcome to Oppia!', [u'Html: <h2>What is 10 times 10?</h2> Prohibited tags: h2, Html: <h1>Congratulations, you have finished!</h1> Prohibited tags: h1']]

I also tested this job on some invalid html. Here is the output for the same:



Recent jobs

Note: This table may be stale; refresh to see the latest state.

Job ID	Status	Time started	Time finished	
ExplorationContentValidationJob-1523083893319-87	completed	April 07 06:51:33	April 07 06:52:05	View Output

Job Output

- [u'Errors', [u'Invalid html: <i><p><b>Congratulations, you have finished!</b></p></i>', u'', u'']]

Step 3: Migrate html content to the desired format

I will write a conversion functions for schema migration which will:

1. Extract the html content from the exploration dict
2. Parse the html content using BeautifulSoup
3. Manipulate the DOM tree to convert html to a suitable format
4. Apply changes to the exploration dict
5. Save the exploration

Here is mock which I have implemented for the same:

These conversion functions will be added to **exp_domain.py**.

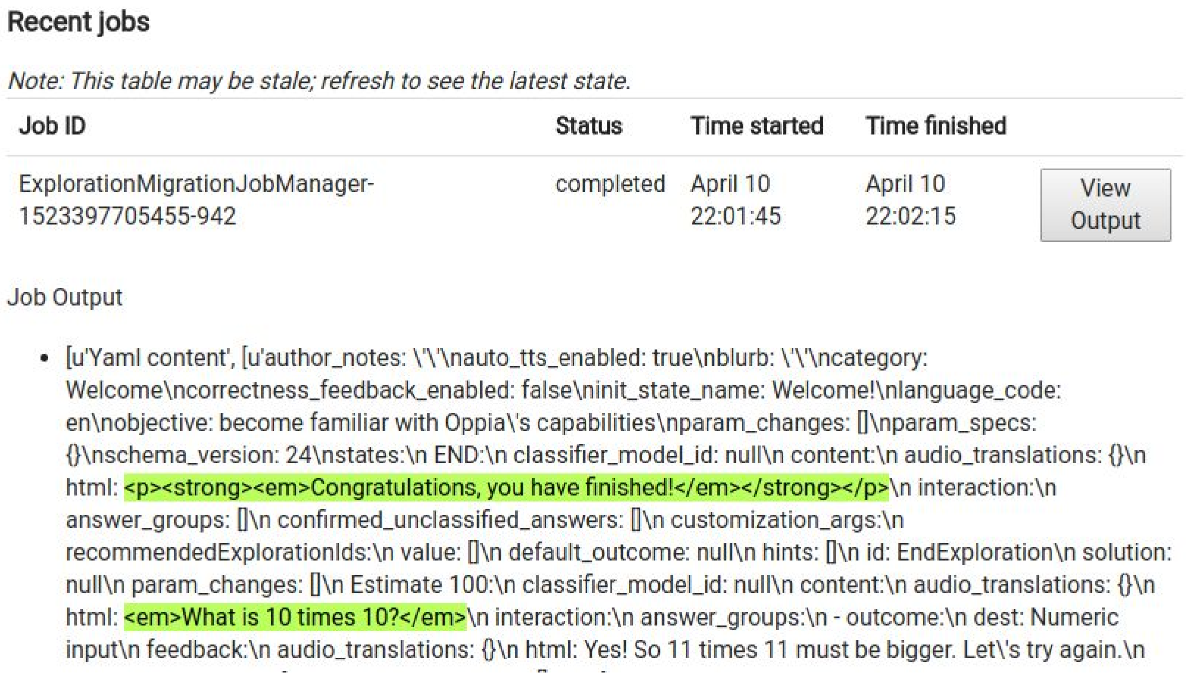
```
def _convert_vN_dict_to_vN+1_dict(cls, exploration_dict):
    """ Converts a vN exploration dict into a vN+1 exploration dict.

    Migrates html content from textAngular to CKEditor format.
    """
    exploration_dict['schema_version'] = N+1

    exploration_dict['states'] = cls._convert_states_vM_dict_to_vM+1_dict(
            exploration_dict['states'])
    exploration_dict['states_schema_version'] = M+1
    return exploration_dict

def _convert_states_vM_dict_to_vM+1_dict(cls, states_dict):
    """Converts from version M to M+1. Version M+1 converts the
    html content from textAngular to CKEditor.

    Args:
        states_dict: dict. A dict where each key-value pair represents,
        respectively, a state name and a dict used to initialize a State
        domain object.

    Returns:
        dict. The converted states_dict.
    """
    for state_dict in states_dict.values():
        html_data = state_dict['content']['html']
        soup = BeautifulSoup(html_data, 'html.parser')

        # Code for only some of the tags in migration details is written
        # here. Other code will be added in the same format.

        for bold in soup.findAll('b'):
            bold.name = 'strong'

        for italic in soup.findAll('i'):
            italic.name = 'em'

        for pre in soup.findAll('pre'):
            pre.name = 'div'
            pre['style'] = 'background:#eeeeee; border:1px solid #cccccc;
padding:5px 10px'

        state_dict['content']['html'] = str(soup)
    return states_dict
```

Current schema version will be set to N so that any exploration which has schema version less than N will be migrated when ExplorationMigrationJobManager is run.

I added the conversion functions and ran the ExplorationMigrationJobManager with the following change to output the yaml content:

```
exploration = exp_services.get_exploration_by_id(old_exploration.id)
yield('Yaml content', exploration.to_yaml())
```

The exploration was migrated as desired. The html tags `<b>` and `<i>` were converted to `<strong>` and `<em>` respectively.



Recent jobs

Note: This table may be stale; refresh to see the latest state.

Job ID	Status	Time started	Time finished	
ExplorationMigrationJobManager-1523397705455-942	completed	April 10 22:01:45	April 10 22:02:15	View Output

Job Output

```
• [u'Yaml content', [u'author_notes: \'\'\nauto_tts_enabled: true\nblurb: \'\'\ncategory:
  Welcome\ncorrectness_feedback_enabled: false\ninit_state_name: Welcome!\nlanguage_code:
  en\nobjective: become familiar with Oppia\'s capabilities\nparam_changes: []\nparam_specs:
  {}\nschema_version: 24\nstates:\n END:\n classifier_model_id: null\n content:\n audio_translations: {}\n
  html: <p><strong><em>Congratulations, you have finished!</em></strong></p>\n interaction:\n
  answer_groups: []\n confirmed_unclassified_answers: []\n customization_args:\n
  recommendedExplorationIds:\n value: []\n default_outcome: null\n hints: []\n id: EndExploration\n solution:
  null\n param_changes: []\n Estimate 100:\n classifier_model_id: null\n content:\n audio_translations: {}\n
  html: <em>What is 10 times 10?</em>\n interaction:\n answer_groups:\n - outcome:\n dest: Numeric
  input\n feedback:\n audio_translations: {}\n html: Yes! So 11 times 11 must be bigger. Let\'s try again.\n
```

Testing

This phase will be time consuming. It will require analysing the data, writing tests for scripts (script validation) and then checking that whether the content has migrated to a form compatible with CKEditor (post validation). Same testing will be used to check whether the content has migrated from jWysiwyg to TextAngular.

Testing will consist of three phases:

Step 1: Pre-Migration Testing

Task 1: In this phase, I will test if the one-off job script produces desired output on dummy explorations.

I will write tests for checking the content migration job which will migrate dummy explorations and compare the output with desired format.

Dummy exploration:

```
states:
  State1:
    classifier_model_id: null
    content:
      audio_translations: {}
      html: <blockquote><p><b><i>Hello, this is
state1</i></b></p></blockquote>
    interaction:
      answer_groups: []
      confirmed_unclassified_answers: []
      customization_args: {}
      default_outcome:
        dest: State1
        feedback:
          audio_translations: {}
          html: ''
        labelled_as_correct: false
        param_changes: []
        refresher_exploration_id: null
      hints: []
      id: null
      solution: null
    param_changes: []
  State2:
    classifier_model_id: null
    content:
      audio_translations: {}
      html: <pre>Hello, this is state2</pre>
    interaction:
      answer_groups: []
      confirmed_unclassified_answers: []
      customization_args: {}
      default_outcome:
        dest: State2
        feedback:
          audio_translations: {}
          html: ''
        labelled_as_correct: false
        param_changes: []
        refresher_exploration_id: null
      hints: []
      id: null
      solution: null
    param_changes: []
```

Desired migrated exploration

```
states:
  State1:
    classifier_model_id: null
    content:
      audio_translations: {}
      html: <blockquote><p><strong><em>Hello, this is
state1</em></strong></p></blockquote>
    interaction:
      answer_groups: []
      confirmed_unclassified_answers: []
      customization_args: {}
      default_outcome:
        dest: State1
        feedback:
          audio_translations: {}
          html: ''
        labelled_as_correct: false
        param_changes: []
        refresher_exploration_id: null
      hints: []
      id: null
      solution: null
    param_changes: []
  State2:
    classifier_model_id: null
    content:
      audio_translations: {}
      html: '<div style="background:#eeeeee; border:1px solid #cccccc;
padding:5px
        10px">Hello, this is state2</div>'
    interaction:
      answer_groups: []
      confirmed_unclassified_answers: []
      customization_args: {}
      default_outcome:
        dest: State2
        feedback:
          audio_translations: {}
          html: ''
        labelled_as_correct: false
        param_changes: []
        refresher_exploration_id: null
      hints: []
      id: null
      solution: null
    param_changes: []
```

Test for Content Migration Job:

This test will be added to tests in **ExplorationMigrationJobTest** class in **exp_jobs_one_off_test.py**.

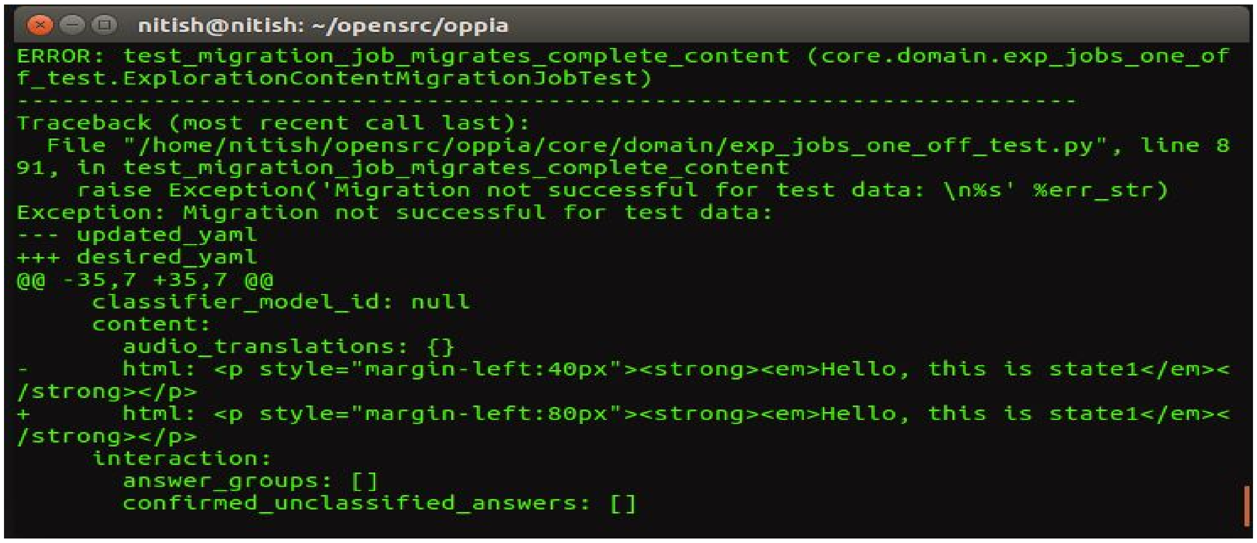
```
def test_migration_job_migrates_complete_content(self):
    """Tests that the exploration migration job migrates the content
    without skipping any tags.
    """
    exploration =
exp_domain.Exploration.create_default_exploration(self.VALID_EXP_ID,
title='title', category='category')
    exploration.add_states(['State1', 'State2'])
    state1 = exploration.states['State1']
    state2 = exploration.states['State2']
    content1_dict = {
        'html': '<blockquote><b><i>Hello, this is
state1</i></b></blockquote>',
        'audio_translations': {}
    }
    content2_dict = {
        'html': '<pre>Hello, this is state2</pre>',
        'audio_translations': {}
    }
    state1.update_content(content1_dict)
    state2.update_content(content2_dict)
    exp_services.save_new_exploration(self.albert_id, exploration)

    # Start migration job on sample exploration.
    job_id = exp_jobs_one_off.ExplorationMigrationJobManager.create_new()
    exp_jobs_one_off.ExplorationMigrationJobManager.enqueue(job_id)
    self.process_and_flush_pending_tasks()
    # Verify that migration produces the desired output.
    updated_exp = exp_services.get_exploration_by_id(self.VALID_EXP_ID)
    updated_yaml = updated_exp.to_yaml()
    desired_yaml = ("""Desired_yaml_content""")
    self.assertEqual(updated_yaml, desired_yaml)
```

I ran the backend tests and the above test produced the desired results.

Task 2: I will also add a feature to output the lines which differ in case the test fails. This will make it easier to debug the script. Here is a mock for doing so:

```
try:
    self.assertEqual(updated_yaml, desired_yaml)
except AssertionError as e:
    err_str = ""
    for line in difflib.unified_diff(updated_yaml.splitlines(),
desired_yaml.splitlines(), fromfile='updated_yaml', tofile='desired_yaml',
lineterm=''):
        err_str += line + "\n"
    raise Exception('Migration not successful for test data: \n%s'
%err_str)
```

Here is one of the false cases which I displayed using the above mock code:



```
nitish@nitish: ~/opensrc/oppia
ERROR: test_migration_job_migrates_complete_content (core.domain.exp_jobs_one_of
f_test.ExplorationContentMigrationJobTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/nitish/opensrc/oppia/core/domain/exp_jobs_one_off_test.py", line 8
91, in test_migration_job_migrates_complete_content
    raise Exception('Migration not successful for test data: \n%s' %err_str)
Exception: Migration not successful for test data:
--- updated_yaml
+++ desired_yaml
@@ -35,7 +35,7 @@
     classifier_model_id: null
     content:
       audio_translations: {}
-      html: <p style="margin-left:40px"><strong><em>Hello, this is state1</em><
/strong></p>
+      html: <p style="margin-left:80px"><strong><em>Hello, this is state1</em><
/strong></p>
     interaction:
       answer_groups: []
       confirmed_unclassified_answers: []
```

Step 2: Migration testing on test server data

I will perform this step on backup copy of the server data.

Task 1: In this phase, I will first perform migration on the test server data and then use the Content Validation job to test if any of the tags used by TextAngular (not supported in CKEditor) are still present in the migrated data.

I have attached the mock code for the Content Validation Job - written above in implementation details for content migration.

This job will validate the html content and provide the list of prohibited tags and invalid html if found after migration.

Task 2: Debug the Content Migration job based on the error cases found in task 1.

After a successful completion of phase 1 and phase 2, I will migrate the actual data and then move to phase 3.

Step 3: Post Migration Testing

This will also firstly occur on a backup copy of production data.

In this phase, I will test whether the actual data has migrated into the required format. I will test whether any tags used in TextAngular (not used by CKEditor) are still present in migrated data. When there are no errors left I will replace the production data with the backup copy on which the migration has been performed after integration of CKEditor.

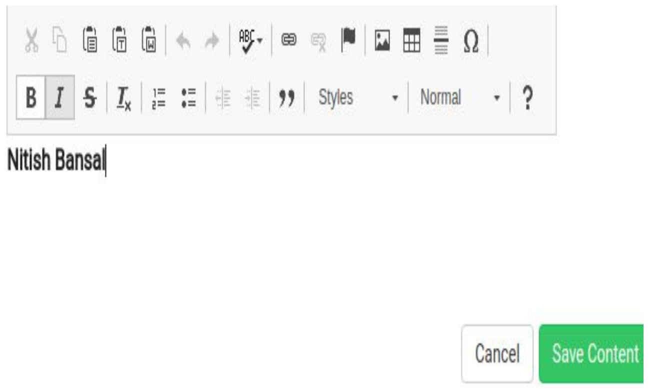
CKEditor Integration

Step 1: Creation of CKEditor RTE Directive

I will begin this milestone by implementing the CKEditorRte directive. I will use the CKEDITOR.inline() function to enable inline editing directly on html elements. I wrote the below code to implement basic CKEditor.

```
oppia.directive('ckEditorRte', [
  '$compile',
  function($compile) {
    return {
      restrict: 'E',
      scope: {
        uiConfig: '&'
      },
      template: '<div id="ckeditor" contenteditable="true"></div>',
      require: '?ngModel',
      link: function(scope, el, attr, ngModel) {
        var ck = CKEDITOR.inline('ckeditor');
 }}}
]);
```

This is the output that I got on using the code above mentioned. There are many additional options in the toolbar. I will configure the toolbar to use only basic styling options and the rich text components.



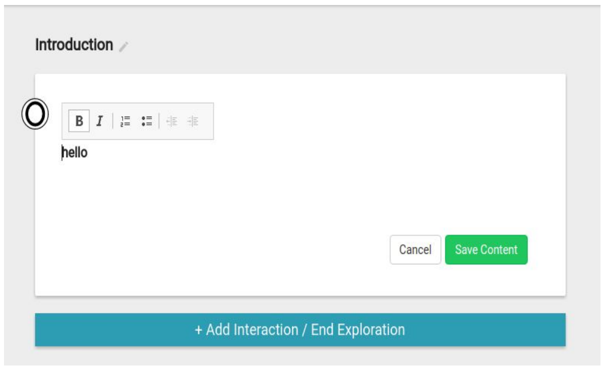
Step 2: Configure toolbar for CKEditor

CKEditor standard provides a variety of plugins available in the toolbar. Since we need basic styling only, I will configure the toolbar to use the groups for basic styling. This is what I tried to configure the toolbar.

```
toolbar: [
  {name: "basicstyles", items: ['Bold', 'Italic', 'Underline']},
  {name: "paragraph", groups: ['list', 'indent'],
     items: [ 'NumberedList', 'BulletedList', '-','Outdent', 'Indent']},
]
```

This is the configured toolbar which I got using the above code. It provides the options for basic styling:

- Bold
- Italics
- Ordered list
- Unordered list
- Increase indent
- Decrease indent



Step 3: Creating plugins for Rich Text Components in CKEditor

In this step I will add the rich text components. The code for the components will remain the same i.e. I will be using the code defined in the folder extensions/rich_text_components as it is. I will add a custom plugin based on the following mock:

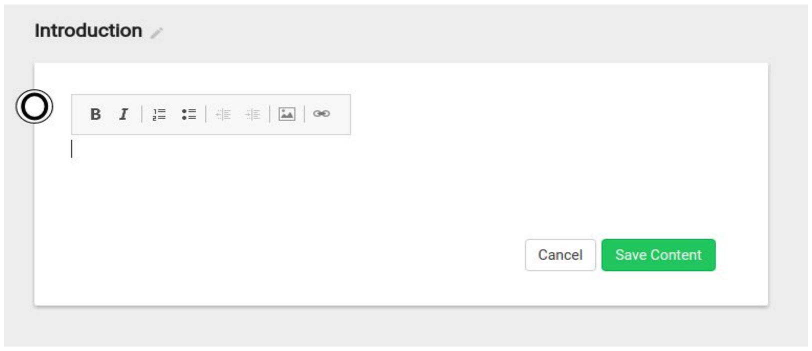
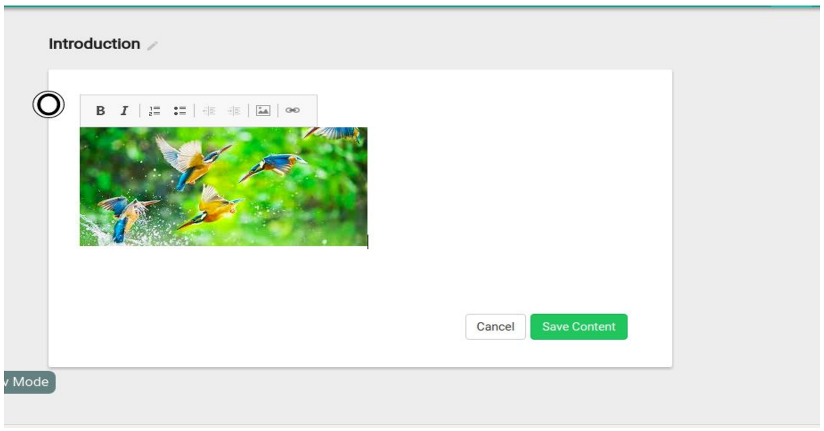
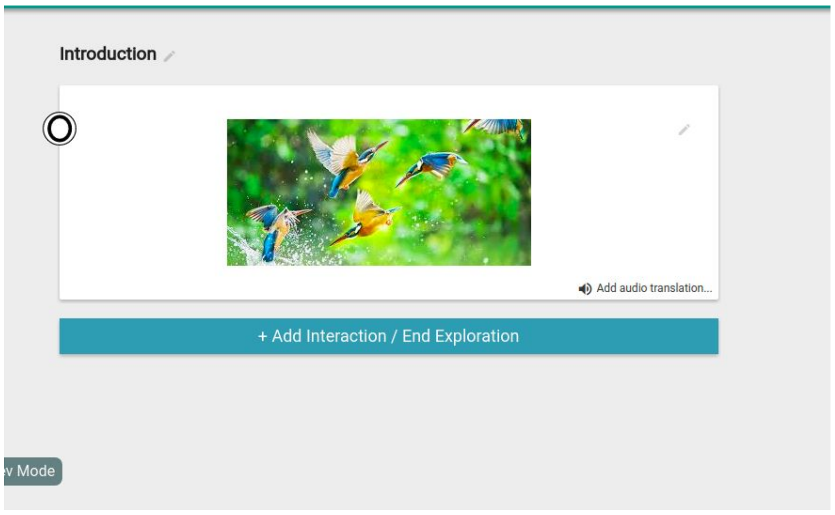
```
CKEDITOR.plugins.add('image', {
    icons: 'image',
    init: function( editor ) {
        // Logic for image plugin
    }
});
```

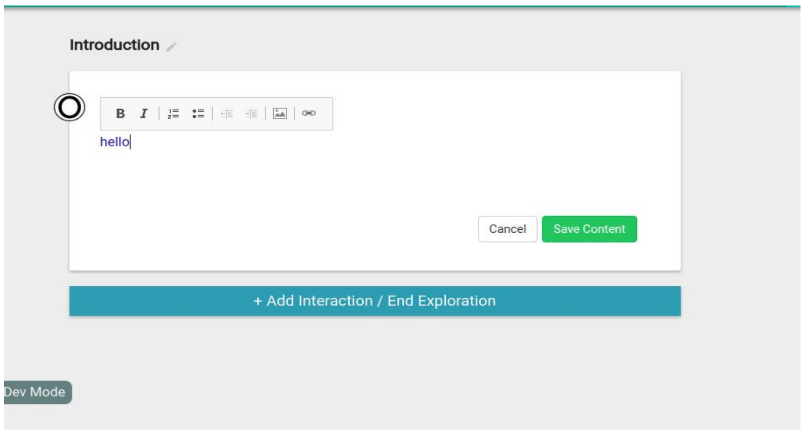
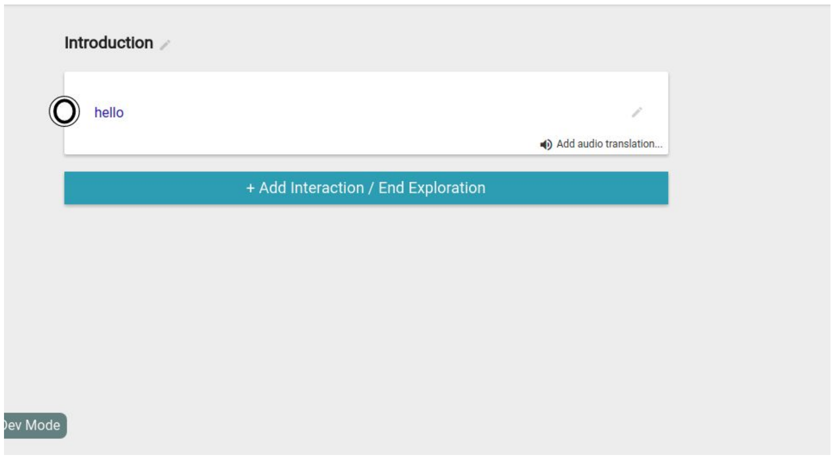
A custom widget will be created for the plugin to open a customization window when the plugin icon is clicked.

I will add the custom widgets based on the following mock:

```
editor.widgets.add('image', {
    button: 'Insert Image',
    inline: true,
    // Logic for image widget
})
```

The created plugin will be added as an extra plugin and integrated in the toolbar. I tried adding plugins for image and link and here is a demonstration of that:

Toolbar after adding custom plugins image and link.	
Image in preview mode i.e without clicking the save button.	
Image after clicking the save button.	 Introduction Add audio translation... + Add Interaction / End Exploration Mode

Link in preview mode. It is now fixed and shows the actual link and not a link marker as shown previously in textAngular.	 Introduction B I hello Cancel Save Content + Add Interaction / End Exploration Dev Mode
Link after clicking the save button.	 Introduction hello Add audio translation... + Add Interaction / End Exploration Dev Mode

I will add all the rich text components namely Image, Link, Video, Collapsible, Math and Tabs. The UX for any component will use the customization modal as was the case with TextAngular RTE. Here is the mock for the same:

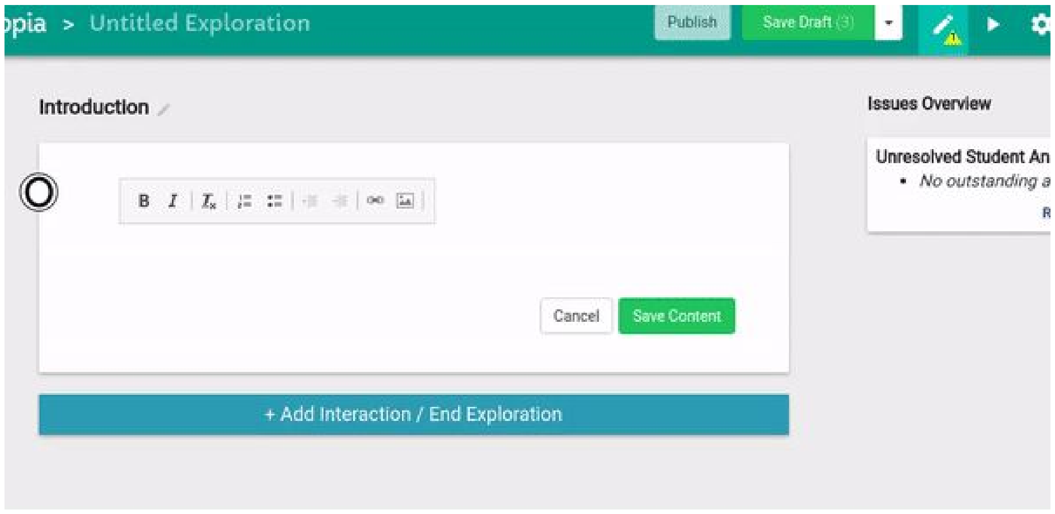
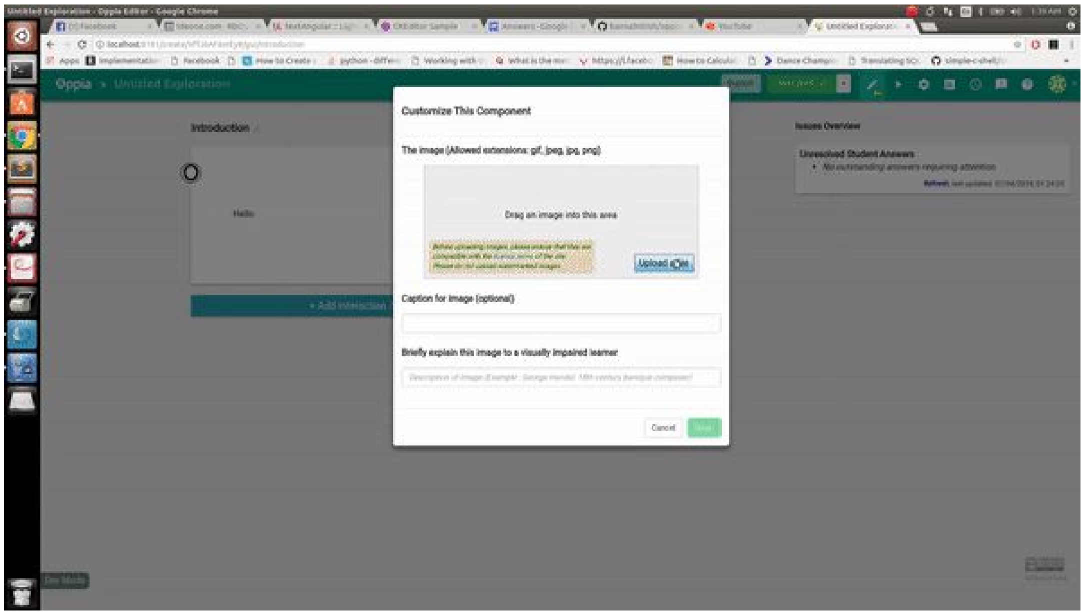
Link	 ppia > Untitled Exploration Publish Save Draft (3) Introduction Issues Overview Unresolved Student An No outstanding a B I Cancel Save Content + Add Interaction / End Exploration

Image	 Oppia Untitled Exploration Introduction Hello + Add interaction Customize This Component The image (Allowed extensions: gif, jpeg, jpg, png) Drag an image into this area Upload a file Caption for image (optional) Briefly explain this image to a visually impaired learner Cancel Issues Overview Unresolved Student Answers

After the integration of CKEditor, I will migrate the content to a form compatible with CKEditor.

Testing the CKEditor and resolving bugs

In this phase, I will test the newly added feature manually as well as take the inputs from the mentors regarding any improvements to be made.

Bugs will be resolved accordingly to produce a perfect Rich text editor. This phase would involve more manual work to check various plugins and remove the bugs accordingly.

Documentation

Documentation will involve the addition of following information:

1. Introduction: This section will contain a small introduction about CKEditor. It will explain the basic features of CKEditor.
2. Reasons to Migrate from textAngular Editor to CKEditor: Advantages of CKEditor over TextAngular. I will describe all the issues that were present in TextAngular editor in brief.
3. Usage of CKEditor: Here I will mention how to use the CKEditor and will define each of the attribute required to use this editor.
4. Third Party: This section will contain all the third party libraries that our editor will be using.

5. Code: I will explain how the code for the editor works, where the code is present and how it is linked up in the files. I will also explain how to configure the editor like changing the theme of the editor, customising the toolbar and adding a plugin to the editor.
6. Making a Custom Plugin for CKEditor: This will contain a detailed description of how to create and add a custom plugin to CKEditor.
7. Upgrading the editor: This final section will contain the information about which all tests are to be performed after upgrading the editor to make sure that our editor has upgraded correctly.

Timeline

Task	Estimated Time	Estimated Period
I will explore the code and get more familiar with the code structure of Oppia, I will do so by • Solving issues • Communicating with the mentor • Reviewing PR's • Organising Creator Experience Page	7 Days	May 7 - 13
I will start the content migration task and it will consist of following: • Finding more differences (if left) between HTML output of jWysiwyg, TextAngular editor and CKEditor. • Update the one-off job script according to the new findings and cases.	10 Days	May 14 - 23
I will write tests for the content migration from jWysiwyg to TextAngular and also for TextAngular to CKEditor during this period. It's work consists of: • Test the one off script on dummy data	18 Days	May 24 - June 10

• Test the migration on test server backup data After the tests are successful, test server data will be migrated to TextAngular format. Then I will test the migration on backup of production data and then migrate the actual production data. Till this time we will have all the content in TextAngular format along with a ready one off job script to perform migration to CKEditor. (June Release)		
I will start with CKEditor integration. • I will integrate CKEditor in this phase. • I will also document this integration.	15 Days	June 11 - 25
I will perform content migration from TextAngular to CKEditor in this period. Firstly migration will occur on test server data and then on production data. (July Release)	15 Days	June 26 - July 10
I will remove all issues that come in way after CKEditor integration and content migration.	21 Days	July 11 - 31
Submission of final work for evaluation and take final feedback from mentors before submission. I will make fixes based on feedback and ensure that everything works as expected.	14 Days	August 1 - 14

Releases

June Release: MIgration from JWysiwyg to TextAngular. Also till this time I will be having both the one-job off scripts (JWysiwyg to TextAngular, TexAngular to CKEditor) ready.

July Release: Integration of CKEditor along with content migration to CKEditor. In this release we will have CKEditor integrated in Oppia.

Summer Plan

Timezone - IST (Indian Standard Time) (UTC +05:30)

Time I can commit to the Project

Period	Hours per week
May 7 - 13	21 - 28
May 14 - July 31	35 - 42
August 1 - August 13	30 - 35

What jobs, summer classes, and other obligations might you need to work around?

I have no commitments in the summer. I'll be staying back home for the most part of it from May 10 - July 20. From 21st July I'll be returning back to my college so the number of working hours may be reduced by 1-2 hours or so as I would be engaged in my classes. However I would try to cover this up by working more on weekends. I have mentioned my typical working hours above and on an average I will be able to spend 40 hours per week on the project.

Communication

Name : Nitish Bansal

Phone : +91 8360172271

EMail : nitishbansal2297@gmail.com

I would like to have meetings on google hangouts. We can fix a schedule and will be notified every week whenever we have meetings. Since Oppia community is very active on gitter that too can be used to have meetings. This is not something that is fixed, I am open to any platform that mentors would like to have a discussion on. I will also maintain daily devlogs to document my daily work.

It would be good to have meetings twice a week with the mentor. This can also be changed by having a chat with the mentor.