

Google Summer of Code 2020

Generate images for LaTeX expressions

Abhith Krishna

About You

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

Oppia's mission is to "provide high-quality education to those who lack access to it." I wish to contribute to this vision with all my efforts so that students all across the world have access to high-quality scalable online lessons.

I have seen and used many online learning platforms but found Oppia to be the most unique. I have always wanted to work on projects which involve creating an end-to-end user flow and creating an impact on the users of the product. I believe, after the completion of this project the experience of the users will improve significantly during lessons.

Prior experience

I have been contributing to oppia for the last 4-5 months and I am familiar with most parts of the codebase. I have been working with Python for the last two years. I am also involved as a backend developer in my university Fests and other events. I have been involved in web development since the past year.

Links to previous PR's

1. Update answer group rules when choices in multiple-choice input interaction are deleted #8682 ([link](#))
2. Added a custom lint check to ensure that there is an empty line between imports and fileoverview. #8409 ([link](#))
3. Added modal to allow user to add commit messages when saving an edit to a question. #8538 ([link](#))
4. modified checks for enabling and disabling the submit button in the suggestion modal #8251 ([link](#))

5. Added validation for numeric-input rule type `IsInclusivelyBetween`. #8742 ([link](#))

- **The link to all my contributions to oppia is here ([link](#))**

Contact info and timezone(s)

Email: abyk476@gmail.com

f20170941@pilani.bits-pilani.ac.in

Github: [donosco98](#)

I will stay in India throughout the summer. The time zone will be Indian Standard Time (GMT+5:30)

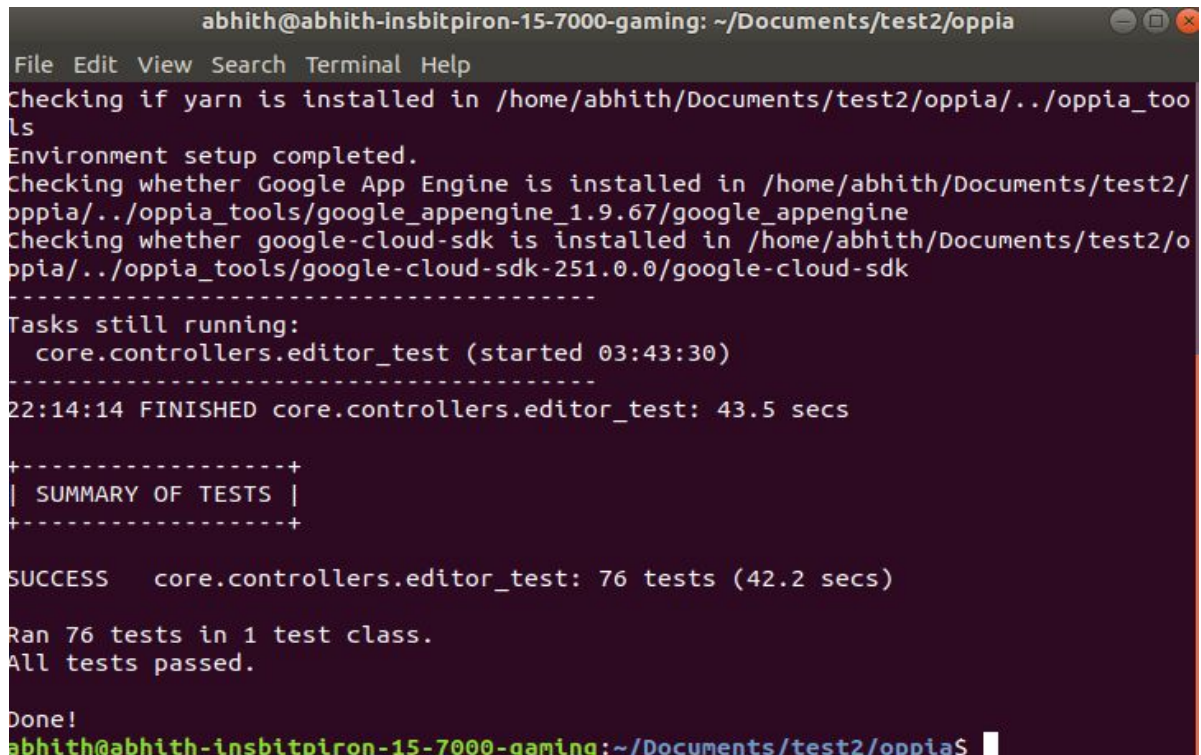
Time commitment

I will dedicate 7-8 hours on a daily basis and will work for more time if any unexpected delays come.

Essential Prerequisites

Answer the following questions:

- I am able to run a single backend test target on my machine. (Show a screenshot of a successful test.)



```
abhith@abhith-insbitpiron-15-7000-gaming: ~/Documents/test2/oppia
File Edit View Search Terminal Help
Checking if yarn is installed in /home/abhith/Documents/test2/oppia/./oppia_tools
Environment setup completed.
Checking whether Google App Engine is installed in /home/abhith/Documents/test2/oppia/./oppia_tools/google_appengine_1.9.67/google_appengine
Checking whether google-cloud-sdk is installed in /home/abhith/Documents/test2/oppia/./oppia_tools/google-cloud-sdk-251.0.0/google-cloud-sdk
-----
Tasks still running:
  core.controllers.editor_test (started 03:43:30)
-----
22:14:14 FINISHED core.controllers.editor_test: 43.5 secs
+-----+
| SUMMARY OF TESTS |
+-----+

SUCCESS core.controllers.editor_test: 76 tests (42.2 secs)

Ran 76 tests in 1 test class.
All tests passed.

Done!
abhith@abhith-insbitpiron-15-7000-gaming:~/Documents/test2/oppia$
```

- I am able to run all the frontend tests at once on my machine. (Show a screenshot of a successful test.)



```
abhith@abhith-insbitpiron-15-7000-gaming: ~/Documents/test2/oppia
File Edit View Search Terminal Help
HeadlessChrome 76.0.3809 (Linux 0.0.0): Executed 1955 of 1970 SUCCESS (0 secs /
HeadlessChrome 76.0.3809 (Linux 0.0.0): Executed 1956 of 1970 SUCCESS (0 secs /
HeadlessChrome 76.0.3809 (Linux 0.0.0): Executed 1957 of 1970 SUCCESS (0 secs /
HeadlessChrome 76.0.3809 (Linux 0.0.0): Executed 1958 of 1970 SUCCESS (0 secs /
HeadlessChrome 76.0.3809 (Linux 0.0.0): Executed 1959 of 1970 SUCCESS (0 secs /
HeadlessChrome 76.0.3809 (Linux 0.0.0): Executed 1960 of 1970 SUCCESS (0 secs /
HeadlessChrome 76.0.3809 (Linux 0.0.0): Executed 1961 of 1970 SUCCESS (0 secs /
HeadlessChrome 76.0.3809 (Linux 0.0.0): Executed 1962 of 1970 SUCCESS (0 secs /
HeadlessChrome 76.0.3809 (Linux 0.0.0): Executed 1963 of 1970 SUCCESS (0 secs /
HeadlessChrome 76.0.3809 (Linux 0.0.0): Executed 1964 of 1970 SUCCESS (0 secs /
HeadlessChrome 76.0.3809 (Linux 0.0.0): Executed 1965 of 1970 SUCCESS (0 secs /
HeadlessChrome 76.0.3809 (Linux 0.0.0): Executed 1966 of 1970 SUCCESS (0 secs /
HeadlessChrome 76.0.3809 (Linux 0.0.0): Executed 1967 of 1970 SUCCESS (0 secs /
HeadlessChrome 76.0.3809 (Linux 0.0.0): Executed 1968 of 1970 SUCCESS (0 secs /
HeadlessChrome 76.0.3809 (Linux 0.0.0): Executed 1969 of 1970 SUCCESS (0 secs /
HeadlessChrome 76.0.3809 (Linux 0.0.0): Executed 1970 of 1970 SUCCESS (0 secs /
cs / 15.579 secs)
TOTAL: 1970 SUCCESS
TOTAL: 1970 SUCCESS
30 03 2020 03:51:50.845:WARN [launcher]: ChromeHeadless was not killed in 2000 m
s, sending SIGKILL.
Done!
```

- I am able to run one suite of e2e tests on my machine. (Show a screenshot of a successful test.)

```

abhith@abhith-insbitpiron-15-7000-gaming: ~/Documents/test2/oppia
File Edit View Search Terminal Help

Classroom page functionality
  ✓ should search for explorations from classroom page

1 spec, 0 failures
Finished in 52.586 seconds

Executed 1 of 1 spec SUCCESS in 53 secs.
[03:49:32] I/launcher - 0 instance(s) of WebDriver still running
[03:49:32] I/launcher - chrome #01 passed
Killing /usr/bin/python /home/abhith/Documents/test2/oppia/../../oppia_tools/google_appengine_1.9.67/google_appengine/dev_appserver.py --host 0.0.0.0 --port 9001 --clear_datastore=yes --dev_appserver_log_level=critical --log_level=critical --skip_sdk_update_check=true app_dev.yaml ...
Killing java -Dwebdriver.chrome.driver=/home/abhith/Documents/test2/oppia/node_modules/webdriver-manager/downloads/chromedriver_2.41 -Dwebdriver.gecko.driver=/home/abhith/Documents/test2/oppia/node_modules/webdriver-manager/downloads/geckodriver_0.26.0 -jar /home/abhith/Documents/test2/oppia/node_modules/webdriver-manager/downloads/selenium-server-standalone-4.0.0-alpha-1.jar -role node -servlet org.openqa.grid.web.servlet.LifecycleServlet -registerCycle 0 -port 4444 ...
abhith@abhith-insbitpiron-15-7000-gaming:~/Documents/test2/oppia$

```

Other summer obligations

I have no other commitments during summer. In the second week of August, my university will reopen So I would spend 4-5 hours on a daily basis during this phase, but I will cover for this at the weekend by working extra hours.

Communication channels

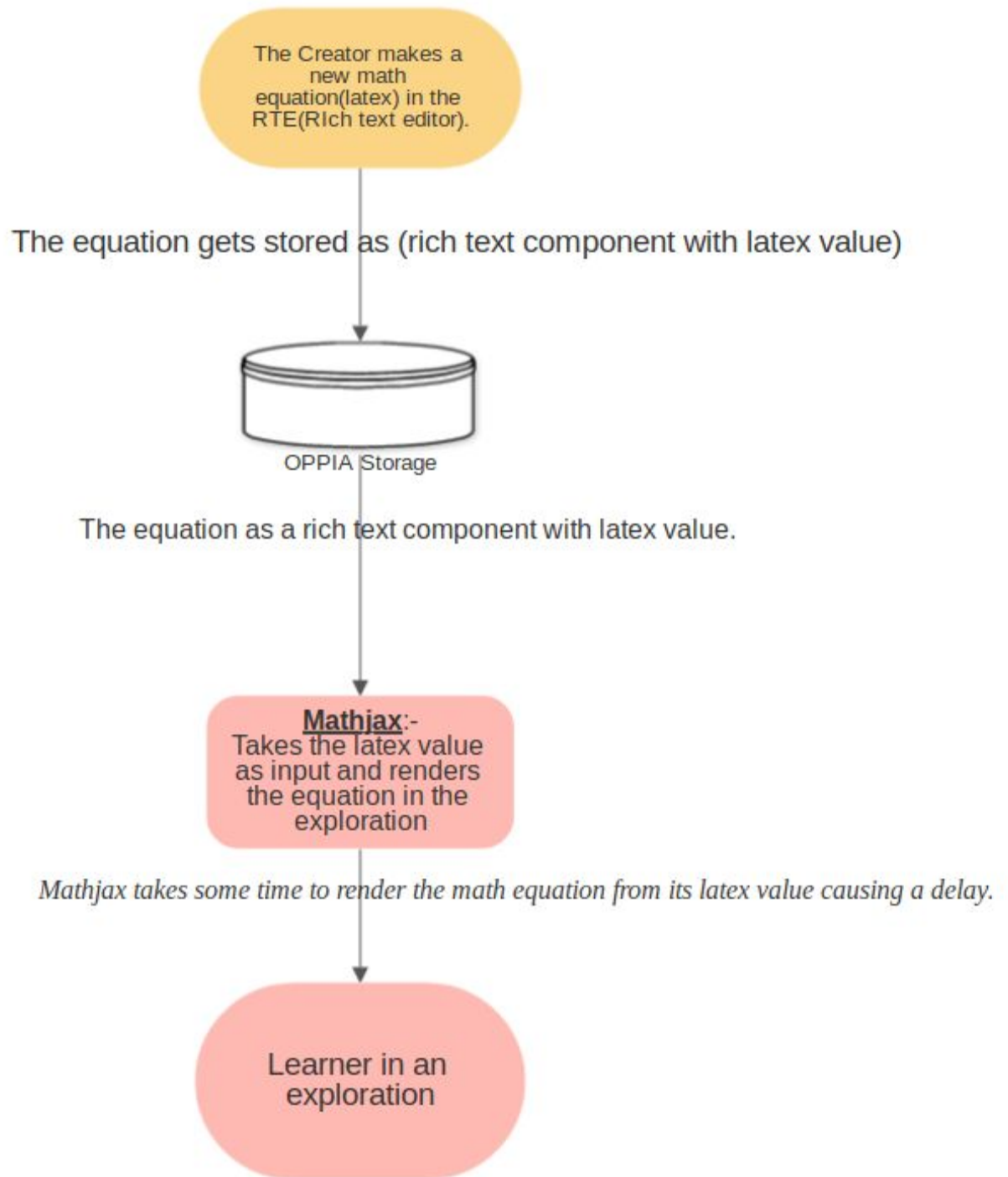
Oppia members mainly use google-hangouts to discuss ideas and meet. I too favor communication through google-hangouts. I am also comfortable with e-mails.

Project Details

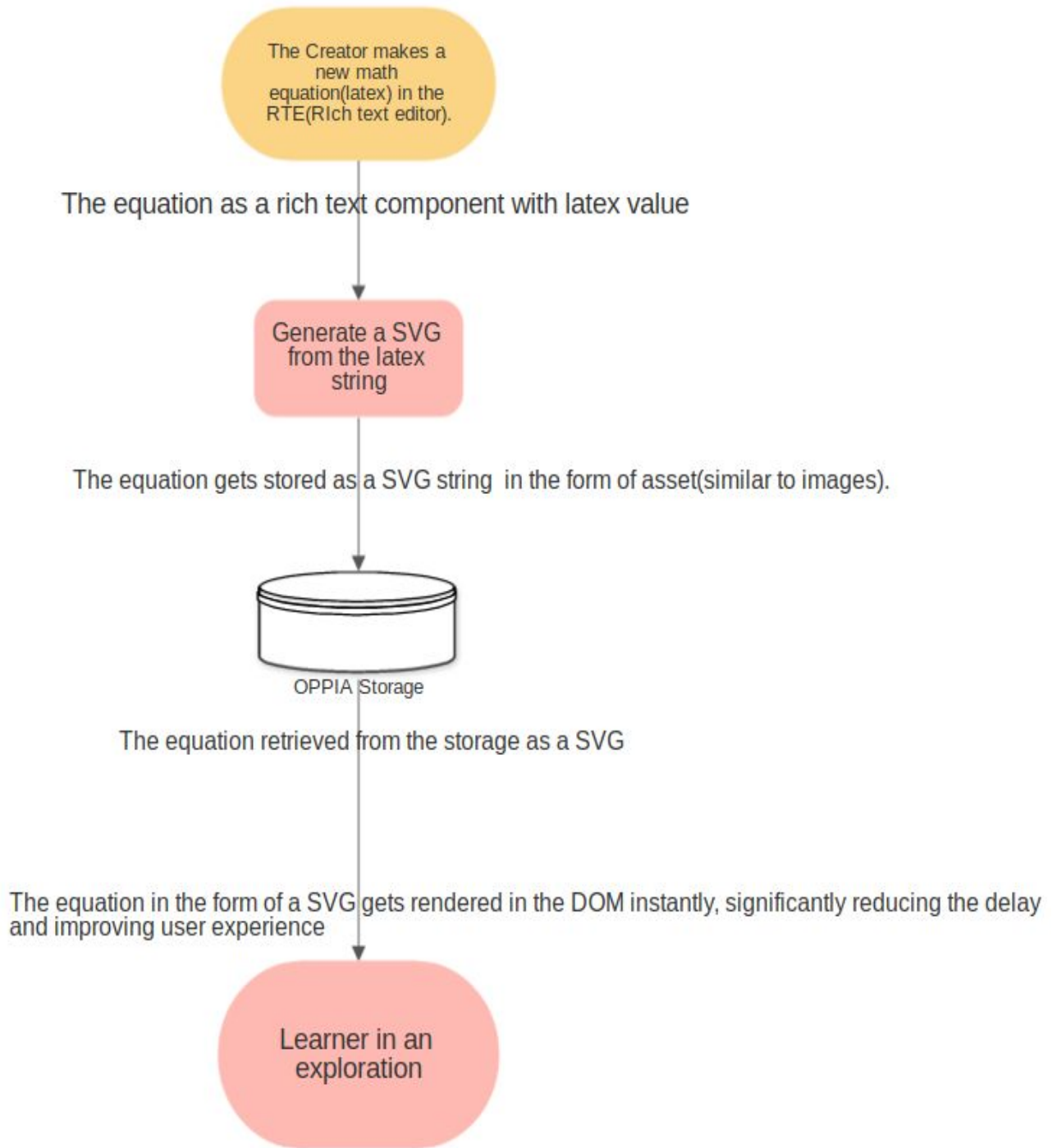
Product Design

- This project aims to introduce SVG rendering for all LaTeX expressions in explorations.
- Currently, we use mathjax to render latex equations in the frontend pages.
- But using Math-Jax to render the latex equations in the frontend causes the pages to take more time to load. This can sometimes compromise the user-experience.
- SVGs will be generated by third party libraries from the latex equations when a new math expression is created, these SVGs will then be stored in the backend
- The SVGs generated and stored will be used for displaying math equations on the learner pages.
- The stored SVGs will be rendered directly in the DOM.
- By doing this, we can remove mathjax as a dependency for rendering latex equations in lessons and explorations from learner pages.
- This can help in explorations to load faster and thereby improving the experience of the learners.

The current implementation for rendering math equations(using mathjax)



The proposed implementation for rendering math equation(using SVGs)



Technical Design

Extend the current "math expression" rich-text editor component to include an SVG field

The `raw_latex` field in `customization_arg_specs` of math Rich Text will be replaced by `math_expression_contents`, the `math_expression_contents` will have a value in the form of an object having key-value pairs of `raw_latex` and `svg_filename`.

```
{  
  "raw_latex": " " ,  
  "svg_filename": " "  
}
```

- The Rich text definition of the Math component will look like the following [rich_text_components_definitions.ts](#)

```
"Math": {  
  "backend_id": "Math",  
  "category": "Basic Input",  
  "description": "A math formula.",  
  "frontend_id": "math",  
  "tooltip": "Insert mathematical formula",  
  "icon_data_url": "/rich_text_components/Math/Math.png",  
  "is_complex": false,  
  "requires_fs": true,  
  "is_lesson_related": false,  
  "is_block_element": false,  
}
```



```

"customization_arg_specs": [{
  "name": "math_expression_contents",
  "description": "Data required to render the expressions."
  "schema": {
    "type": "custom",
    "obj_type": "MathExpression"
  },
  "default_value": [{
    "raw_latex": "",
    "svg_filename": ""
  }]
}]

```

- For the frontend, Mathjax is the best tool for converting Latex to SVG because:
 - Most of the online tools used for converting latex equations to SVG are based on Mathjax.
 - MathJax supports a wide array of latex equations.
 - MathJax is well-maintained and updated by developers. So less, likely to run into bugs.
 - Also, MathJax is currently a dependency on oppia.
 - Output generated by mathjax works in all browsers including Internet Explorer 6
 - MathJax is supplemented with extensive documentation which can be helpful for future developers
- In Python Matplotlib is the best option to convert Latex to SVG:
 - Matplotlib has a powerful module called " Mathtext " for parsing and rendering Latex math equations in different formats (Eg SVGs)
 - Matplotlib can convert Latex into SVG in very few steps.
 - Matplotlib is well maintained and supported by developers
 - Matplotlib is well-documented.
 - Matplotlib is purely written in Python.
 - There are very few other reliable alternatives to convert Latex to SVG files. Most of these alternatives are either written in other languages or are deprecated.

- In the coming section, we will compare the SVGs generated by both Mathjax and Matplotlib to see which yields the best result.
- Based on the Comparison we will decide which of the following 3 approaches to adopt.
 - a. SVG generation for new expressions and Migration for existing expressions both on the client-side using Mathjax.
 - b. SVG generation for new expressions on the client-side using Mathjax, But migration for existing expressions in the backend using Matplotlib.
 - c. SVG generation for new expressions and Migration for existing expressions both in the backend using Matplotlib.

COMPARISON OF SVG GENERATION BY MATHJAX AND MATPLOTLIB

1.USER EXPERIENCE (CREATORS)

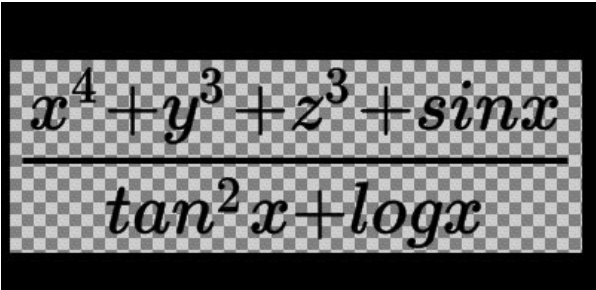
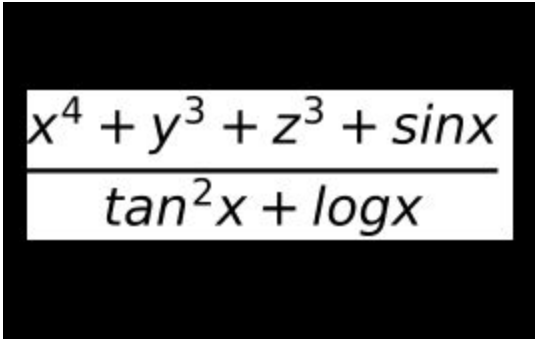
<u>MATHJAX</u>	<u>MATPLOTLIB</u>
<ul style="list-style-type: none">• The generation of SVGs happens dynamically in the client-side• The SVGs can get generated as soon as the user changes or enters the value in the math-expression editor.• This dynamic generation has the following benefits:• We don't need to change the current flow of the math-latex-equation editor.• The SVG generation happens automatically in the background as the user changes the value of Latex string• As there is no user intervention in SVG generation, the user experience is improved.• Another benefit of using Mathjax is that the creator can Preview the generated SVG before saving it.• The preview gets displayed as soon as the user changes the LAtex value.	<ul style="list-style-type: none">• The generation of SVGs happens in the backend.• For SVGs to be generated a request to the server has to be made. Due to this reason, the generation of SVGs cannot be dynamic and immediate.• We would need to change how the math-latex-equation editor works to account for the fact that matplotlib cannot generate SVGs dynamically as the user changes the value of the Latex string.• A major Drawback for Matplotlib is that there is no easy way to preview the SVGs generated.• All of these can lead to poorer user-experience.

2.EFFICIENCY AND PERFORMANCE

<u>MATHJAX</u>	<u>MATPLOTLIB</u>
<ul style="list-style-type: none">• The Time taken to generate an SVG string given a Latex String is almost the same as that of Matplotlib.(Time taken by Mathjax might slightly vary with the resources available at the client-side)• The conversion of Latex string to SVGs is more indirect and harder as compared to Matplotlib.	<ul style="list-style-type: none">• The Time taken to generate an SVG string given a Latex String is almost the same as that of Mathjax• The conversion of Latex string to SVGs is easy and direct in Matplotlib as compared to Mathjax.• Backend calls are required for Generating SVGs.• Harder to handle errors that can occur during the SVG generation

<ul style="list-style-type: none"> As the generation of SVGs happen in client-side no requests to the server is required(except for saving the generated SVGs) Easier to handle errors and warnings that may occur during the SVG generation. For example, If a particular Latex string entered by the user is not supported or is invalid, we can immediately show this warning to the user. 	<ul style="list-style-type: none"> Here It's harder to warn the user if a particular Latex string is invalid or not supported.
--	---

3.FEATURES OF THE SVGs GENERATED

MATHJAX	MATPLOTLIB
<div>  </div> <ul style="list-style-type: none"> The above is an SVG generated by Mathjax The Size of this file generated By mathjax is 10.5kb The File generated by MathJax is 15% smaller as compared to the file generated by Matplotlib. Mathjax is highly configurable through the config file. <div> <pre> <svg xmlns:xlink="http://www.w3.org/1999/xl: width="12.795ex" height="4.388ex"~ viewBox="0 -1224.9 5508.7 1889.3"~ role="img" focusable="false"~ style="vertical-align: -1.543ex;">~ <defs>~ <!-- Paths and defs to shape the SVG -->~</svg>~ </pre> </div> <ul style="list-style-type: none"> The above is the SVG string generated by Mathjax. 	<div>  </div> <ul style="list-style-type: none"> The above is an SVG generated by Matplotlib. The Size of this file generated by Matplotlib is 12.4 kb. We can see that the file generated by Matplotlib is roughly 20 % larger than the file generated by MathJax <div> <pre> <svg height="20pt" version="1.1"~ viewBox="0 0 65 20" width="65pt"~ xmlns:xlink="http://www.w3.org/1999/xlink">~ <defs>~ <!-- Paths and defs to shape the SVG -->~ </svg>~ </pre> </div> <ul style="list-style-type: none"> The above is the SVG string generated by Matplotlib

<ul style="list-style-type: none"> Mathjax SVGs are custom made to be displayed inline with other Surrounding Texts. The attributes width, height, viewBox, and Style are generated in terms of ex values(1 ex is the height of a lowercase font) rather than absolute values. The height and width are automatically generated by mathjax based on other Fonts existing on the web page. Mathjax automatically styles and positions the SVGs in order for it to be perfectly aligned inline with the surrounding text. (Styles like “vertical-align” make this possible) Mathjax SVGs when used in line with surrounding text <p>This is the Equation generated by Mathjax : $\frac{x^4+y^3+z^3+\sin x}{\tan^2 x+\log x}$</p> <ul style="list-style-type: none"> As we can see Mathjax SVGs are aligned properly with surrounding text 	<ul style="list-style-type: none"> Matplotlib SVGs are meant to be used to display plots rather than displaying equations inline in HTML pages. The attributes width, height, viewBox are harder to configure to match the desired values. The attributes are by default generated in absolute units. Matplotlib SVGs are not styled to display equations inline. The SVGs generated don't have any styles to position the SVG inline. Matplotlib SVGs when used in line with the surrounding text. <p>THIS is the equation displayed by Matplotlib: $\frac{x^4+y^3+z^3+\sin x}{\tan^2 x+\log x}$</p> <ul style="list-style-type: none"> Matplotlib SVG is not perfectly aligned with the surrounding text.

- Given below is the code segment from the codebase of mathjax which is used to position and style the SVGs.(Just a piece of additional information).

(line 2219
third_party/static/MathJax-2.7.5/unpacked/jax/output/SVG/jax.js)

```
//
//  Style the <svg> to get the right size and placement

var l = Math.max(-svg.l,0), r = Math.max(svg.r-svg.w,0);
var style = svg.element.style, px = SVG.TeX.x_height/SVG.ex;
var H = (Math.ceil(svg.H/px)+1)*px+SVG.HFUZZ, // round to pixels and add padding
D = (Math.ceil(svg.D/px)+1)*px+SVG.DFUZZ;
var w = l + svg.w + r;
svg.element.setAttribute("width",SVG.Ex(w));
svg.element.setAttribute("height",SVG.Ex(H+D));
```

```

style.verticalAlign = SVG.Ex(-D);
if (l) style.marginLeft = SVG.Ex(-l);
if (r) style.marginRight = SVG.Ex(-r);
svg.element.setAttribute("viewBox",SVG.Fixed(-l,1)+" "+SVG.Fixed(-H,1)+" "+
                        SVG.Fixed(w,1)+" "+SVG.Fixed(H+D,1));

```

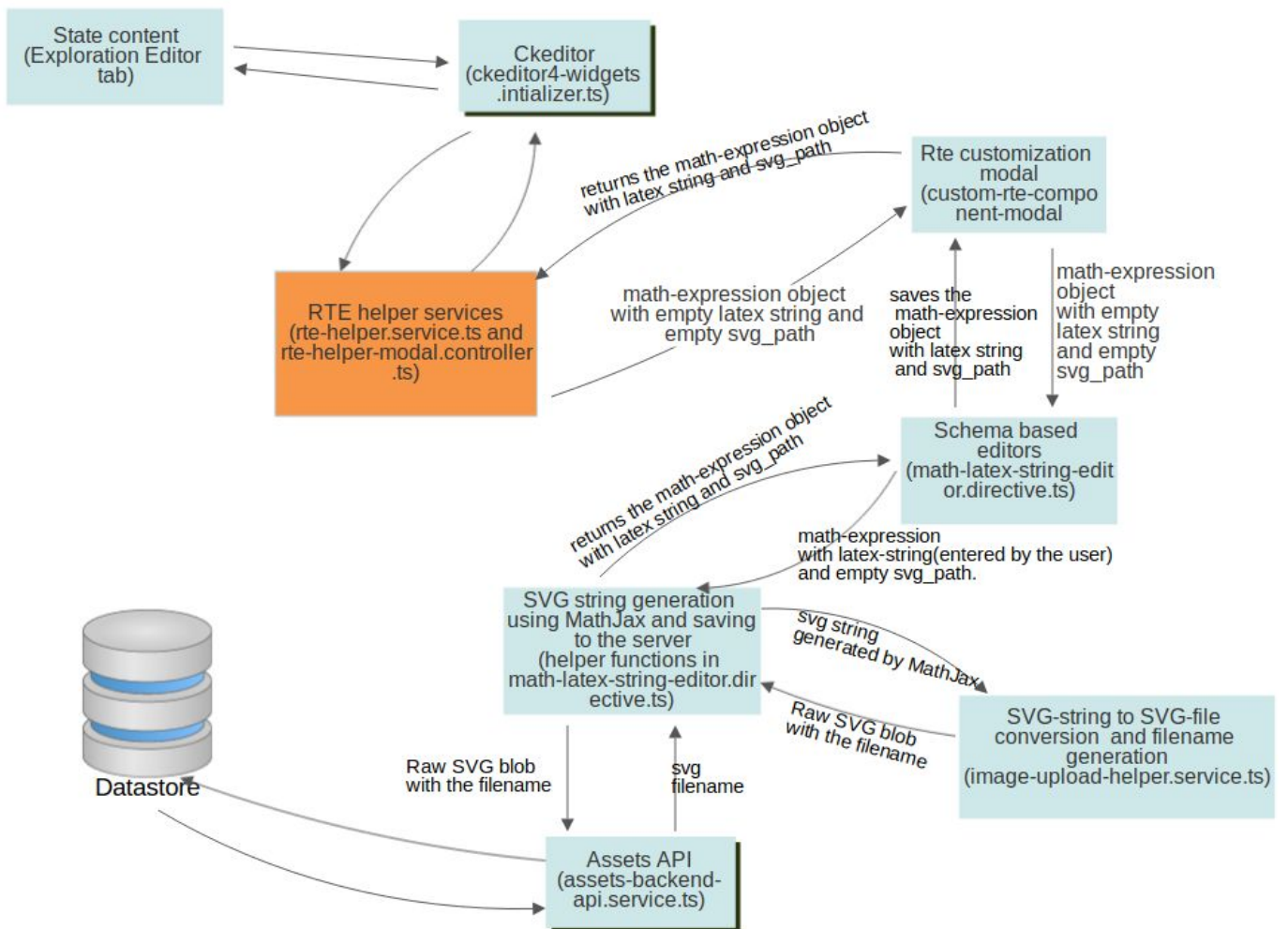
4.MIGRATION AND SUPPORT FOR EXISTING MATH EQUATIONS.

<u>MATHJAX</u>	<u>MATPLOTLIB</u>
<ul style="list-style-type: none"> • The process of generating and saving SVGs for existing Math-expressions is trickier because we have to migrate the explorations from the client-side • This is one of the drawbacks of using only Mathjax. • But one major reason to use Mathjax is that all the existing expressions are supported by Mathjax (Because Oppia codebase currently uses Mathjax to typeset equations) • So there won't arise a case where a particular expression's SVG cannot be generated by Mathjax. • This ensures that no expression in the server is left without an SVG image. 	<ul style="list-style-type: none"> • The process of generating and saving SVGs for existing Math-expressions is easier to code. • Since Matplotlib is a python library it can be directly used for migrations in the backend. • One major problem with using Matplotlib is that it doesn't support all the latex equations supported by Mathjax. • Matplotlib cannot generate SVGs for all the existing expressions on the server. • So this can lead to a case where some math-expressions already in the server will not have an SVG(Because Matplotlib may not support these Latex equations).

- From the above comparison, we can infer that it is better to do the SVG generation using Mathjax and Migrations also using Mathjax
- Some compelling reasons to choose Mathjax over Matplotlib.
 - The SVGs can be automatically and dynamically generated as the user enters the math-expressions Latex value.
 - The generated SVG can be previewed before saving
 - Easier to track and handle errors.
 - SVGs generated by Mathjax are better than Matplotlib for our purpose(in terms of file size, styling, etc.).
 - All the existing Expressions in the server will be supported by Mathjax, leading to a smooth and complete migration.
 - Matplotlib is a heavy library but If we use Mathjax, no need to add any new libraries.

- We see that SVGs generated by MathJax are more suited to our purpose than SVGs generated by matplotlib.

Architectural Overview



The architectural overview of the SVG generation and storage.

BRIEF EXPLANATION OF THE ABOVE FLOW.

- When the creator clicks on the math icon of Ckeditor, a customization modal for RTE components opens. [`rte-helper.service.ts`]
- The `rte-helper.service.ts` passes the `customizationArgs` to the customization modal.
- At this stage the customization arg for the math-expression, both `raw_latex` and `svg_filename` fields are empty.
- The creator enters a math equation as a latex string in the RTE for math equations. [`math-latex-string-editor.directive.ts`]
- As the value of the latex string is entered or changed by the user an SVG string corresponding to the Latex string is generated by Mathjax.
- We do this by placing the Mathjax SVG generation function inside a `$scope.$watch` and generate a new SVG each time the creator enters/changes the Latex String in the editor.
- We assign the SVG string generated corresponding to the latest Latex string entered by the user in a variable.
- When the user clicks on the Done button the SVG string stored in the variable is processed and saved to the backend
- We convert the SVG string to a raw Blob and generate a filename with the help of helper functions in `image-upload-helper.service.ts`
- Also for saving the SVG to the backend, we need to use `async` and `await` for the function which saves the SVG to the backend.
- In this way, we can ensure that the SVGs are saved before we close the RTE customization modal.
- After the SVG is saved to the backend we assign the filename(which is received as a response from the backend.) to the customization arg `svg_filename`.
- Hence the newly created Math-expression has a corresponding SVG file saved in the backend.

Implementation Approach and Details

Generating SVG string from the latex string.

- We will be using MathJax 2.7.5 for converting the latex string into an SVG string(Mathjax 2.7.5 is already being used in oppia).
- First, we have to configure MathJax to give SVG output.
- This can be done by changing the configuration file (mathjaxConfig.ts).

```
window.MathJax = {
  skipStartupTypeset: true,
  jax: ["input/TeX", "output/SVG"],
  extensions: ["tex2jax.js", "MathMenu.js", "MathZoom.js"],
  showMathMenu: false,
  showProcessingMessages: false,
  messageStyle: "none",
  SVG: {
    useGlobalCache: false
  },
  TeX: {
    extensions: ["AMSmath.js", "AMSsymbols.js",
"autoload-all.js"]
  }
};
```

- Define a function in `math-expression-editor.directive.ts` named `convertLatexStringToSvg()`, which takes `(latexString)` as an argument .

```

var convertLatexStringToSvg = function(inputLatexString) {
    var emptyDiv = document.createElement("div");
    var outputElement = angular.element(emptyDiv)
    // we need to append the element with a <script>
    // tag in order for mathjax to work on that element
    var $script = angular.element(
        <script type="math/tex">'
        ).html(inputLatexString);
    outputElement.html('');
    outputElement.append($script);
    MathJax.Hub.Queue(["Typeset", MathJax.Hub,outputElement[0]])
    MathJax.Hub.Queue(function() {
        ctrl.svgString = outputElement[0].innerHTML;

    });
}

```

- *Explanation of the above code*
 - First, an empty div element is created. This element will be typeset to an SVG string by MathJax
 - MathJax runs asynchronously, So Mathjax provides us a way to integrate it with our code.
 - This is done by using *MathJax Queues*. [link](#)
 - Any function added to the Mathjax Queue executes synchronously one after the other
 - So first we add the typeset function “Typeset” to the queue along with the element to be typeset.
 - Then we add the second set of instructions to the queue.
 - I.e extracting the SVG string from the typeset element and assigning it to a variable that will be later processed and saved to the backend.
 - Note that we can’t add these instructions outside of the queue, Because the second queue ensures that the previous operation(i.e Typesetting) is completed before executing the next set of instructions(i.e extracting the SVG string).

- Regarding alternate methods to convert Latex to SVG in Mathjax, there is no direct way to convert a Latex string to Svg string in MathJax 2.7.
- Hence we use this method of creating a custom element and then typesetting it.
- The above method used is similar to what is already used currently in (`mathJax-bind.directive.ts`)
- Mathjax 3 supports the conversion of a Latex string to SVG string directly without much effort, But Mathjax 3 is still in development and all the features from Mathjax 2.7 are not yet added in Mathjax 3.
- Mathjax 3 does not support typesetting of individual components in a page(Which is what we need), rather Mathjax 3 only allows Typesetting of the page as a whole.
- Mathjax 3 is still in beta.

Generating SVG data and filename from the SVG string.

- Define `convertSvgStringToSvgFile()` and `generateSvgFilename()` in `image-upload-helper.service.ts`
- In `convertSvgStringToSvgFile()`

```
let svg = svgString;
```

Here `svgString` before converting to raw data is of the form :

```
<svg xmlns:xlink="http://www.w3.org/1999/xlink" width="12.795ex"
height="4.388ex" viewBox="0 -817.3 768 1379.7" role="img"
focusable="false" style="vertical-align: -1.306ex;"><defs><path
stroke-width="10" id="E2-MJMATHI-78" d="M52 289Q59 331 106 386T
.....
.....
.....</path></defs><g
stroke="currentColor" fill="currentColor" stroke-width="0"
transform="matrix(1 0 0 -1 0 0)"><g
transform="translate(120,0)"><rect stroke="none" width="528"
height="60" x="0" y="220"></rect><use transform="scale(0.707)"
xlink:href="#E2-MJMATHI-78" x="84" y="594"></use><use
transform="scale(0.707)" xlink:href="#E2-MJMATHI-79" x="122"
y="-488"></use></g></g></svg>
```

- We will next extract the following attributes from the SVG and store it in the filename. (We need these parameters in order to display the SVG inside an image tag).
 - Height
 - Width
 - Style-vertical-align (vertical_padding)
- When the SVGs are loaded these attributes will be extracted from the filename and appropriately used to style the image tag containing the SVG.
- Now convert this svgString to raw data in the following way.
- The main step in converting the SVG string to Blob is:

```
let blob = new Blob([svg], {type: 'image/svg+xml'});
```

- This is the data that is sent to the backend for saving

Generation of filename in (*generateSvgFilename()*).

- The filename will be generated based on random numbers and time.

```
generateSvgFilename: function() {
  var date = new Date();
  return 'svg_' +
    date.getFullYear() +
    ('0' + (date.getMonth() + 1)).slice(-2) +
    ('0' + date.getDate()).slice(-2) +
    '_' +
    ('0' + date.getHours()).slice(-2) +
    ('0' + date.getMinutes()).slice(-2) +
    ('0' + date.getSeconds()).slice(-2) +
    '_' +
    Math.random().toString(36).substr(2, 10) +
    '_height_' + height +
    '_width_' + width +
    '_vertical_padding_' + vertical_padding +
    '.svg';
}
```


Saving the generated SVG File and Filename to the server(Backend).

- We use a function similar to saving images(Which is already present)
- The upload handler for images and SVG is already present in the backend.
- We have to slightly modify the backend to make SVGs as a supported file type. (`feconf.py`)
- As the saving to the server is done after the user clicks on the “Done” button, we have to put the function which saves SVG to the backend in an `async` function and return the promise.
- Then we will use `await` in order to wait for the promise to resolve and assign the response filename to the customization-arg `svg_filename`.

```
ctrl.data.mathExpressionContent.svg_filename= response.data
```

- Hence the Generated SVG is linked to the latex string corresponding to a math-expression

Changing the oppia-non-interactive-maths

- The `oppia-noninteractive-math.directive.ts` has to be modified to extract the `raw_latex` property from the modified customization arg.
- We will use `mathjax` and `latex` to render the SVG until the migrations are completed
- Once the migrations are completed we can use the `svg_filename` to render the images.

```
angular.module('oppia').directive('oppiaNoninteractiveMath', [  
  'HtmlEscaperService',  
  function(HtmlEscaperService) {  
    return {  
      restrict: 'E',  
      scope: {},  
      bindToController: {},  
      template: require('./math.directive.html'),  
      controllerAs: '$ctrl'  
      controller: ['$attrs', function($attrs) {
```

```

var ctrl = this;
ctrl.$onInit = function() {
  ctrl.mathExpressionsContent = HtmlEscaperService.escapedJsonToObj(
    $attrs.mathExpressionsContentWithValue);
  ctrl.rawLatex = ctrl.mathExpressionsContent.raw_latex;
}
};
}]
};

```

- `ctrl.mathExpressionsContent` has both `rawLatex` and `svgFilename` in it.
- The `oppia-noninteractive-math.directive.html` will use the `svg_filename` only after the complete migration.

Migrate all existing expressions on the server to include the new field `svg_filename`.

OVERVIEW OF THE MIGRATION

- We need to migrate all the existing expressions on the server to include the new field `svg_filename`
- **Note:- This migration has to be done as soon as we change the `rich_text_components_definitions.ts` file to include the new SVG field**
 - **Although at this stage the `svg_filename` will be empty.**
 - **We will populate the `svg_filename` with the reference to an SVG after this migration is completed.**
 - **This populating of SVG filename will be done through the Admin tab from the client-side (discussed in the coming section) after the existing math expressions are migrated to comply with the new rich text definition.**
- We need to carry out the migration for the HTML content at the following places
 - Explorations
 - Questions
 - Add Question suggestions
 - Translation suggestion.
- We will define the migrations in the respective domain file for each of the objects mentioned above.

- We will use a new helper function mainly defined in (mainly in [html_validation_services.py](#)) for adding the new attribute to the math-expressions RTE component.

HELPER FUNCTION NEEDED TO BE USED IN THE MIGRATION

- This function which when given an HTML string finds all the ([oppia-noninteractive-math](#)) tags and replaces the attribute ([raw_latex-with-value](#)) with the new attribute ([math_expression_contents-with-value](#)).
 - The function ([add_svg_filename_to_math_rte_components](#)) will be defined in ([html_validation_services.py](#))
 - The function takes an HTML string as an argument and returns the converted HTML string with the new attribute(with the empty `svg_filename` field).
 - The function will use *beautifulsoup4* for parsing the HTML strings. (This library is already being used in oppia in many places)

```
def add_svg_filename_to_math_rte_components(html_string):

    soup = bs4.BeautifulSoup(
        html_string.encode(encoding='utf-8'), 'html.parser')

    for math in soup.findAll(name='oppia-noninteractive-math'):
        raw_latex = json.loads(unescape_html(math['raw_latex-with-value']))
        # Generate an SVG file and filename from the given Latex string
        math_expression_contents_dict = {
            "raw_latex": raw_latex,
            "Svg_filename": " "
        }
        # delete the attribute raw_latex-with-value
        del math['raw_latex-with-value']
        # add the new attribute math_expression_contents-with-value
        math['math_expression_contents-with-value'] = escape_html(
            json.dumps(math_expression_contents_dict))

    return python_utils.UNICODE(soup)
```

DETAILS OF THE MIGRATION.

Migrating the Explorations

- All the places in explorations where HTML content(with math Rich Text component) is used needs migration.
- Migrating the states (states_dict) of an exploration
 - We will use the function `convert_html_fields_in_state` already defined in `state_domain.py` for migrating the states.
 - This function takes a general conversion function and applies this conversion function to all the HTML contents in the states
 - The following values of the exploration state_dict have HTML content:
 - `state_dict['content']['html']`
 - `state_dict['interaction']['default_outcome']['feedback']['html']`
 - `state_dict['interaction']['answer_groups']`
 - `state_dict['interaction']['hints']`
 - `State_dict['interaction']['solution']`
 - `State_dict['interaction']['customization_args']['choices']['value']` in the case of 'ItemSelectionInput' and 'MultiChoiceInput' interaction.
 - `state_dict['written_translations']`
 - The Function `def convert_html_fields_in_state()` used for migrating.(Already defined)

```
def convert_html_fields_in_state(cls, state_dict, conversion_fn):  
    """Applies a conversion function on all the html strings in a state  
    to migrate them to a desired state.
```

```
    Args:
```

state_dict: dict. The dict representation of State object.
conversion_fn: function. The conversion function to be applied on
the states_dict.

Returns:

dict. The converted state_dict.

"""

```
state_dict['content']['html'] = (
    conversion_fn(state_dict['content']['html']))
if state_dict['interaction']['default_outcome']:
    interaction_feedback_html = state_dict[
        'interaction']['default_outcome']['feedback']['html']
    state_dict['interaction']['default_outcome']['feedback'][
        'html'] = conversion_fn(interaction_feedback_html)

for answer_group_index, answer_group in enumerate(
    state_dict['interaction']['answer_groups']):
    answer_group_html = answer_group['outcome']['feedback']['html']
    state_dict['interaction']['answer_groups'][
        answer_group_index]['outcome']['feedback']['html'] = (
        conversion_fn(answer_group_html))
    if state_dict['interaction']['id'] == 'ItemSelectionInput':
        for rule_spec_index, rule_spec in enumerate(
            answer_group['rule_specs']):
            for x_index, x in enumerate(rule_spec['inputs']['x']):
                state_dict['interaction']['answer_groups'][
                    answer_group_index]['rule_specs'][
                        rule_spec_index]['inputs']['x'][x_index] = (
                            conversion_fn(x))

for hint_index, hint in enumerate(
    state_dict['interaction']['hints']):
    hint_html = hint['hint_content']['html']
    state_dict['interaction']['hints'][hint_index][
        'hint_content']['html'] = conversion_fn(hint_html)

if state_dict['interaction']['solution']:
    solution_html = state_dict[
        'interaction']['solution']['explanation']['html']
```

```

        state_dict['interaction']['solution']['explanation']['html'] = (
            conversion_fn(solution_html))

    if state_dict['interaction']['id'] in (
        'ItemSelectionInput', 'MultipleChoiceInput'):
        for value_index, value in enumerate(
            state_dict['interaction']['customization_args'][
                'choices']['value']):
            state_dict['interaction']['customization_args'][
                'choices']['value'][value_index] = conversion_fn(value)
    return state_dict

```

- We will use the helper function `add_svg_filename_to_math_rte_components` which we defined earlier in `html_validation_service.py` as a conversion Function.
- So the whole migration for states will be as done by defining (`def _convert_states_v32_dict_to_v33_dict`):

```

def _convert_states_v32_dict_to_v33_dict(cls, states_dict):
    for state_dict in states_dict.values():
        for key, state_dict in states_dict.items():
            states_dict[key] =
                state_domain.State.convert_html_fields_in_state(
                    state_dict, html_validation_service.
                        add_svg_filename_to_math_rte_components)
    return states_dict

```


- Migrating the Draft changes (draft_change_list) of an exploration by the user
 - We'll define a migration function in (`class DraftUpgradeUtil`) in the file (`draft_uupgrade_services.py`)
 - We modify the `draft_change_list` in the following cases
 - When the user edits the state property i.e when change `cmd` is equal to `CMD_EDIT_STATE_PROPERTY`(Look in `class ExplorationChange` in `exp_domain.py` for details).
 - When the property being edited is `answer_groups`.
 - Note that, We need to handle the case of “ItemSelectionInput” separately because in “ItemSelectionInput” the property (`rule_spec['inputs']['x']`) allows RTE components.
 - There is no direct way to get the interaction type from the (`draft_change_list`).
 - We can check whether an interaction is “ItemSelectionInput” by checking the type of the property (`rule_spec['inputs']['x']`).
 - If the type of (`rule_spec['inputs']['x']`) is string then we know that the interaction is “ItemSelectionInput”.
 - If this property does not exist or is not a string, then the interaction is not “ItemSelectionInput” and we don't need to handle this case.
 - When the property being edited is the main (`content`).
 - When the property being edited is (`customization_args`).
 - We need to upgrade the draft if the interaction type is either “MultiChoiceInput” or “ItemSelectionInput”.
 - This is because only these interactions allow RTE components in their choice(`customization arg`).
 - In this case, we can check whether an interaction is “ItemSelectionInput” or “MultiChoiceInput” by checking the type of the property (`choices['value']`)
 - If this property exists and is a String(HTML) then the interaction is either “ItemSelectionInput” or “MultiChoiceInput”.
 - We have to define helper functions

convert_html_fields_in_draft_change_list_answer_group() in draft_upgrade_services.py for applying the conversion function we defined earlier to the (draft_change_list) **answergroups**.

```
def convert_html_fields_in_draft_change_list_answer_groups(
    draft_change_list, conversion_fn):
    for answer_group_index, answer_group in enumerate(
        draft_change_list):
        answer_group_html = answer_group['outcome']['feedback']['html']
        draft_change_list[answer_group_index]
        ['outcome']['feedback']['html'] = (
            conversion_fn(answer_group_html))
        # we check whether the interaction type is ItemSelectionInput by passing
        # the rule_spec property to a helper function which returns a boolean.
        if check_is_interaction_ItemSelectionInput(answer_group['rule_spec']):
            for rule_spec_index, rule_spec in enumerate(
                answer_group['rule_specs']):
                for x_index, x in enumerate(rule_spec['inputs']['x']):
                    state_dict['interaction']['answer_groups'][
                        answer_group_index]['rule_specs'][
                            rule_spec_index]['inputs']['x'][x_index] = (
                                conversion_fn(x))
    return draft_change_list
```

- We have to also define a helper function

(**convert_html_fields_in_draft_change_list_customization_arg()**) in (draft_upgrade_services.py) for applying the conversion function we defined earlier to the (draft_change_list) to the customization_args if required.

```
def convert_html_fields_in_draft_change_list_customization_args(
    draft_change_list, conversion_fn):
    if check_is_interaction_ItemSelection_or_MultiChoice(
```

```

        draft_change_list['choices']):
    for value_index, value in enumerate(
        draft_change_list['choices']['value']):
        draft_change_list['choices']['value']
            [value_index] = conversion_fn(value)
    return draft_change_list

```

- The main migration function to migrate `draft_change_list` :function in (`class DraftUpgradeUtil`) in the file (`draft_uupgrade_services.py`)

```

def _convert_states_v33_dict_to_v34_dict(cls, draft_change_list):
    for i, change in enumerate(draft_change_list):
        if (change.cmd == exp_domain.CMD_EDIT_STATE_PROPERTY and
            change.property_name ==
                exp_domain.STATE_PROPERTY_INTERACTION_ANSWER_GROUPS):
            draft_change_list[i] = exp_domain.ExplorationChange({
                'cmd': exp_domain.CMD_EDIT_STATE_PROPERTY,
                'property_name': (
                    exp_domain.STATE_PROPERTY_INTERACTION_ANSWER_GROUPS),
                'state_name': change.state_name,
                'New_value':(
                    convert_html_fields_in_draft_change_list_answer_groups(
                        change.new_value,
                        add_svg_filename_to_math_rte_components)),
                'Old_value':(
                    convert_html_fields_in_draft_change_list_answer_groups(
                        change.old_value,
                        add_svg_filename_to_math_rte_components))
            })
        elif (change.cmd == exp_domain.CMD_EDIT_STATE_PROPERTY and
            change.property_name ==
                exp_domain.STATE_PROPERTY_INTERACTION_CUST_ARGS):
            draft_change_list[i] = exp_domain.ExplorationChange({
                'cmd': exp_domain.CMD_EDIT_STATE_PROPERTY,
                'property_name': (

```

```

        exp_domain.STATE_PROPERTY_INTERACTION_CUST_ARGS),
    'state_name': change.state_name,
    'new_value':(
        convert_html_fields_in_draft_change_list_customization_args(
            change.new_value,
            add_svg_filename_to_math_rte_components)),
    'old_value':(
        convert_html_fields_in_draft_change_list_customization_args(
            change.old_value,
            add_svg_filename_to_math_rte_components))
    })
elif (change.cmd == exp_domain.CMD_EDIT_STATE_PROPERTY and
      change.property_name ==
      exp_domain.STATE_PROPERTY_CONTENT):
    draft_change_list[i] = exp_domain.ExplorationChange({
        'cmd': exp_domain.CMD_EDIT_STATE_PROPERTY,
        'property_name': (
            exp_domain.STATE_PROPERTY_CONTENT),
        'state_name': change.state_name,
        'new_value':add_svg_filename_to_math_rte_components(
            change.new_value),
        'old_value':add_svg_filename_to_math_rte_components(
            change.old_value)
    })

```

Migrating the Questions, Question Suggestions, and Translation.

- Migrations to be performed to questions are very similar to explorations.
- Questions don't have multiple states as compared to explorations.
- The Places where RTE components are used in Questions and require migrations.
 - State data (Similar to explorations)
 - Question Content
- Define the migration function in (question_domain.py).
- We would be reusing the helper functions we defined for migrating explorations in order to migrate the questions.

- Also, We need to migrate suggestions of type
SUGGESTION_TYPE_TRANSLATE_CONTENT,
SUGGESTION_TYPE_ADD_QUESTION ,
SUGGESTION_TYPE_EDIT_STATE_CONTENT .
- The migration process will be similar to the migration of questions. (state_dict)

POPULATING THE SVG FILE-PATH FIELD IN MATH RTE COMPONENTS.

OVERVIEW OF THE POPULATION OF SVG FIELDS

- As decided in the comparison section, we will be populating the SVG filenames for existing math RTE components on the client-side using Mathjax.
- We need to populate the `svg_filename` field which we added to all the `oppia-noninteractive-math` tags in the previous migration.
- We will carry out this procedure from the Admin tab.

Mocks for exploration migration form

Generate SVGs for explorations

Use this form to generate SVGs for Explorations.

Number of explorations in the Server : 50

Index of the first exploration to migrate

Index of the last exploration to migrate

[Generate SVGs](#)

Mocks for question migration form.

Generate SVGs for questions

Use this form to generate SVGs for Explorations.

Number of Questions in the Server : 80

Index of the first question to migrate

Index of the last question to migrate

[Generate SVGs](#)

POPULATING THE EXPLORATIONS AND DRAFT CHANGE LISTS WITH SVGs

- Add a function in (editable-exploration-backend-api.[service.ts](#)) to fetch 'X' number of explorations from a given starting index.

- The function takes the indices as an argument and returns an object with the given number of explorations and associated draft change lists.
- We need to define a controller in the backend which will return the required number of explorations and associated draft change lists.
- The controller will return a dict with the key 'exploration_data', whose value is a list of all the required exploration data and corresponding draft_changes of users.
- The algorithm used inside the controller to create the output_array:
 - Get the list of all the explorations, sorted according to when they were first created. (using the property `created_on` in `BaseModel`)
 - Create a list of these explorations and assign an index to each exploration ID.
 - add all the explorations specified in the given index range to the output array.
 - In this way, the index of the explorations always remains the same. (because they are sorted according to the dates when they were first published)
 - Now for each of these exploration IDs find out which users have a draft change list associated with a given exploration ID.
 - Fetching of the users '`draft_change_lists`;' we can do this in 2 ways.
 - i. The first approach: Get all the `user_ids` and iterate through them to check whether a particular `user_id` has an `ExplorationUserDataModel` for the given `exp_id`, this can be checked by using the method `def has_reference_to_user_id()` in the `ExplorationUserDataModel`.

code for the query(1st approach)

```
exploration_model = exp_models.ExplorationModel.get(exp_id)
for user_id in user_ids:
    if exploration_model.has_reference_to_user_id(user_id):
        draft_change_list = user_models.ExplorationUserDataModel.get(
            user_id, exp_id)
```

- ii. The Second Approach: Get all the `ExplorationUserDataModels` and iterate to find all the `user_ids` with a draft_change for a particular `exp_id`.

code for the query(2nd approach)

```
users_draft_change_lists = []
exploration_user_data_models = user_models.ExplorationUserDataModel.get_all()
for exploration_user_data_model in exploration_user_data_models:
    if exploration_user_data_model.exploration_id == exp_id:
        user_draft_change_list = {
            'user_id' : exploration_user_data_model.user_id,
            'draft_changes': exploration_user_data_model.draft_change_list
        }
        users_draft_change_lists.push(user_draft_change_list)
```

- Add the draft change lists of all the users(If exists) along with exploration_dict in the output array of an exploration.
- The final output dict is of the form:

```
{
  'exploration_data':
    [{
      'exploration_dict' : {
        // exploration dict for the exploration with index X
      },
      'users_draft_change_lists': [
        // Draft change lists of multiple users who have edited the
        // exploration with index X.
        // Each element of this array is an object with keys user_id
        // and draft_change_list_dict
        {
          'user_id' :
          'draft_changes': []
        }, {
          'user_id' :
          'draft_changes': []
        }
      ]
    }
  ]
}
```

```

    },{
      'exploration_dict' : {
        // exploration dict for the exploration with index X+1
      },
      'users_draft_change_lists': [
        // draft change lists of multiple users who have edited the
        // exploration with index X+1
      ]
    }]
  }
}

```

- We have to loop through all the components of the `exploration_dict` having HTML content similar to what we did in the migration of adding `svg_filename` attribute to all math RTE components(Similar to page 24)
- We will use a DOM parser to find oppia-non-interactive tags in the HTML string and add the SVG file name with the help of the SVG generator function defined using MathJax.(As previously considered)
- As the above process is asynchronous we need to use `async` and `await`.
- Since the exploration update in the backend requires a change list, we will maintain a change list (initially empty) and add all the changes we make to the `exploration_dict` into the change list.
- The above process of adding objects to the change list will be similar to what we do in functions `editExplorationProperty()` and `editStateProperty()` defined in `(ChangeListService)`.
- Similarly, The `draft_change_list` can be populated by looping through the objects using the same logic that we followed in the migration of `draft_change_list`.
- In the case of `draft_change_list`, we don't need to maintain a change list as we did in the case of `state_dict`, we can directly update the HTML.
- We also need to define a controller for saving this migrated `exploration_dict` and `draft_change_list`.
- Handling the updated exploration data.
 - The data which we use to update the explorations will be similar to the `output_dict` which we fetched in the previous step, the only difference is that inside the key `'exploration_dict'` it will have only the `change_list` and the `exp_id`.
 - Define a controller in `admin.py` to handle the saving of the updated exploration

- The controller will iterate through each `exploration_dict`'s `change_list` and the corresponding user `draft_change_list`.
- We update the explorations with the help of function `update_exploration()` in `exp_services.py`.
- We update the `draft_change_list` using a function similar to `create_or_update_draft()` in `exp_services.py`.
- Saving the generated SVGs.
 - In order to avoid multiple calls to the backend, we will save the SVG images in batches.
 - The Max size of each batch can be predefined. (for reference, Max size of image upload allowed currently in oppia is 1mb)
 - The size of an SVG for a typical equation is around 10-15kb so we can save many SVGs in a batch.
 - Saving these small SVGs individually is very inefficient and requires many calls to the server, so saving in batches is better.
 - We will maintain an object of the following form, and push newly generated SVGs in the list 'svg_images'.

```
{
  'svg_images':[
    {
      'raw_data': ,
      'file_name': ,
      'entity_type': ,
      'entity_id':

    },{
      'raw_data': ,
      'file_name': ,
      'entity_type': ,
      'entity_id':

    }...
  ]
}
```

- We also maintain a variable '`total_batch_size`' to store the size of the object.
- We will update the variable every time a newly generated SVG is added to the object.
- When the variable '`total_batch_size`' exceeds the max size(1mb) we will save this data to the backend.
- The payload given will be the object we created in the previous step.
- In the backend, we create a handler that will iterate through each SVG file data and save the SVG using functions in `fs_service.py` and `fs_domain.py`.
- While saving the files in the backend we will maintain an `error_dict` and return this a response.
- The error dictionary will have the filename, `entity_id`, and `entity_type` of all the SVGs which have not been saved properly.
- This process will be repeated every time the variable '`total_batch_size`' exceeds the max size.

POPULATING THE QUESTIONS, QUESTIONS, AND TRANSLATION SUGGESTIONS WITH SVGs.

- The questions will be populated with SVGs in a very similar manner to how we populated `exploration state_dict` and `exploration content`.
- We need to define services in `editable-question-backend-api.service.ts` to save fetch 'X' number of questions as requested by the admin tab.
- We also need to define a controller in the backend which will return the required number of questions
- We need to maintain a change list and loop through the question `state_dict` object and update the change list.
- Finally, we update the questions in the backend using the change lists generated in the previous step
- We need to define new services and controllers for fetching the suggestions of type `SUGGESTION_TYPE_TRANSLATE_CONTENT` and `SUGGESTION_TYPE_ADD_QUESTION` from the server.

- The migration process will be similar to the migration of questions. (state_dict

AUDITING AND VALIDATING THAT THE MIGRATIONS HAVE BEEN SUCCESSFUL

- We need Validation one-off jobs for all the Domain objects we have migrated in the previous section.
- The one-off jobs for the validation will be added in respective one-off job files for the objects.
- Validation Job mainly to be added are in explorations states, (Draft_change_list), and questions states and (change_list)
- The Validation jobs will be using helper functions defined in html_validation_services.py.
- The function to be defined in (html_validation_services.py) achieve the following things:
 - The arguments given will be an HTML list containing all the HTML strings in a particular object. (Ex exploration states, exploration draft_change_lists etc.) and an err_dict(to store invalid tags).
 - The function first finds all the tags having name "oppia-noninteractive-math"
 - Checks whether the attribute "math_expressions_content-with-value" has the value "svg_filename" populated
 - If the value "svg_filename" exists then, extract the value "svg_filename" from the attribute "math_expressions_content-with-value"
 - verify using (html_validation_services.py)) and (fs_domain.py) that the filename exists in the datastore
 - If the filename is invalid or non-existent add the tag to the err_dict.
 - The function will then return this err_dict back to the one-off job function.
- HELPER FUNCTIONS:
validate_svg_filenames_in_math_rte_component()in
 html_validation_services.py

```
def validate_svg_filenames_in_math_rte_component(
```

```

        html_list):
    # err_dict is a dictionary to store the invalid tags and the
    # invalid parent-child relations that we find.
    err_dict = {}
    # All the invalid html strings will be stored in this.
    err_dict['strings'] = []

    for html_data in html_list:
        soup = bs4.BeautifulSoup(
            html_data.encode(encoding='utf-8'), 'html.parser')
        is_valid = validate_soup_for_svg_filename(soup)
        if is_valid:
            pass
        else:
            err_dict['strings'].append(html_data)

```

```

def validate_soup_for_svg_filename(soup):
    for math in soup.findAll(name='oppia-noninteractive-math'):
        math_components = json.loads(unescape_html(
            math['math_components-with-value']))
        if math_components:
            svg_filename = math_components['svg_filename']
            validate_svg_filename(svg_filename)
            if validate_svg_filename:
                pass
            else:
                return False
    return True

```

- One-off job to validate exploration state-data:
 - The job will be named (`class ExplorationContentSvgfilenameValidationJob`)

```

class ExplorationContentSvgfilenameValidationJob(
    jobs.BaseMapReduceOneOffJobManager):

```

```

@classmethod
def entity_classes_to_map_over(cls):
    return [exp_models.ExplorationModel]

@staticmethod
def map(item):
    if item.deleted:
        return

    try:
        exploration = exp_fetchers.get_exploration_from_model(item)
    except Exception as e:
        yield (
            'Error %s when loading exploration'
            % python_utils.convert_to_bytes(e), [item.id])
    return

html_list = exploration.get_all_html_content_strings()

err_dict = html_validation_service.
    validate_svg_filenames_in_math_rte_component(html_list)

for key in err_dict:
    if err_dict[key]:
        yield ('%s Exp Id: %s' % (key, item.id), err_dict[key])

@staticmethod
def reduce(key, values):
    final_values = [ast.literal_eval(value) for value in values]
    # Combine all values from multiple lists into a single list
    # for that error type.
    output_values = list(set().union(*final_values))
    exp_id_index = key.find('Exp Id:')
    if exp_id_index == -1:
        yield (key, output_values)
    else:
        output_values.append(key[exp_id_index:])

```



```
yield (key[:exp_id_index - 1], output_values)
```

- The one-off job for exploration state is straight-forward because we already have the function (`get_all_html_content_strings()`) defined .
- This makes getting the HTML string easy
- One-off job to validate exploration `draft_change_list` by a user:
 - The one-job for validating `draft_change_list` is not that straightforward.
 - We first need to define a helper function similar to (`get_all_html_content_strings()`) in (`user_domain.py`) to get all the HTML content in the `draft_change_list` of a user for a particular exploration.
 - We will use logic similar to what we used in migrating the (**`draft_change_list`**) in order to extract the HTML.(see [migrating draft_change_list](#))

```
def get_all_html_content_strings_from_draft_change_list(draft_change_list):  
  
    html_list = []  
    for i, change in enumerate(draft_change_list):  
        #extract HTML from answer_groups of draft_change_list  
        if (change.cmd == exp_domain.CMD_EDIT_STATE_PROPERTY and  
            change.property_name ==  
            exp_domain.STATE_PROPERTY_INTERACTION_ANSWER_GROUPS):  
            #extract all the html from the old answer groups  
            for answer_group_index, answer_group in enumerate(  
                change.new_value):  
                answer_group_html = answer_group['outcome']  
                    ['feedback']['html']  
                html_list.push(answer_group_html)  
                if check_is_interaction_ItemSelectionInput(  
                    answer_group['rule_spec']):  
                    for rule_spec_index, rule_spec in enumerate(  
                        answer_group['rule_spec']):
```

```

        answer_group['rule_specs']):
        for x_index, x in enumerate(
            rule_spec['inputs']['x']):
            html_list.push(x)
#extract all the html from the new answer groups
for answer_group_index, answer_group in enumerate(
    change.old_value):
    answer_group_html = answer_group['outcome']
        ['feedback']['html']
    html_list.push(answer_group_html)
    if check_is_interaction_ItemSelectionInput(
        answer_group['rule_spec']):
        for rule_spec_index, rule_spec in enumerate(
            answer_group['rule_specs']):
            for x_index, x in enumerate(
                rule_spec['inputs']['x']):
                html_list.push(x)

#extract HTML from customization_args of draft_change_list
elif (change.cmd == exp_domain.CMD_EDIT_STATE_PROPERTY and
    change.property_name ==
    exp_domain.STATE_PROPERTY_INTERACTION_CUST_ARGS):
    if check_is_interaction_ItemSelection_or_MultiChoice(
        change.old_value['choices']):
        for value_index, value in enumerate(
            change.old_value['choices']['value']):
            html_list.push(value)
    if check_is_interaction_ItemSelection_or_MultiChoice(
        change.new_value['choices']):
        for value_index, value in enumerate(
            change.new_value['choices']['value']):
            html_list.push(value)
#extract HTML from contents of draft_change_list
elif (change.cmd == exp_domain.CMD_EDIT_STATE_PROPERTY and
    change.property_name ==
    exp_domain.STATE_PROPERTY_CONTENT):
    html_list.push(change.old_value)
    html_list.push(change.new_value)

```

RENDERING THE MATH EXPRESSIONS USING THE SVGs

- Once the migrations and validations discussed in the previous parts are completed we can use the SVG images stored to display math-equations.
- We need to replace the math-Jax bind directive with an image directive.
- We will use the tag to render the SVG
- SVGs will be cached in (AssetsBackendApiService's) cache and loaded similar to images.

CACHING THE SVGs

- The SVGs will be cached with the image files in AssetsBackendApiService's cache.
- The SVG file names will be extracted from the HTML content in [ExtractImageFilenamesFromStateService](#)
- We will add a function in this service to extract the SVG filenames from oppia non-interactive math tags.
- Then we will add these file names along with the image filenames to be cached into the AssetsBackendApiService's cache.

CHANGING THE OPPIA-NONINTERACTIVE MATH DIRECTIVE TO RENDER THE SVGs

- We now change the oppia-noninteractive math directive to use the SVG file-path from the customization args passed to the directive. ([mathExpressionsContentWithValue](#))
- We define helper functions in (assets-backend-api.service.ts) and (image-preloader.service.ts) similar to images, in order to fetch the URL of the SVG to be displayed.
- We define functions to extract dimensions of the SVG from the filename in image-preloader.service.ts.
- We can then use these dimensions to style the SVGs and display them. (Similar to images)

- We will be using an image tag with appropriate styling (using the dimensions extracted from the filename) to display the SVG.

Testing Approach

- The test files will be added whenever changes are made in service.ts files in the frontend along with respective PR
- All the backend migrations PRs will include test files with them
- E2e tests will be added to check the flow of the new math-expression editor described in the product design.
- Unit tests for the following journeys or flows will be added.
 - Click on the add RTE component for Math.
 - If a new expression is created, While in the math-expression-editor expect the svg_filename field to be empty.
 - Also, If we edit an existing math RTE component expect the svg_filename to be not empty in the math-expression-editor.
 - Save the math-expression.
 - While in the editor or exploration player check that the svg_filename field is not empty and has a valid filename.

Milestones

Milestone 1

Key Objective:

- All RTE math expressions have a new “SVG filename” attribute in addition to the LaTeX value. For existing expressions, this filename will be an empty string. For newly-created expressions, the filename will be non-empty and the corresponding SVG file is guaranteed to exist on the server (but will not be used for display yet).

(Note: migrations in this milestone are expected to be run in July release cut)

No.	Description of PR	Prereq	Target date	Target date
-----	-------------------	--------	-------------	-------------

		PR number s	for PR submission	for PR to be merged
1.1	<p>--Modify the rich_text_components_definitions.ts and migrate all expressions in the server to replace the attribute raw-latex-with-value with 'math_expression_contents-with-value'.</p> <p>--Define the helper functions in html_validation_service.py to add the new attribute (which complies with the new math RTE definition) to all the math tags in a given HTML string.</p> <ul style="list-style-type: none"> • Add tests for these functions In html_validation_services_tests.py) • Modify existing tests to account for the new Math component structure. <p>--Migrate the Explorations state data and contents</p> <ul style="list-style-type: none"> • Define the migration function in (exp_domain.py). • Write tests for the migration functions in (exp_domain_test.py <p>--Migrate the Explorations draft_change_list of a user.</p> <ul style="list-style-type: none"> • Define the helper functions for the draft_change_list migrations in (draft_upgrade_service.py) • Write the migration function • Write tests for all the above functions 	None	Jun 7	Jun 12

1.2	<p>–Migrate the Question State data and contents</p> <ul style="list-style-type: none"> • Define the migration function in (question_domain.py). • Write tests for the migration function. <p>–Migrate the Question and Translation Suggestions</p> <p>(Note: the above migrations are expected to run in July's release cut.)</p> <p>–Modify the Protactor.js for math RTE components to accommodate the new Structure.</p>		Jun 12	Jun 17
	<i>After PR 1.1 and 1.2 all the expressions in the server will have the new Math RTE structure. But the SVGs are not yet generated for the old expressions</i>			
1.3	<p>–Add helper functions in (image-upload-helper.service.ts)</p> <ul style="list-style-type: none"> • for generating the SVG filename • converting the SVG string to raw data. <p>– Add tests for these functions in (image-upload-helper.service.spec.ts)</p>	1.1,1.2	Jun 15	Jun 18
1.4	<p>Add helper functions in (assets-backend-api.service.ts) for saving the SVGs to the backend.</p> <p>Add tests for these functions in (assets-backend-api.service.spec.ts).</p>	1.1,1.2	Jun 17	Jun 20
	<i>After 1.3 and 1.4 all the helper functions to generate and save the SVGs to the backend will be ready</i>			
1.5	<p>Add/modify functions in the math-latex-string-editor.directive.ts</p> <ul style="list-style-type: none"> • Modify Mathjax configuration file (mathjax-config.ts) • Function for converting the Latex string to SVG string (convertLatexStringToSvg()) 	1.4,1.3	Jun 21	Jun 24

1.6	Modify the <code>oppia-noninteractive-math.directive.ts</code> to have both <code>svg_filename</code> and <code>raw_latex</code> field but use <code>raw_latex</code> to display math.	1.5	Jun 23	Jun 26
	<i>After 1.5 and 1.6, the math RTEs will use the new structure and all newly created SVGs will have an SVG saved in the backend.</i>			

Milestone 2

Key Objective:

- All RTE math expressions in the server (in explorations, questions, topics, skills, question/translation suggestions and any other RTEs) include a non-empty SVG filename that points to a valid file. There is a validation job that can be used to audit this, and that validation job runs to completion successfully on existing production data.

(Note migrations in this milestone are expected to be run in August release cut)

No.	Description of PR	Prereq PR numbers	Target date for PR submission	Target date for PR to be merged
2.1	Add controllers in the backend to fetch and save Explorations, Questions and suggestions. (admin.py) <ul style="list-style-type: none"> • Define a handler to fetch and save Exploration data and associated <code>draft_change_lists</code>. • Define a handler to fetch and save questions data • Define a handler to fetch and save Questions and suggestions data. • Write tests for the above handlers 	None	Jul 6	Jul 10
	<i>After PR 2.1 all the backend infrastructure for saving SVGs for existing expressions will be ready.</i>			

2.2	Add functions in <code>editable-exploration-backend-api.service.ts</code> and <code>editable-exploration-backend-api.service.ts</code> to interact with the handlers defined in the previous PR 2.1. Add tests for the above functions.	2.1	Jul 9	Jul 13
	<i>After 2.2 frontend services to fetch and save entities to migrate(populate with SVGs) will be ready</i>			
2.3	-Write a one-off validation job to validate the migration of exploration state data and contents <code>exp_jobs_one_off.py</code> -Write a one-off validation job to validate the migration of exploration draft_change_list in the file <code>user_jobs_one_off.py</code> -Write a one-off Job to validate the question migration and write tests. <code>question_jobs_one_off.py</code> -Write a one-off Job to validate the question and translation suggestions migration. -Write tests for the above one-off jobs.	None	Jul 14	Jul 19
	<i>After 2.3 one-off jobs to validate that SVGs exist for all expressions will be ready.</i>			
2.4	Add functions in <code>admin-activities-tab.directive.ts</code> to generate SVGs for all the math-expressions in the server <ul style="list-style-type: none"> Function to generate SVG and Save SVGs for all math RTEs in a given HTML string(using Mathjax). Function to loop through the Exploration_dict and draft_change_lists and find HTML content having RTEs. Function to loop through Questions and find HTML content having RTEs. Function to loop through Question and translation Suggestions and find 	2.1, 2.2	Jul 19	Jul 24

	HTML content having RTEs.			
2.5	Make the changes to admin-activities-tab.directive.html to display the form for migration(as shown in the mock)	2.4	Jul 22	Jul 25
	After 2.4 and 2.5, The form to generate SVGs for existing Math RTEs will be ready in the admin tab.			

Milestone 3

Key Objective:

- All math expressions are displayed using SVGs in the learner view.
- MathJax is fully removed from the learner view.

No.	Description of PR	Prereq PR numbers	Target date for PR submission	Target date for PR to be merged
3.1	Modify AssetsBackendApiService to allow caching of SVGs as assets.	None	Aug 5	Aug 8
3.2	Create/Modify Helper functions in (image-preloader.service.ts) to fetch and load SVGs similar to images.	None	Aug 7	Aug 10
	After 3.1 and 3.2, all the frontend infrastructure to cache and load SVGs like image assets will be ready.			
3.3	Modify the directive for oppia-noninteractive-math.ts to render the stored SVGs	3.1 and 3.2	Aug 10	Aug 13
3.4	Modify the template which oppia-noninteractive-math.ts controls to render the stored SVGs in	3.3	Aug 12	Aug 15

	math.directive.html (similar to images)			
3.5	Remove mathjax from Leaner pages. Mathjax has to be removed from the following places. <ul style="list-style-type: none"> • exploration-player-page.mainpage.html • practice-session-page.mainpage.html 	3.4	Aug 14	Aug 17
	<i>After 3.3,3.4 and 3.5, the Math expressions will be rendered using the SVGs rather than Mathjax</i>			
3.6	Add e2e tests to completely test the new flow of displaying and creation of math RTE components.	3.4	Aug 19	Aug 23
	Manually test the new feature and fix any bugs that come up.		Aug 20- Aug 24.	

Additional Project-Specific Considerations.

Documentation Changes

As the migrations are a bit complex, documentation will be added in oppia-wiki