CIS*2750
Assignment 3
Module 1

1. Preliminaries

You will be provided with the A3 Stub, which includes both the client and the server stubs. See A3 Stub documentation PDF for details. You **must** follow the code organization described in the A3 Stub documentation.

Your Module 1 functionality must be placed into three stub files:

- All HTML code must go into `public/index.html`
- All client-side code must go into `public/index.js`. Do not embed any JavaScript code directly into `index.html`.
- A template CSS file has been provided for you in `public/style.css`. Do not embed any CSS code directly into `index.html`.
- All server-side code JavaScript code (Module 2) must go into `app.js` in the root directory of the stub.

As discussed in the lecture notes, you will be using the de-facto standard of modern Web graphical user interfaces - HTML + JavaScript with the jQuery library. Some jQuery examples are provided for you (see Week 6-7 examples). There are also multiple JavaScript/HTML/jQuery examples in *"Learning PHP, MySQL, JavaScript, CSS & HTML5*".

You have some freedom in how you want to lay out the GUI and determine its exact appearance, but the items below items will be required. Items specified as drop-down lists or buttons must stay that way; you cannot switch them to other UI elements.

The `index.html` file provided for you already includes a number of JavaScript libraries, including jQuery and Bootstrap.

You may wish to use Cascading Style Sheets (CSS) for prettifying your GUI, but you are not required to do so. You will not lose any marks if you use the default CSS file included in the Stub.

You **must** test your code on NoMachine, using **Firefox** installed on the SoCS servers. If your GUI does not work correctly on this browser, you **will** lose marks - up to and including getting a **zero (0)** in this project. Testing it on your home browser is **not** sufficient.

You are welcome to develop your code at home and test your code on a browser of your choice. However, make sure you also you test your code on the Linux server **as you develop it - do not** save the testing until the last minute! Use the NoMachine Graphical Linux Environment: **https://wiki.socs.uoguelph.ca/techsupport/guides/nomachine**

2. Overview of Web GUI

The UI web page has a title showing "SVG Image Viewer". It has a number of required forms and buttons. There will be three major display "panels" (how you implement them in HTML is up to you):

- The File Log Panel for displaying images, along with file summaries and download options. It is also scrollable horizontally.
- The SVG View Panel for display of SVG components, one component per line. You should assume that the SVG image you load from a file could have at least 10 top-level components - i.e. immediate children of `svg` element - and the panel is appropriately scrollable.
- Additional functionality - creating SVG images, adding components to existing image, scaling (see Section 5).

- Various error messages and, occasionally, other data will be displayed as JavaScript alerts, as described in Sections 4 and 5.

- You will also need to display success/error messages into the browser's JavaScript console using the `console.log` method for debugging purposes. Whenever you get a response from the server to a specific GET/POST, display the the response status to the debugging console - e.g. "file successful uploaded", "failed to create SVG file - invalid format", etc.. These messages will not be visible to the user, but they will greatly help you debug things, and we might need to use them when grading.

Sample app layout:

File Log Panel
SVG View Panel
Additional functionality goes here (create SVG, add shapes, scaling)

The two panels and all the forms/buttons described in Sections 3 - 6 and can be organized as you see fit. Do whatever you think makes a good UI. However, all specific tables **must** be organized as discussed in sections 3 and 4.

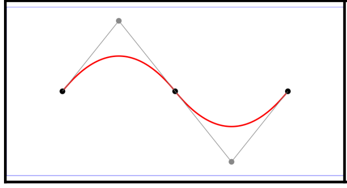
This GUI layout is a bit clunky, but it is simple, and it saves you from working on multiple windows/tabs in your very first Web app.

3. File Log Panel

This panel displays the list of **all** valid SVG files on the server, including all the files uploaded from client and the all files created from scratch by the client. The panel is scrollable vertically and, if necessary, horizontally. The panel contains a link to the downloadable SVG file, and a summary of that file's properties:

- Image thumbnail (clickable download). Scaled to be 200 pixels wide. Must maintain original aspect ratio.
- File name (clickable download)
- File size, rounded to the nearest KB
- Total number of rectangles
- Total number of circles
- Total number of paths
- Total number of groups

Data for these columns is can be easily obtained using A2 functions. Here is a sample row:

Image (click to download)	**File name (click to download)**	**File size**	**Number of rectangles**	**Number of circles**	**Number of paths**	**Number of groups**
	quad01.svg	1KB	1	5	2	3

If there are no files on the server, display the message "No files" in the File Log Panel.

If a particular SVG file is invalid, you cannot display its summary. In this case, do not display the file at all. Only valid files must be displayed in the File Log Panel. Fortunately, there is an A2 function that only loads valid SVG images...

Place an "Upload file" button at the bottom of the File Log Panel. Upload functionality is described below, in Section 6.

Display success/error messages into the browser's JavaScript console using the `console.log` method for debugging purposes.

4. SVG View panel

This panel shows components in the currently open file, one line per component, in the order that `createValidSVG()` returns them. This is intended to be a tabular-looking view with rows and columns (e.g. an HTML table), though how you implement it is up to you. Since there will often be many routes/tracks in a single file, this view must be scrollable, both horizontally and vertically.

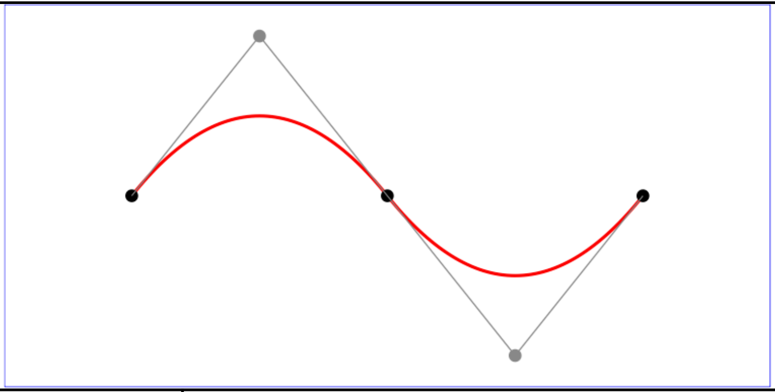
Above the panel, there is a **drop-down list of file names**. The list must contain only valid SVG files. It must include all the valid files currently stored on the server. This list is empty if there are no valid files on the server. When the user selects a file, run `createValidSVG()` on the file, then use the A2 Module 3 functions to get the summary of its data for this panel. If `createValidSVG()` returns an error, go no further (the SVG View panel is unchanged). Otherwise, fill the panel from the file's components.

The panel will display image properties:

- The image itself, scaled to be 800 pixels wide (maintain aspect ratio)
- Title
- Description
- Summaries of image components at the top level, i.e. immediate children of `svg` element. Element summaries are intended to show a few important properties at a glance. The column data for each element type is discussed below. Data for these columns is can be easily obtained using A1/A2 functions:
  - **Component:** rectangle, circle, path, or group. They must be displayed in order discussed on A1 Module 1 . For every component type, number them sequentially, starting from 1. Restart the numbering for each element type. Feel free to add a separate table sub-heading for each element type, if you think it looks better.
  - **Summary:** Names and values of the fields of the appropriate struct. A2 Module 3 functions provide the data there:
    - For numerical data, the A2 functions should have rounded them to 2 decimal points
    - Truncate path data after the first 64 characters, wrap as necessary to avoid very wide tables
    - For groups, display the number of elements this group contains (immediate children - you do not need to recurse into subgroups).
    - Include units where appropriate.

Here is a couple of examples (next page):

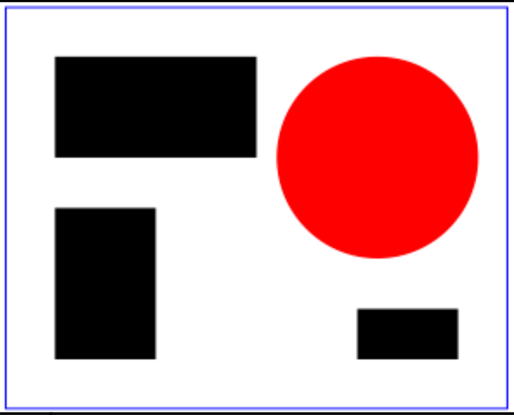
quad01.svg



Tile	Description
Example quad01 - quadratic Bézier commands in path data	Picture showing a "Q" a "T" command, along with annotations showing the control points and end points

Component	Summary	Other attributes
Path 1	path data = M200,300 L400,50 L600,300 L800,550 L1000,300	3
Group 1	2 child elements	1
Group 2	3 child elements	1
Group 3	2 child elements	1

`rect_circ.svg`



Tile	Description
	Rectangles and a circle

Component	Summary	Other attributes
Rectangle 1	Upper left corner: x = 0.5cm, y = 0.5cm Width: 2cm, Height: 1cm	0
Rectangle 2	Upper left corner: x = 0.5cm, y = 2cm Width: 1cm, Height: 1.5cm	0
Rectangle 3	Upper left corner: x = 0.5cm, y = 0.5cm Width: 1cm, Height: 0.5cm	0
Rectangle 4	Upper left corner: x = 0.1cm, y = 0.1cm Width: 4.98cm, Height: 3.98cm	3
Circle 1	Centre: x = 3.7cm, y = 1.5cm, radius = 1cm	1

Updating image properties

The user must be able to perform the following actions for every element. How you implement the UI for them is up to you.

Show attributes: The action is to display the additional properties for a specific component - e.g. fill, stroke, etc..

Edit attribute: Modify the value of an existing attribute, or add a new attribute. You can do this in the same UI you use to show attributes. You can also make the fields for the "required" attributes - e.g. rectangle width, or circle centre - editable right in the summary table, and provide the user with a way to submit these changes.

Edit title / description: The user should be able to edit tile and description. Enforce the length restrictions stated in `SVGParser.h`.

All of these must actually modify the underlying SVG file, so you get the input from the UI, send it to the server, and then use A1/A2 functionality to create an SVG struct from file, edit it, validate the result, and save it back to file. Most of this functionality was already implemented in A1 and A2, so you can put these functions together without too much additional code.

If any of the changes fail for any reason - e.g. image fails to validate - display an appropriate alert (and remember to add an error message to the console log). If the request to the server to update the file succeeds, update the contents to the File Log Panel entry for that file. The changes - e.g. new title, or the new fill colour for a rectangle - should be reflected in the image itself and its summary.

Display success/error messages into the browser's JavaScript console using the `console.log` method for debugging purposes.

5. Working with SVG data and SVG files

When your web client starts, it parses all `.svg` files uploaded to the server, displays their summaries in File Log Panel, and adds them to the list of files in SVG View Panel, and all the drop-down lists of server file names. If there are no `.svg` files on the server, the File Log Panel must display an appropriate message (See Section 3).

For all functionality below, display success/error messages into the browser's JavaScript console using the `console.log` method for debugging purposes.

Your Web client GUI must have the following interactive functionality:

- **Upload an SVG file**: Obtain a filename (see details below) and upload the `.svg` file to the server. If the server request for the upload is unsuccessful - i.e. server returns some sort of error - then go no further. The File Log Panel would remain unchanged. Display the error in an alert.

  If the server request is successful - i.e. server returns some sort of OK message - add a row to the File Log Panel using the file's components obtained from the server. In addition, add the file name to all the drop-down lists of server file names - i.e. in SVG View Panel, and the Create SVG module (see below). If the file you are trying to upload already exists on the server, don't overwrite it, and display an appropriate message in the an alert. Also, you are welcome to validate files on upload, and prevent the user from uploading invalid files.

  Obtaining a filename*:* you must open a file browser for the user to select a filename, then submit it. Use HTML forms for this. If an input file does not exist, display the error in an alert.

- **Download a file:** This is done by clicking on the image thumbnail or file name in the File Log Panel (See Section 3).

- **Create SVG**: Creates a SVG struct, which is saved to a .svg file. A2 bonus functions come in handy here. The newly created SVG struct is saved to a file on the server using `writeSVG()`, after validating is using `validateSVG().` User must provide the file name, and you can verify that the file name is unique. The user must not be able to create invalid SVG files.

  Depending on what happens on the server, the file may or may not be saved. If the server response indicates an error, file was not saved. Display the error message in an alert. If the server response indicates success, file was saved. If the file was saved, update the contents to the File Log Panel to include the summary of the new file, and add the file name to the drop-down list in SVG View Panel, and all the other drop-down lists of file names.

  The user input must be entered using forms, and there must be a “Create SVG” button.

  A SVG image without at least one visible element is valid, but it is not very useful. As a result, you will need to populate newly created empty SVG images with some simple elements, or add elements to any to any other existing SVG file.

- **Add shape**: add a shape - a rectangle or a circle - to one of the files currently uploaded to the server. Again, A2 bonus functions come in handy here. File name must be selectable with a **drop-down list** - do not force the user to type in the file name. The list must contain only valid `.svg` files. The user be able to select which shape they are trying to create.

  You will need to let the user set the required attributes of the shape, as well as its units. How you set this up is up to you. To make things more fun, you might want to let the user set the fill of the shape on creation.

  If the request to the server to insert a shape into a file succeeds, update the contents to the File Log Panel entry for that file. The change to the image must also be visible in the thumbnail. If the request to the server fails, display an appropriate error message in an alert.

The user input must be entered using forms, and there must be an “Add shape” button. You can have separate buttons for creating rectangles and circles.

- **Scale shapes**: you need to allow the user to scale all rectangles or circles in the image by the same factor. The user must be able to specify the scale factor. The scaling must be uniform along x and y dimensions. For example, you can let the user scale all rectangles by a factor of 2, or circles by a factor of 4.

  The user input must be entered using forms, and there must be an “Scale shape” button. You can have separate buttons for scaling rectangles and circles.

Bonus (3%)

For bonus marks, you can also allow the user to scale the entire image, instead of just specific shapes. This way, the user can double the whole image in size. For this scaling, all the relative locations of elements and their relative sizes must be preserved.

Scaling paths is not easy, so for the sake of simplicity, the bonus functionality only needs to work on images consisting of only rectangles, circles, and groups. Keep in mind that the bonus marks will be given only if the required basic scaling works perfectly. No partial marks will be given for the bonus.

7. Defensive programming

Validate user input, and explicitly report errors to the user using alert message. If your GUI silently accepts invalid user input or fails for any reason, you will lose marks. If you allow invalid input and create invalid files - e.g. files with an extension other than “`.svg`” or files that violate SVG the specification - you will also lose marks.