Group 8

Undergraduate Project

Software Design Document

Author: Ben Miloshoff

Group Members: Ben Miloshoff, Arjun Agrawal, Joe Cullinan, Geonhyuk Im, Brett Malmquist, Ekenechukwu Nwannunu, Adam Richard, Christian Schubert

Date: (04/09/2021)

TABLE OF CONTENTS

1. Introduction

1.1 Purpose

This software design document describes the architecture and system design of the Omnibus Visualization System. It is intended to be viewed and understood by the software development team and the customer. This document is segmented into two parts. The first is a preliminary design in which the overall system architecture and data architecture is defined. The second is the detailed design stage. This is where more detailed data structures are defined and algorithms are developed for the defined architecture.

1.2 Scope

This product is a web-based data visualization system. It generates various diagrams and charts out of user specified JSON data. This system contains files written in HTML, CSS, and JavaScript. It also uses the JavaScript libraries Webix and Plotly for easier CSS formatting and graphing. This software is modular and can be expanded as needed.

1.3 Definitions, Acronyms, & Abbreviations

Loose coupling: Organized to minimal dependence between elements.

High cohesion: Organized so like-minded elements are grouped together.

2. References

Webix Documentation: https://docs.webix.com/

Plotly Documentation: https://plotly.com/javascript/

3. Decomposition Description

3.1 Module Decomposition

3.1.1 Module 1 Description

This system is decomposed into seven parts, all being separate HTML files. One of which is the MainMenu.html file. This file connects all of the others together through a series of buttons. There are six buttons and each one leads to a page that can display the specified diagram.There are six HTML files that each can calculate and display a different graph given JSON. Depending on which diagram the user is trying to create,

they can navigate to the necessary page. On the page there is a box that the user can input the JSON and then click the “Draw Diagram” button to display the diagram.

3.1.2 Module 2 Description

Each HTML file also contains a webix object. This object is responsible for the page’s layout and CSS formatting. It is a javascript object that takes parameters and can transform those parameters into CSS when the page is rendered. The object is required to contain a form for user input, buttons for page navigation, and a div that can contain the diagram that is created.

3.2 Concurrent Process Decomposition

This system does not have concurrent processes but does have two processes that run sequentially.

3.2.1 Process 1 Description

There are two buttons on each diagram page. One of which is the “Draw Diagram” button which is connected to the drawGraph() function. The other is the “Main Menu” button which returns the user to the main menu.

3.2.2 Process 2 Description

When the drawGraph() function is clicked, it will pull the string from the textarea and try to parse it to JSON. If it succeeds, then the diagram will be created using the values in the JSON and sent to the div in the webix code.

3.3 Data Decomposition

When the user hits the “Draw Diagram” button, the JSON in the text field is sent as a string to the JavaScript function drawGraph(). This function parses the string into JSON that can be further manipulated. The JSON values are then used as input for the trace variables. These trace variables represent individual data points to be plotted. Once all the traces are created, they are placed together into a list called data. The data points are finally plotted with a call to Plotly.newPlot() which takes data as a parameter. As of now, this system is purely front end so the data is not stored permanently but can be easily connected to backend databases at a later time.

3.3.1 Data Entry 1 Description

Data dictionary for drawGraph() function.

Entity Name	Type	Parameters	Description
data	array[object]	NONE	An array containing the traces for the graph.
drawGraph	function	NONE	A function called that creates the graphs.
e	exception	NONE	Exception that is thrown and caught when JSON is invalid.
getElementById	function	string	Gets the HTML element with the same ID as the parameter.
input	string	NONE	Gets the JSON string that the user inputs.
json	object	NONE	The JSON object returned from the parse function.
layout	object	NONE	The object that contains all of the layout options for the newPlot function.
newPlot	function	string, object, object	The function that creates the diagram from its parameters.
parse	function	string	The function that parses the user's input string to a JSON object.
trace	object	NONE	The object that contains the information to build the diagram.

3.3.2 Data Entry 2 Description

Once the data is processed by the drawGraph() function, it is then inserted into the div html element embedded in the Webix code. This div is formatted so it can contain both the graph and an exception message if the JSON was not parsed correctly. It also has CSS styling to output the exception message in red, so it is more clearly visible to the user.

4. Dependency Description

4.1 Inter-module Dependencies

This system is partitioned into seven parts, each being their own HTML file. There is a file for the main menu, and one file for each of the six diagrams. Each HTML file is connected to the main menu by a button and the main menu is the only place the user can see a button for each diagram. This allows the relationship between each component loosely coupled. Given that the main menu is the only page that can access all the other pages, the system can be easily expanded.

4.2 Inter-process Dependencies

The drawGraph() function is dependent on the user input text box as well as the div in the webix code where the diagram Is outputted. The function pulls the input string directly from the textarea in the form when the “Draw Diagram” button is clicked. The drawGraph() function must check for errors in JSON formatting as well as any edge cases such as an empty input.

4.3 Data Dependencies

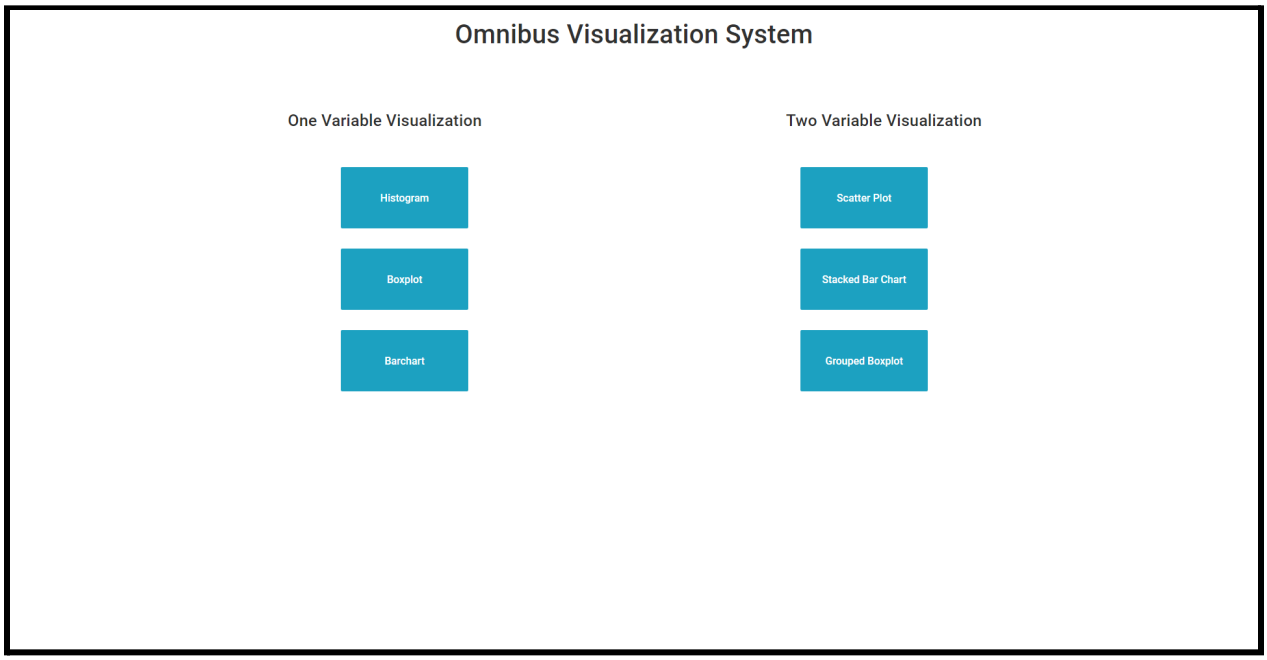
The JSON entered in the form’s textarea must have the correct format for the diagram it is supposed to draw. It is up to the user to know how to correctly format the JSON so the diagram is created successfully. If the JSON is incorrectly formatted, an exception will be thrown and caught by the drawGraph() function and the exception message will be displayed where the diagram is normally outputted.

5. Interface Description

5.1 Module Interface

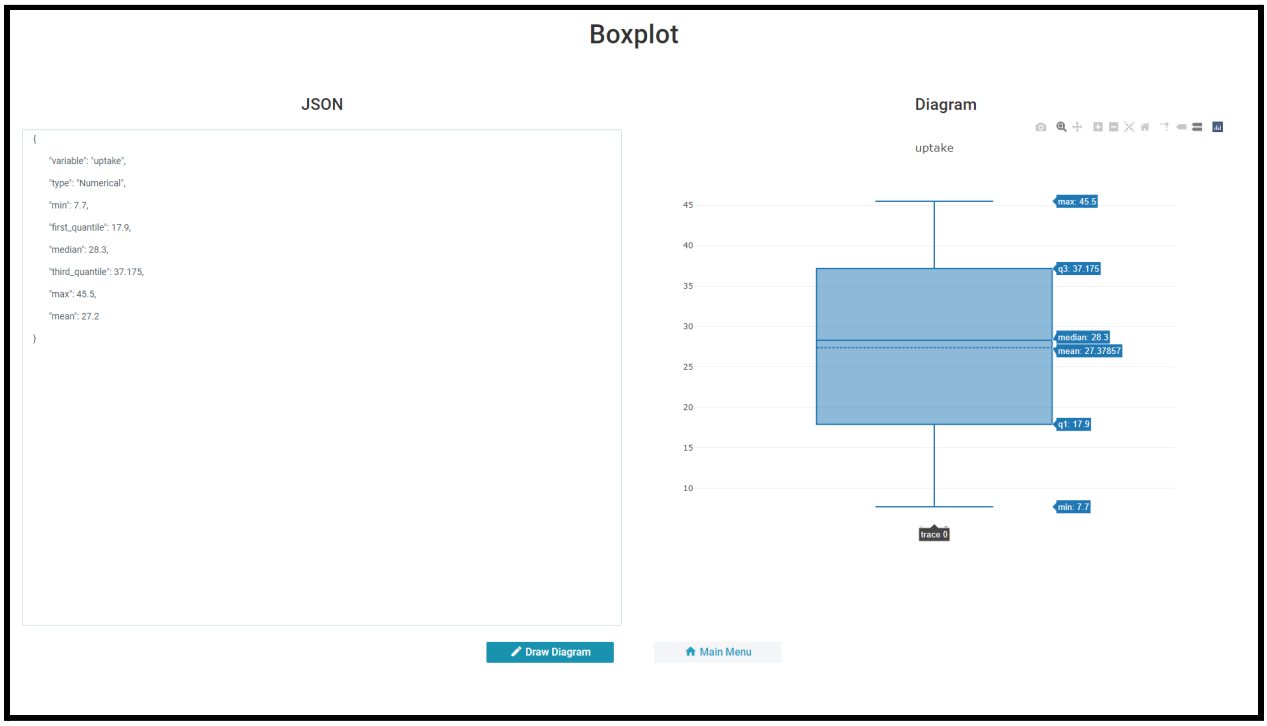
5.1.1 Module 1 Description

The user will start from the main menu page and can navigate to any of the six pages to draw their diagram. Once the user clicks the button for the diagram they want to create, the system will take them to the page. On each diagram's page, there will be a box to input JSON, an empty space to display the graph, a button to draw the diagram, and a button to return to the main menu.



5.1.2 Module 2 Description

When the user inputs the JSON into the text box, they can then click the "Draw Diagram" button to submit the JSON to draw the diagram. If the JSON formatting was not sufficient to draw the diagram, an error message will be printed in place of the diagram. The user can then re-enter the JSON and try again. Once the diagram is displayed correctly, the user can manipulate the diagram to their liking. There are buttons above the diagram that allows the user to download, zoom, pan, select, and scale the diagram. After the user is finished, they can click the main menu button and select a different diagram.



5.2 Process Interface

5.2.1 Process 1 Description

The buttons shown below are both connected to javascript functions. The “Draw Diagram” button calls the drawGraph() function on a click event. The “Main Menu” button calls an anonymous function that returns the user to the main menu.

5.2.2 Process 2 Description

The screenshot below shows the box where the user can input the JSON string. It also shows where the diagram will be displayed after the “Draw Diagram” button is clicked.

6. Detailed Design

6.1 Module Detailed Design

6.1.1 Module 1 Detail

Pseudocode for webix layout in main menu.

```
webix.ui({
      rows:
      [
          { // Page Title },
          { // Button columns label
              cols:
              [
                  {
                     // One variable column
                  },
                  {
                     // Two variable column
                  }
              ]
          },
          { // First row of buttons
              cols:
```

```
            [
                 { // First button
                    href: “FileName.html”
                    click:function() {window.open(“FileName.html”);}
                 },
                 { // Second button
                    click:function() {window.open(“FileName.html”);}
                 }
            ]
        }
    ]
 });
```

6.1.2 Module 2 Detail

Pseudocode for webix layout in diagram pages.

```
webix.ui({
      rows:
      [
          { // Page header},
          { // JSON and Chart type label },
          { // JSON input box and chart row
              cols:
              [
                  { // JSON input box
                      elements:
                      [{view: "textarea", id: "jsonText"}]
                  },
                  { // Chart div
                      <div id='chart'></div>
                  }
              ]
          },
          { // Button row
              [
                  { // Draw Diagram Button
                      click: drawGraph // Call function when button is clicked
                  },
                  { // Main Menu Button
                      click:function() {window.open(“MainMenu.html”);}
                  },
              ]
          }
      ]
});
```

6.2 Data Detailed Design

6.2.1 Data Entity 1 Detail

Pseudocode for the drawGraph() function.

```
drawGraph() {
        var input = text from user input box
        try {
                var json = parse input string to JSON
                var trace  = { JSON values from user input }
                var data = [combined traces if more than one]
                var layout =  { give additional diagram options }
                newPlot('Div ID', data, layout) // Build and return diagram
        }
        catch (e)
        {
                print exception message
        }
}
```

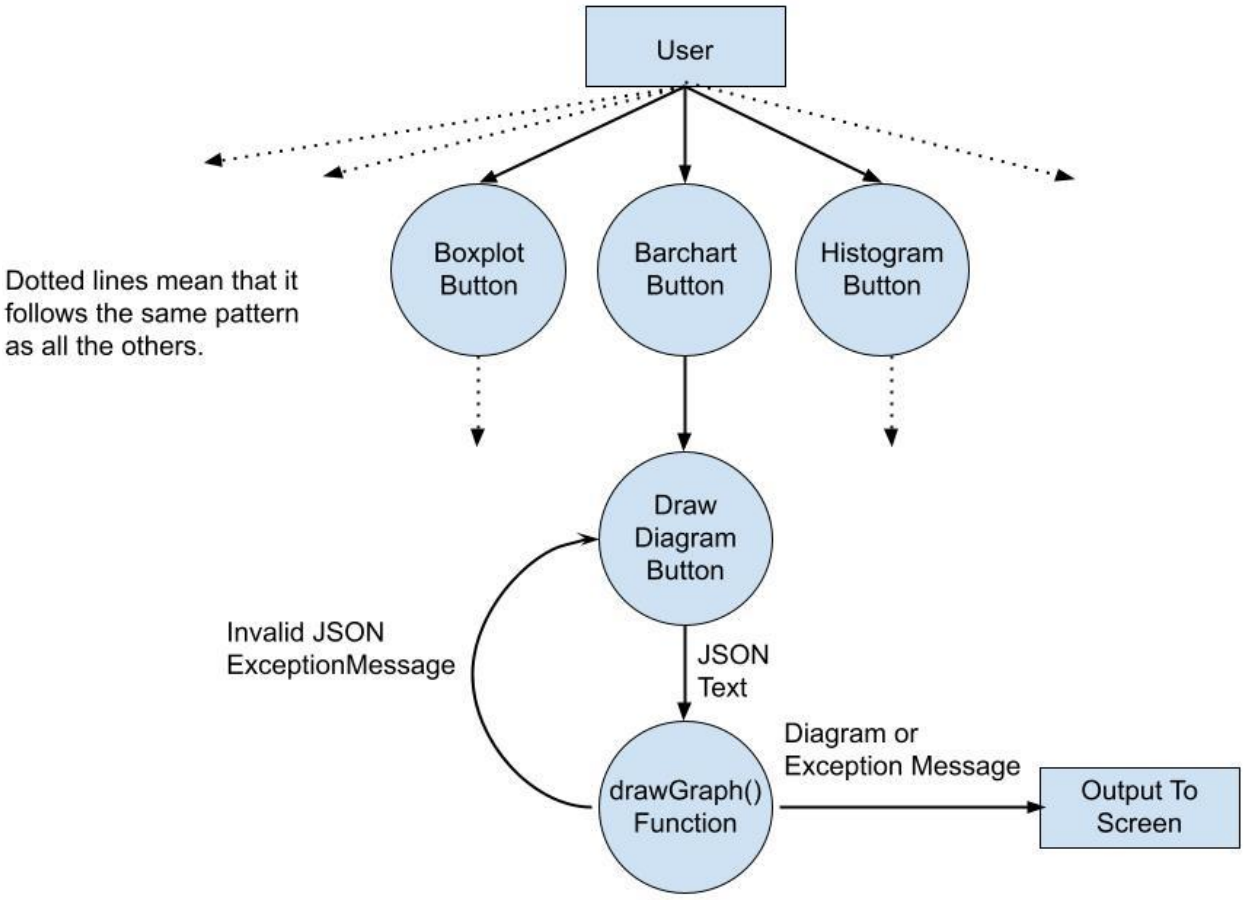
6.2.2 Data Entity 2 Detail

Pseudocode for the div that the diagram is inserted into.

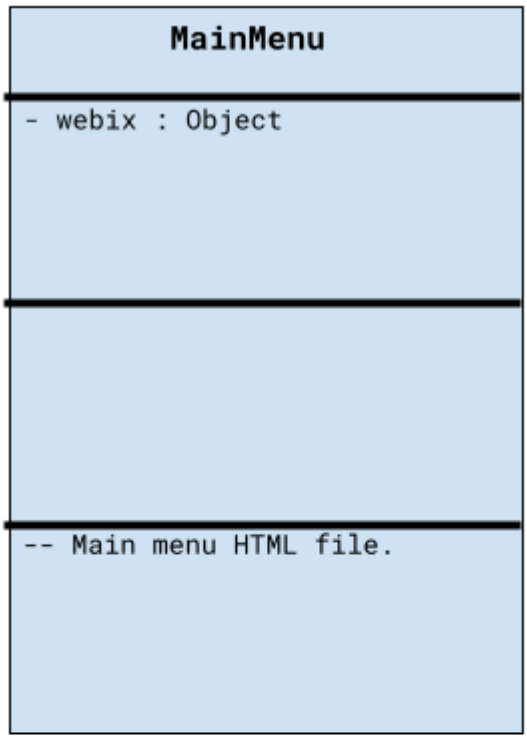
```
webix.ui({
     { // Chart div
       view: "template",
       template: "<div id='chart' style='width:900px;height:700px;color:red;'></div>"
     }
});
```

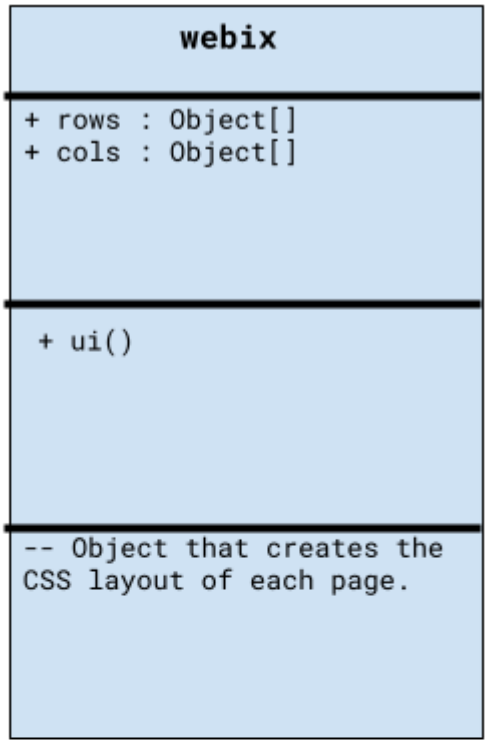
7. Appendix

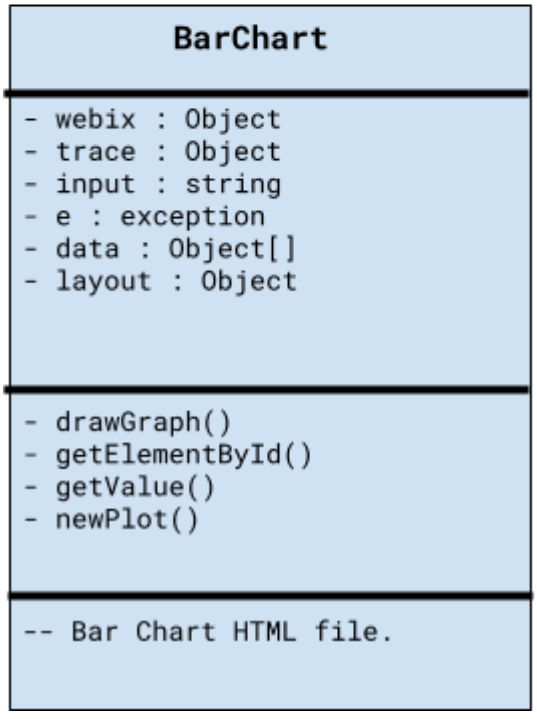
Data Flow Diagram:



Class Diagram:







All other HTML files have the same class diagram as the BarChart