|  |
| --- |
| VISUALIZATION OF THE COMPOSITION OF  CLOUD COMPUTING SERVICES |
| A Thesis  Presented to  The Faculty of the Department of Computer Science  California State University, Los Angeles |
| In Partial Fulfillment  of the Requirements for the Degree  Master of Science  in  Computer Science |
| By  Davis Louie  December 2018 |

© 2018

Davis Louie

ALL RIGHTS RESERVED

The thesis of Davis Louie is approved.

Jiang Guo, Committee Chair

Zilong Ye

Raj Pamula, Department Chair

California State University, Los Angeles

December 2018

# ABSTRACT

Visualization of the Composition of Cloud Computing Services

By

Davis Louie

Visualization of the Composition of Cloud Computing Services also known as VTCCS is a web-based application that provides a way for users to create composite web services out of existing web services. In modern technology, web services are quite common, and many companies use cloud computing web services which are services provided remotely. Those web services are often used together with many others that it would require some programming and effort to make them work. VTCCCS simplifies the process and allows users not only to generate composite web services but also to monitor the performance of the composite web service. VTCCCS lets users benchmark each individual web service and the overall composition.

# ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Jiang Guo for his assistance and guidance on the subject matter. His help has been very valuable for the direction of this project.

Great thanks to the former NASA Direct-Stem team who are: Bill Pham, Andrew McLees, and Patrick Flinner. If it weren’t for them, this project wouldn’t be created and ready.

I would also like to thank the senior design that graduated last spring who aided and assisted me in creating this web application: Zolangi Ramirez, Gonzalo Serrano, Hongsuk Choi, Sudip Baral, and Johnson Truong.

Additional thanks to Dr. Jiang Guo’s current NASA Direct-Stem team who consist of Isaac Cano, David-senpai-San, Ricky Vargas, Adekola Togunloju, and more. I would also acknowledge the assistance of the past Direct Stem graduate students, Shay Nguyen and Adam Berman. Their advice was very helpful.

TABLE OF CONTENTS

Abstract iv

Acknowledgments v

List of Tables viii

List of Figures ix

List of Terms x

Chapter

1. Introduction 1

2. VTCCCS Technical Background and Framework 3

2.1 VTCCCS as a Web Application 3

2.2 Overview of Angular 4

2.2.1 Angular Modules 4

2.2.2 Angular Components 5

2.2.3 Angular Services 6

2.3 HTML Used in Angular 8

2.4 CSS Used in Angular 9

2.5 Jersey Framework and JAX-RS 10

2.6 MySQL 14

3. VTCCCS Design and Structure 16

3.1 Presentation Layer 16

3.1.1 Admin Component 17

3.1.2 Debug Component 18

3.1.3 Demo-app Component 19

3.1.4 Home Component 19

3.1.5 Main-nav Component 20

3.1.6 Projects Component 21

3.1.7 Shared Component 21

3.1.8 Split-panel-login Component 22

3.1.9 Webservice-config-menu Component 22

3.2 Application Layer 23

3.3 Data Layer 24

4. Installation Guide 26

4.1 Node.js and NPM 26

4.2 Angular CLI 27

4.3 Git 28

4.4 Java Development Kit 28

4.5 Installing MySQL 29

4.6 Apache Maven 29

4.7 Project Deployment 31

5. VTCCCS User Manual 33

5.1 Register/Login 33

5.2 Project Creation 34

5.3 Real-Time Performance 41

5.4 Visualization 42

6. Conclusion 44

References 45

# LIST OF FIGURES

Figure

1. A Portion of Code from the Root Module Called App.module.ts 5

2. Example Code for Component Metadata from Main-nav.component.ts 6

3. Small Snippet of the User.service.ts 7

4. Provider Snippet of Code from App.module.ts 7

5. Example HTML code from Bar-chart.component.html 8

6. CSS Code Example from main-nav.component.html 9

7. Example of RESTful Web Services 11

8. Sample Code of Pom.xml 12

9. Sample Snippet of Web.xml 13

10. The Response from Getting all the Projects 13

11. Sample Code of SQL from Schema.sql 14

12. VTCCCS Multitier Architecture 16

13. Angular Architecture for VTCCCS 17

14. Web Page of the Administrator Component 18

15. Debug Component Web Page 18

16. Web Page of the Demo-app Component 19

17. Home Component Web Page 20

18. The Main-nav Component Shown on the Website 20

19. Showing the List of Projects in the Project Component 21

20. The Modal Window for the Split-panel-login Component 22

21. The Web Service Builder 23

22. EER Diagram of the Database Directstem 25

23. Sample Output of a Successful Installation of Node.js and NPM 27

24. Output of Checking if the Angular CLI is installed on 27

25. Modifying the Environment Variables in Windows 10 29

26. Adding the Environment Variables for Maven 30

27. Showing the Path Variable for Maven Directory 30

28. Displaying the Sample Output of Mvn -version that is Installed 31

29. Location of Project Download 31

30. Example Output after Npm Install Command 32

31. Registering a New Account 33

32. Logging into VTCCCS 34

33. Successful Login 34

34. Project Details Editor 35

35. Menu Options 35

36. Selected Node Options 36

37. Example of Input, Regular, and Output Node Interconnected 37

38. Performance Chart of a Web Service 37

39. Example of Linking a Remote API to the Node 39

40. Example of a Successful Response from a Web API 40

41. The Real-Time Performance Comparison Monitoring 5 Websites 41

42. Radial View of a Project 42

43. Tree View of a Project 43

LIST OF TERMS

VTCCCS Visualization of the Composition of Cloud Computing Services

HTML Hypertext Markup Language

CSS Cascading Style Sheets

CLI Command Line Interface

DI Dependency Injection

IDE Integrated Development Environment

SQL Structured Query Language

SPA Single-Page Application

NPM Node Package Manager

OS Operating System

API Application Program Interface

JAX-RS Java API for RESTful Web Services

REST Representational State Transfer

URI Uniform Resource Identifier

CRUD Create, Read, Update, and Delete

XML Extensible Markup Language

JSON JavaScript Object Notation

HTTP HyperText Transfer Protocol

URL Uniform Resource Locator

EER Enhanced Entity-Relationship

# CHAPTER 1

# Introduction

VTCCS which is an acronym for Visualization of the Composition of Cloud Computing Services, is a web application created to construct complex composition of web services that would normally be hard to visualize. Our web application creates a nice model for the user to create and link web services to form the composition of their choice. Programmers come across these challenges when they work with multiple web services and each service requires an output from another web service to use as an input. It is a tedious process in which these services need to run in a certain order and in parallel if possible. An example of this sequence is akin to cooking. The cook needs a certain amount of ingredients in and they all need to be prepared at different times. Once the cook gathers everything up and put them together, the cook retrieves the result which is the finished dish which would be the web service composition.

Now what is a web service? Many people get confused and believe surfing the web is a web service because the internet is offering a service which is displaying information onto their phone or web browser, but that is not a web service. It is an electronic service offered by one device to another device. If humans were cyborgs or part machine, that might be considered a web service, but for now it is only services that electronic devices can offer each other. The most common web services are Web APIs.

Another feature our application has is the ability to monitor these web services. Our web application monitors the run time of each cloud computing web service and records them. Our users will be able to determine which web services have the fastest run time speeds. Users might ask why is run time important. Take in account that run time makes a huge difference in the overall composition of the web service. If people have one web service that takes up too much time completing its task, that means another web service that needs the output of the slow web service must wait for the full run time for its service to even start. This is what we call a bottleneck where one web service holds up the whole composition from being completed. Fortunately, users can identify the bottle neck with our web application. Without that feature, users would have to run each web service one by one to see where the bottle neck lies and with higher order compositions, that would be very time consuming.

An augment to the previous monitoring feature is the ability to monitor the web service in real time. It is very common for a web service to be bogged down by many requests from multiple users across the globe. Users need to know when the web service will receive heavy traffic. VTCCCS can monitor and keep track of a web service’s run time speed throughout the day by constantly sending a request. That way, users will be able to determine which hours of the day are best for using a web service and can also be used as a comparison between similar web services that do the same things.

CHAPTER 2

VTCCCS Technical Background and Framework

Since VTCCCS requires real-time execution of web services and the ability to create composite web services, it is important to choose the right technologies and frameworks that allows us to accomplish our objective and give users a friendly experience. With that in mind, VTCCCS is implemented to be a web application.

The VTCCCS web application uses HTML, CSS, Angular 4, Node.js, Angular Command Line Interface also known and Angular CLI, D3.js, and Chart.js for the front end. The back end uses Java, Maven, and MySQL for the back end. Angular uses typescript which is a superset of JavaScript. The project was built by using Maven for the back end and a combination of the Angular CLI with Node.js for the front end. The Integrated Development Environments or IDEs used were IntelliJ, Visual Studio Code, and the MySQL workbench for the SQL scripts.

**2.1 VTCCCS as a Web Application**

A web application or web app is a client-server computer program which the client runs in a web browser [1]. VTCCCS requires using a modern browser such as Google Chrome, Firefox Mozilla, Microsoft Edge, or Apple Safari to be able to access and execute this application.

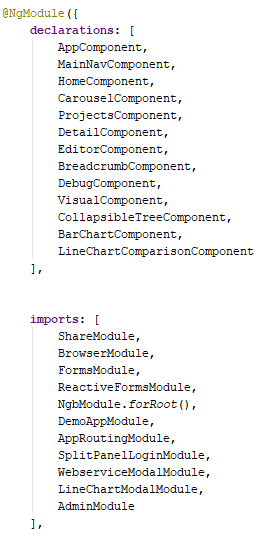
The reason why VTCCCS is a web application is because of accessibility. Users only need a web browser either on their smart phones or their computers to be able to use it which means the barriers of entry is low. Of course, users need internet access in order to utilize a web browser, but that is a trivially thing in this modern age. There are no other requirements.

**2.2 Overview of Angular**

Angular is a TypeScript-based open-source front-end web application platform [2]. Angular is cross platform and has great speed and performance. Angular uses a hierarchy of components and subcomponents for its architecture. The reason why angular is so fast is because it is a single-page application or SPA. Instead of rewriting the page to load a new one, SPAs dynamically change or swap components which requires less processing power and without reloading any redundant data like the header or footer. Angular runs only on the client side so it is responsible for loading the HTML, CSS, and typescript.

**2.2.1 Angular Modules**

Angular web applications are modular, and they are built from individual modules called NgModules. NgModules are like containers that hold a chunk of code that has completes a job or is closely related to the job. They have components, service providers, and other necessary code such as models. Every Angular application has a root module and that root may have child modules that stem from it. As you can see from Figure 1 below that this root module uses 13 components and it has imported 11 modules. The reason for the huge imports is because the components need the classes in order to function.



*Figure 1.* A Portion of Code from the Root Module Called App.module.ts.

**2.2.2 Angular Components**

The way Angular creates a component is first creating the metadata is using the @Component decorator in the typescript file. An example of this can be seen below.

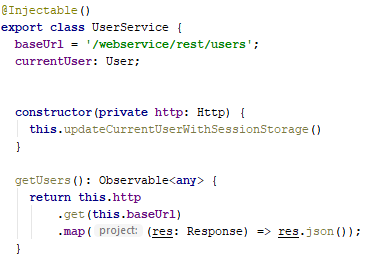


*Figure 2*. Example Code for Component Metadata from Main-nav.component.ts.

The angular metadata tells Angular the location of the building blocks of the component are in the file structure. Figure 2 shows that the blocks are referenced using relative paths and that the HTML and CSS files are in the same directory by templateUrl for HTML and styleUrls for CSS. Selector is the CSS selector that identifies this directive in a template and triggers instantiation of the directive [3]. The component metadata also can register a service for the component to use. A service is generally the way components pass data from one to the other.

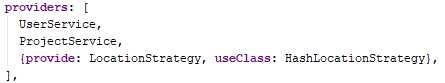
**2.2.3 Angular Services**

Angular Services cover a broad category of what it provides for the web application. It functions as a helper that completes a specific task such as fetching data from the server or validating the input forms. Components can delegate the tasks to the services. Some people may ask why should we even use services? It is mostly to reduce redundant code. If multiple components need to retrieve information from the same database, it is more efficient to create a service rather than type out the same code repeatedly for each component. An example of a service can be seen in Figure 3.



*Figure 3.* Small Snippet of the User.service.ts.

The @Injectable( ) indicates DI or dependency injection. DI is in the Angular framework and it is used everywhere. It basically allows you to inject the service to any of the components that need that service. Also, Angular needs to register a provider for that service in the module. To look at how to register providers, the figure below demonstrates this.



*Figure 4.* Provider Snippet of Code from App.module.ts.

In Figure 4, two service providers have been registered, UserService and ProjectService. Whenever a component needs the UserService, the provider will tell the injector how to create the dependency. An injector creates dependencies on services, so the component can use them whenever they need to. A component may need both the UserService and the ProjectService so the provide tells the injector how to create them and then the injector adds both UserService dependency and the ProjectService dependency. Now the component can use the services it needs.

**2.3 HTML Used in Angular**

HTML is the standard markup language for creating web pages and web applications [4]. HTML uses elements which are building blocks for web pages such as head, body, p for paragraph, img for image, and many more. HTML is the source or text of the web page. Angular uses HTML to create their component or subcomponent. If the component needs text rendered, then HTML will contain all the text and will be solely responsible for that portion of the web page. Angular stores the text in an external HTML file in which angular can access it when use that component. A sample of HTML code is shown in Figure 2.

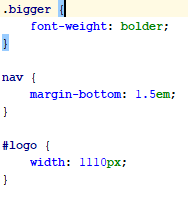


*Figure 5*. Example HTML code from Bar-chart.component.html

In the figure, the table element or tag creates a table for the website. Since there are 2 tr tags which stands for table row, there will be 2 rows. Counting the td which is table data or th which is table header, there will be 3 columns in this table. Within these tags or elements are text that will be placed in the table depending on their position. The first header will have the text Node. After that, it will be Color and so on.

**2.4 CSS Used in Angular**

CSS is the styling of the web page. That means CSS is responsible for the font, text size, alignment, color, etc. CSS is a part of the whole angular component and like HTML, the CSS is kept in an external stylesheet. CSS is like a set of instructions while the text is being rendered telling the computer to make that text color red or make that font to Courier. CSS may not seem important, but it has its benefits. Styling the web page in a more presentation manner would make the web page easier to read. For example, if the web page is single spaced and 50 pages long with size 9 font, that would be too difficult to read. A Simple example can be seen below.



*Figure 6*. CSS Code Example from main-nav.component.html.

From the figure, you can see the CSS selectors which selects the portion of the HTML text to be stylized. The first selector is .bigger, which is a class selector. Basically, it will affect all the HTML tags that has class=”bigger” in them and will make the text bolder. The second selector is directly selecting the HTML element which is the nav element. It is making the bottom margin of the nav 1.5 em. The em unit is the height of the font in nominal points or inches [5]. In simple terms, it will add the bottom margin relative to the font size. The last selector is like the first one. It is an id selector. If the HTML element has id=”logo”, then it will have a width of 1110px. The px is the shorthand for the word pixel.

**2.5 Jersey Framework and JAX-RS**

The Jersey RESTful Web Services framework is an open source framework that provides support of the development of RESTful APIs and can serve as the standard JAX-RS API, but extra features. In order to understand this much more, REST needs to be understood. REST is a software architectural style that defines a set of constraints to be used for creating web services [6]. Now how REST works is that you can make requests to the web service’s URI and the resource will return a response in either HTML, XML, JSON, or something else. The most common operations are GET, POST, PUT, and DELETE, which are CRUD HTTP methods. Some of the methods can be seen below in the figure.



*Figure 7.* Example of RESTful Web Services.

In the figure, three operations are being used: GET, POST, and DELETE. Now the @Path annotation above the class declaration helps us name the URI, so if some application needs access to this resource, it would use /webservice/rest/projects. Now some may wonder where webservice and rest come from since project is only stated in the path. The 2 figures below help create the base URI. In the first figure, the pom.xml states the first URI which is /webservice. Since Jersey is built with maven using the pom.xml as a blueprint, /webservice URI is the base. The next figure is the web.xml that states the next URI which is /rest/\*. That means the next URI is attached to the base while tomcat renders the REST API.

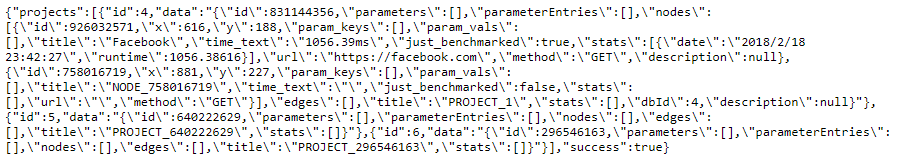


*Figure 8.* Sample Code of Pom.xml.



*Figure 9.* Sample Snippet of Web.xml.

Going back to the next lines of code in the Figure is the getProjects() function. Now this function fetches all the projects in the database and returns that as the response. The @Produces allows the API to return the response in a JSON format. When the requested URI is called, the response the server will send back will look like this in the figure below.

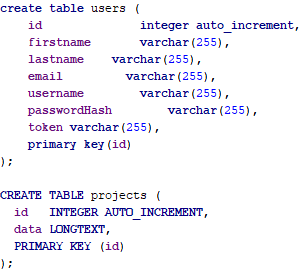


*Figure 10.* The Response from Getting all the Projects.

The next function is getProjectById(). This function is slightly different than the first function because it has a @Path notation. That means in order to use this function, the URI would be localhost:8080/webservice/rest/projects/{id} where {id} is a number. It basically adds on to the base URI and allows the application to search for a specific project with that id. Now the last two functions allow the web application to add or delete data to the database. The delete URI needs an id to indicate which project needs to be deleted and the add URI needs the /add at the end to indicate that the application is adding a new project.

**2.6 MySQL**

MySQL is a relational database management system that uses SQL as the language. Using MySQL may be a daunting task at first, but it is not too difficult. First, a database needs to be created. In order to create a database, the command create database database\_name should be used. After the database is created, the command use database database\_name has to send right after in order to select that database to use it.



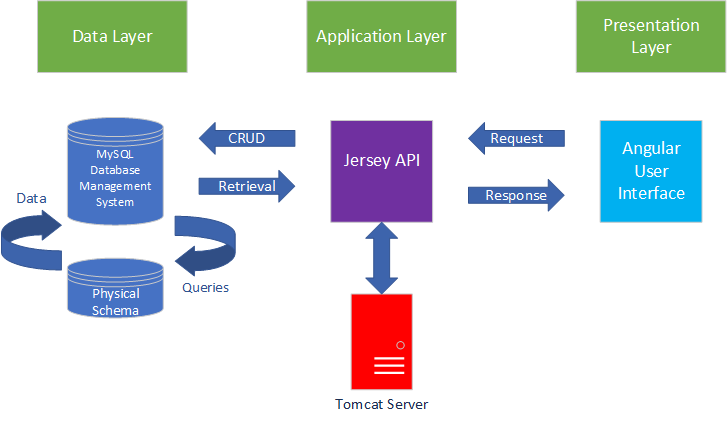
*Figure 11.* Sample Code of SQL from Schema.sql.

From the sample code, there are 2 tables being created. The first one is the users table with attributes of id, firstname, lastname, email, username, passwordHash, and token. Generally, the query follows the same command to create a table. The query starts with create table and then the table’s name. Parenthesis is added and then the list of attributes along with the datatype is added and separated with commas. Tables are essential for storing data in MySQL. As the figure shows, most of the attributes of the user name is varchar. Varchar means variable character which means a character length of many different sizes. The number 255 is the maximum number of characters that attribute can hold. This is necessary to control how much data can be stored in the database.

CHAPTER 3

VTCCCS Design and Structure

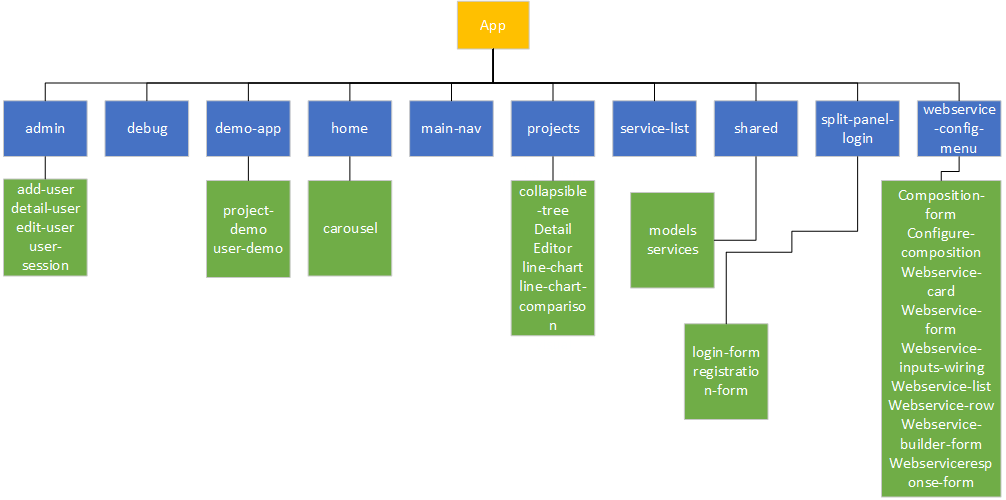
VTCCCS is made up of three layers: presentation, application or logic, and data. The presentation layer is the top level of the application. This is what users see in our application. They would login, create projects, build composite web services, and collect speed data in this layer. The application or logic layer controls functionality by performing processing tasks. An example of this would be the REST API in this project where it controls what projects are displayed and which users have access to specific information. The data layer includes all the data persistence mechanisms such as database servers. This would be the MySQL database system this project has set up.



*Figure 12.* VTCCCS Multitier Architecture.

**3.1 Presentation Layer**

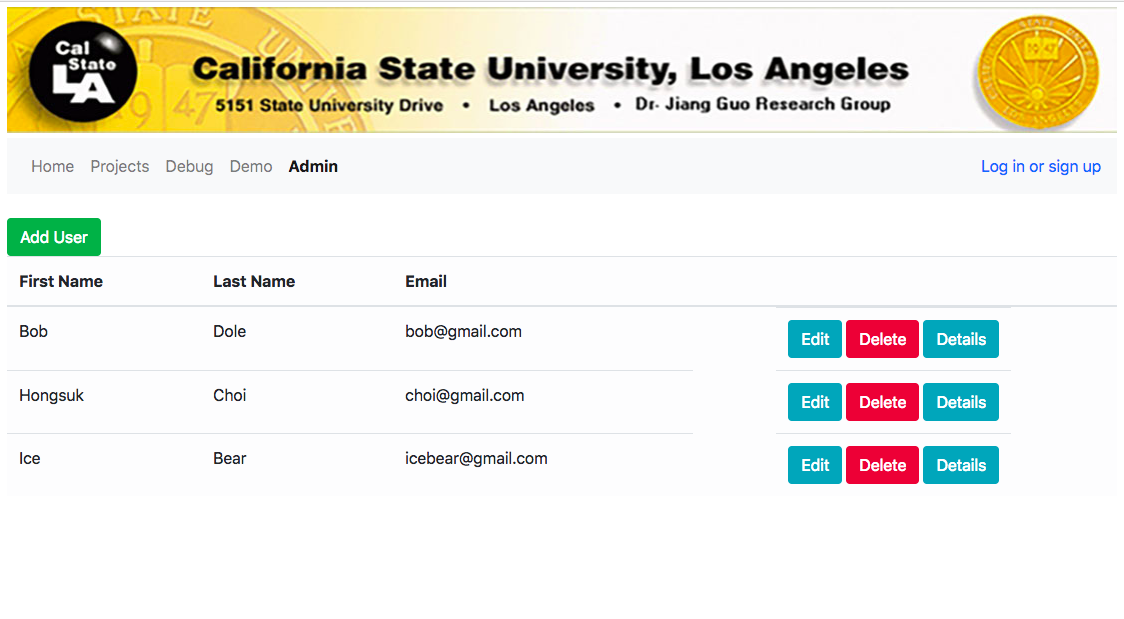
The presentation layer is what the user can see and interact with in the interface. Now this is all created with the Angular framework. There are many components in VTCCCS that make up the user interface and many subcomponents that make up the component. In the figure below, VTCCCS application is made up of 10 components currently. They are color-coded as blue. Those are: admin, debug, demo-app, home, main-nav, projects, service-list, shared, split-panel-login, and webservice-config-menu. Each sub-component is color-coded as green.



*Figure 13.* Angular Architecture for VTCCCS.

**3.1.1 Admin Component**

The admin component is responsible for managing the users that are creating projects and using the resources provided. Only an administrator can access this page and currently can add and delete users as well as modifying their information.



*Figure 14.* Web Page of the Administrator Component.

**3.1.2 Debug Component**

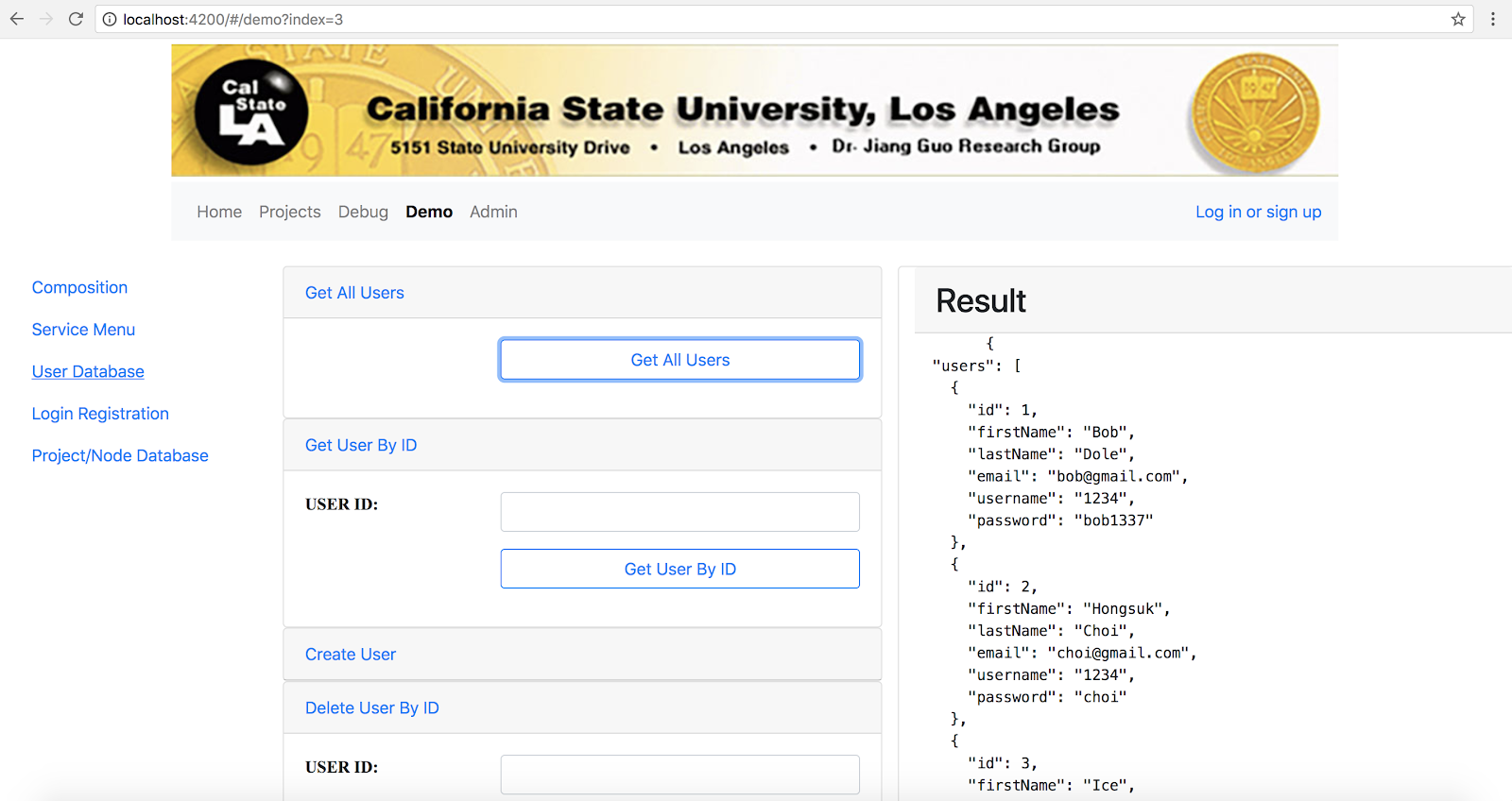
The debug component is created for obvious reasons, that is to debug when errors are encountered. Only an administrator can access this page and it allows the admin to check the project’s structure. Since each project is stored in a JSON format, administrators can check which projects is causing errors.



*Figure 15.* Debug Component Web Page.

**3.1.3 Demo-app Component**

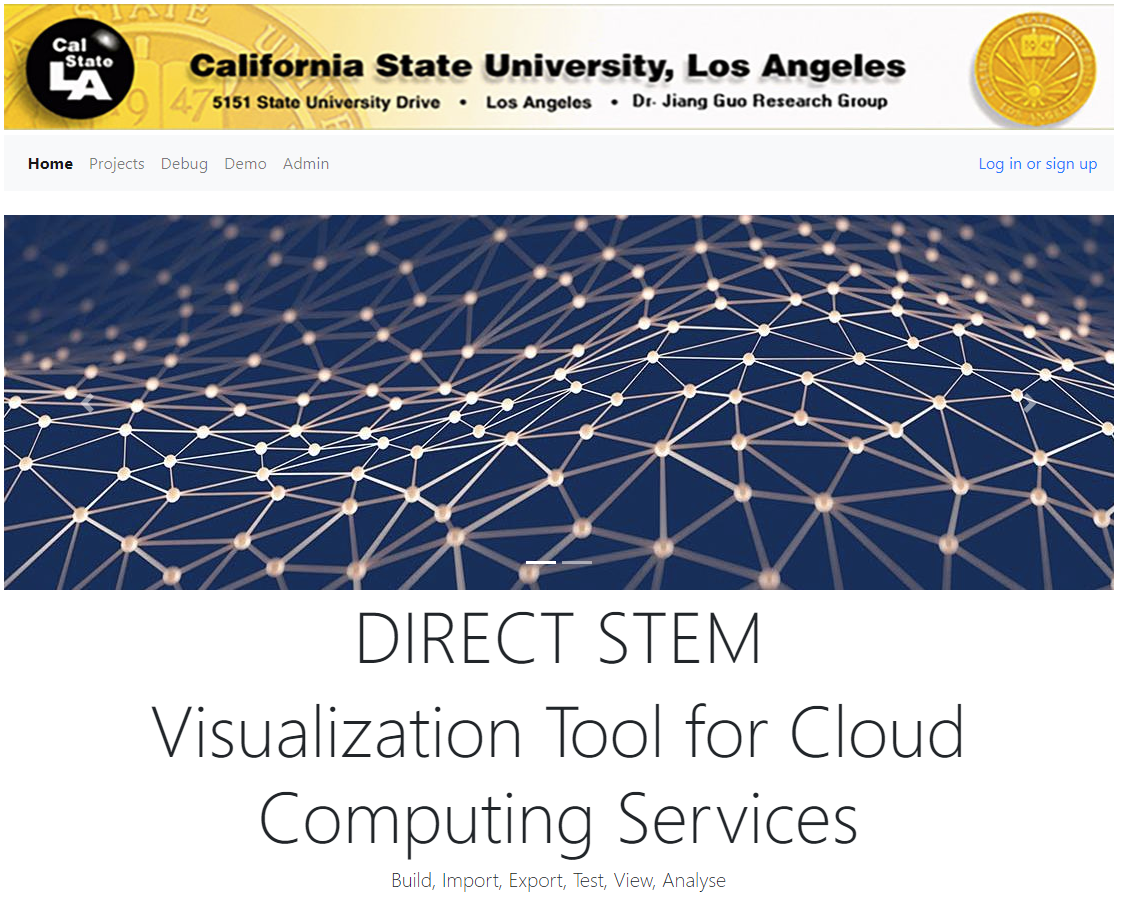
The demo-app component’s purpose is to test the REST API and clearly see the result of what the API is returning. This page is also restricted to admins currently just like the other 2 components but will be restricted to only the programmers later. This page helps programmers locate any input and output configuration errors.



*Figure 16.* Web Page of the Demo-app Component.

**3.1.4 Home Component**

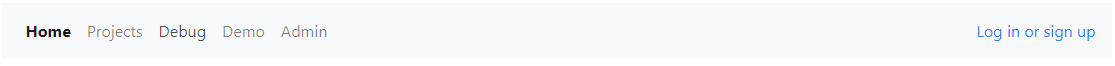
The home component is the base webpage for our project. This webpage is what every user sees when they first arrive at the website. This page has a carousel that can display different images above the title. This page is mainly for aesthetics and show how neat the website is.



*Figure 17.* Home Component Web Page.

**3.1.5 Main-nav Component**

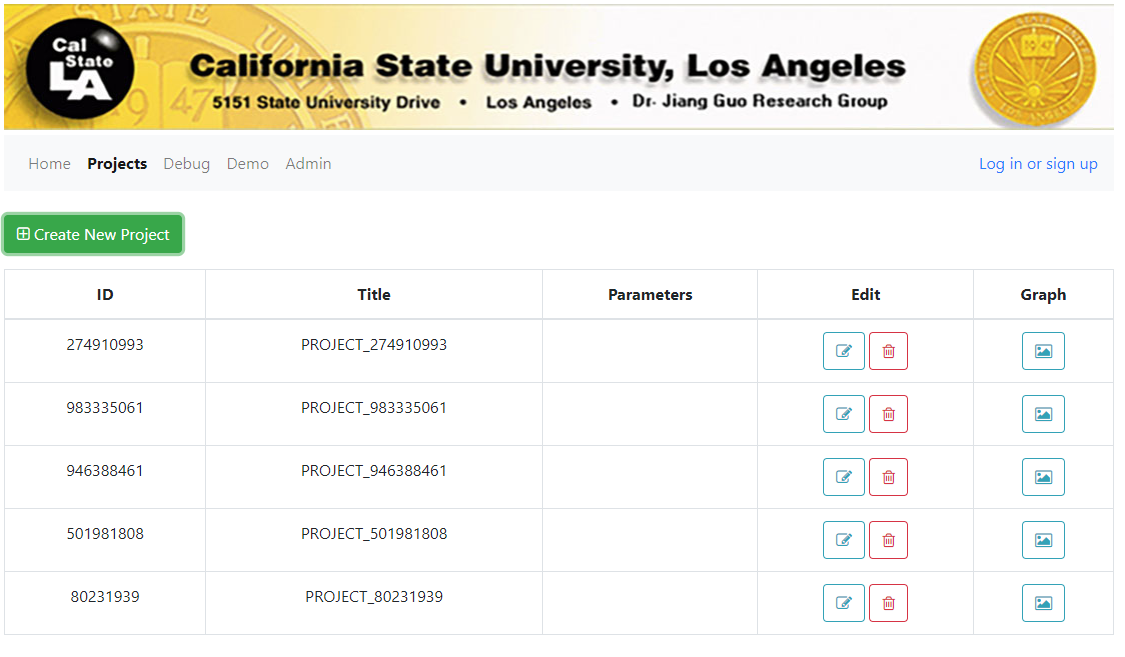
The main-nav component is a navigation bar that can be seen throughout the whole website. The main-nav component allows users or administrators to switch to a different component of the website. Currently, there are 5 tabs that can be seen in Figure 18: Home, Projects, Debug, Demo, and Admin. There is also a way for users to log in to their accounts on the right hand side.



*Figure 18.* The Main-nav Component Shown on the Website.

**3.1.6 Projects Component**

The projects component simply lists out all the projects available in a table. There is also a button at the top to let users generate a new project. Now the ID of the project is automatically generated from 1 to 1 billion and the title is automatically set to be PROJECT\_ProjectID where ProjectID is the number randomly generated. There are 3 buttons that each project row has which are edit, delete, and graph. The edit button brings up the project editor sub-component which allows users to edit the project name and description. The delete button removes the project from the application. The graph button will bring up the composite web service tool which will allow users to create their compositions.

****

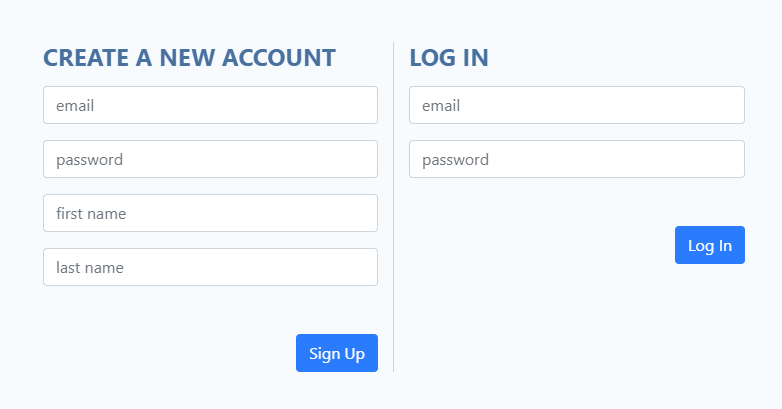
*Figure 19.* Showing the List of Projects in the Project Component.

**3.1.7 Shared Component**

The shared component is not a traditional Angular component, but it has all the resources that is shared by many of the components in this project. It contains the models of this project which are the users, projects, nodes, edges, and more. It also has all the services the project use which are the user service and the project service.

**3.1.8 Split-panel-login Component**

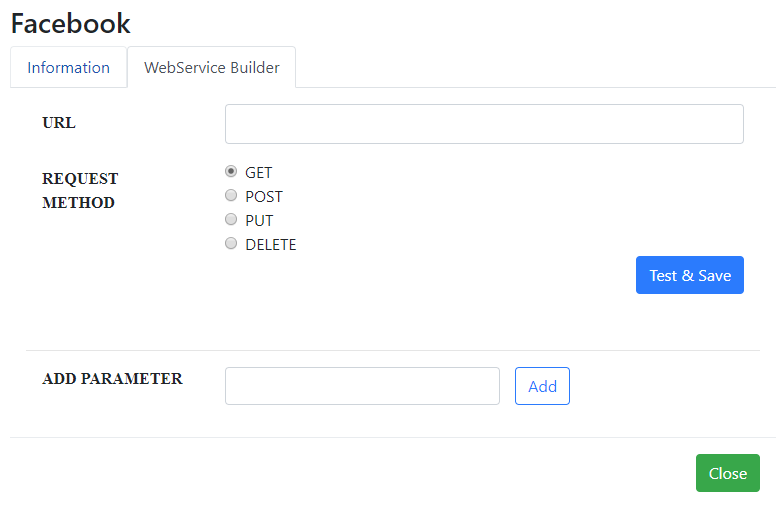
The split-panel-login allows the users to log on the website. This component can be brought up by clicking the Log in or sign up text on the right of the navigation bar. The component appears in a modal window which blacks out the background and allows the user to focus on the split-panel-login. The figure below shows that the left side allows the user to sign if they need an account or to log in on the right side.

****

*Figure 20.* The Modal Window for the Split-panel-login Component.

**3.1.9 Webservice-config-menu Component**

Now the webservice-config-menu component is made up of many sub-components which can be seen in Figure 13. This component is mainly to set up the API call to the remote webservice. Since there are many sub-components, only the webservice-form sub-component will be explained. In the webservice-form sub-component, there is the URL textbox which lets the user indicate which API they shall be using. There are the request methods that lets the user choose which CRUD operation they will be performing. Generally, most people would use GET since POST, PUT, and DELETE usually involve having access to those procedures. There is an add parameter textbox if an API resource requires them. The user can press the add parameter to create more textboxes for parameters if more than one is needed. Also, the title of the webservice is at the top and there is a close button to close this component. Figure 21 shows everything that was mentioned.

****

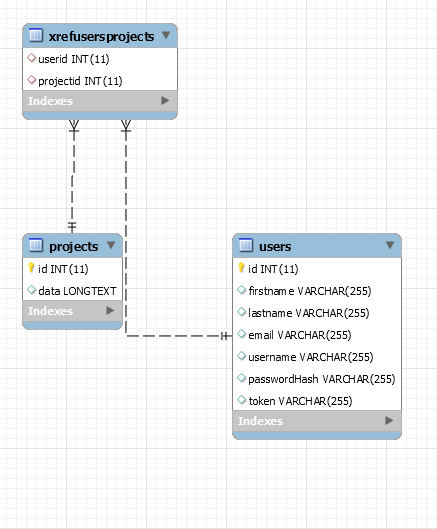
*Figure 21.* The Web Service Editor.

**3.2 Application Layer**

The application layer is the shared communication and interface methods. It is basically all the logic and message sending the project’s REST API handles. The REST API is written in Java and used the Jersey Framework in order to build it. Now this project has many endpoints. Going through them, the API has a getProjects(), getProjectById(), createProject(), deleteProjectById(), updateProject(), getUsers(), getUserById(), createUser(), deleteUser(), updateUser(), and queryWS(). The projects and users are all simple CRUD operations. The queryWS() is part of the logic for testing webservices, because it sends the query to the endpoint to test it.

**3.3 Data Layer**

The data access layer is the layer that provides simplified access to data stored in persistent storage of some kind, such as an entity-relational database [7]. Now VTCCCS’s data persistent is using the relational database MySQL. In MySQL, the project has the directstem database. In the directstem database, there are 3 tables currently. They are projects, xrefusersprojects, and users. Users have attributes such as id, firstname, lastname, etc. that was discussed earlier, but projects only have two attributes. It only has id and data. The reason for the simple table is because of the lack of requirements collected. The table was kept simple and will be changed later once all the requirements are fully understood and a final version of the table will be made. The data attribute is in JSON format and has kept all the information of the projects since it is a LONGTEXT datatype. The reason why long text was used because the maximum length is 4,294,967,295 according to the MySQL documentation [8]. Projects can generate a lot of data especially with their real-time monitoring and performance collection capabilities so having enough space is necessary. The last table is the xrefusersprojects which stands for the references between users and projects. This table is sole purpose is to keep track of which users own which projects. This is necessary so users do not interfere with each other’s projects.



*Figure 22.* EER Diagram of the Database Directstem.

CHAPTER 4

Installation Guide

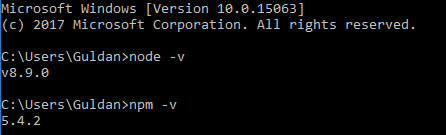
This portion will cover how to get the project running on different platforms. This is mainly to help those who want to use the project and see what it is capable of. This installation guide will be based on using the Windows platform. If you are using a different OS, research how to install these tools onto your machine. Also, these instructions aim to run the project locally and not from a remote server.

**4.1 Node.js and NPM**

The first thing the project needs is Node.js. Node.js is a platform for JavaScript. It is built on Chrome’s V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model to remain lightweight and efficient. Node.js is needed to use Angular CLI which is one of our main tools for this project.

Installing Node.js also automatically installs NPM. NPM stands for Node Package Manager. NPM lets users install and manage their project’s dependencies. All our project’s front-end dependencies are listed inside our package.json file and running npminstallwill immediately have all the dependencies installed.

1. Download the Windows installer from the Node.js website.
2. Run the installer.
3. Follow the directions of the installer. Choose the default values for the installer if you are not sure of what options to select.
4. Test if Node is properly installed by running the command node -v in the Windows command prompt.
5. Test if NPM is also properly installed by running the command npm -v in the Windows command prompt.



*Figure 23.* Sample Output of a Successful Installation of Node.js and NPM.

**4.2 Angular CLI**

Angular CLI means Angular Command Line Interface. Angular CLI is a tool to initialize, develop, scaffold and maintain Angular applications. Angular is a TypeScript-based open-source front-end web application framework.

1. Open a command prompt and run the command npm install -g @angular/cli.
2. Check if Angular CLI is properly installed by running the command ng –version. The prefix ng stands for Angular and it is how start using the Angular CLI.



*Figure 24.* Output of Checking if the Angular CLI is installed on.

**4.3 Git**

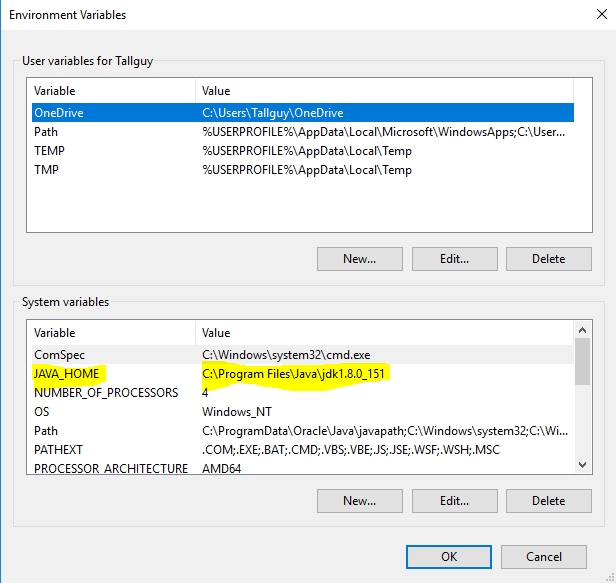
Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. GitHub is where developers store their projects and network. The project is stored on GitHub for easy access.

1. Download the Windows installer from the git website.
2. Run the installer.
3. Follow the directions of the installer. Choose the default values of the installer if you are unsure of what options to select.
4. Open a command prompt or use git bash if you chose to add it in during the installation process.
5. Run two commands to configure your username and email for git: git config -global user.name “John Doe” and git config -global user.name johndoe@gmail.com.
6. You need to generate a new SSH key for authentication so follow the instructions in <https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/>.

**4.4 Java Development Kit**

Java Development Kit or JDK is a software development environment used for developing Java applications. This project requires at least JDK 1.7.

1. Download the JDK installer for Windows from the Oracle website.
2. Run the installer.
3. Add JAVA\_HOME to the system environment variables in the control panel.



*Figure 25.* Modifying the Environment Variables in Windows 10.

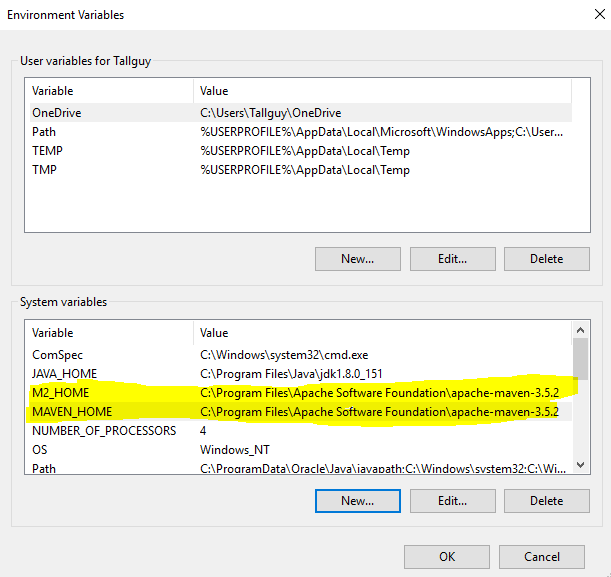
**4.5 Installing MySQL**

1. Download the Windows installer for MySQL community edition.
2. Go through the installer to get MySQL installed. All the default options are fine, but you only need the server installed.
3. Either start MySQL workbench or access MySQL through the MySQL command line so you can run the script located at CSULA-DIRECTSTEM-Webservices\java-server\sql.

**4.6 Apache Maven**

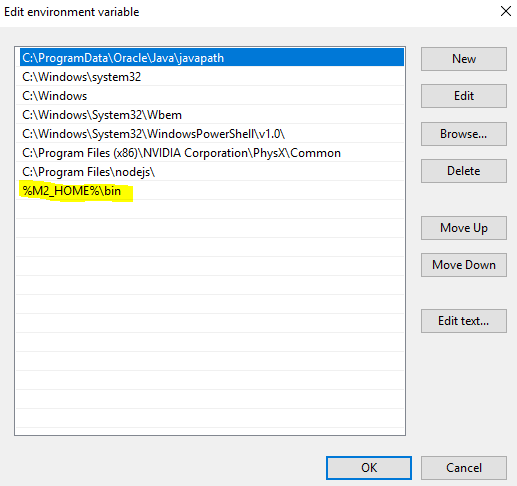
Apache Maven is a software project management and comprehension tool. Maven is needed to build the REST API in order to access the database.

1. Download the apache-maven-3.5.2-bin.zip file from the Apache Maven website.
2. Extract the zip file to your desired location.
3. Add M2\_HOME and MAVEN\_HOME to the system variables in the control panel.



*Figure 26.* Adding the Environment Variables for Maven.

1. Update the path variable to include %M2\_HOME%\bin so you can run the maven command.



*Figure 27.* Showing the Path Variable for Maven Directory.

1. Finally, to verify if maven is installed, run the command mvn -version in the command prompt.

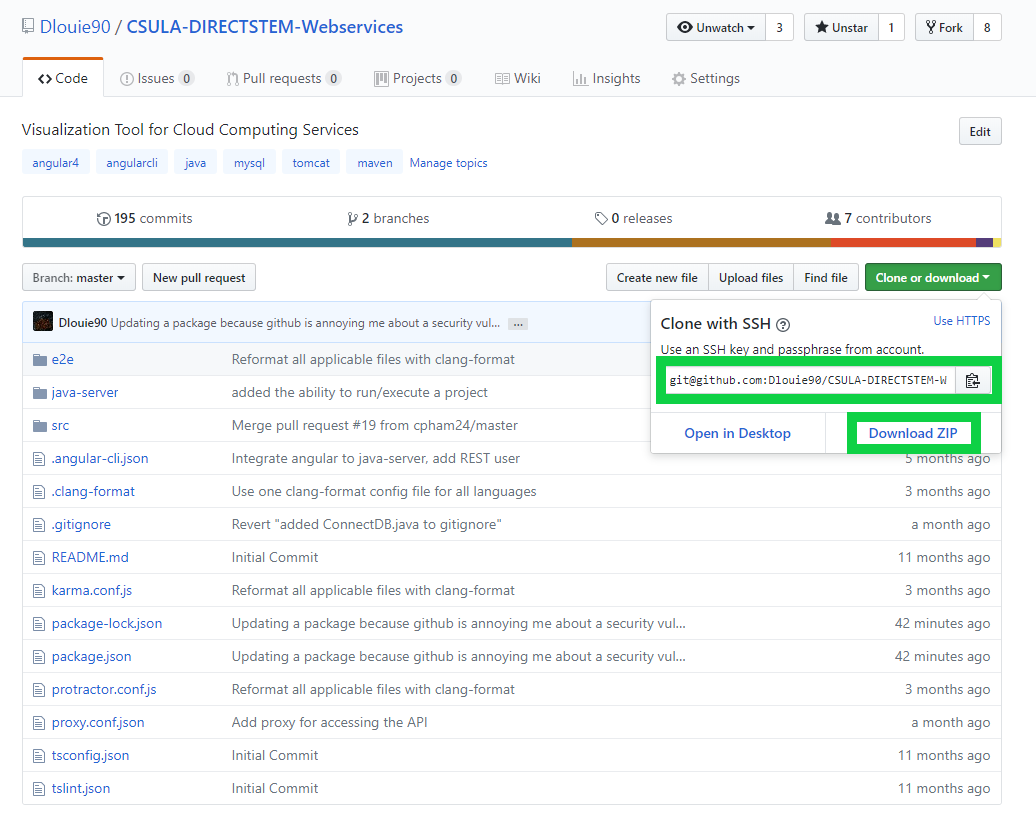


*Figure 28.* Displaying the Sample Output of Mvn -version that is Installed.

**4.7 Project Deployment**

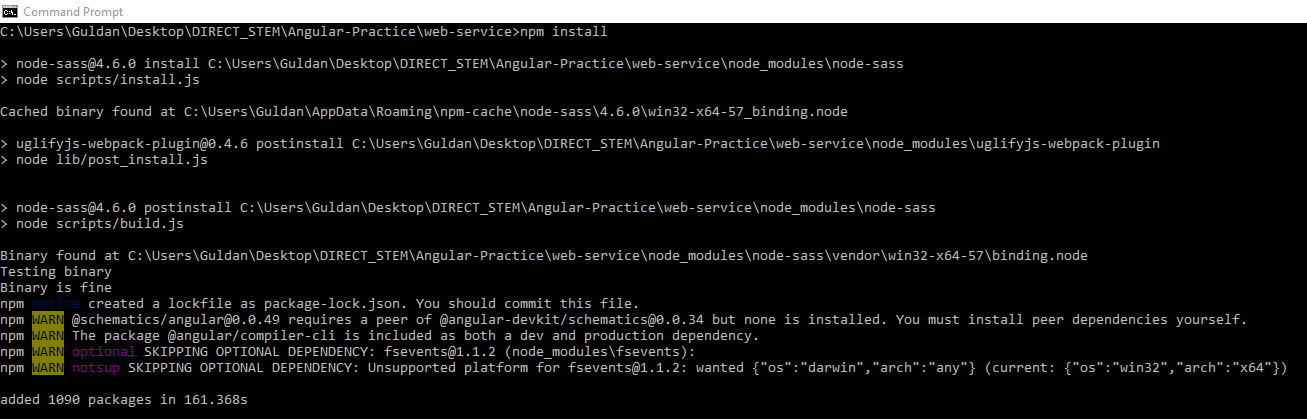
In order to deploy the VTCCCS project, the back end and front end needs to start. Usually it is best to run the back end first, because the front end will make calls to the REST API and will return errors if the back end has not been fully booted.

1. Download the project from [https://github.com/Dlouie90/CSULA-DIRECTSTEM-Webservices](https://github.com/Dlouie90/CSULA-DIRECTSTEM-Webservices%20) as a zip file and extract the files. Another option is to clone the repository if using git is preferred. See the figure below for download location.



*Figure 29.* Location of Project Download.

1. Open either the Windows command prompt or a terminal and change the directory to the location of the project.
2. Run the command npm install in the project’s directory where the package.json file is located to install the front-end dependencies required for the project. There should be a node\_modules folder in the directory of the project after the dependencies have been installed.



*Figure 30.* Example Output after Npm Install Command.

1. Run the command ng build to compile the Angular application in the directory.
2. Now confirm that MySQL is running by either using the MySQL Notifier application if it was installed or the list of services in Windows.
3. Use the command prompt or any terminal to go to CSULA-DIRECTSTEM-Webservices\java-server directory. The pom.xml file can be seen in this directory.
4. Run the command mvn tomcat7:run. Note: Ctrl + C will stop the server if you need to do so.
5. Open another terminal and go back to the project directory.
6. Use the command npm start to start the front end of the project.

Npm start runs a custom command written inside the package.json. It should automatically open the default browser to the website locally. If not, open a browser and go to localhost:4200. The web application is being served there.

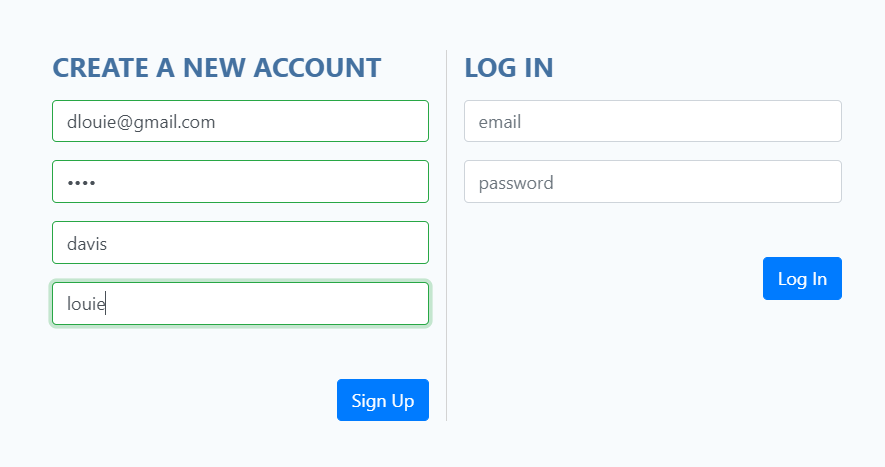
CHAPTER 5

VTCCCS User Manual

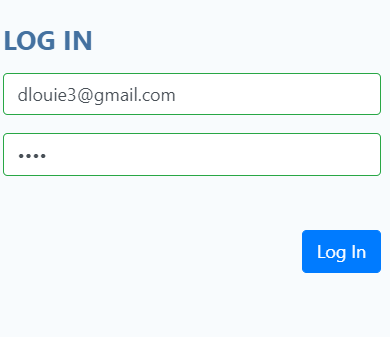
This guide is to aid users in using VTCCCS web application. Jumping into this application without a guide is possible, but several features may be missed at first glance.

**5.1 Register/Login**

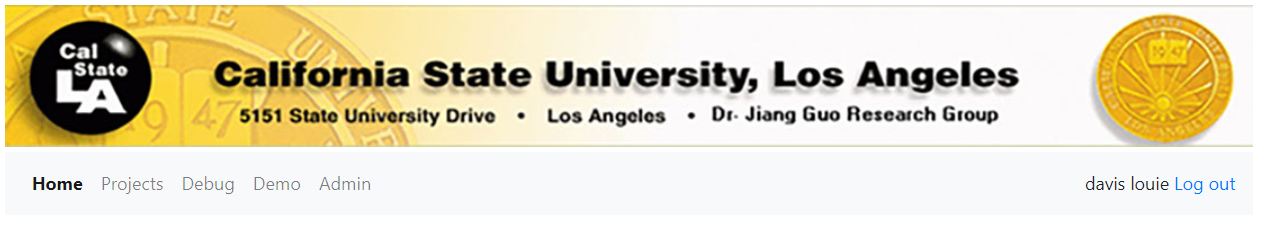
The first step is to login to the web application. Click on the Log in or sign up text in the right-hand side of the navigation bar. This should bring up the login/register modal window. Now, if the users are new to the website, they will use the left side to create a new account. If the users already made an account, they can use the right side of the modal window to log in. After the users log in, they can start creating their projects.



*Figure 31.* Registering a New Account.



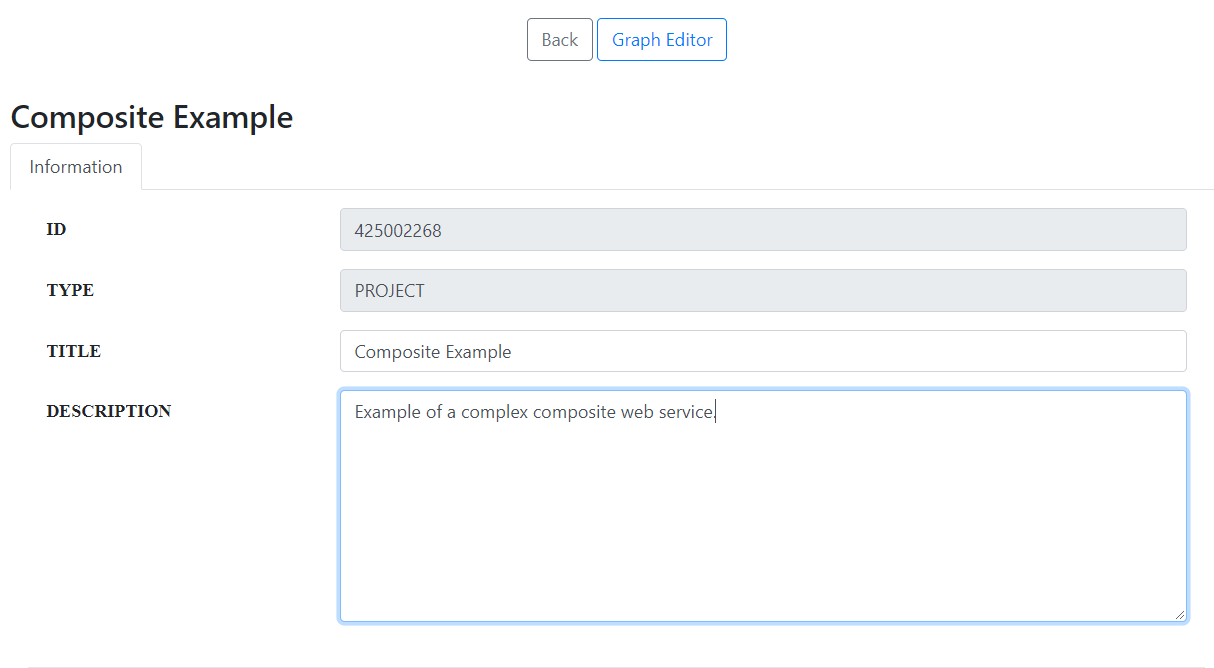
*Figure 32.* Logging into VTCCCS.



*Figure 33.* Successful Login.

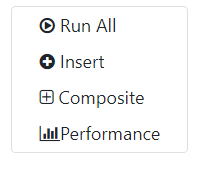
**5.2 Project Creation**

Now the user will click on the Projects tab in the navigation bar or click on the start here button to go to the projects page. The user may press the edit button to edit details of the project like the title or description.



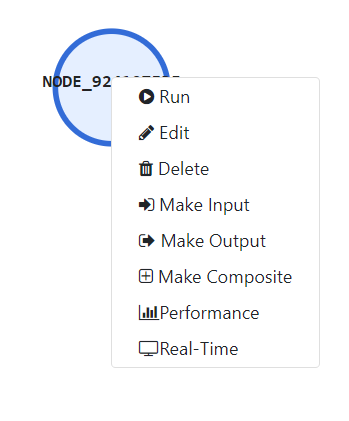
*Figure 34.* Project Details Editor.

Now the user can move on to the graph editor by pressing the graph button on the far right in the graph column of the table. This will take the user to the graph editor of the project. The user can create a node by right-clicking and selecting insert. This will create an empty node. There are other menu options such as Run All, Composite, and Performance. Run All would run all the web services. Composite would allow the user to choose a different project and make that node that composite web service. Performance would display metrics gathered such performance and availability, but more will be covered later.

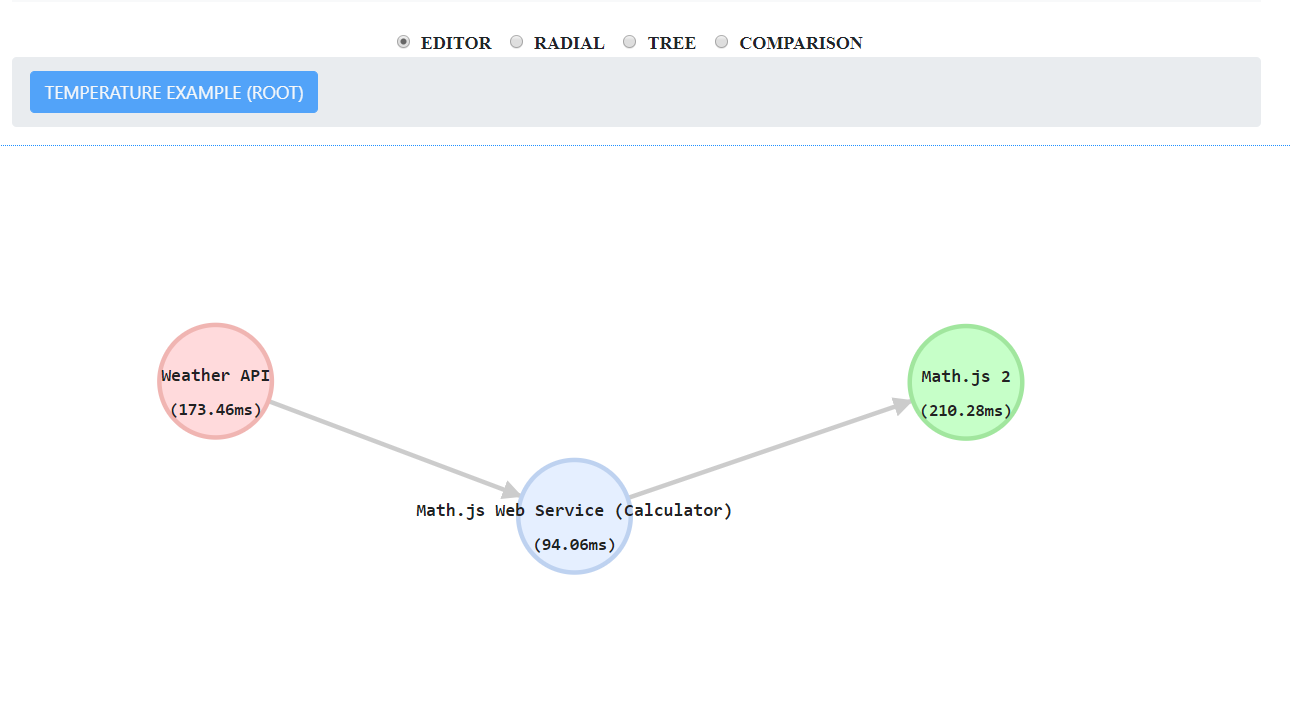


*Figure 35.* Menu Options.

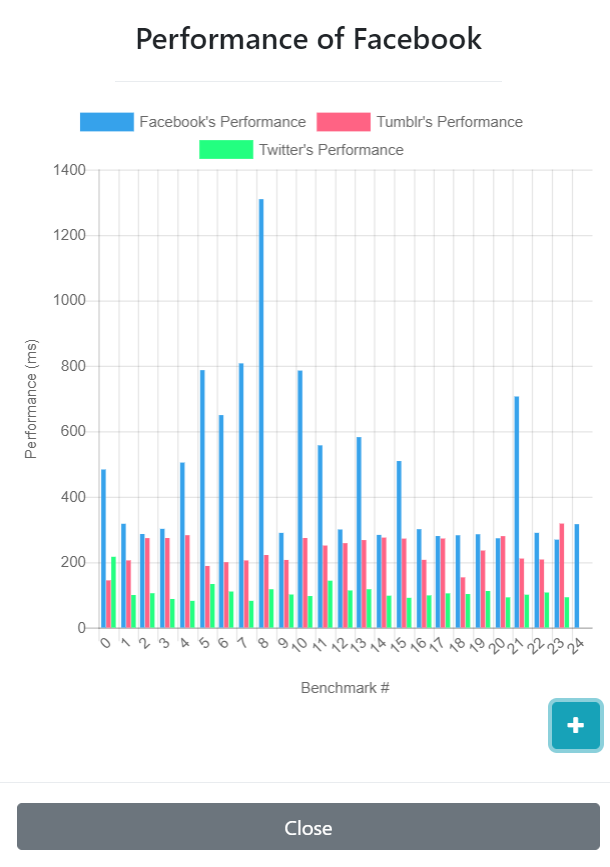
This node can be configured by selecting the node. The border will be thicker to show that you have selected the node. Now if you right-click when the node is selected, there will be more options. Run just runs the individual web service and records the run-time performance. Edit will allow the user to edit the node’s properties and insert a web service. Delete removes the node from the graph editor. Make Input will make the node an Input Node which will pass the input parameters on to the next web service. This is mainly for complex compositions since we need input nodes. Make Output is the result of the current project and it is what this project is sending as output to other web services as inputs. Performance displays the most recent performance metrics. Finally, Real-Time brings up the real-time monitoring window. Also, if the user wants the nodes to be connected, they must click on the node, hold CTRL, drag the mouse to the node the user wants to connect.



*Figure 36.* Selected Node Options.

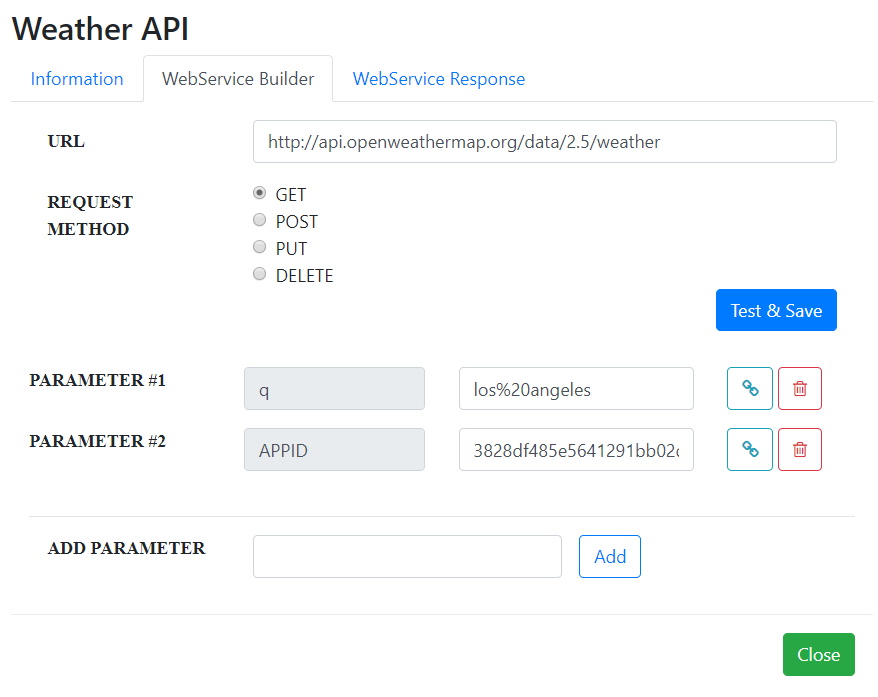


*Figure 37.* Example of Input, Regular, and Output Node Interconnected.

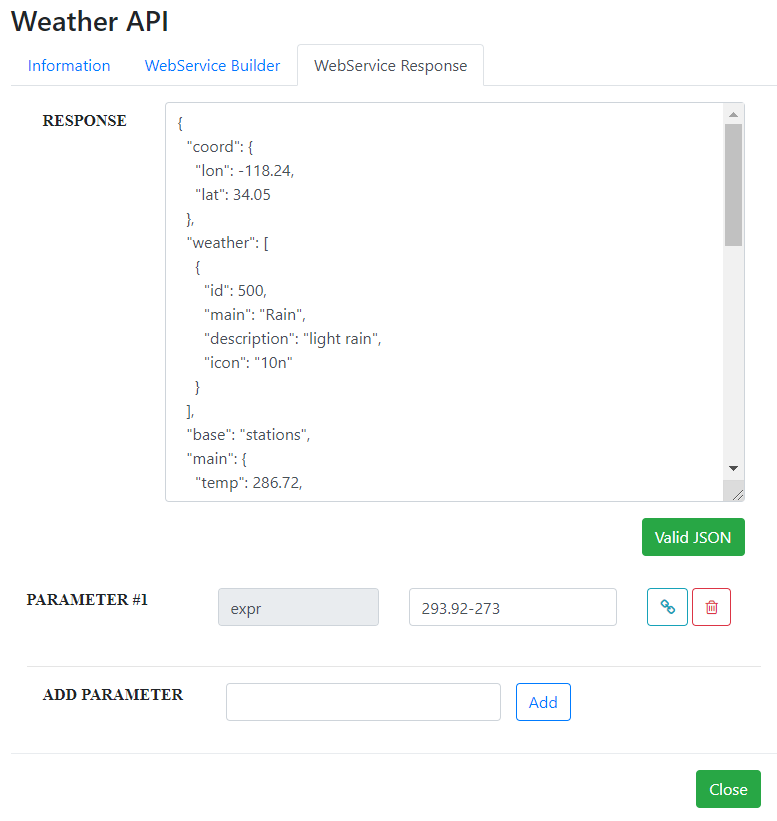


*Figure 38.* Performance Chart of a Web Service.

Now in order to add a web service, the user must set the URL to one of the endpoints of a web API of their choice. In the figure below, this node is using Weather API. Next, the user sets the request method. Now if the web API needs parameters, the key and value must be set properly. For the Weather API, the key is q which stands for query and the value is Los Angeles. If additional parameters are needed, the add parameters textbox can be used to add more parameters. Once the user is done configuring the web service, press the test & save button. This will test to see if the web service is valid. If it is indeed valid, the WebService Response tab will appear, and no error dialogue will pop up. There are also link buttons and delete buttons to the right of each parameter. The link button will allow you to link that parameter to an input that current node has. The only way for a node to see options to link is when another node is giving their output to this node.



*Figure 39.* Example of Linking a Remote API to the Node.

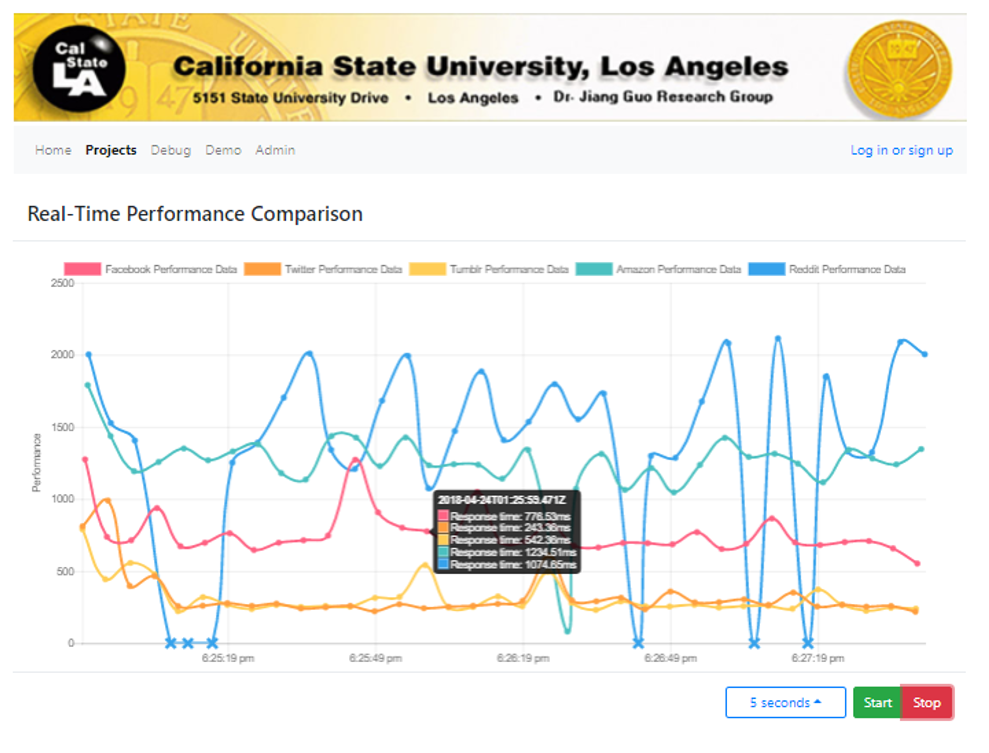


*Figure 40.* Example of a Successful Response from a Web API.

In the WebService Response tab for the Weather API, it returned a JSON file. Now if users want outputs in this node, they must add them manually. It is a similar process to adding parameters for the web API, but the value must be typed. The user must decide what they want to be as outputs.

**5.3 Real-Time Performance**

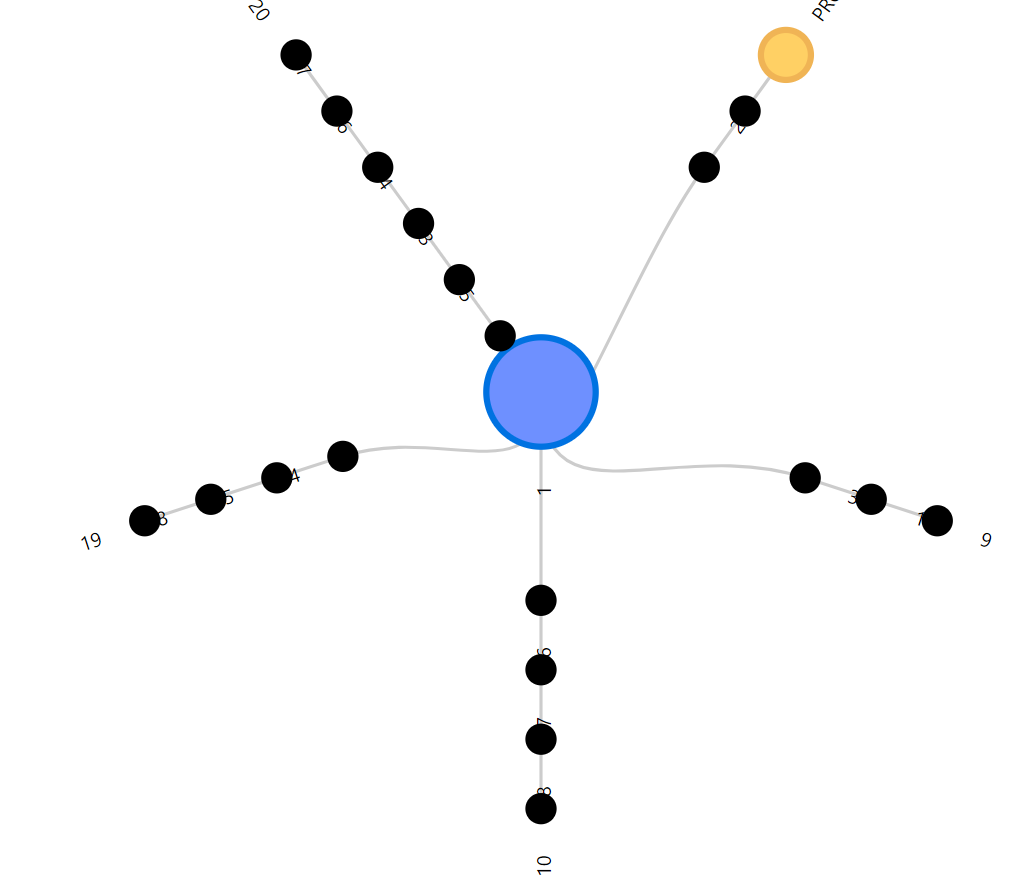
The real-time performance is how this web application collects performance data for web APIs. Without this feature, the user would be manually clicking on run for data. This feature will make the API call every x seconds where x is the chosen interval for this task. This feature also is not restricted to just web APIs. As seen below, the figure shows that this feature can measure response times of websites even though this project’s intended target is web services. There are two different ways of using this feature. The first way is individually testing one web service by right-clicking a node and selecting real-time. The other way is a group comparison by clicking the comparison radio button at the top of the graph editor. In that menu, the user can select which nodes in the current project they want to be monitored together.

****

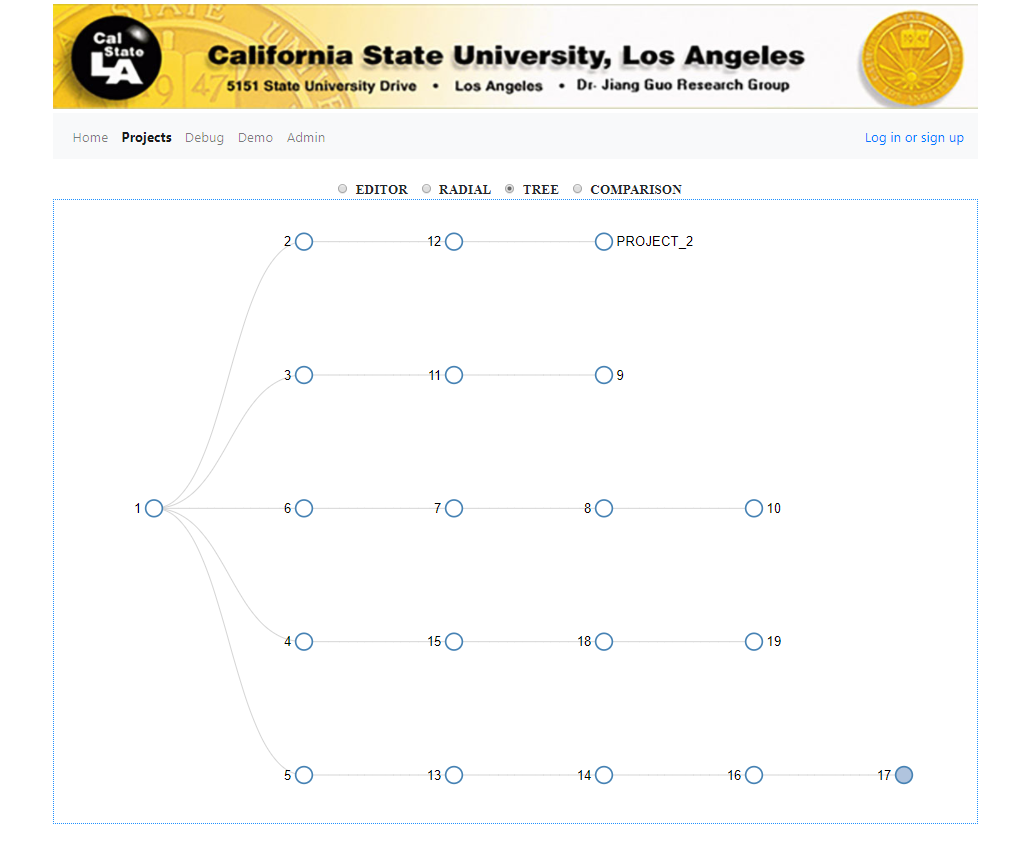
*Figure 41.* The Real-Time Performance Comparison Monitoring 5 Websites.

**5.4 Visualization**

Currently, VTCCCS only has two ways to visualize the project. The first one is the Radial View and the second one is the Tree View. Currently, the Radial View and Tree View can display simple compositions, but if the composition gets too complex, the graphs may not give a great visualization. The user can access these views with the radio buttons at the top of the graph editor.



*Figure 42.* Radial View of a Project.



*Figure 43.* Tree View of a Project.

CHAPTER 6

Conclusion

The creation of composite web services would greatly aide web programmers to generate web services people would not be able to visualize. Keeping track of hundreds of web services is near impossible without structure which VTCCCS aims to solve. The real-time monitoring feature aims to weed out all the underperforming web services. As the internet grows more complex, many web services get left alone without maintenance or improvements. If one web service gets delayed, then the whole performance of the composition gets decreased greatly. So, in order to counteract this possible bottleneck, the real-time tool was created so our users will get the best compositions they can possibly make.

# REFERENCES

[1] “Web application.” *Wikipedia*, 30 November 2018, <https://en.wikipedia.org/wiki/Web_application>.

[2] “Angular (application platform).” *Wikipedia*, 23 November 2018, <https://en.wikipedia.org/wiki/Angular_(application_platform)>.

[3] “Component.” *Angular*, Google, <https://angular.io/api/core/Component>.

[4] “HTML.” *Wikipedia*, 27 November 2018, <https://en.wikipedia.org/wiki/HTML>.

[5] “Em (typography).” *Wikipedia,* 2 January 2018, <https://en.wikipedia.org/wiki/Em_(typography)>.

[6] “Representational state transfer.” *Wikipedia,* <https://en.wikipedia.org/wiki/Representational_state_transfer>.

[7] “Data access layer.” *Wikipedia,* 21 August 2017, <https://en.wikipedia.org/wiki/Data_access_layer>

[8] “11.1.3 String Type Overview.” *MySQL,* Oracle, 2018, <https://dev.mysql.com/doc/refman/8.0/en/string-type-overview.html>