Analysis Me

**Fall**

2015

Technical Report

Service Oriented Computing (18-655)

Team 1

Enclosed in this document is the technical report of the Climate Service Management sponsored by NASA.

Table of Contents

[1. Introduction 3](#_Toc438238987)

[2. Motivation 6](#_Toc438238988)

[3. Related work 7](#_Toc438238989)

[4. System design 9](#_Toc438238990)

[5. System implementation 12](#_Toc438238991)

[6. Experiments and analysis 15](#_Toc438238992)

[7. Conclusions and future work 23](#_Toc438238993)

[8. Contribution of each team member 25](#_Toc438238994)

[9. Tutorials 26](#_Toc438238995)

# Introduction

Climate report has been an indispensable part of our daily lives. Reliable climate services management plays an important role in providing customer with accurate and useful climate services. During the last 5 weeks, members of Team1 implemented a climate service management service. We implemented necessary functions to help manage many different types of climate services, build interactions between service providers and consumers and improve user experience.

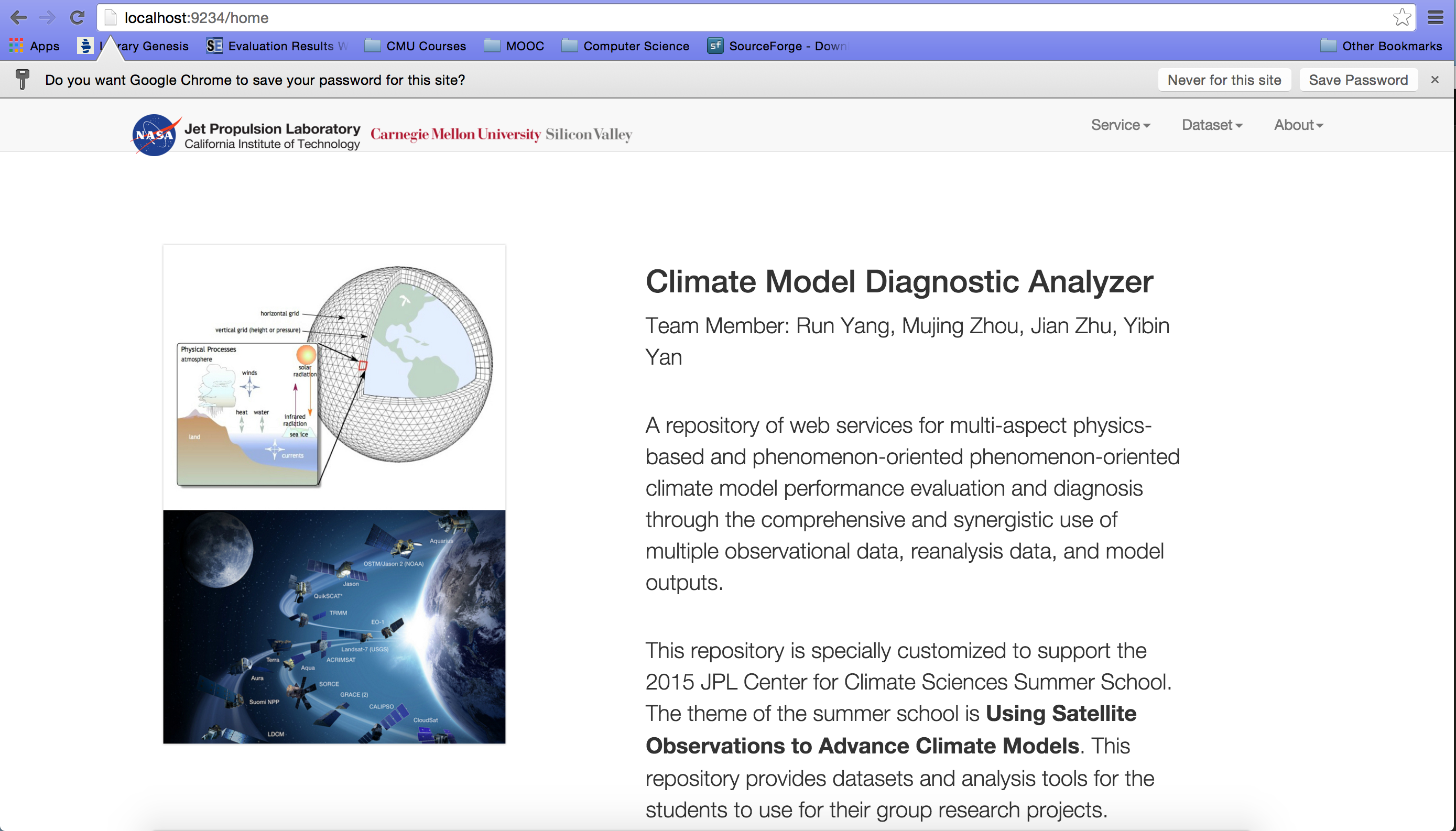


Figure 1.1

Technologies we have used include but not limit to Hibernate, AJAX, Docker and MySQL. We use Docker to build our running environment to improve reusability and efficiency, which is quite efficient as we did not need to reconfigure the running environment every time. Hibernate and MySQL database are selected to build back-end of our service. Hibernate is an extremely helpful framework as we can operate on our database directly with Java. AJAX is used in our implementation of front-end functions.

We implement basic functions like “services keyword search” and “service grade” to improve user experience. “Services keyword search” means that people can enter keyword and use it to search for the first three related climate services. “Service grade” means that people can rate a climate service by giving it a score. Climate services will be ranked according to their grades and services with high grades will be preferred by users.

Moreover, version management is also supported by our system. All the previous service versions can be shown in the Climate Service page; people can choose to use their favorite version if they like. At the same time, top3 climate services can be got according to their grades and used frequency, this will offer users access to their most recently visit service quickly.

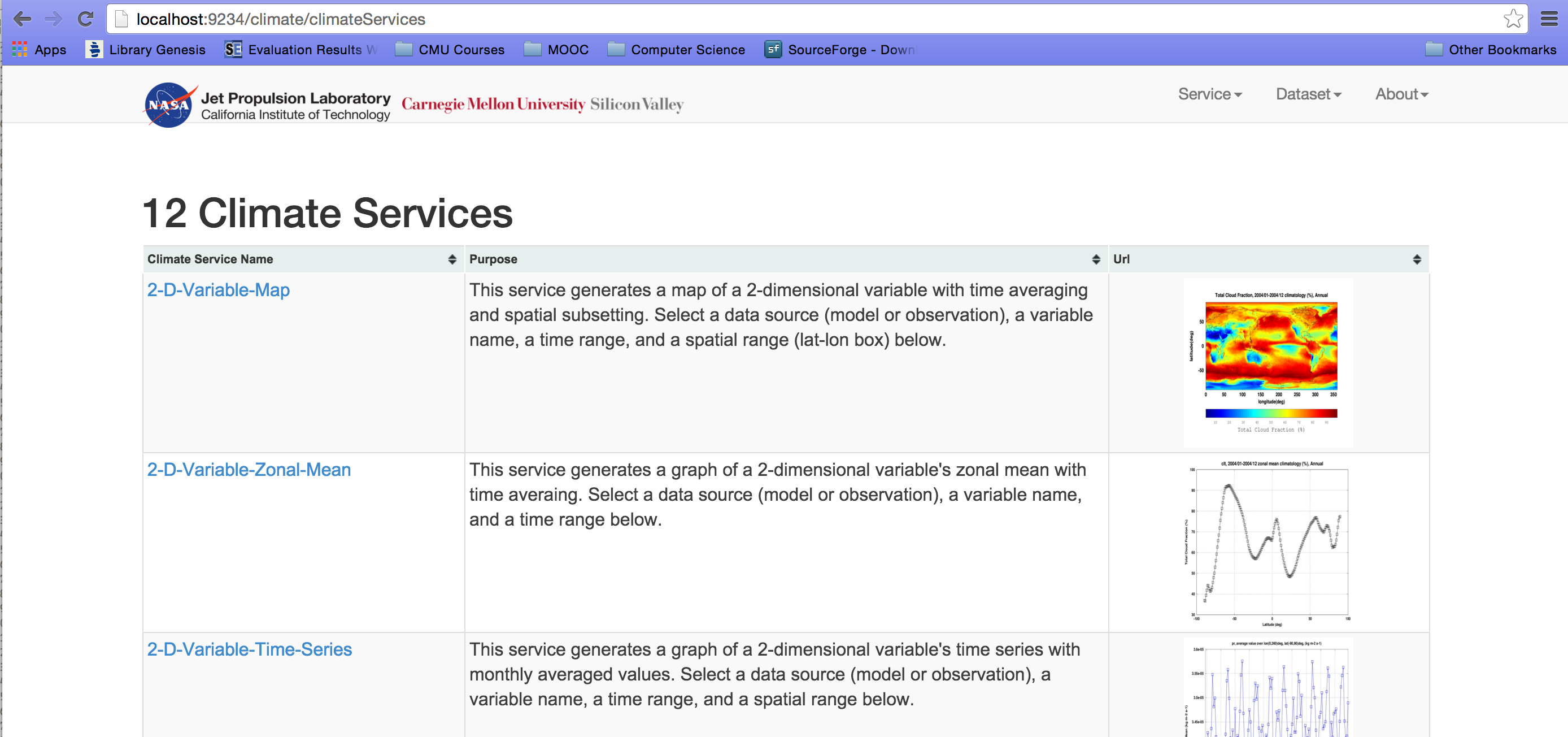


Figure 1.2

In addition, to help user interact with their friends better, we define special functions with sign “@” and “#” as well. Sign “@” can be used to mention a friend directly. And “#” can be used to mention a specific climate service. We choose to use MySQL database as our back-end database. Hibernate is used as the backend framework to manage MySQL database. With hibernate, we can map tables in database to java class directly and operate on database with API, which is very convenient for implementation.

What’s more, we also added two useful functions - tag and recommendation function for your convenience. Tags are added with comment to describe a climate service. Climate services will be categorized by their tags for users to search for quickly. People can also choose to ‘like’ a climate service and system will recommend suitable climate service for user according to history records and user’s favor. Those functions will give our user awesome use experience.

Our report will start from an overview of the motivation and introduction to the whole project. After that, we will introduce how we manage our time and allocate our tasks. We will give a detailed description of our system design and implementation and experiments will be shown steps by steps. Then, we will give the conclusion of our project and explain some future work. A detailed tutorial will be attached at the end of this report.

# Motivation

This project is an interesting and meaningful attempt to brush up our programming skills and system design concepts. We learn various programming frameworks and development techniques during the project, so the motivations behind this project are various.

First, we would like to enhance usability of climate service based on the prototype provided. The original version of Climate Service Management System is too simple and difficult to use. It simply gives a portal for the users to view different climate services. As a hub, from our understanding, it should be intuitive and easy to use. As a result, we plan to give a concise but elegant UI to our users. A straightforward UI can be easy to use and improve the productivity significantly.

Second, we would like to build the bridges among the researchers. Since this climate service management system is developed for the related researchers and scientists, it is pivotal to enable and improve the interaction among the users. As is known to all, communications and interactions are essential to the scientific research. How to enable the interoperability and interaction among several services in the management system is one key design consideration in our development process.

Finally, we would like to improve the work efficiency and productivity of researchers and scientists further by polishing up our system. We try to provide some feedback in our individual service page, thus we could gather information of each service and give recommendations to our users. In our opinions, a precise and sophisticated recommendation system should know what the user need exactly and provide some concrete suggestions. For example, our system should know what service is popular, what services a user need. Based on the recommendation system, user can easily find the services he or she needs, and speed up the research further.

# Related work

Technologies we have used include but not limit to Docker, Play Framework, Hibernate, Ajax and so on. Below is a detailed introduction to each technology.

Docker Container - Simplify development environment configuration. Set highly reusable developing environment for our project.

Play Framework - High velocity web framework for Java and Scala. Help us develop front-end quickly and efficiently.

Hibernate - Object/Relational Mapping (ORM) framework. Map tables of MySQL database to java classes, help develop back-end efficiently.

Ajax - Avoid reloading the whole web page. Useful in implementation of “@” and “#” functions.

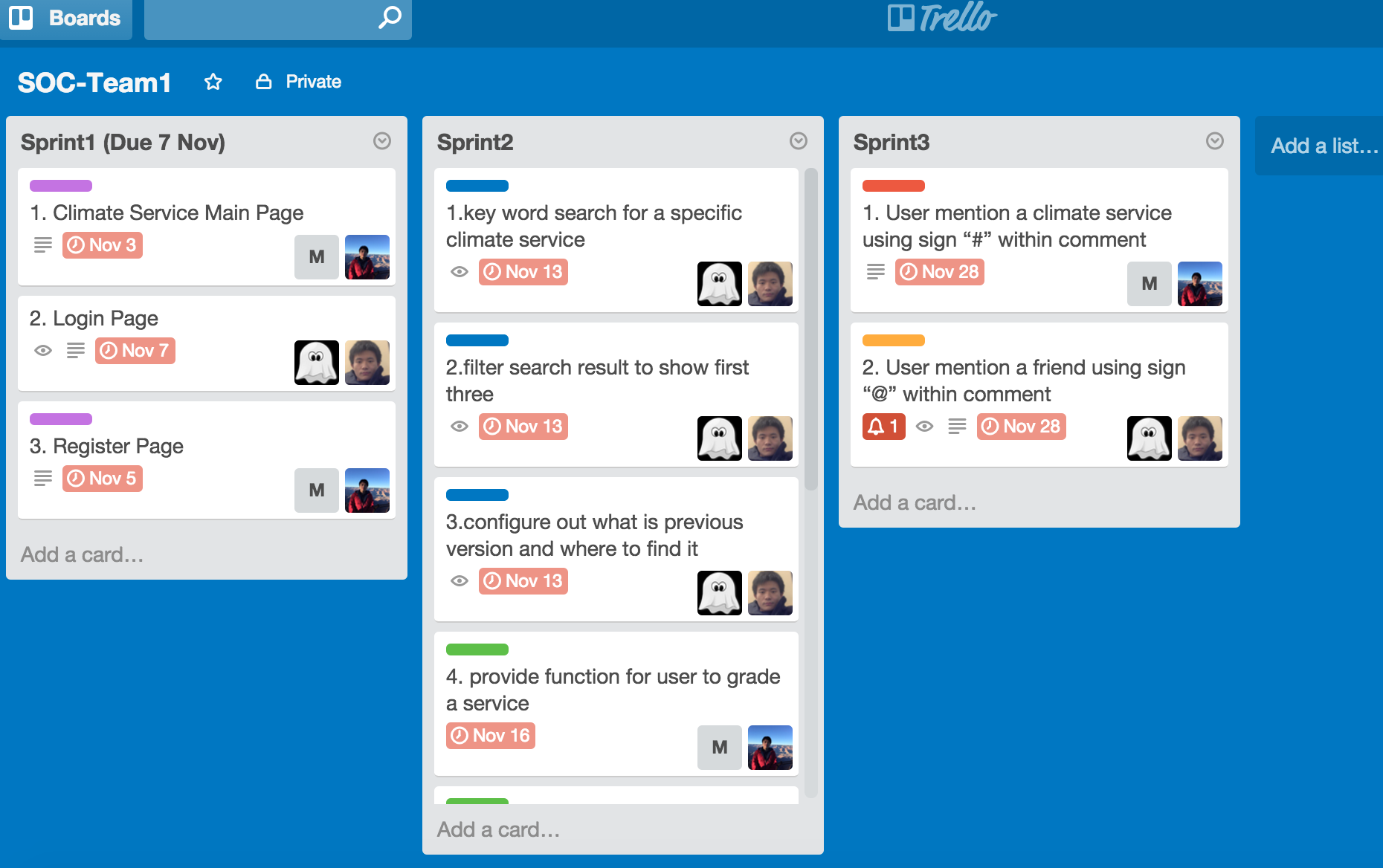
As shown below, we have separated our project into 3 sprints in Trello and shared it with our TAs and professors.

Figure 3.1

Sprint 1

In Sprint1, we got familiar with the whole climate service framework and redesigned the main page of the climate services management service. We spent about one week on this sprint and implemented our own Login and Register function.

Sprint 2

In Sprint2, we spent one week to implement some basic required functions like Keyword Search, Grade a Service and Version Control. We provide users with the function to search three related climate services with keyword. People can also rate a climate service by giving it a scores. At the same time, people can select their preferred version of climate services to use.

Sprint 3

In Sprint3, we spent about 2 weeks to implement functions of “#” and “@”. We used AJAX technology in this part. People can mention a specific service using ‘#’ and mention a specific user using ‘@’ just like what you can do in Facebook.

Extra Implementation

After implementing all the required functions, we have about one week left. In addition to all the required functions, we spent one week to implement two additional functions - Tag a Service and ‘Like’ And Recommend. People can add tags when writing comments to a service. Tags will be shown along with comments and services will be categorized by the tags. People can also choose to ‘like’ a friend by clicking the ‘like’ button. After ‘like’ a friend, the system will automatically recommend this friend’s favorite climate services to you. These functions will improve the interaction between user and service system.

# System design

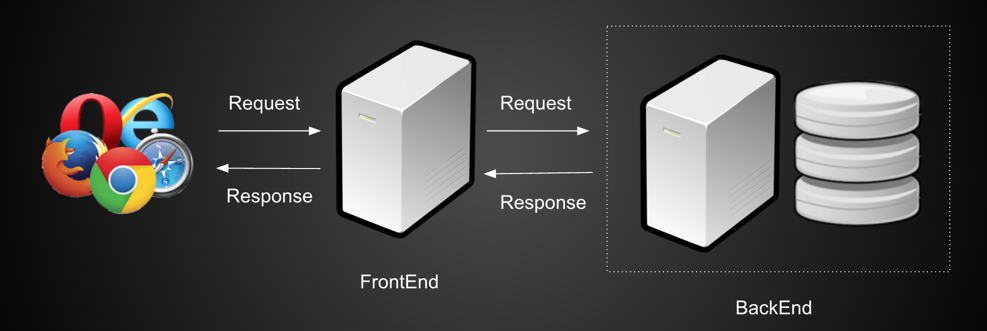


Figure 4.1 System Architecture

There are two servers in our system. One is frontend server, which would process request sent from browser and redirect the request to the backend server. Backend server is in charge of the interaction with database. After Receiving requests from frontend server, backend server would do the corresponding CRUD (Create, read, update and delete) process. Altering retrieving result from database, backend server would forward the result to the frontend server. And finally, result would be showed on website.

Both frontend server and backend server are built upon Play Framework (we would introduce more in System Implementation section). They communicate with each other through HTTP Request. By doing this, we make our system loose coupling and easily scalable.

We use MySQL as database. And the schema is following.

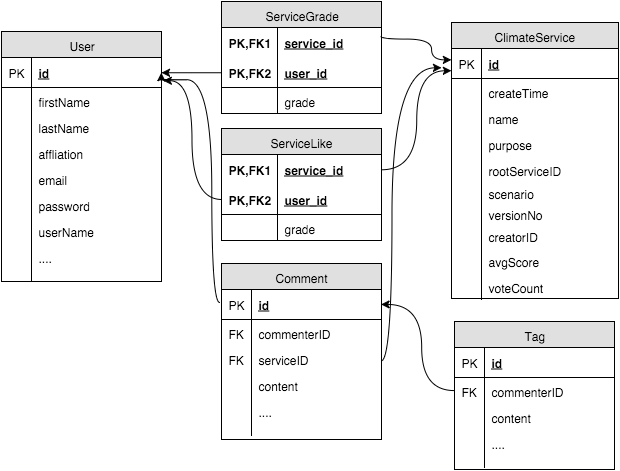


Figure 4.2 database schema

When designed the schema, we make sure all relations are in BCNF (Boyce–Codd normal form). By following this normalization form, we can make sure there is no redundancy in the system and also make sure each table is lossless and dependency preserved. Details would be discussed in System Implementation section.

Following is our UML of our system.

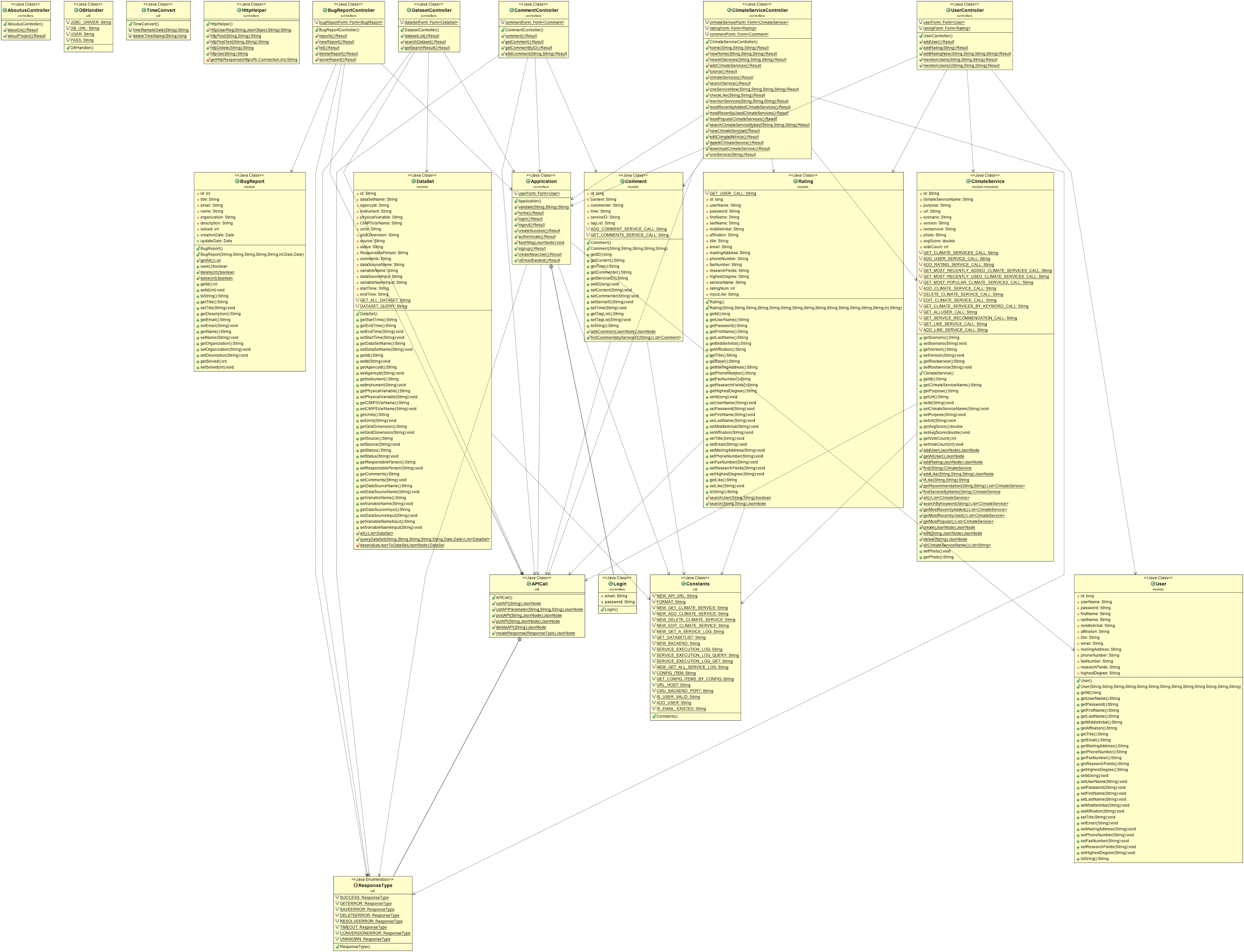


Figure 4.3 UML Frontend

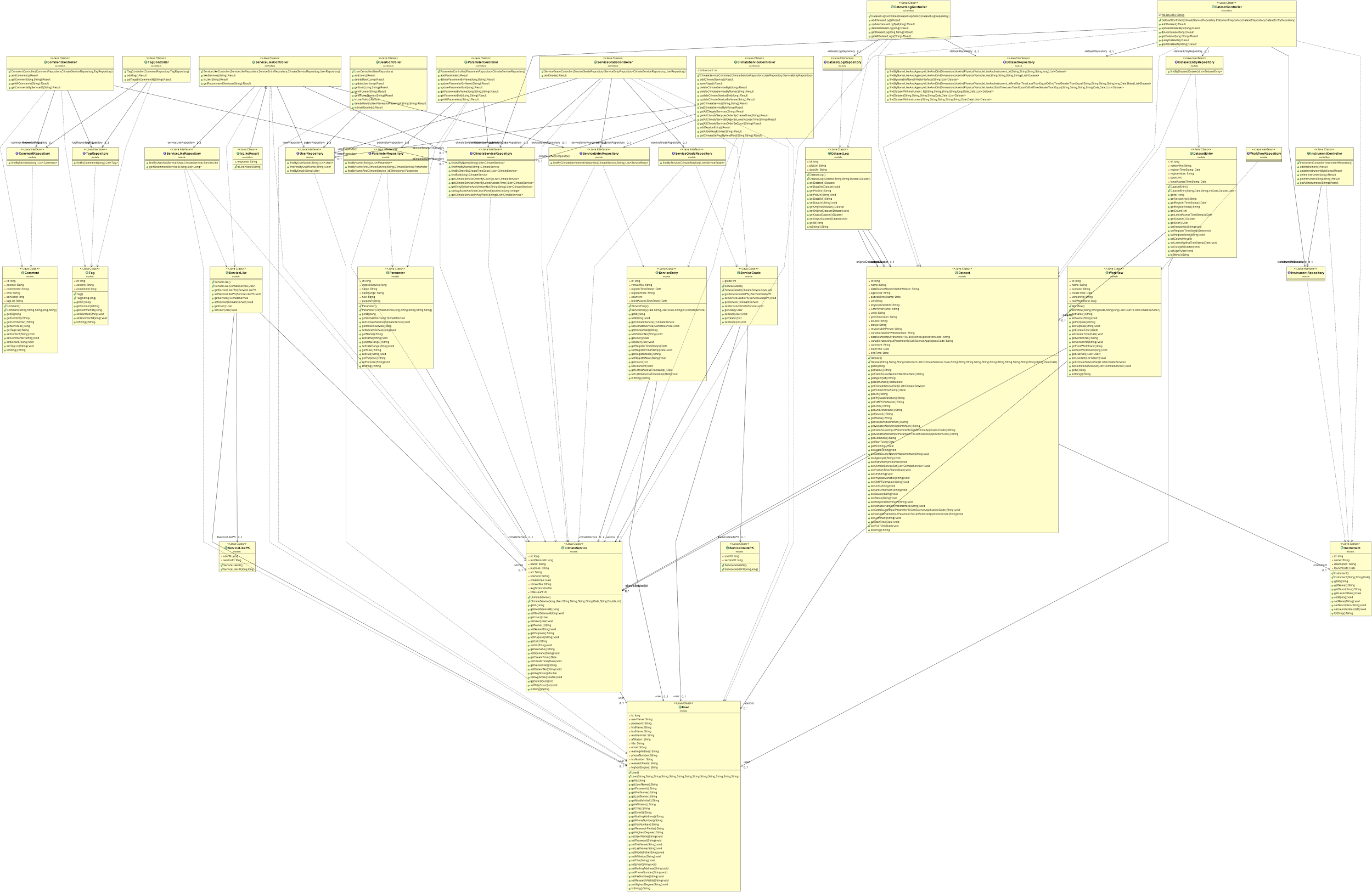


Figure 4.3 UML Backend

# System implementation

**5.1 Language**

**Frontend**: Java, JS, Scala, HTML

**Backend**: Java, SQL

**5.2 Frameworks**

In this project, both frontend server and backend server are built using Play Framework. Play Framework is an open source web application framework, written in Scala and Java, which follows the model–view–controller (mvc) architectural pattern. It aims to optimize developer productivity by using convention over configuration, hot code reloading and display of errors in the browser. It is a framework that is easy for deployment and debug. Besides, it has good performance and scalability.

In addition, we also use Hibernate in order to make it easier and more convenient to manipulate database. Hibernate is an object-relational mapping framework for the Java language, providing a framework for mapping an object-oriented domain model to a traditional relational database. Hibernate solves object-relational impedance mismatch problems by replacing direct persistence-related database accesses with high-level object handling functions.

**5.3 Database Implementation**

MySQL is used as database in this project. MySQL is an open source RDBMS that relies on SQL for processing the data in the database. MySQL provides APIs for the languages C, C++, Eiffel, Java, Perl, PHP and Python. In addition, OLE DB and ODBC providers exist for MySQL data connection in the Microsoft environment. A MySQL .NET Native Provider is also available, which allows native MySQL to .NET access without the need for OLE DB. MySQL is most commonly used for Web applications and for embedded applications and has become a popular alternative to proprietary database systems because of its speed and reliability.

There are 5 tables in our database.

|  |  |
| --- | --- |
| Table Name | Description |
| User | Store user-related data. Such as email and password, etc. |
| ClimateService | Store service-related data. Such as service name, version, etc. |
| ServiceGrade | This table is for recording user’s rating for each table. |
| ServiceLike | This table is for recording user’s liking a specific service. |
| Comment | Store users’ comments for services. |
| Tag | This table is for user to tag a comment. |

**5.4 Implemented Features**

|  |  |
| --- | --- |
| Features | Description |
| Registration | Allow user to register account. When a new account created, a record would be inserted into [user] table. |
| Login | Check if login information entered by user exists in [User] table. If exist, user can enter system. |
| Make Comment | User can make comments for services. If comment successfully, a record would be inserted into [Comment] table. When user browses a service, all comments would be listed. |
| Make ratings | User can rate a service. Rating score can range between 0 - 5. Once user successfully rate a service, a record would be inserted to [ServiceGrade] table. In the meantime, the ‘avgScore’ column of this service would recalculated. |
| Version control | Allow service to have multiple versions. Each version of service is independent to each other. |
| Search a Service | Search service by keyword. We use the ‘Like’ statement of SQL to accomplish this function. |
| Mention a Climate Service in Comment | User can mention a service using ‘#’. We use Ajax in the frontend to retrieve the corresponding data from database and users don’t have to refresh the page. |
| Mention a Friend using @ | User can mention a friend using ‘@’. We use Ajax in the frontend to retrieve the corresponding data from database and users don’t have to refresh the page. |
| Tag a Comment | User can tag a services in her comments. By tagging a service, we can search services through tags. When tag successfully, a record would be inserted into [tag] table. |
| Like a service and recommendation | User can ‘like’ a service. When successfully, a record would be add into [ServiceLike] table. In a addition, our system would recommend services to user based on the common ‘like’ between different users. |

# Experiments and analysis

**Feature 1 Registration**

In Sprint 1, we implemented our own Register page on which we provide users with places to enter their personal information like username, first name, last name, email and password. After clicking submit button, user registration information will be sent from frontend to backend. Valid user information will be inserted into the User table in the testdb database.

If the username has already existed, an error will be prompted and successfully registered user will be directed to the climate service page.

In this part, we use callAPI() method to send request from frontend to backend. User registration data will also be passed to the backend in JSON format. Valid user registration information will be stored and if duplicates take place, error will be returned from backend to frontend.

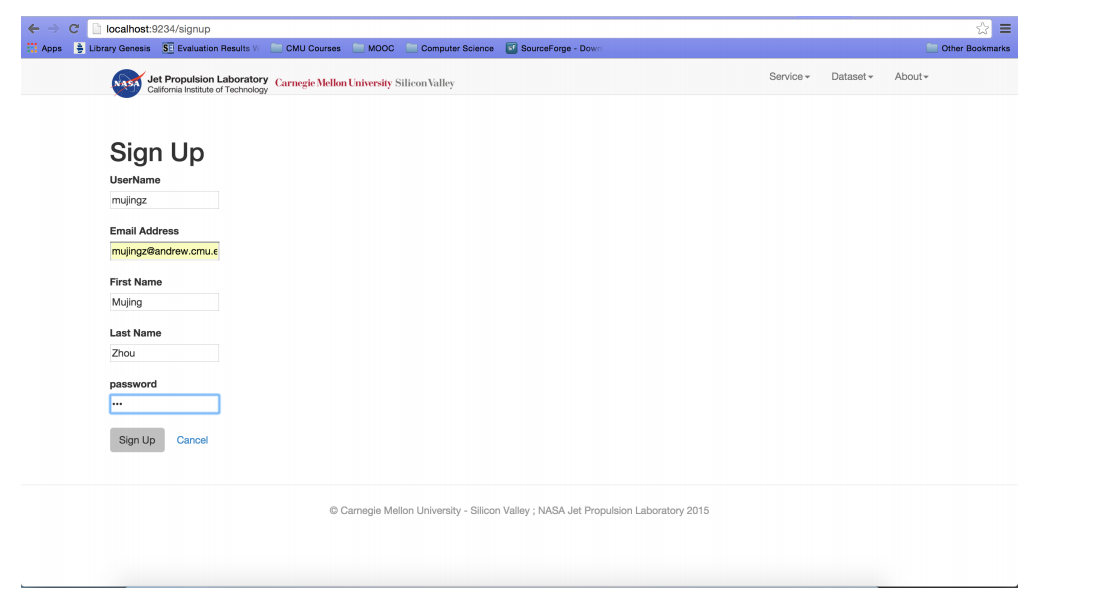
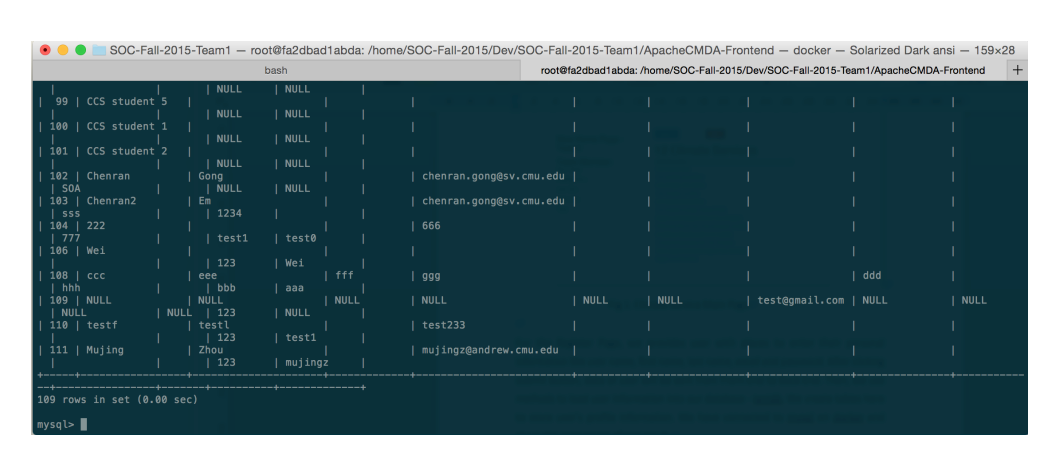


Figure 6.1.1 Register

Figure 6.1.2 Valid Registration

**Feature 2 Login**

In Sprint 1, we also implemented our own login page. In this page, user can enter his username and password to log into his own account. After clicking the “login” button, input information will be validated with the database at backend. If user login information is correct, user will be directed to his own main page. If the input login information is incorrect, user will be direct to another page that shows “Fail to login”.

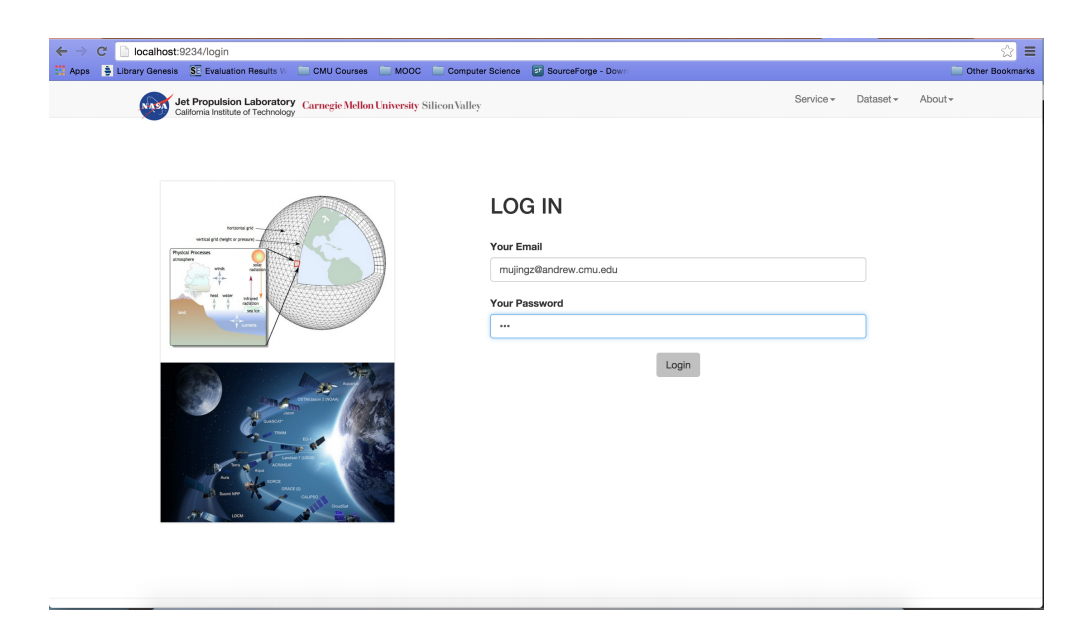
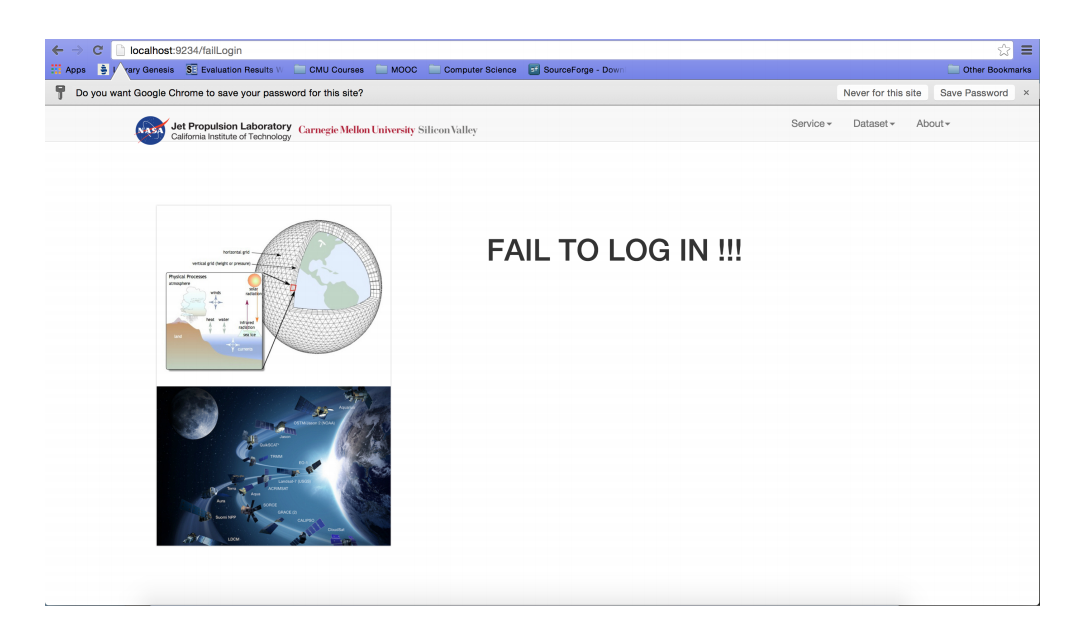
In this part, we user callAPI() method to send request from frontend to backend. Request is received by backend and redirected to UserController through routes. In UserController, input email and password will be parsed to judge if user has existed or not. Result will be sent back as response to front and be shown to the user.

Figure 6.2.1 Login Page

Figure 6.2.2 Invalid Login

**Feature 3 Make comments**

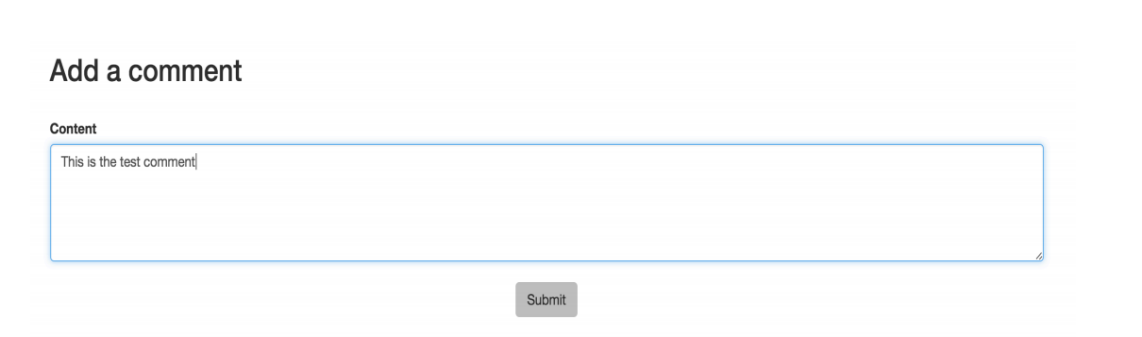
On the service list page, users can click into any one of the services in the service list. For a single service, the above input text area can be used to add a comment. After clicking submit, the comment containing the comment content, the user who makes the comment as well as the time user makes the comment will be passed to the backend and store in the comment table we establish in the database.

Figure 6.3.1 Add comments

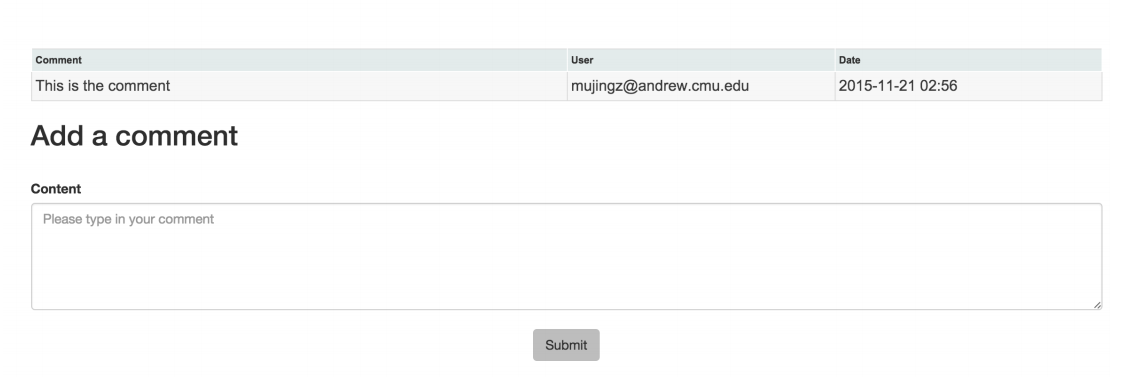


Figure 6.3.2 Display comments

**Feature 4 Make ratings**

In Sprint 2, we also implement the rating feature. Users can rate the service in the service page. We use JavaScript to implement visual rating widgets instead of just letting user enter a rating text.

The added rating will then be passed to the backend and stored in the ServiceGrade table in the database. Average rating will be computed based on the previous average rating and the total number of ratings that have been made for this service.

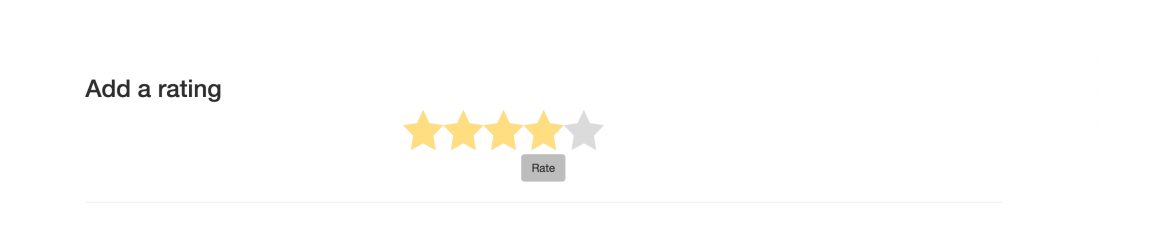


Figure 6.4.1 Make ratings

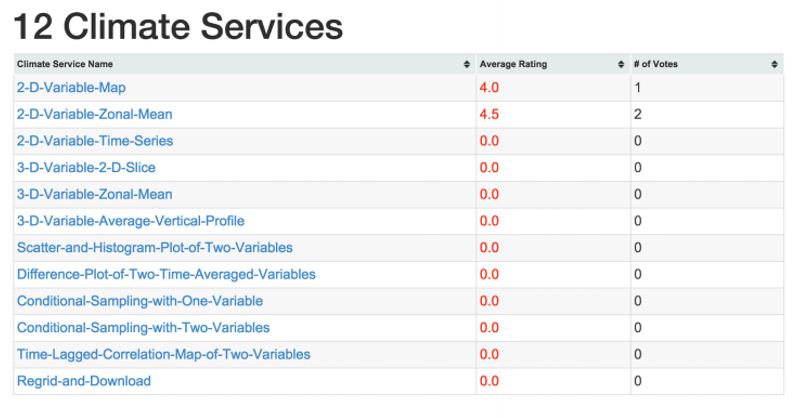


Figure 6.4.2 Display average rating

**Feature 5 Version control**

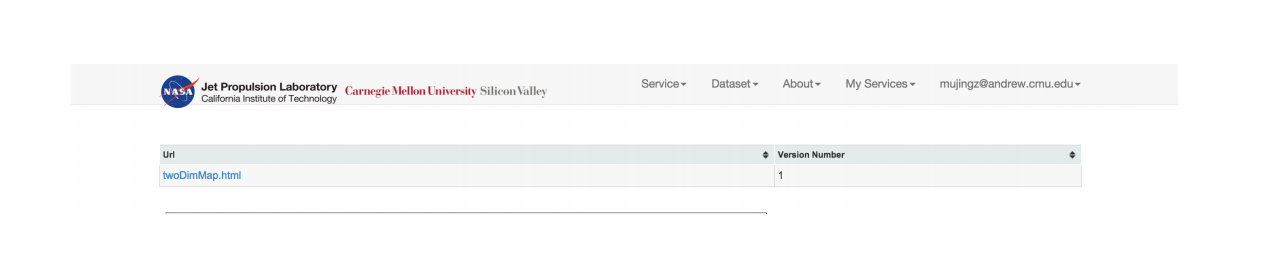
Whenusers click into a single service and all the versions related with this service will be displayed at the top of the page. User can click into any of these versions to view a specific version of the climate service. Also, the most recent version of climate service will be displayed just in this service page to give the user more convenience.

Figure 6.5.1 Version controls

**Feature 6 Search a Service**

Users who have successfully logged in can also search an available service. At the navigation bar, users can click the “Service” tab and then find the “Search Service” feature. User can enter service keywords (For example, “Variable”), and the top three services containing the “Variable” keyword will be displayed on the result page. We write the sorting code to sort all the services we fetch containing the keywords and return the top three of the search result to the frontend.

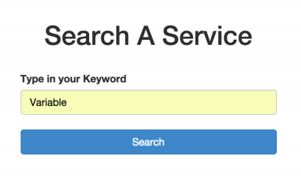


Figure 6.6.1 Search a service

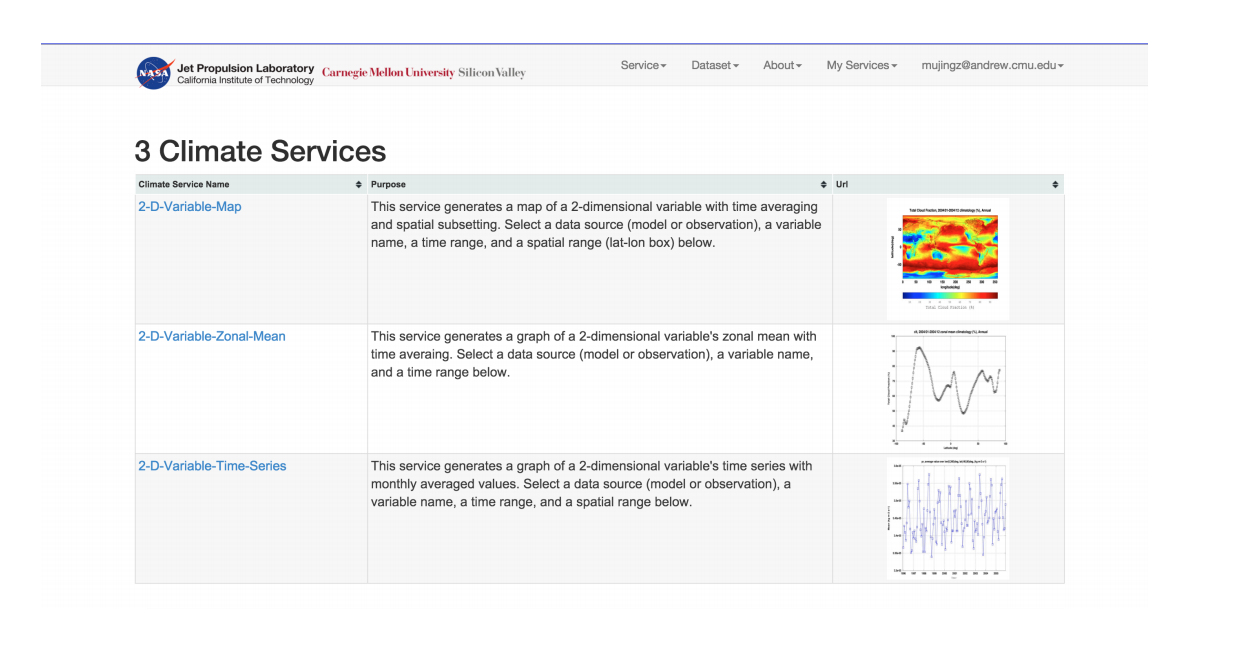


Figure 6.6.2 Search result

**Feature 7 Mention a Climate Service in Comment**

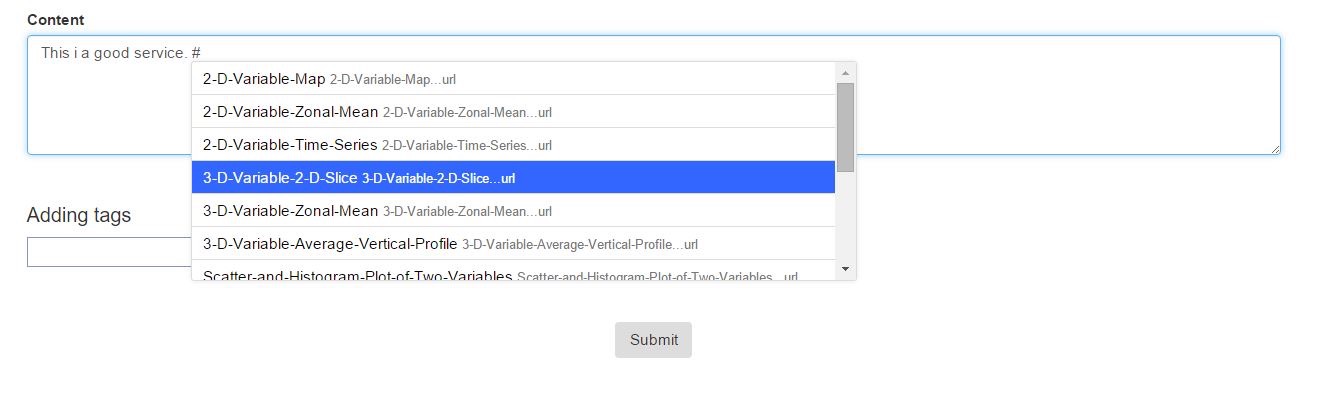


Figure 6.7.1 Mention a service

Based on the comment feature we added in Sprint 2, we add the hashtag functionality in the comment box. Figure 1 shows the screenshots of this feature. When a user is commenting on a climate service, the user can use a hashtag ‘#’ to mention a climate service. We use jQuery to implement this feature in order to show a real-time recommendation. As you can see from the screenshots, when the user types in a hashtag into the comment box, we will give a list of services dynamically. The user can then mention a climate service in his or her comment, which may indicate a relationship between two similar climate services. Figure 6.7.2 below show the results of the hashtag functionality.

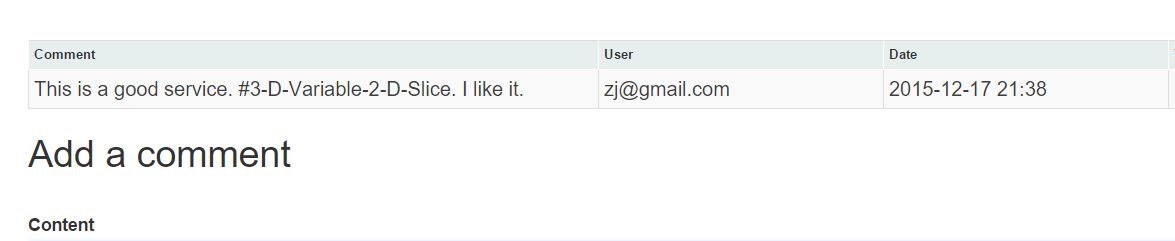


Figure 6.7.2 Results for hashtag

**Feature 8 Mention a Friend using @**

In this functionality, a user can mention a friend using a sign ‘@’. Since we team one does not focus on implementing a social network, our ‘@’ functionality will give the full list of users that have been stored in the backend database. Like the hashtag feature described above, this feature is also implemented using jQuery AJAX, which can only reload a small part of the webpage and display the required information in a real-time manner. The Figure 6.8.1 shows the screenshots of this feature.

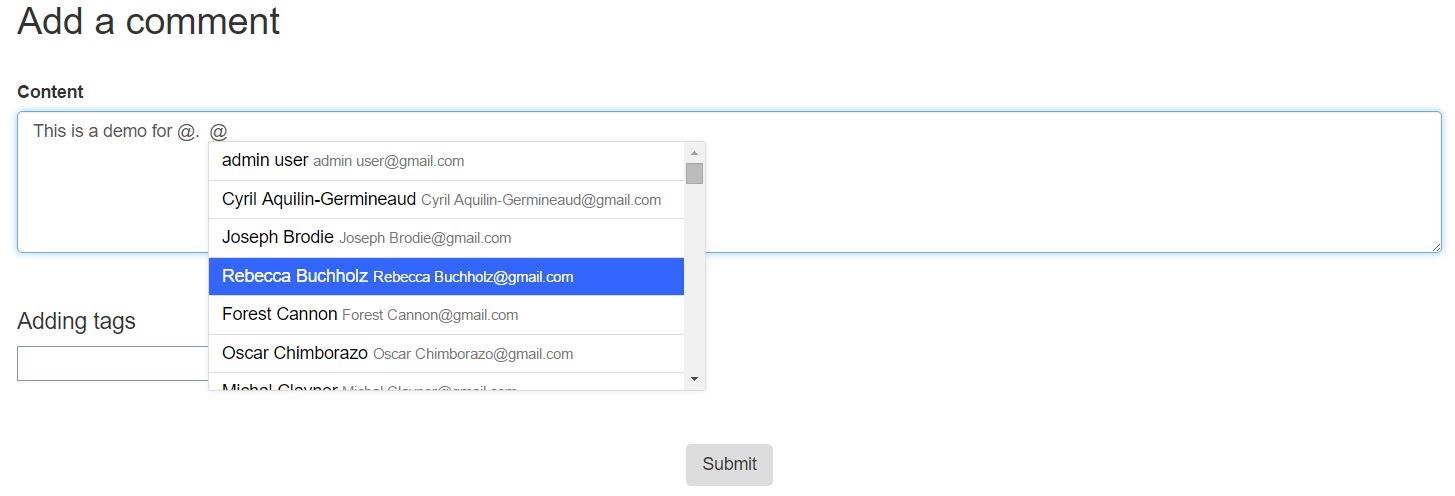


Figure 6.8.1 Mention a user



Figure 6.8.2 Results for @ a user

**Feature 9 Tag a Comment**

This feature is designed by ourselves. Our design allows our user to add several tags to a comment. As you can see in Figure 6.9.1, there is an input box below the comment area. We provide four tags (funny service, boring service, heat service, map service) for our users to add to their comments. These tags can be a kind of classification of the services. We may use this tags to provide recommendation functionality later.

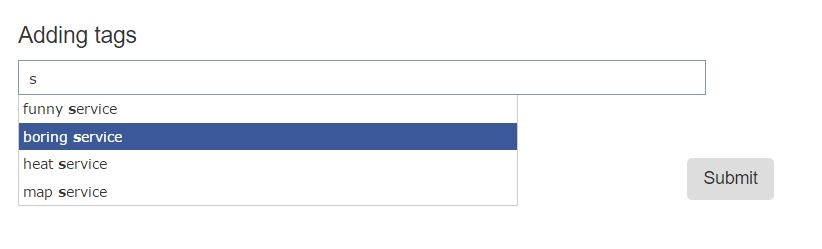


Figure 6.9.1 Add a tag

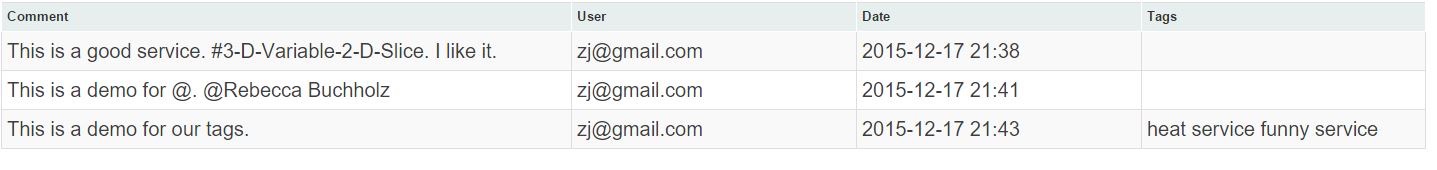


Figure 6.9.2 Results for adding a tag

**Feature 10 Like a service and recommendation**

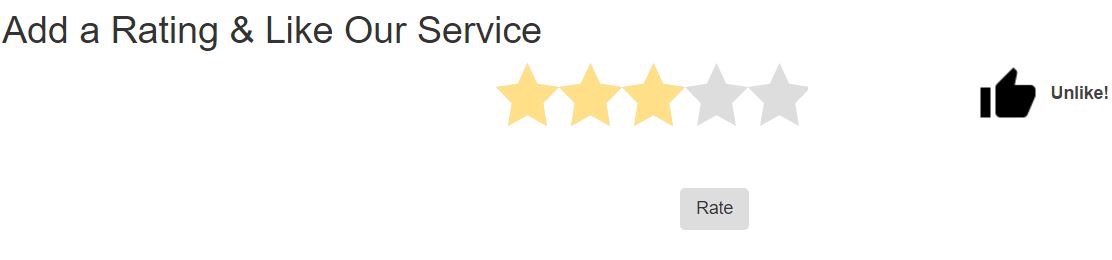


Figure 6.10.1 Like a service

This feature is designed by ourselves. In the individual service page, a user can choose to like a service when he or she rate a service. As is shown in Figure 6.10.1 above, by click on the Like button near the rating area, our user can submit his/her likeness with the rating results. Then our system will give recommendation of services based on this likeness. For instance, if Alice likes services S1, S2, S4, and Bob likes Service S1, S2. It is highly possible that Bob would also like S4. So our system will recommend service S4 to Bob. Figure 6.10.2 below is a test case for our recommendation functionality.

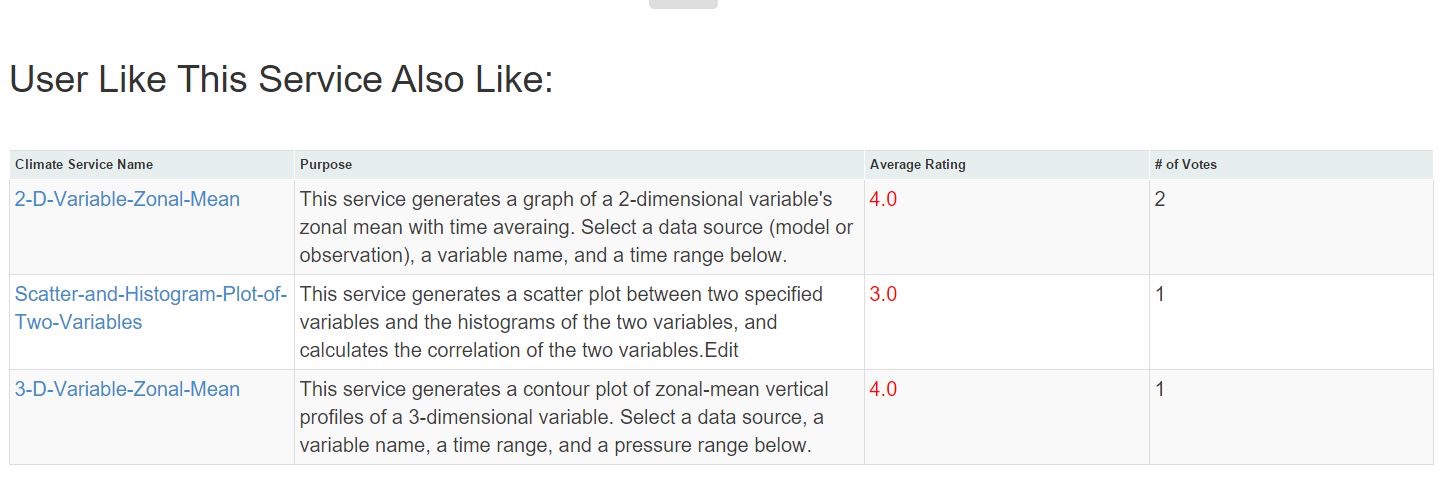


Figure 6.10.2 Recommend services based on Like

# Conclusions and future work

From this project, we valid our knowledge of service-oriented computing and learn a lot from both Frontend and Backend structure of a climate web service. We implemented and tested our project in Docker environment.

At Front-end, we handle necessary Frontend design language like HTML and JavaScript. We used AJAX technology in the implementation of “@” and “#” to refer to specific person or service. At Backend, we get familiar with Hibernate framework and understand how to map database tables to java classes. MySQL database is selected to be our Backend and we use Hibernate API and SQL to operate on database. We also gain experience of operation between Frontend and Backend and learn a lot about Play Framework.

Our climate service management system still can be improved in many aspects in the future. It should display a more user-friendly UI and provide more functionalities to enhance the interaction between users and machine.

First of all, we will design a more elegant UI for our web services. Since we only have half semester to learn frontend technologies and design the UI for our climate service management system, our User Interface can be improved or redesigned in many aspects. We will try to provide more dynamic contents in order to include interactive elements in our web services. By using Ajax technologies more widely, we could make our service more interesting and user-friendly. By utilizing more CSS templates, we can give a more exquisite UI and maintain the usability at the same time. We may provide a real-time comment and chat functionality for our system, so that our user can receive the feedback from other user as soon as possible.

Furthermore, we would like to reconstruct our climate service management system and build a serve management ecosystem. We may introduce the concept of friend circle and researchers group to provide some features from social networks. As is known to all, researchers and scientists need to communicate with each other, exchange their up-to-date research outcomes, in order to accelerate the steps of innovation and discovery. We hope more elements from social networks in our web service can help this communication. In addition, since researchers are often gathered into different groups who show interests and concentrations on different topics. By introducing the mechanism of researchers group, we can push and recommend our climate service in a precise way.

Finally, we plan to build a more sophisticated and powerful recommendation system. We will design more metrics in our management system. For instance, we could log how many times that a service is viewed in the last one month and rank the services by the viewing times. Depending on more and more metrics we collect from our web service, we can create a more powerful recommendation system and provide more precise suggestions on our services. Thus we can improve the productivity and work efficiency of our user.

# Contribution of each team member

**Mujing Zhou** - Implementation of “@” (friends mention function) and “#” (special service mention function) functions. UI design and front-end implementation of register, service rate, tags functions.

**Run Yang** - implementation of login function back-end; database design and operation of keyword search function and service version control; back-end database design and ‘save’, ‘get’ methods implementation for tags functions.

**Jian Zhu** - front-end design and implementation of login, keyword search, service version control functions. UI design and function implementation of ‘like’/recommendation functions.

**Yibing Yan** - back-end design and implementation of register, service grading. Design of database schema and implementation of ‘like’/recommendation function back-end.

# Tutorials

1. Make sure you have installed Docker correctly.
2. Open Oracle VirtualBox and select the “default” VM. Choose the Settings menu.
3. Choose the Network submenu and edit the Port Forwarding configuration.

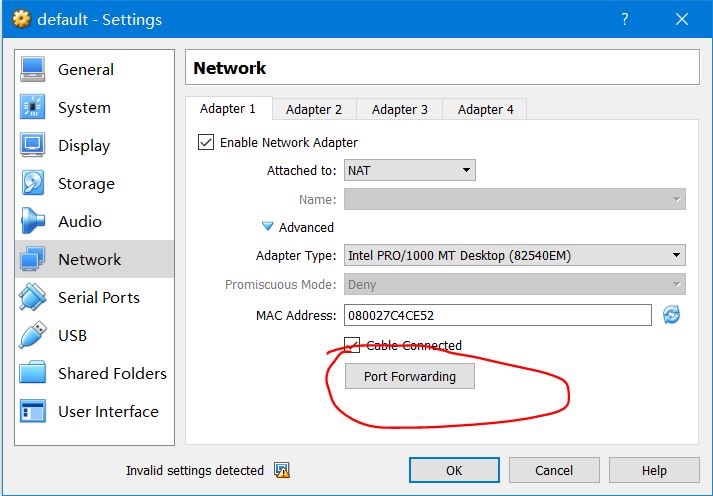


Figure 9.1

1. Set up port forwarding from guest port 9234 to host port 9234

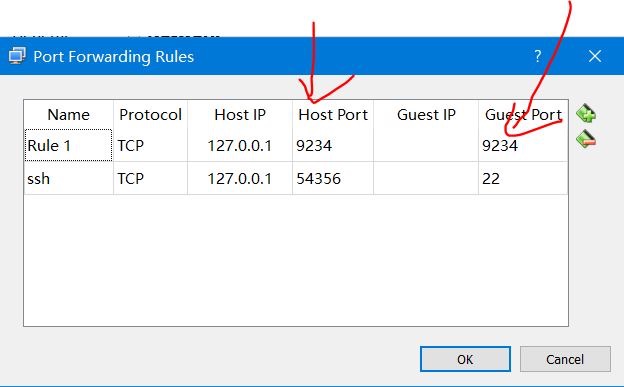


Figure 9.2

1. Create a development directory in your local machine and put the project under the directory, e.g., C:\Users\GitHub\Final\SOC-Fall-2015-Team1

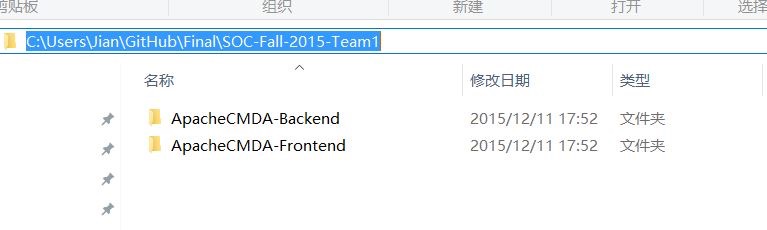


Figure 9.3

1. Open Docker Terminal and mount the local directory to the container by

docker run -it -p 9234:9000 -v /c/Users/GitHub/Final:/home/SOC-Fall-2015/Dev cmusvsc/apachecmda:1.0

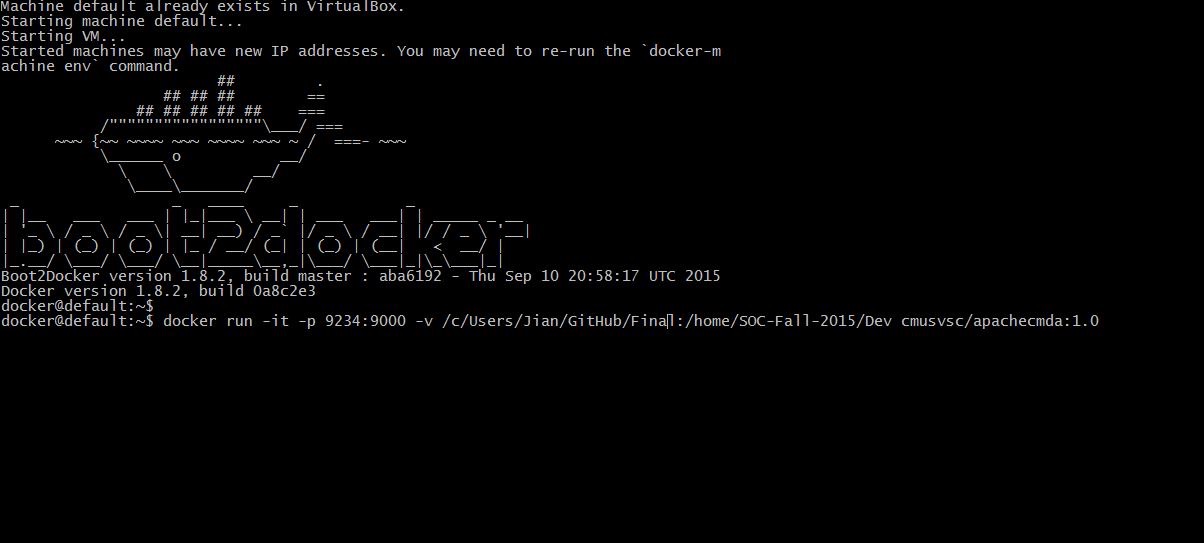


Figure 9.4

1. Change the current directory to the backend directory ApacheCMDA-Backend

cd /home/SOC-Fall-2015/Dev/SOC-Fall-2015-Team1/ApacheCMDA-Backend

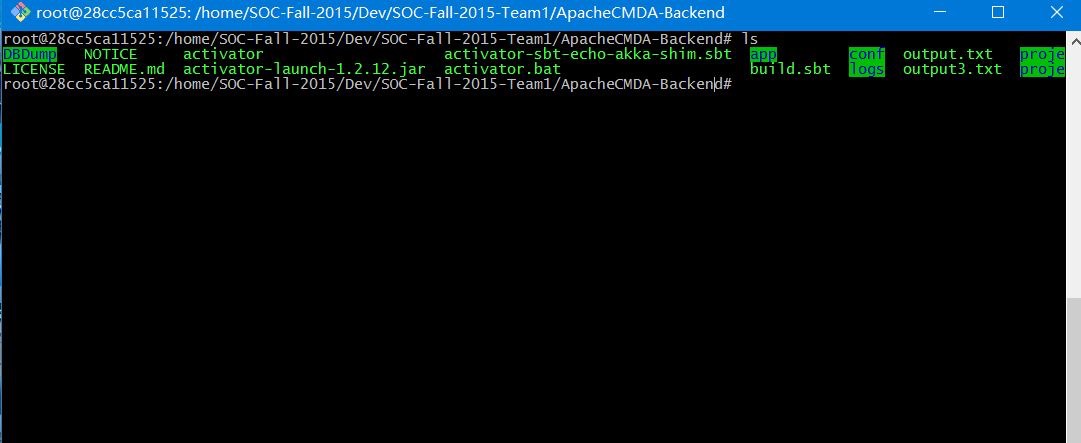


Figure 9.5

1. Run the MySQL database.

sudo service mysql start

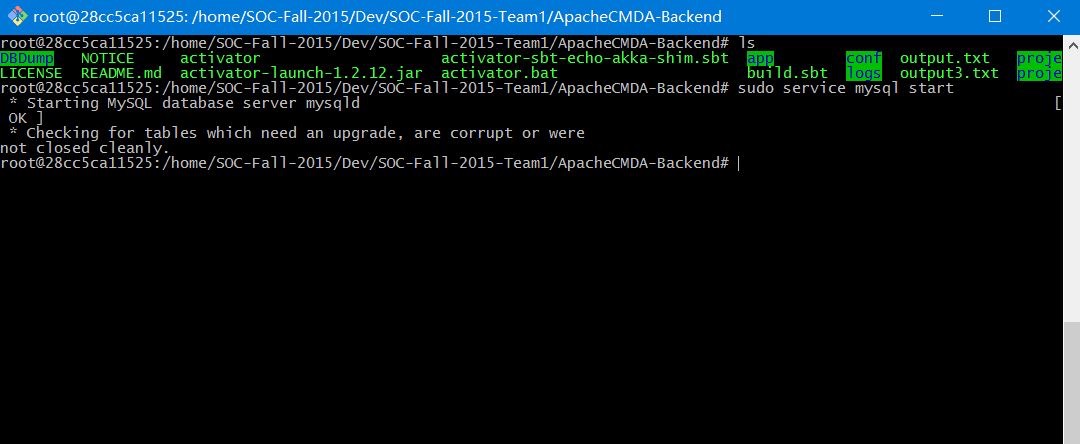


Figure 9.6

1. Create a Screen Session to start the backend.

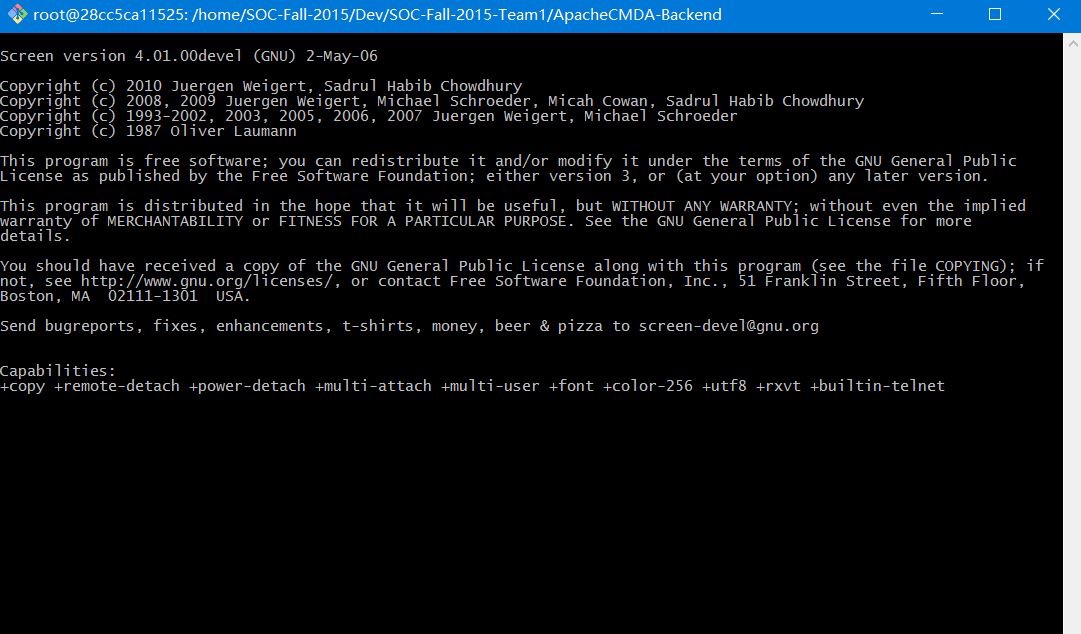


Figure 9.7

1. Start the backend server with port number 9034, then detach the session

./activator "run 9034"

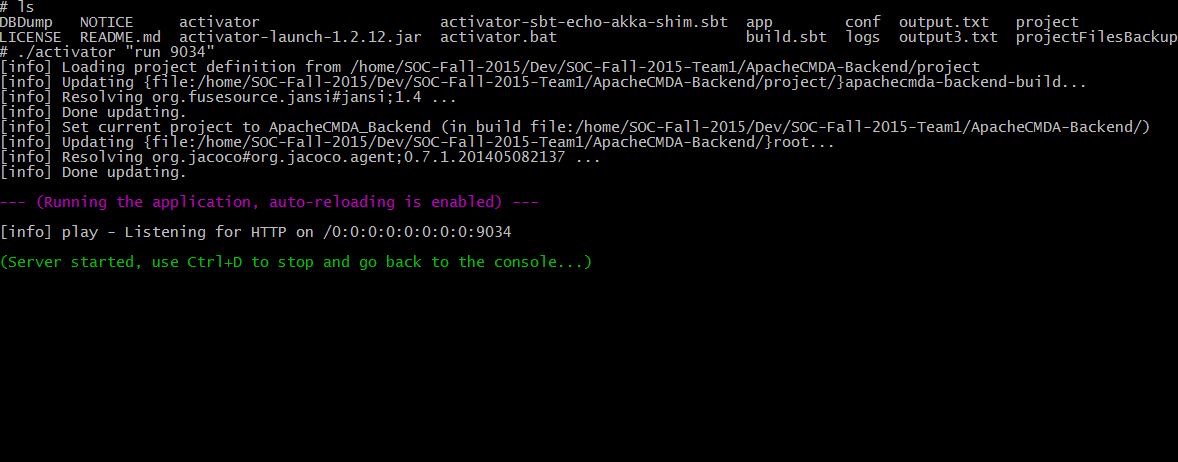


Figure 9.8

1. Detach the session by: CTRL + A + D
2. Go to the Frontend directory

cd /home/SOC-Fall-2015/Dev/SOC-Fall-2015-Team1/ApacheCMDA-Frontend

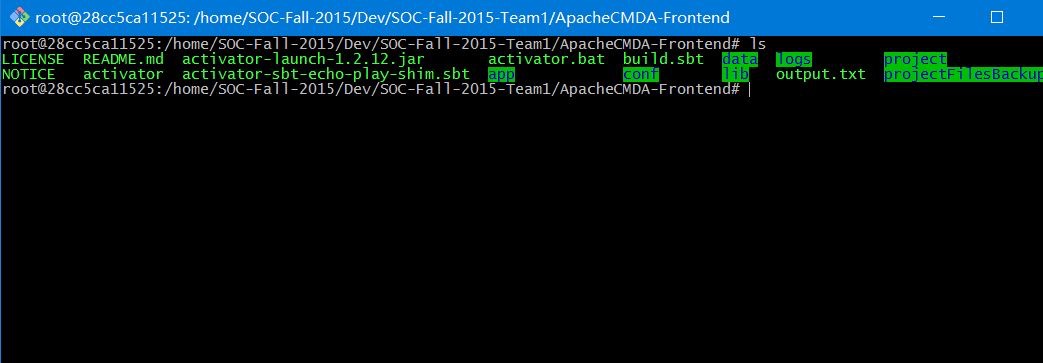


Figure 9.9

1. Run the frontend server: ./activator run

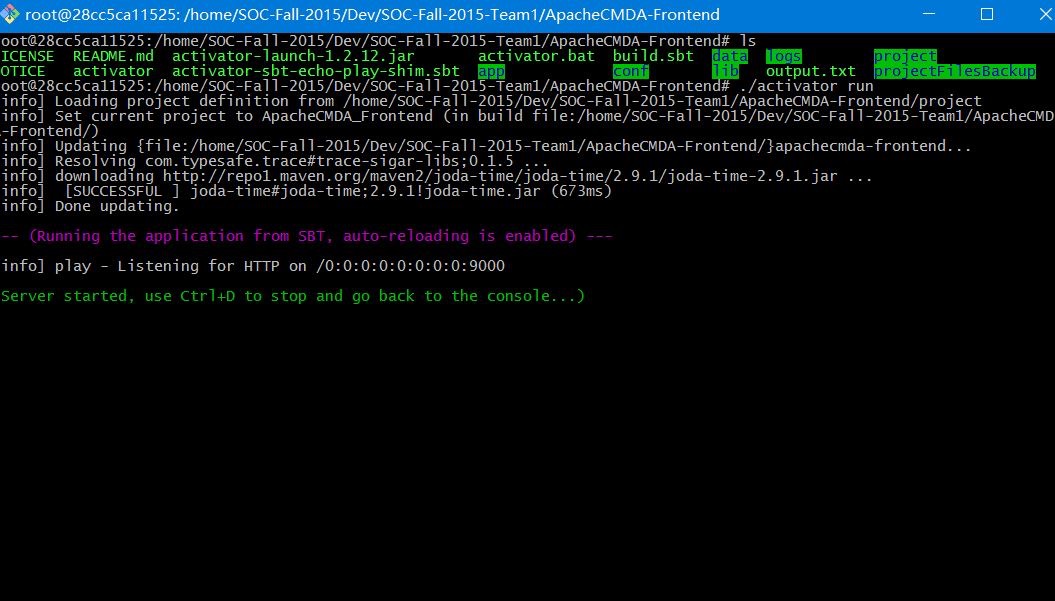


Figure 9.10

1. Open the URL below using your host’s browser:

<http://localhost:9234/>

1. Click on the Login button on the top right corner of the web page. You will be redirected to the Sign In page. If you have not signed up for our system, choose Sign Up a new account.

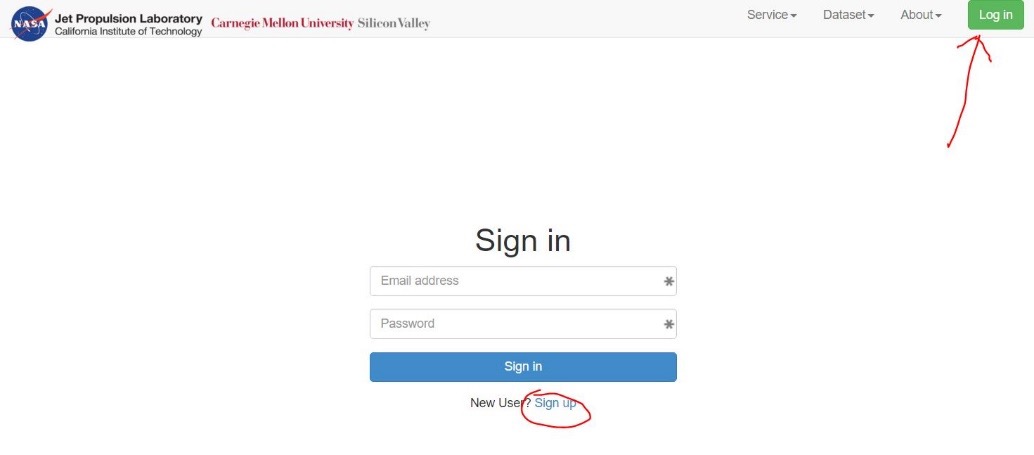


Figure 9.11

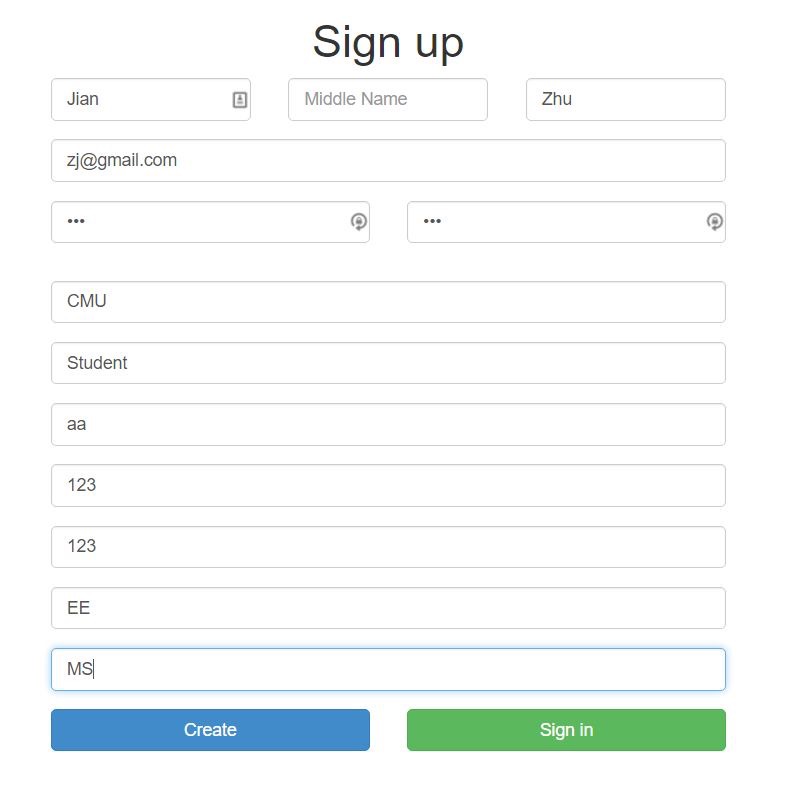


Figure 9.12

1. You will need to use the email and password you just registered to sign in our system. After signing in the system, you will be redirected to the home page of our climate service management system. Now You can choose the function from the menu in the header of the web page.

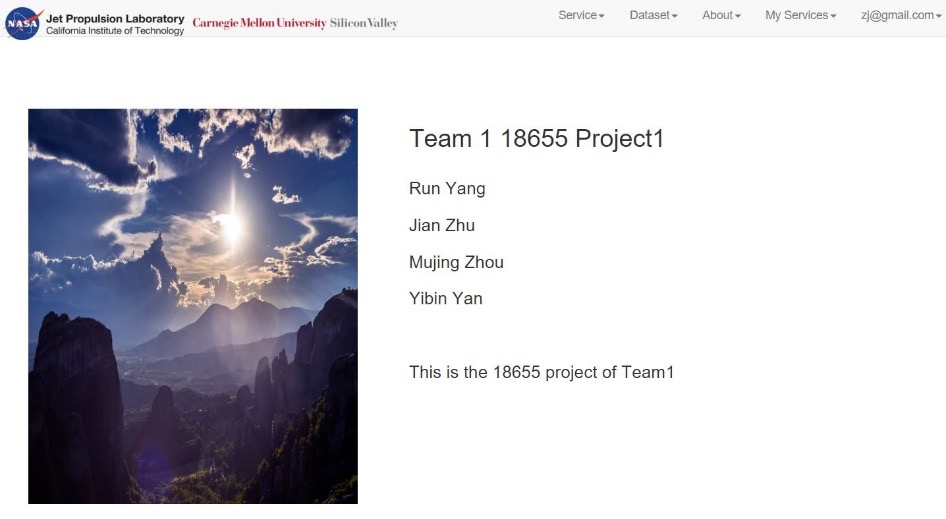


Figure 9.13

1. In the header of main page, you can click on “Service” to view the menu.



Figure 9.14

1. Choose Service List, and you will be redirected to our new service list page.

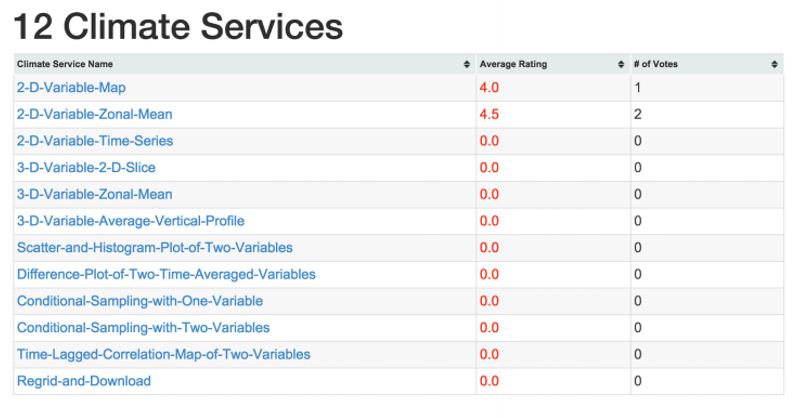


Figure 9.15

1. Choose one of the services, you can go to the individual service page.



Figure 9.16

1. You can comment, tag, mention a service with #, mention a user with @.

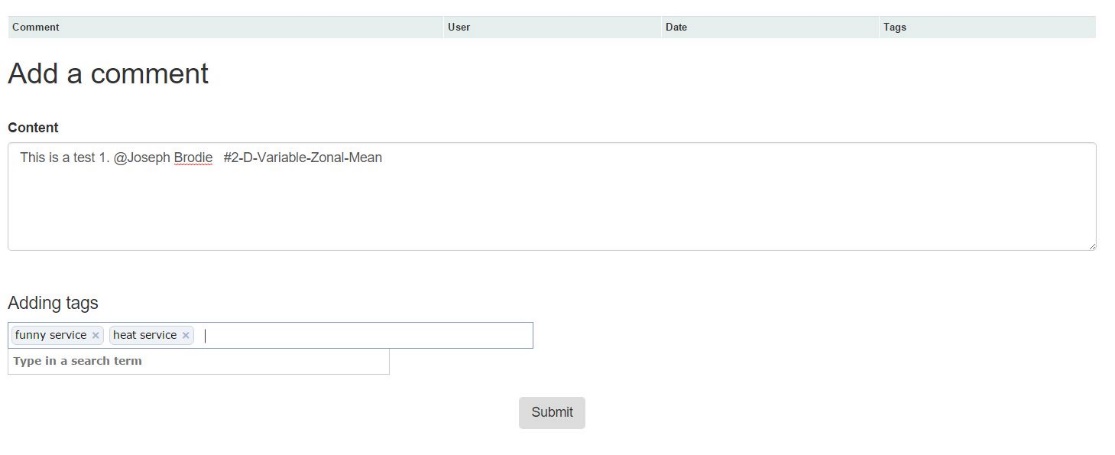


Figure 9.16

1. You can also rate and like a service.

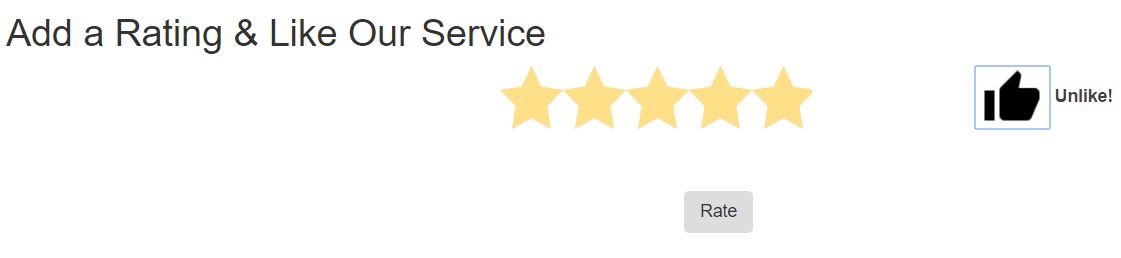


Figure 9.17

1. Choose the “Search Service” in the header menu. You can search the key word.



Figure 9.18