**18655 Final Project Report**

**Project 3 workflow-centric scientific social network**

**Team 7**

**Team members:**

**Huanwen Chen**

**Ge Jin**

**Jiewen Ma**

**Team 8**

**Yiming Zhang**

**Lai Wei**

**Hsueh-Hung Cheng**

**Yangdi Zhou**

1. **Introduction**

In this project, we would like to build a social network and forum that scientists collaborate and communicate on scientific workflows or scientific experiments. A scientist publishes and shares her experiments, and other scientists may either communicate or comment on the entire workflow. In other words, the project provides a platform for scientists to post their scientific experiements and workflows, so that other scientists who are interested in the workflow can redo the experiements and make comments. Like any other social network web application, this project supports multiple user interaction.

In this whole project, we mainly focus on workflow system and user system. Based on the former work, we have designed a more specific and powerful system. Until now, we have make it true of several functions, like: user login and signup functionality, user profile, user group functionality, user access control, subscription functionality, workflow popularity, workflow list, adding and deleting workflow, commenting, marking answers and so on.

In the whole process of this project, we also use many technique support to help the project work better. We use Trello Board to divide up all the missions and targets we’d like to realize. It is also a reminder that reminds us when and what to do. We use Docker as a virtualization platform to run the project. In this project, we use MVC framework to connect each part. And more specicfic, we use Hibernate for mapping all the object-oriented domain models to to database in the backend. And we use scala language to programing the front end.

1. **Motivation**

Workflow management has become such a common need that most companies have multiple ways of creating and scheduling job internally. At the same time, a workflow can be useful not only for resolving business but also for scientific issues. However, nowadays most of the platforms of workflow mainly focus on the bussiness use, and workflows in science are seldom well utilized in a proper way. That’s why we need to create a platform for scientist to discuss workflows.

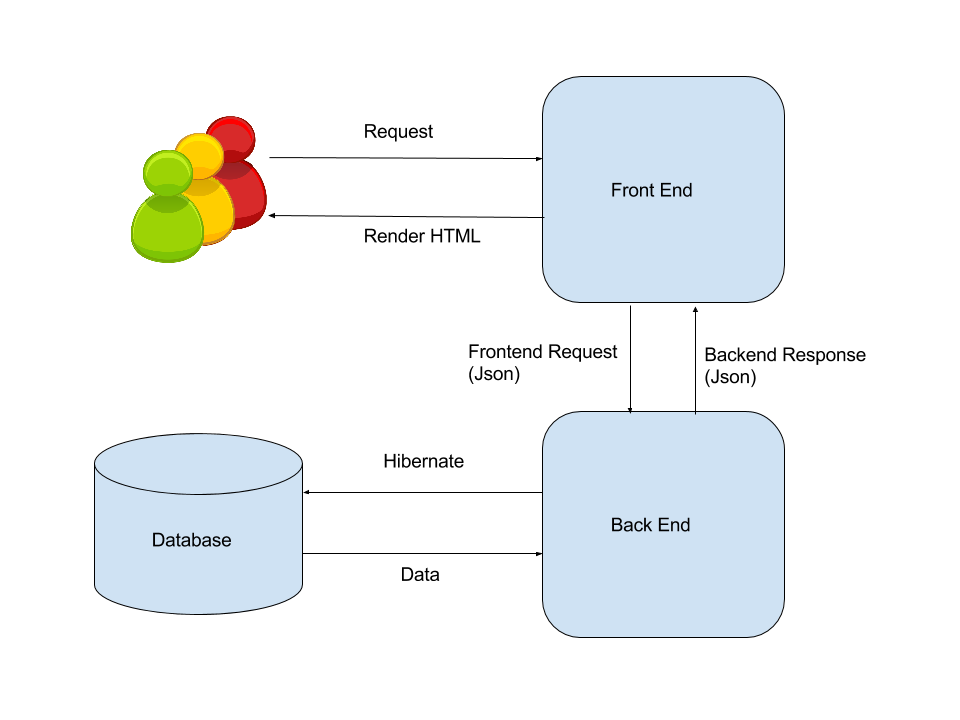
A workflow platform does not, by definition, have the full set of freatures offered by today’s typical workflow products. Rather, the workflow platform considered here focuses on supporting the concept of a workflow as a model of the organization of work item as well as the discussion among scientists in a specific group internally. Therefore, the workflow platform has multple points of interaction. All the user can post workflow, which can be reviewed and discussed by other users via leaving comments. In addtion, different usesr belong to different groups, so that scientific issues can be dealed with effectively and according workflow can be revised via authority aprroved. Moreover, good and impressive comments can be marked so that the users develop the workflows can get encoragement or improve their workflows accordingly.

All in all, the purpose we want to achive is to develop such a platform of workflow that is well organized, full of interaction and really helpful for managing scientfic issues among scientest groups for multiple projects.

1. **Related work**

This project is basically related to the Apache project - Apache Climate Model Diagnostic Analyzer (CMDA), which provides web services for multi-aspect physics-based and phenomenon-oriented climate model performance evaluation and diagnosis through the comprehensive and synergistic use of multiple ovservational data, reanalysis data, and model outputs. Since CMDA is an incubation project, we develop our project based on what have done in CMDA, indicating that we develop a workflows’ platform for the scientists of CMDA into their workflow discussion.

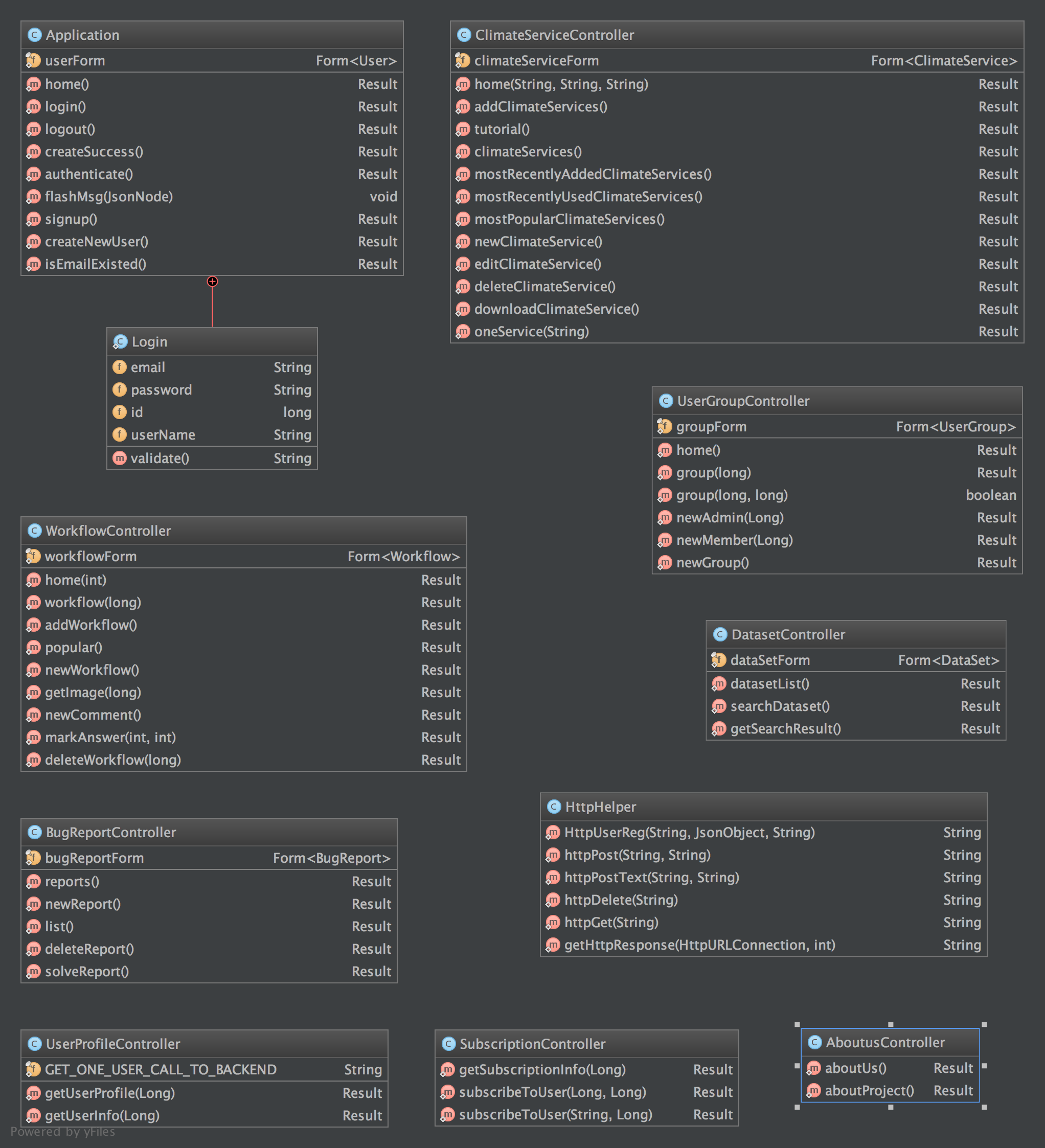
1. **System design**



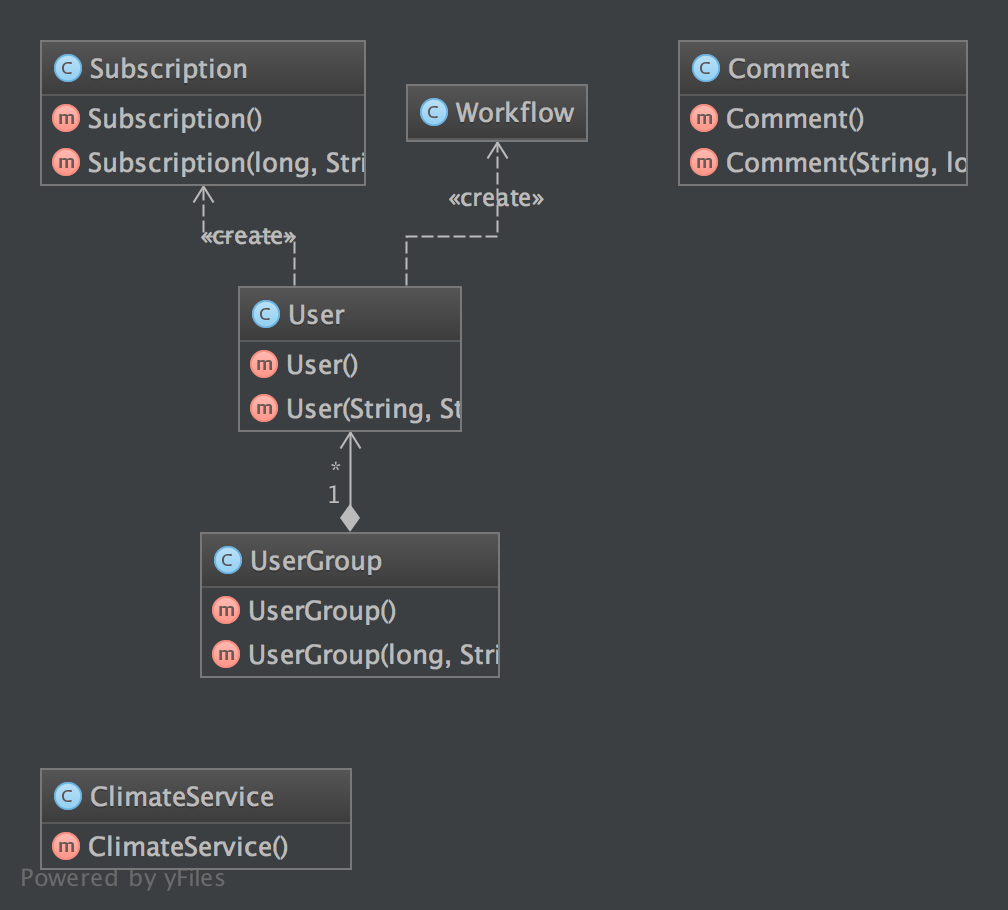
When client send HTTP request to the front end, front end controller use the method in the corresponding model class and call backend API to send request to backend. Backend controller look into the database and populate or fetch the data into or from database. Controller then read data into model, serialized the model into json format and send to front end as response. The front end controller receive the reply and then render the HTML to the client.

**4.1 Class Diagram**

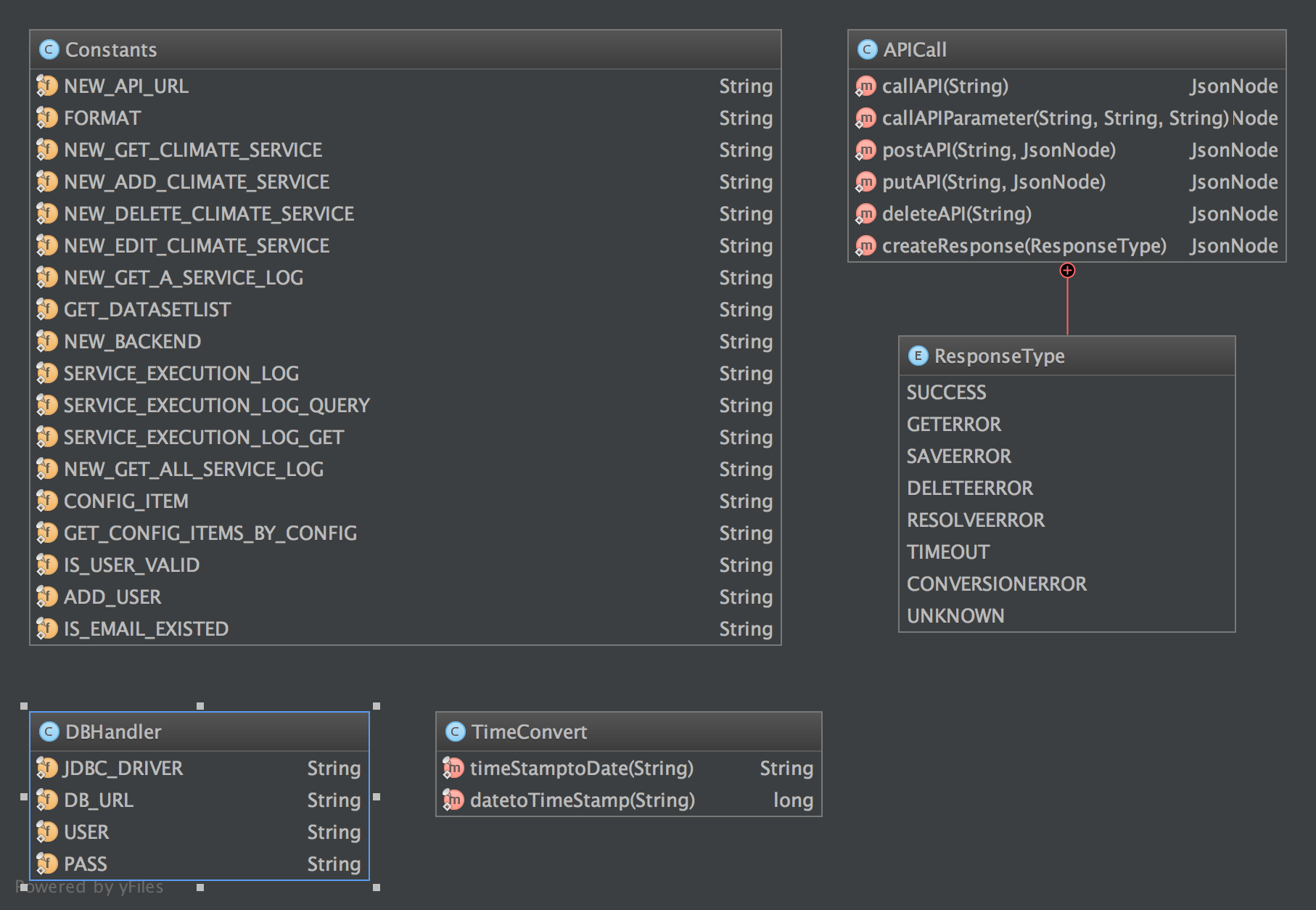
Following is the class diagram of our controllers package, on top of the code base provided, we have implemented several classes including: WorkflowController, UserProfileController, UserGroupController, SubscriptionController. In addition, we added methods of login and authenticate in ApplicationController. These controllers are the Controller part in the MVC framework which in charge of manipulate models and update views.



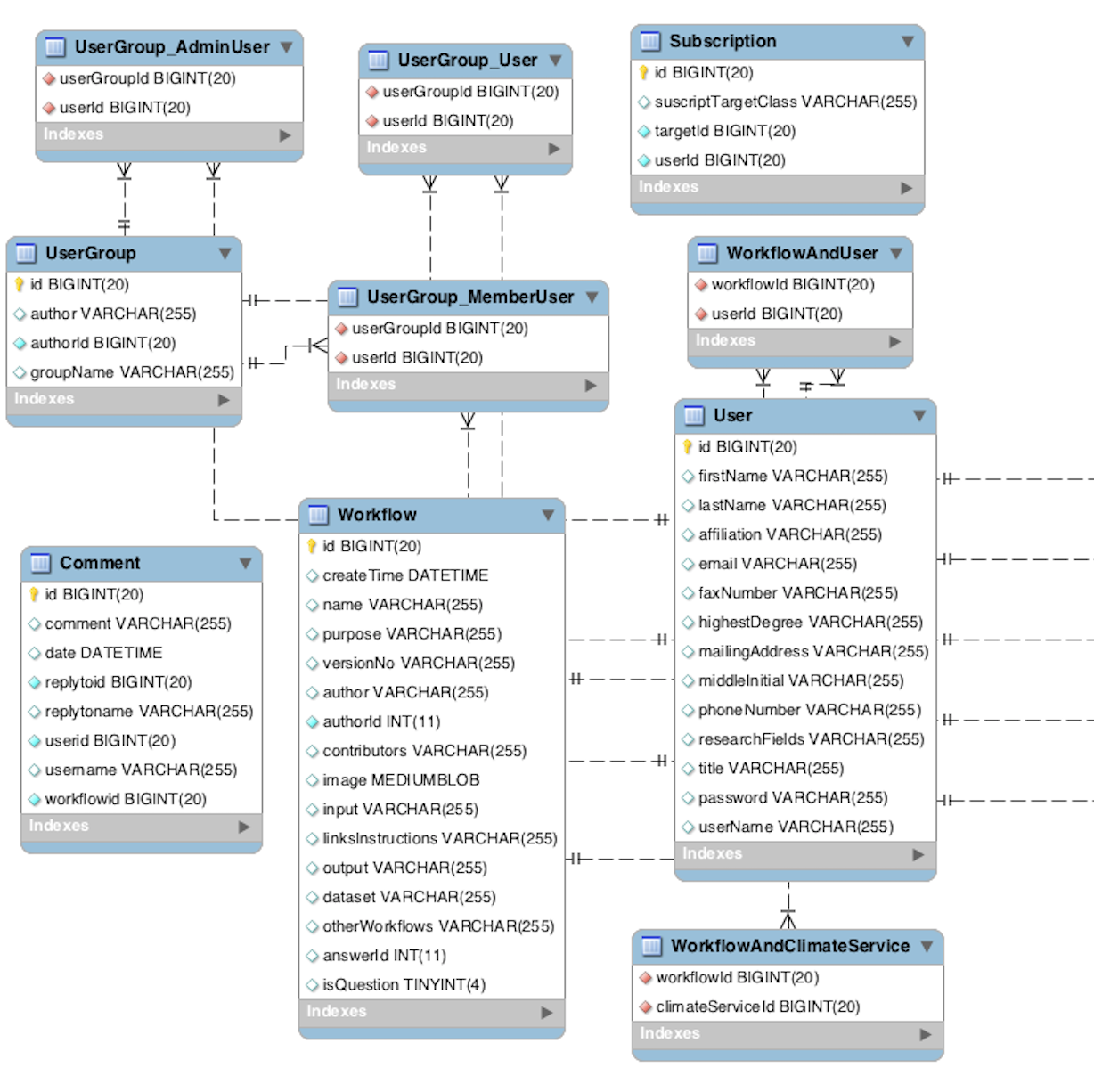
The figure below demonstrated the structure in our Metadata package under Models. They store contents and data that will be used by this project and were also used when accessing the back end.



Following is the class diagram of the util Package, the APICall class was used when making API calls to the back end, and the Constant class were used to store frequent used constants.



**4.2 Database Design**



Above is our database design UML. This UML only shows the new tables that we created for the project.

To implement the functions for workflows, we created the Workflow table to store the properties of the workflow, such as name, version number, contributors, image, etc. The other information of the workflow (Which climate services is included, which user is related to the workflow) are stored in separate tables (WorkflowAndClimateService, WorkflowAndUser), connecting to the workflow table using foreign key.

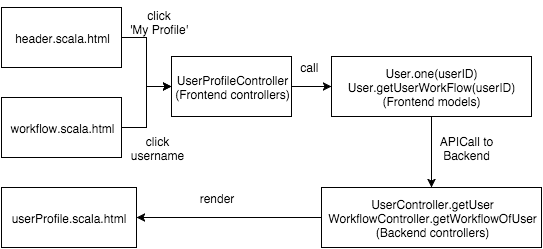
To implement the functions for UserGroup, we created the UserGroup table to store the properties of the group, such as author, description, etc. The group-user relationship is stored in 3 tables, connecting to the usergroup table using foreign key. The UserGroup\_AdminUser table stores the administrator member list information of a group; The UserGroup\_MemberUser stores the ordinary member list information of a group; The UserGroup\_User stores the creator information of a group.

As for the Subscription functions, we set a column named subscriptionTargetClass to store which kind of information is being subscripted(Workflow, UserGroup, User, etc.). It also stores the subscriber’s id and user id.

For the comment functions, we create the table comment. It stores the comment information, the id of the workflow that the comment belongs to, and also the replyToID information, so that it can be displayed in the proper workflow page, with the proper reply relationship.

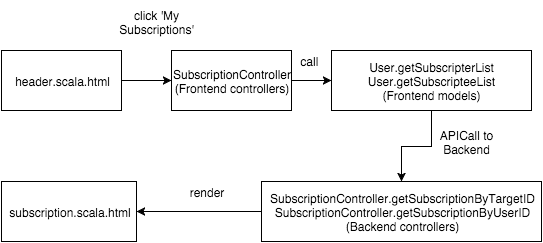
1. **System implementation**

**User Profile Implementation**

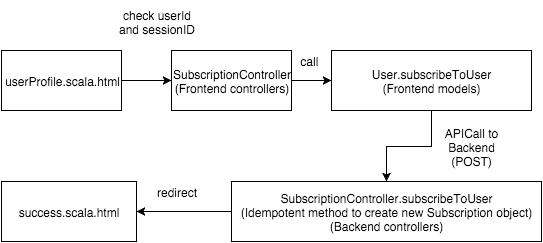


The user profile can be accessed from multiple places: the navigation bar, workflow page, and subscription page. The frontend server will fetch the User object from the backend server and render the content with userProfile.scala.html. Currently it does not support profile edit and deletion, so the users are only allowed to register and view profiles.

**Subscription Implementation**

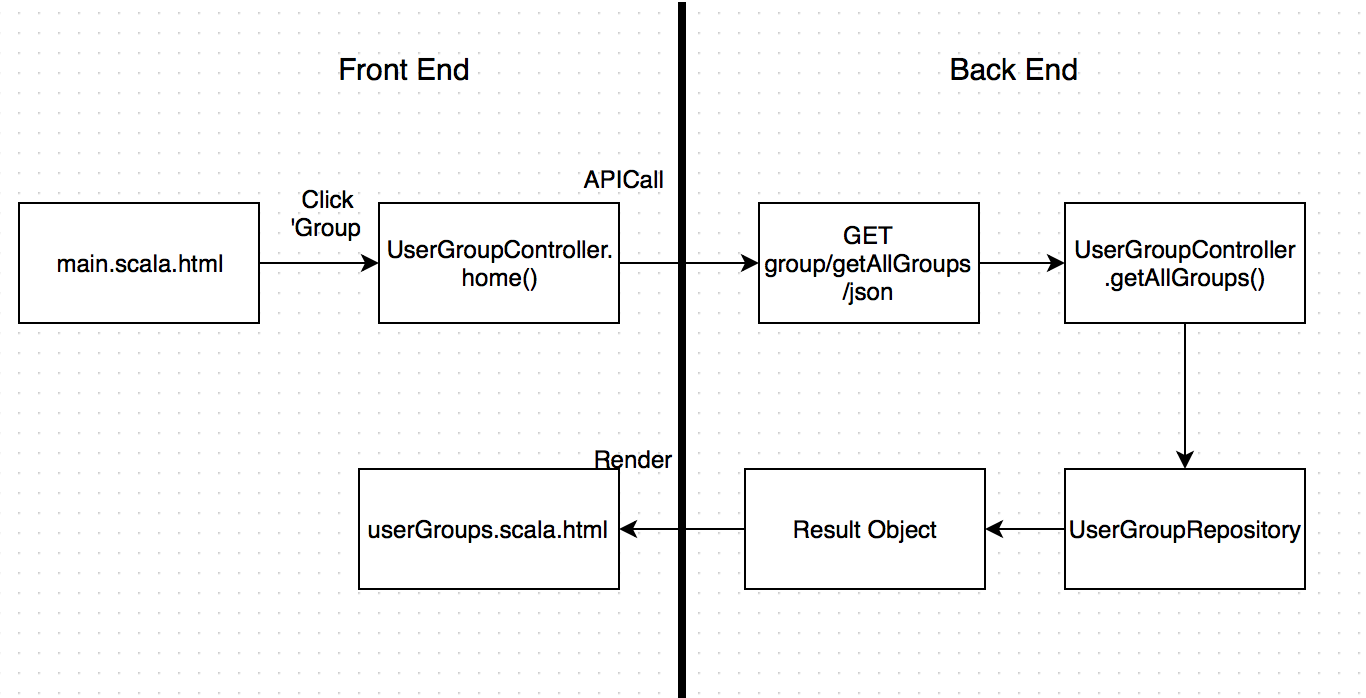


The above diagram shows the flow to get the list of a user’s subscription. Listing a specific collection’s members is a common operation in RESTful design, so it will not be explained in detail. Then the list of subscription will be rendered via subscription.scala.html.

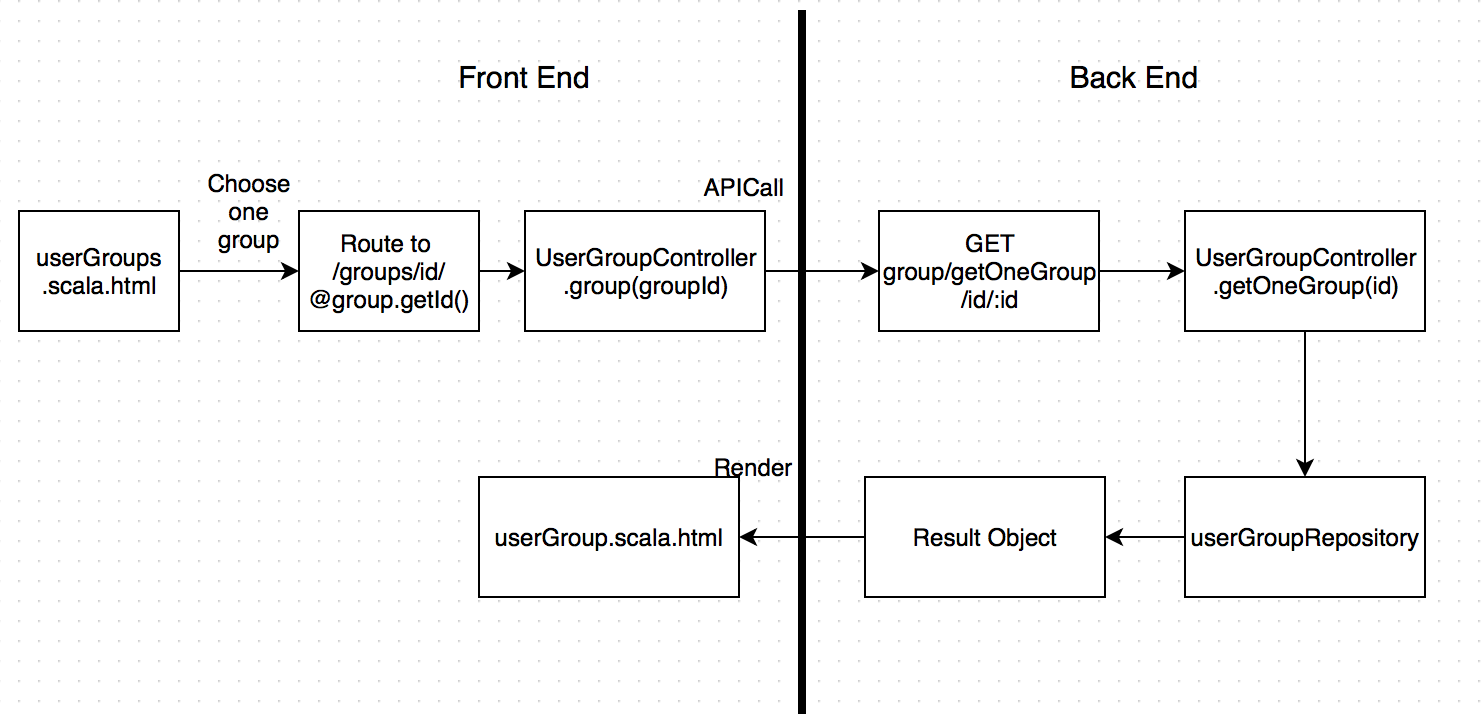


To subscribe a user, the follower has to click the subscription button on the followee’s profile page, and the request will be processed by the SubscriptionController. A recorded will be added into the subscription table.

**UserGroup Implementation**



User group can be accessed in main page by clicking on the “Group” button in the header. The page will be redirected to the the front end controller, which calls home() method and makes an API call to back end router on the URL “group/getAllGroups/json”. The router will calls the back end controller on getAllGroups() method, which makes a query to the database and get the group list as a Result object. Then the object can be rendered to the userGroup.scala.html and serves as a view on the browser.



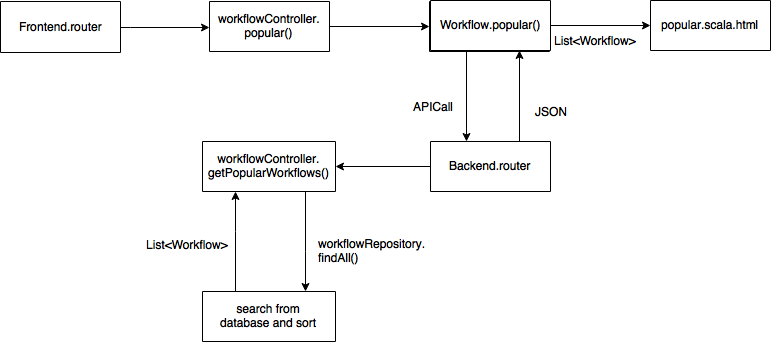
Within userGroup page, a list of available groups is rendered on the browser. User can select a group on the page to view the group details, or create his/her own group on the page.

When clicking on one group, the page will call the front end route on “/groups/id/:id”, which calls the User Group front end controller’s group() method. It makes an API call to the backend router to perform a GET operation on the URL /group/getOneGroup/id/:id, which calls the backend controller’s getOneGroup() method. This makes a query to the backend database, and get the group information as a Result object, which is then rendered to the userGroup.scala.html.

At the end of userGroup page, there is an area for creating a user group. A user can type in the new group name and click “create group” button. Then the page would refresh itself and a new group will be created. The creator is by default the administrator of the group.

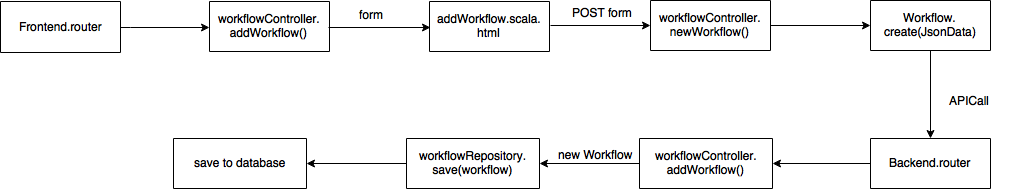
Within a group page, the basic group information is provided. When the creator is visiting this page, he/she will be shown the additional content to add new administrator or add new ordinary member. When a member logs in and visits this page, he/she could only visit the basic information. When a user is not logged in or he/she is not part of the group, he/she will be redirect to the error page, stating that he/she does not have the proper permission to view this information.

**Popularity Implementation**



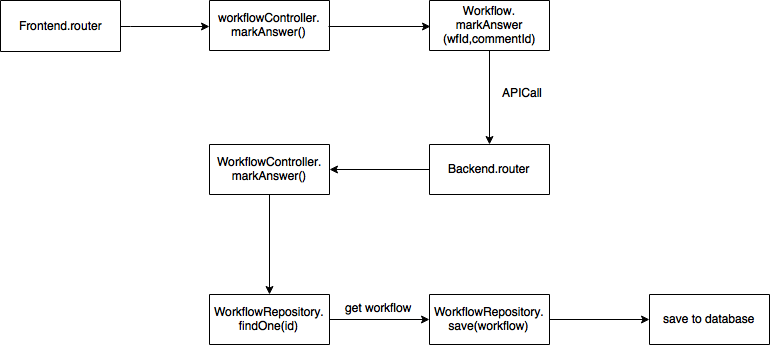
We provide a page with which the user can browse the top popular workflows. The popularity is calculated simply based on the number of views. The more views the workflow has, the more popular the workflow is. When accessing popularity browsing page, everything is identical as browsing the workflows except that the workflows are sorted based on their popularity.

**Post Workflow**



When client want to post a new workflow, he access the /workflow/new/workflow, filled out the form and post to the same url. When front end receive the post request, the newWorkflow() method is called and form data are synthesized into a model, then serialized into a json format string and sent to backend/workflow/newWorkflow API. Backend controller addWorkflow() is called, a new workflow model is created and use the Workflow model class’s save method to populate the data into database.

**MarkAnswer Implementation**



Our implementation is different from the requirement because we misunderstand its meaning. In our implementation, a user can mark a workflow as a question during the creation. The author can pick up one comment associated with the workflow as the answer. The operation behind this functionality is simply to set the comment id in a workflow.

1. **Experiments and analysis**

We tested our project mainly manually right after each sprint was completed and before the demonstration to the professor and teaching assistants. Based on the results of experiments and testing, our implementation works very well as our expectation. Problems may occur if the user types unexpected inputs or values. Such situations are prevented by JavaScript on the client side and Java on the backend side. However, it is still possible that some corner cases are not covered. So continuous testing and debugging are required in the future.

In terms of the performance analysis, our project may not work well as the traffic and the number of users grow. It is because we used only one database and one backend server in our design. For better scalability, multiple replications and a load balancer are needed to handle the growing traffic.

1. **Conclusions and future work**

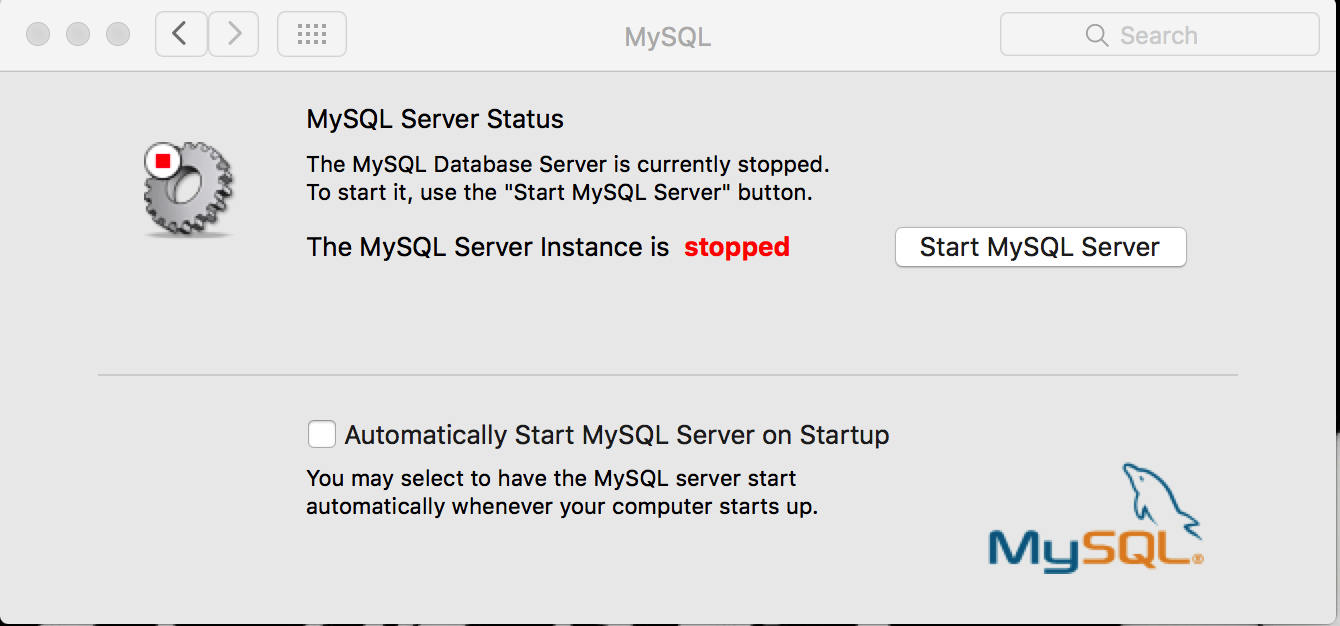
Although there are many complex functionalities in this project, we complete this job at last. Thanks to the professor Jia and TAs. Each time we have problems or difficulties, we usually search help from them and they could give us huge help. From this project, we have learnt a lot like how to run a real service, how to use MVC to build the whole structure, how to allocate tasks correctly and conveniently, how to merge two projects together from different versions and so on. And I think it is so useful for our later work in life.

But there still exist some other work that we would prepared to do but did not do, such as: it needs a search page on the workflow list that we can type in the author’s name or workflow’s title and then it can return a list of workflow. What’s more, we can also add one functionality that when leaving comments in one workflow, he can also mention another user like facebook or twitter’ sign @. This functionality makes the interoperability between users better and more convenient. And one functionality of sharing could be also added into this project, to make the service more popular to the public.

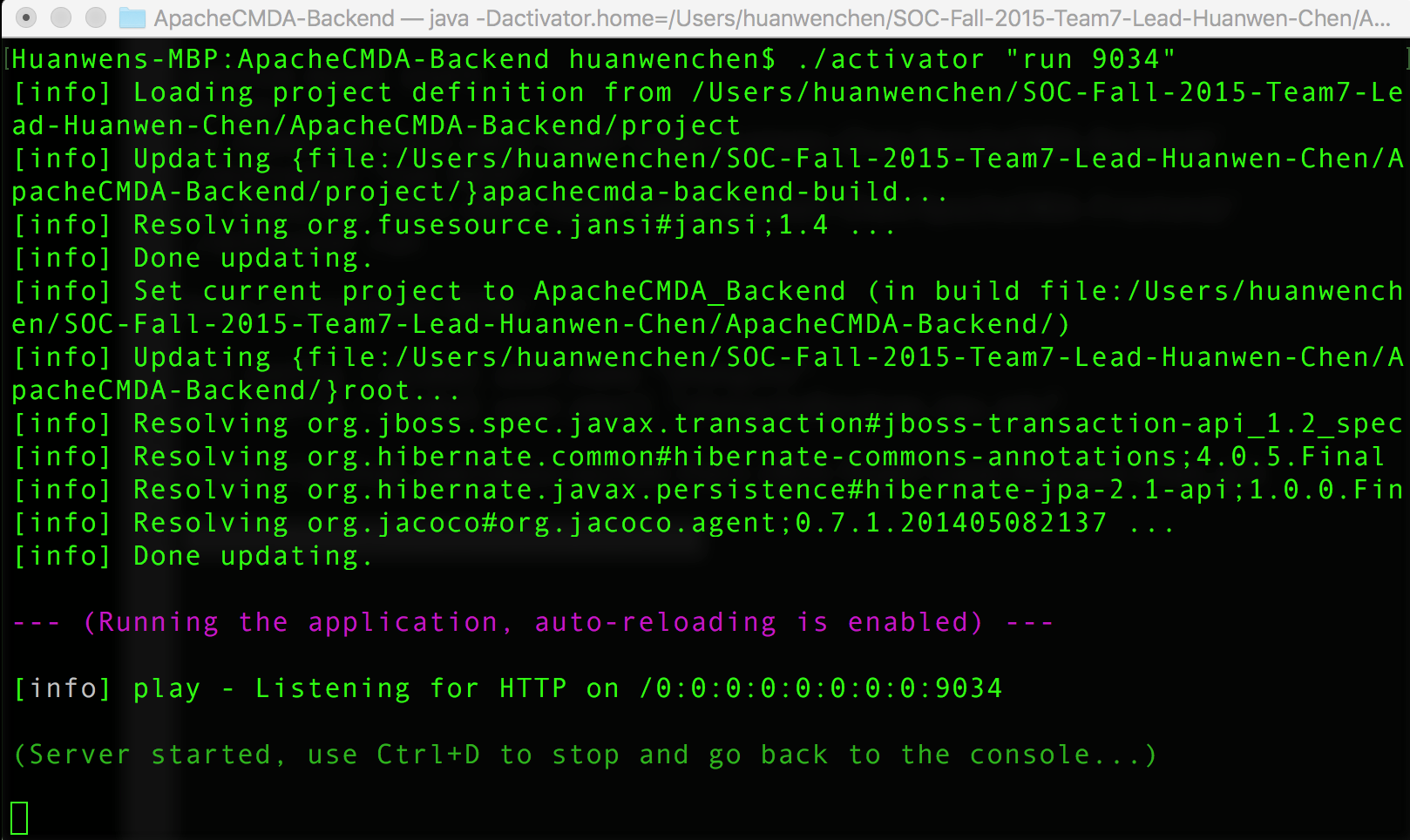
1. **Contribution of each team member**

|  |  |
| --- | --- |
| team 7 | contribution |
| Huanwen Chen | getting workflow list and one workflow |
| Wenjie Ma | getting comment from the workflow and writing comment |
| Zhongchuan Xu | adding/deleting a new workflow and marking answers |
| Ge Jin | listing popular workflow and showing the top in regular list |
| team 8 |  |
| Yiming Zhang | User |
| Lai Wei | User Group |
| Hsueh-Hung Cheng | Access control |
| Yangdi Zhou | UI improvement |

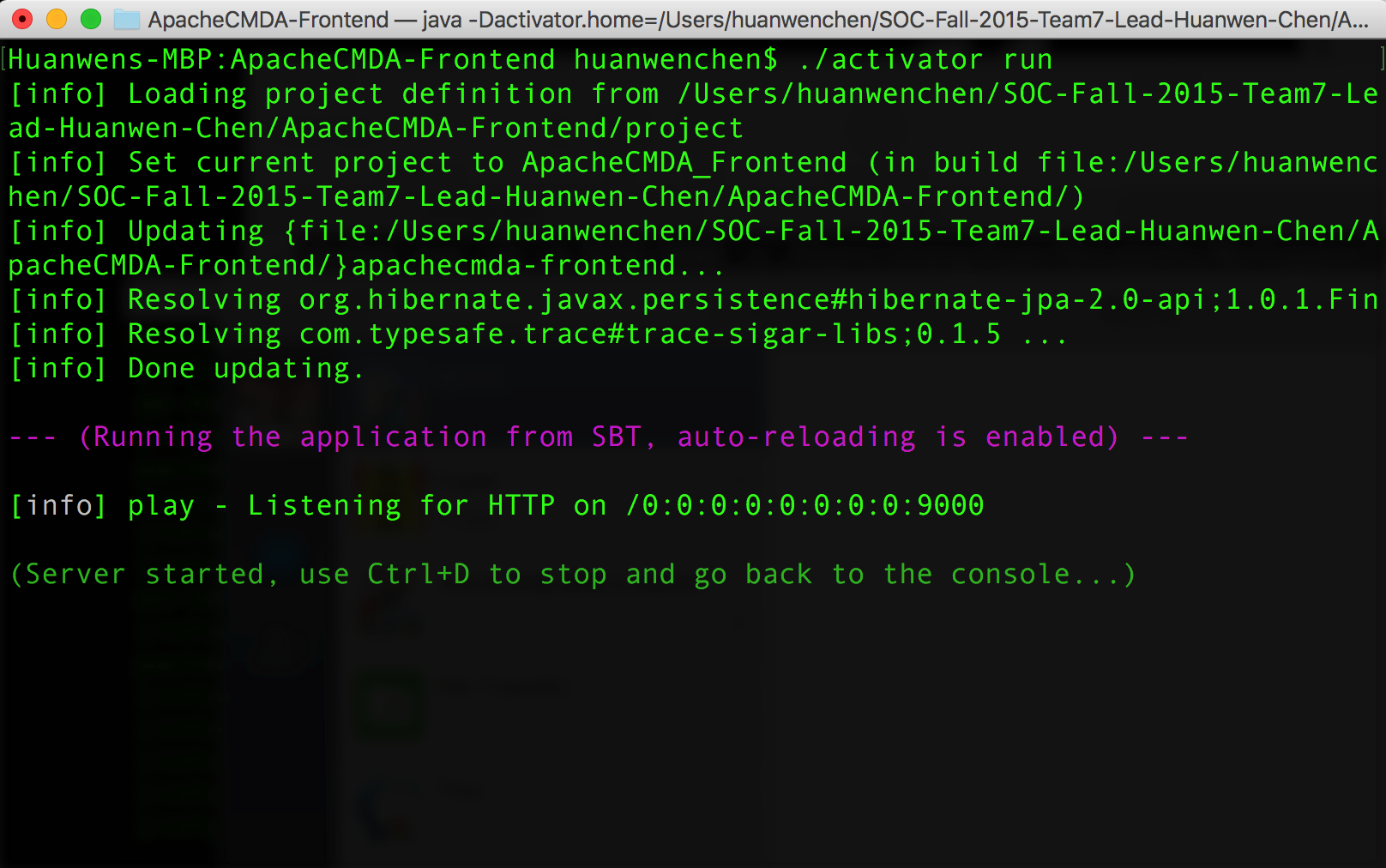
1. **Tutorial**
2. Install mysql, please use brew to install myql. Then run the mysql.server restart to initialize the mysql. If you have installed the mysql with Mac, you can start the mysql server via System Preferences → MySQL → Start MySQL Server, which is showing as below:



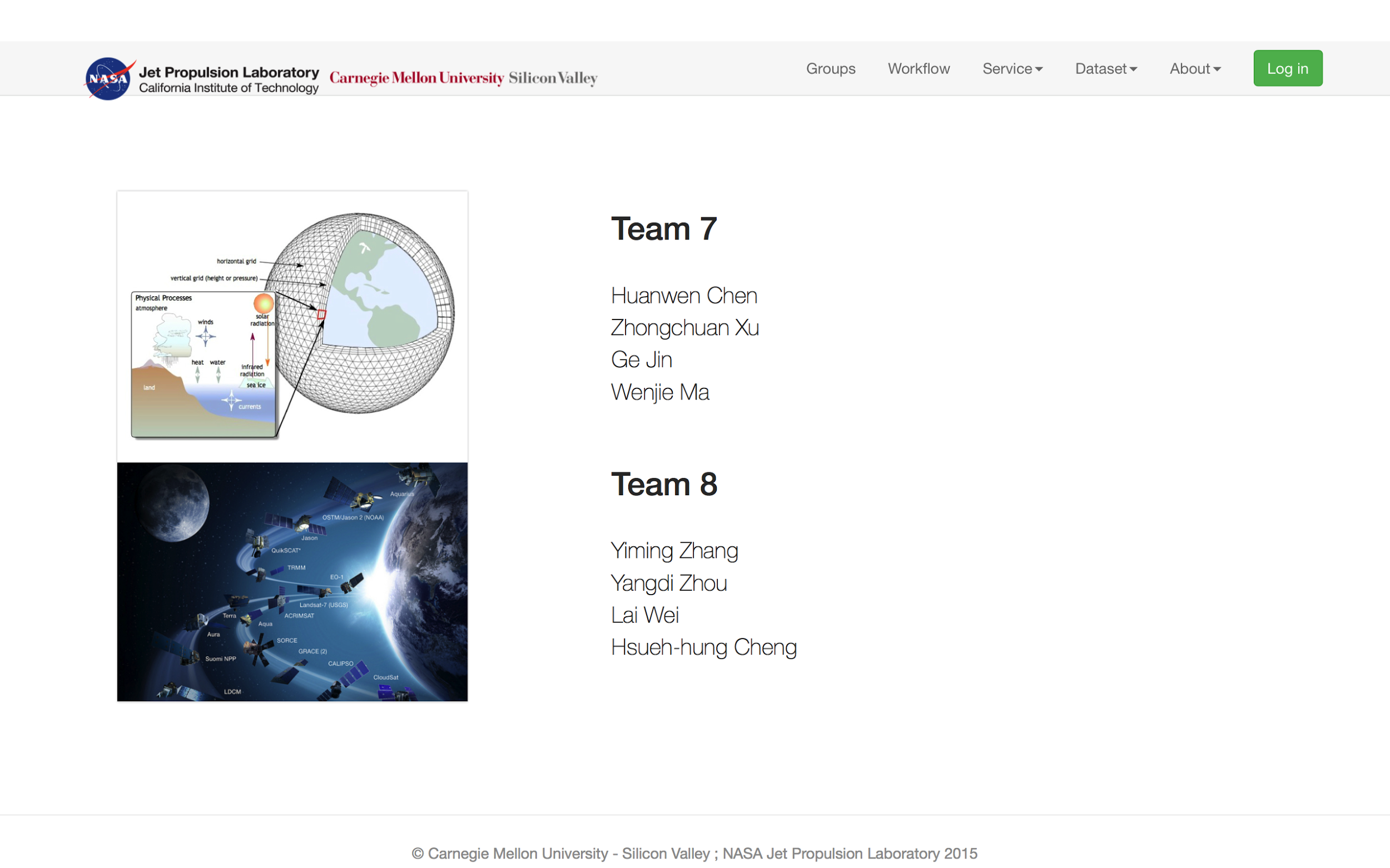
1. Connect to the mysql (make sure mysql has running on port 3306) and create schema called test\_db with a username is “root” and password is void. No tables needed to be created because they will be created automatically by backend. Before you login into mysql, you can set global environment by PATH=”$PATH”:/usr/local/mysql/bin.
2. cd into [ApacheCMDA-](https://github.com/cmusv-sc/SOC-Fall-2015-Team13-Bowen-Zhang/tree/master/ApacheCMDA-Frontend)Backend, run ./activator "run 9034", then will see the following picture



1. cd into [ApacheCMDA-](https://github.com/cmusv-sc/SOC-Fall-2015-Team13-Bowen-Zhang/tree/master/ApacheCMDA-Frontend)Frontend, and run the frontend with ./activator run, then will see the following picture



1. Open the browser and go to localhost:9000, you will see the main page showing as below:



1. Then click Log in to register your account and begin the journey of the website.