Analysis Me

# Introduction

This project aims to develop a web-based project management system to help the decision maker to verify the projects and approve features.

The system allows users to create projects and send them for verification by the administrator.

The system is useful for project management in big companies as they require strict approval processes. This system is is completely cloud based and follows service oriented protocols.

The system has two types of users - regular users and administrator, the regular users can add features or projects and the administrator is charged with approving the project or to reject the project.

# Motivation

Big companies require strict project management on the features added in their repository and there has to be a system with varied user access control with multi-tiered access for different types of users.

Large project require through planning and the administrators need to keep a close check on what all is developed. This project allows the administrator to approve or reject the different features proposed by the users.

This project is required because the teams can be cross country or across different countries and in large companies there has to be a system for easy communication on the project apart from just normal communication channels. This project provides a context based communication channel, emails or slack don’t suffice as it’s easy to loose track of what project is being talked about and it’s hard to search and recap what was discussed.

# Relative Work



**Figure 1 Play! Framework**

We use Play! Framework as our development platform. Play! Framework is based on a lightweight, stateless, web-friendly architecture, and makes web development easier.



**Figure 2 MySQL**

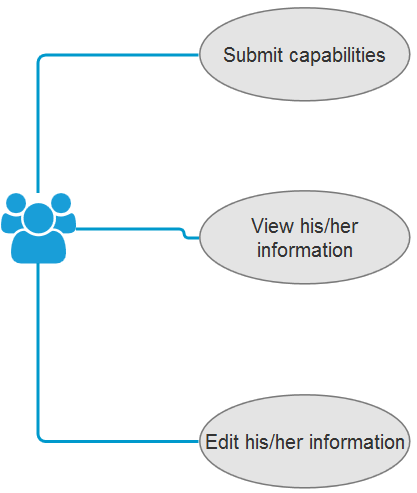
We use MySQL for our database because play! framework does not have its own database. MySQL is one of most popular open source databases.

There are multiple systems that are used for project management like trello or asana etc. , but few provide a strict access control for different types of users and therefore this project is important.

# System Design

## Requirement Analysis

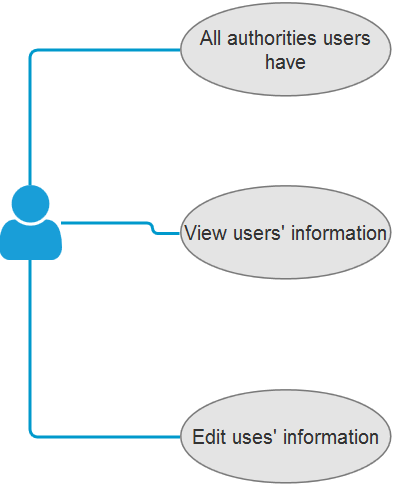
### Ordinary User



**Figure 3 users**

The basic user is given permissions to create capabilities and update his information. So, The normal user can create and modify his capabilities and is required to get approval every time he makes changes to the capapbility.

### Administrator



**Figure 4 admin**

The administrator is charged with approving capabilities. The administrator can edit capabilities or approve/reject capabilities. There is just one admin user currently but more administrator can be added easily in the system, just by changing the permissions in the database. The code is dynamic enough to allow multiple administrators.

### User Friendly

we created the project on Bootstrap which provides support for different browsers and mobile devices. The project is run using REST calls so it’s easily supported by all modern browsers and this ensures that the system is as user friendly as possible. Bootstrap also ensures that there is a consistent design scheme and color scheme throughout the system this makes it easy to understand. We added alert messages which give clear problems or updates on different steps if the system this is crucial as users should always have a clear view of what happened on each step.

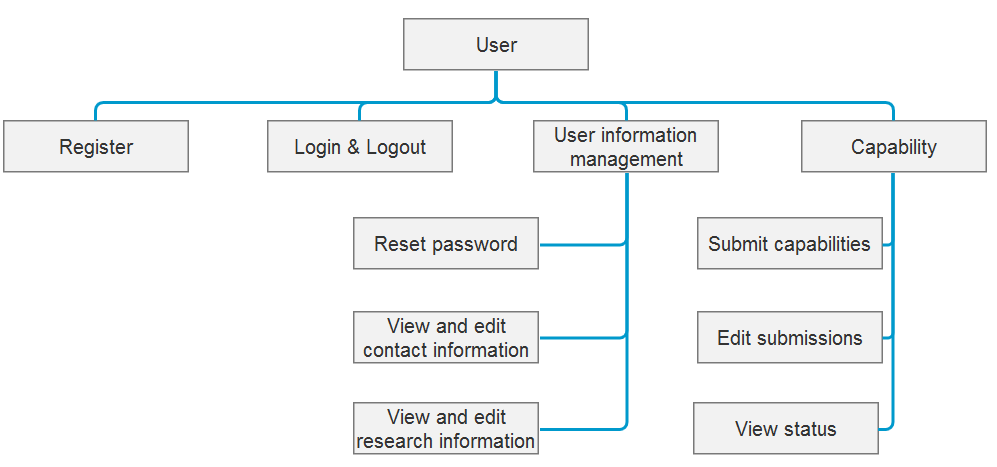
### Security Requirement

The system uses state of the art security protocols. The system keeps all the passwords encrypted using bcrypt which requires both time and space so it’s a great deterrent against brute force attacks. The project also uses sessions to log in the users and id something goes wrong then the sessions can be invalidated and the user will need to log in again. This protects users in case of a breach and invalid users can be logged out from the backend. The system also validated the users in all calls and doesn’t store the data in the sessions this ensures that any change in the system is reflected instantaneously.

The deployment process will ensure other security protocols. The system should be deployed in a virtual private cloud with only the front end server having access to the internet and the database server and the backend server being shielded behind the firewall. This will ensure that the inter server communications are not leaked and are protected against any man in the middle attacks.

## Design

### Design on User Management

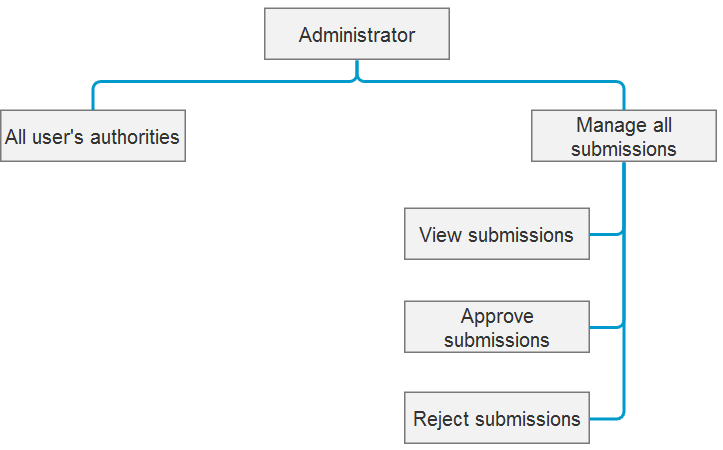


**Figure 5 Design on user mode**

The user management design is straightforward and has three major parts-

* login system - This system has calls that are not protected by the sessions and are required for the users to create a new users or login into the system.
* User information system - This system allows the user to edit his information in the system and this system is only accessible after logging in. The users can edit all his information using this page except his email and password.
* Capability management system - This system allows a logged in user to keep track of his submitted capabilities and add new capabilities in the system. This part notifies the users when a decision has been reached on a capability.

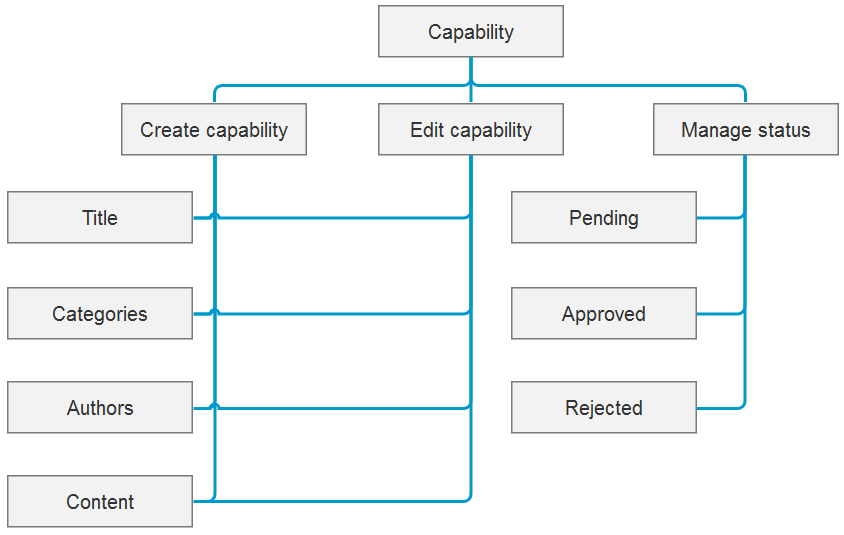
### Design on Administrator Mode



**Figure 6 Design on administrator mode**

The administrator has the same parts as the regular users and has one additional functionality which allows him to approve or deny a capability. This part allows the admin user to view all the submissions made and then either to make changes on them or approve the capabilities.

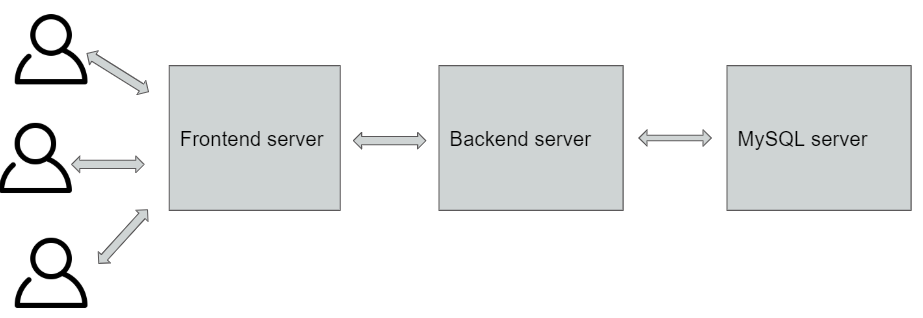
### Design on Submissions



**Figure 7 Design on submissions**

Figure 7 shows the design on capability includes create a capability, edit a submitted capability and manage the status of a submitted capability. Any user including the administrator can create a capability with the information of capability title, categories, at least one authors, the abstract of the capability and some detail description about the capability. After submitting, only the owner of a capability can edit the information of the submission. Both the owner of a capability and the administrator can view the submission information, i.e., the administrator can view all submissions in the system.

### Design on Architecture



**Figure 8 Design on pipeline**

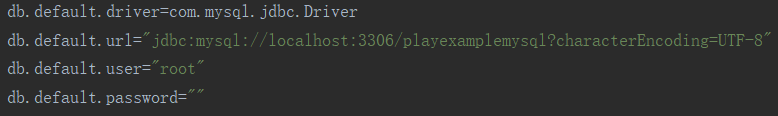
We use a three tier system where the whole system is broken down into three parts. One part is responsible for the frontend and the delivery of the content to the users. This part doesn’t talk to the database and all access to the database is through the backend server which uses a REST protocol to allow the frontend server to change different systems in the server. The third part is the MySQL server which sits separately than the frontend and the backend systems. This ensures that the systems are separate and breach of one system will not compromise the whole system. This also allows for easy scalability of the system as more servers can be added on each tier easily and the load can be distributed using load balancers among the different systems.

We follow a MVC architecture for the project which is facilitated by the play framework. The models controllers and view are located in separate folders which provides easier understanding of the system and allows for new users to jump in and work on separate parts of the system.

# System Implementation

All codes are in Github public repositories. The front-end and back-end parts are separate into two independent projects.

## Configuration



**Figure 10 conf**

First, we configure the environment for MySQL server in “conf – application.conf” as Figure 10 above.



**Figure 11 dependencies**

Then, we add the library dependencies in “build.sbt”. We have 6 dependencies as follow:

HikariCP and Ebean is for database. We use HikariCP as the connection pool for Java Database Connection and use Ebean for the data structure and processing methods.

Javax.mail API is used to send email using Java automatically.

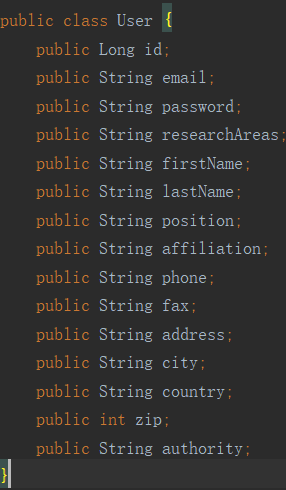
Unirest is used for HTTP calls, such as POST, PUT, GET and DELETE.

Play-bootstrap 4 is a front-end library to provide HTML web components like a well-designed CSS. It is for a better user experience and user interface.

Jbcrypt is an encryption library. We use it to encrypt the password for security.

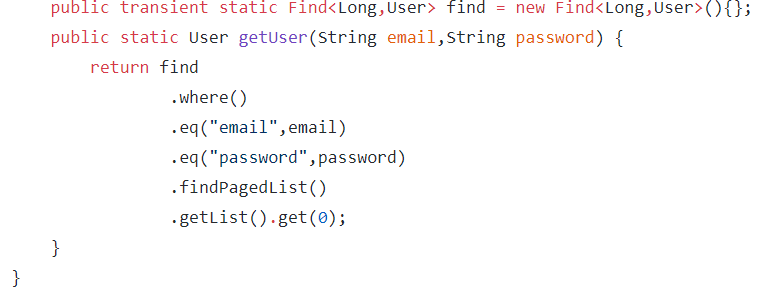
## Data structure

### User



**Figure 12 user**

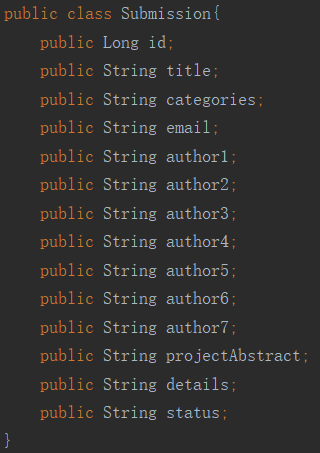
Figure 12 shows the attributes of a user in both front-end and back-end projects. A user has the id, email, password, etc. All java class fields above will be attributes in the user table in the database. And the primary key is user id.



**Figure 13 user**

Figure 13 shows the difference of models – user in back-end project. There are two methods for get the target user by receiving information.

### Submission



**Figure 14 submission**

Figure 14 shows the structure of a submission. Fields in Submission.java will be the attributes in table “submission” in database. The primary key is also the id.

The difference between front-end and back-end projects are the same as “user”.

Since the play framework does not have its own database, we need to create a database in MySQL before running it. No table needs to be created in MySQL.

## Register

### Interface



In front-end project, “app - view - createUser.scala.html” is developed to show the user interface of register part. We use “b3.form” in Play-bootstrap 4 for the form which can receive user’s input information.

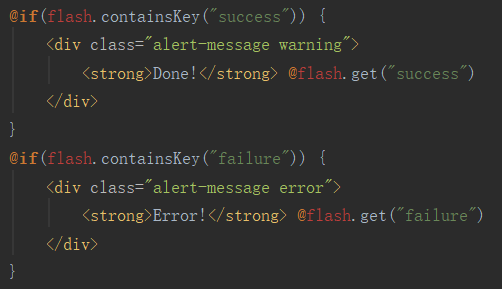
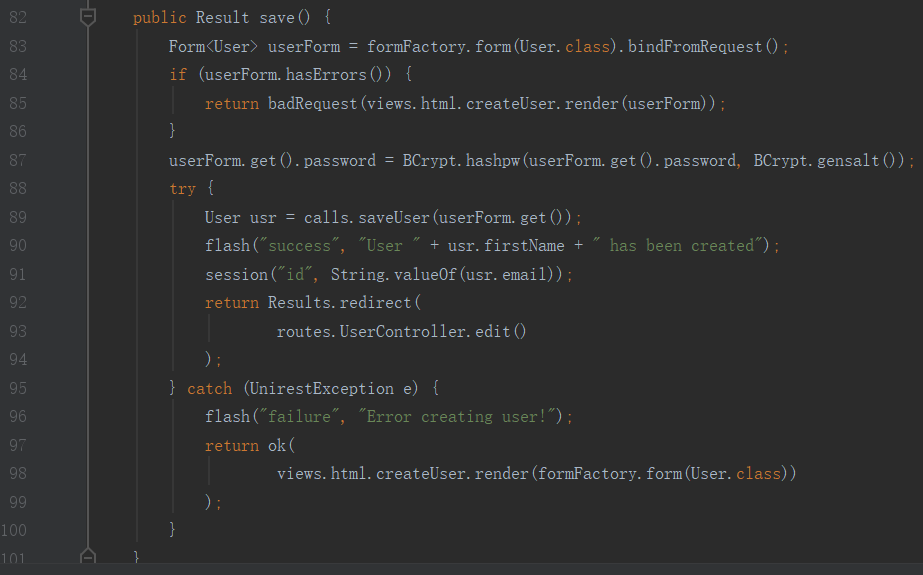


Figure above shows the code for displaying a warning message on a successful registration, and an error message on a failure registration. This part is almost the same in each creating and editing page, so it will not be described again later.

### Logic

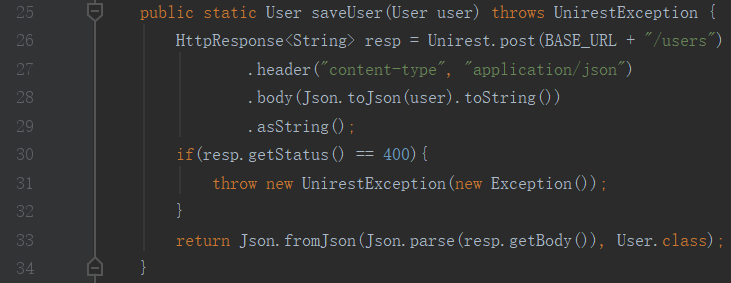


In front-end, “app – controller – UserController.java” is developed to process the HTTP request sending from “app - view - createUser.scala.html”. There is a method named “save” in this Java file.

Line 83 shows that it will read the request form and store the data into memory in a form of user class.

Line 87 shows that it will encrypt the password and save the new password by bcrypt.

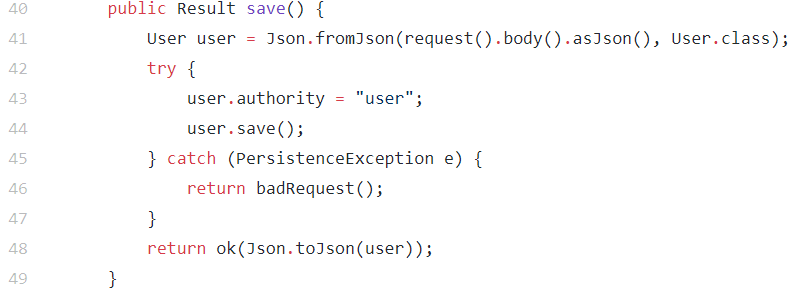
Line 89 shows that it will try to save the new form data into database via “app – model - BackendCalls”.



“app – model - BackendCalls” has a method named “saveUser”.

Line 26 shows that it will call a back-end method via sending HTTP POST to the URL “http://localhost:9001/users” with the JSON of user inputs using Unirest.

Line 33 shows it will return the data in the format of java class “User”.



In the back-end project, “app – controller – UserController.java” has a method called “save”. Line 44 shows it will save the data into database via a method “save” in Ebean. Line 48 shows the saved data will be returned as the format of JSON, and this JSON will be the response data for the front-end project.

After that, Line 92 and Line 97 in “app – controller – UserController.java” in front-end project shows that it will redirect to another java class for editing the user information if save data in database successfully.

## Reset password

### Interface



In front-end project, “app - view - login.scala.html” is developed to show the start interface. One of the forms refers to resetting the password. We also use “b3.form” to receive user’s input action, i.e., clicking the button.



In front-end project, “app - view - forgot.scala.html” is developed to receive the email account using “b3.form”.

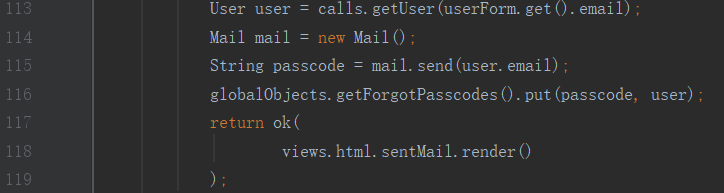


“app - view - sentMail.scala.html” is developed to show the state of sending an email, and turn to login page after sending the email successfully.

### Logic



In front-end, line 164 in “app – controller – UserController.java” turns the page into “app - view - forgot.scala.html”. And as the above figure, it will go to “forgotPost” method in “UserController”.

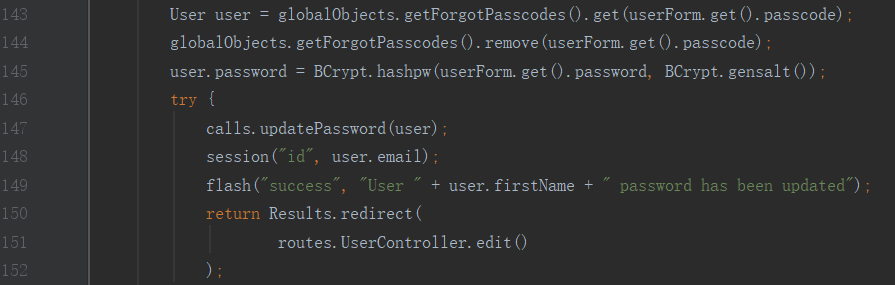


Line 115 shows it will call a method named “send” in Java class “Mail”.



In “app – models - Mail”, “email” is a “HtmlEmail” in Javax.mail. Line 22 shows how to send to a given email account (mailAddress) a given link. This link refers to “controllers.UserController.resetPassword()”.

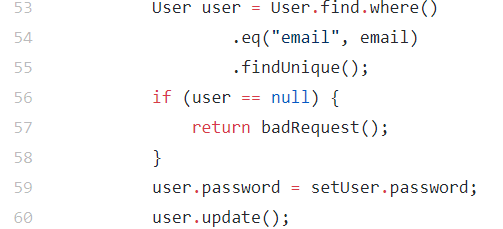
Line 118 in “UserController” shows after sending email successfully, it will turn to “app - view - sentMail.scala.html” page.



Line 145 in “UserController” shows the new input password will be encrypted.

Line 147 calls a method called “updatePassword” in “app – model - BackendCalls”.

Similar as “save” method in “BackendCalls”, it will call a back-end method via sending HTTP POST to the URL “http://localhost:9001/users/reset”.



In back-end project, Line 53 in “UserController” finds the target user in database; Line 60 update the user with new password into the database.

## Login

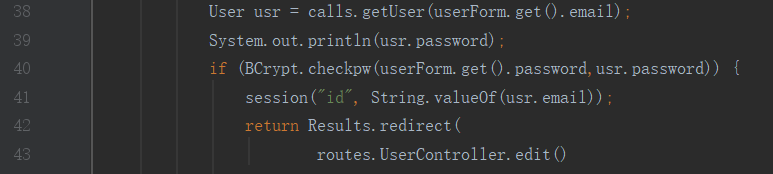
### Interface



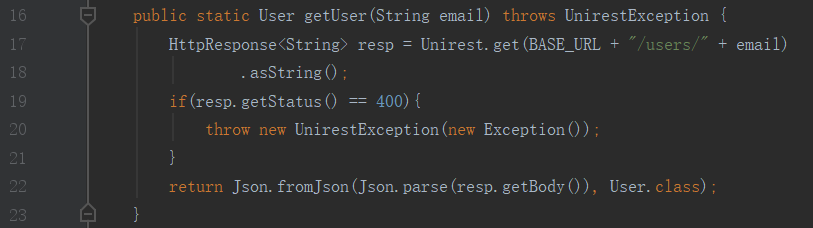
In front-end project, “app - view - login.scala.html” can be used to login by “b3.form”.

Since all the user interface have the similar programming skills, we will not talk more on other interfaces later if not necessary.

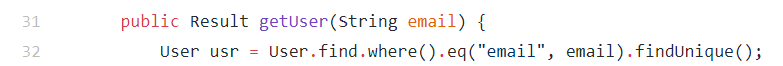
### Logic



In front-end project, Line 38 in “app – controller - UserController” calls the “getUser” method in the “app – models - BackendCalls”.



Line 17 in “BackendCalls”, “getUser” method will send HTTP Request (GET) to “http://localhost:9001/users/[email]”.



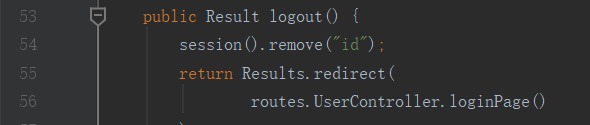
This link will call “getUser” method in “app – Controllers - UserControllers” in back-end project, and “getUser” method in back-end project will return the target user as the format of JSON.

And “BackendCalls” in the front-end project will return the target as the format of java class “User”.

Since the passwords are stored in the database after encryption.

Line 40 in “UserController” compares the encrypted input password and password in database. If these two passwords are equal, the session of that user will be opened, and it will call “edit” method to show the editing personal information page.

## Logout

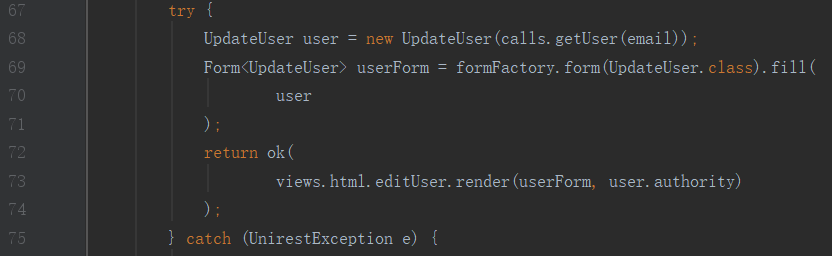


In “logout” method in “UserController” in the front-end project, it just delete the current session with the session id, and return to login page, i.e., “app - view - login.scala.html”.

## Edit personal information



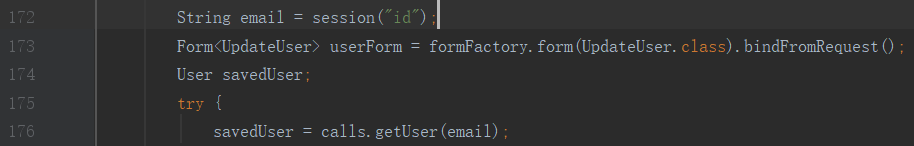
Line 9 in “app - view - editUser.scala.html” shows that the role of user, i.e., user and admin will be displayed on the top of web page.



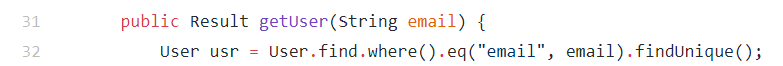
In “edit” method in “UserController” in the front-end project, it will use a java class called “UpdateUser” to store the user which is wanted to be edited and transmit it to “editUser.scala.html”. “authority” here means the role of user, which will be used in Line 9 in “editUser.scala.html”.



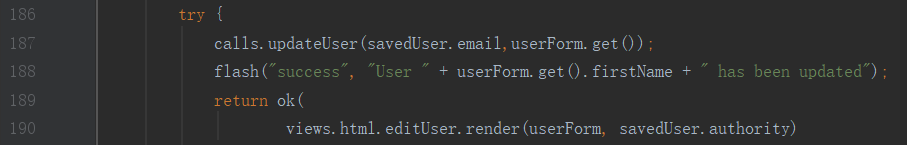
“app - view - editUser.scala.html” gets the new user personal information and sends it to “update” method in “UserController”.



Line 176 in “UserController” calls “getUser” method in the “app – model - BackendCalls”, and “getUser” method in the “app – model - BackendCalls” calls “getUser” method in back-end project.



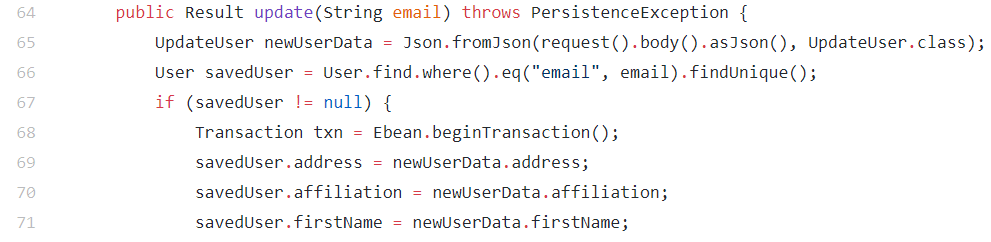
In back-end project, Line 32 gets the user by its original email address which is stored in the database and may be changed in the new user personal information now.



In front-end project, Line 187 calls the “updateUser” method in the “app – model - BackendCalls”.



In “BackendCalls”, Line 47 sends HTTP POST to the URL “http://localhost:9001/users/edit”

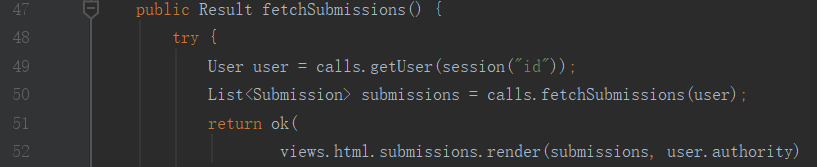


In the back-end project, Line 65 gets the new personal information, and Line 64 gets the target user in database.

Line 68-82 change the old information to the new information in database via a transaction using Ebean.

Line 190 in “update” method in “UserController” shows it will turn to “app - view - editUser.scala.html” with a message after updating successfully, because of the “savedUser.authority”.

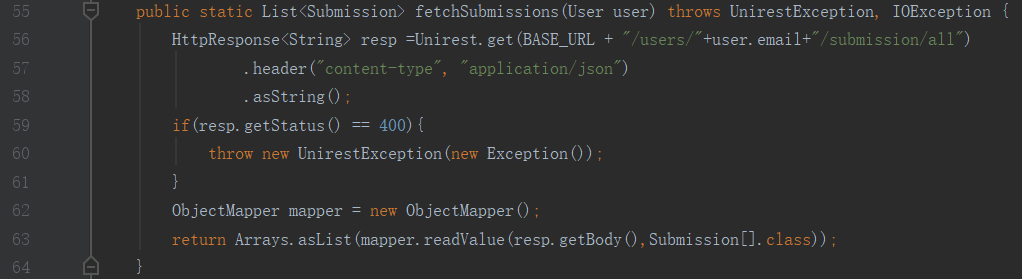
## View submissions



In front-end project, Line 187 in “app – controller - SubmissionsController” calls the “getUser” method in the “app – model - BackendCalls”.

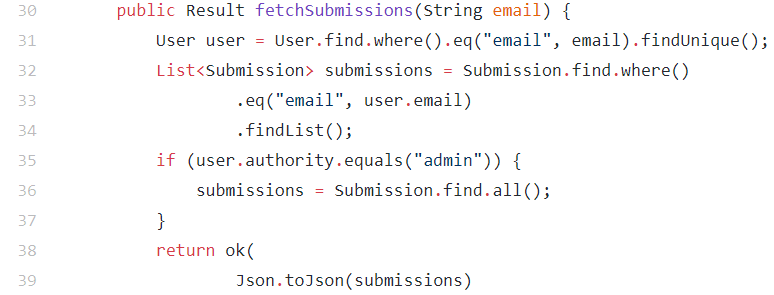
As we have discussed in 5.7, “getUser” method will call “getUser” method in back-end project, and “getUser” method in back-end project will return the target user as the format of JSON. And “BackendCalls” will return the target as the format of java class “User”.

Line 50 in “SubmissionsController” calls the “fetchSubmissions” method in the “BackendCalls”.



Line 56 in “BackendCalls” shows that it will call a back-end method via sending HTTP POST to the URL “http://localhost:9001/users/[email]/submission/all” using Unirest.

In back-end project, this URL means to call the “fetchSubmissions” method in “app – controllers - SubmissionsController”.



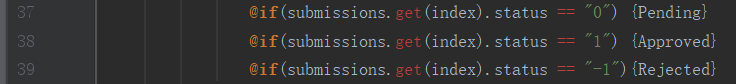
Line 31 in “fetchSubmissions” method gets the target user will its email address as we disscussed.

Line 32 gets a list of submissions with the given user’s email address from the table “submission” in the database via “app – models - Submission” in 5.2.2.

Line 35 shows the administrator mode, which we will talk about in 5.12.

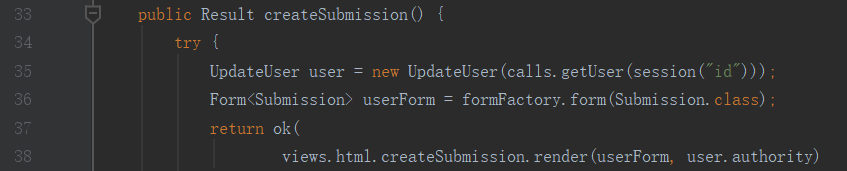
Line 39 shows “fetchSubmissions” method will return all obtained submissions.

In front-end project, Line 56 in “BackendCalls” gets what are returned from Line 39 in “fetchSubmissions” method in back-end project, and Line 63 returns these submissions as a format of a list of “Submission”.



In the “submissions.scala.html”, we use “pending”, “Approved” and “Reject” as submission status, instead of just shwoing “0”, “1”, “-1” in database.

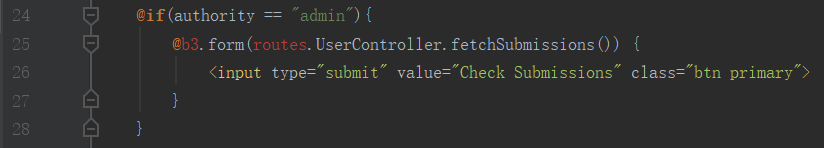
## Create a submission



In front-end project, Line 35 in “app – controller - SubmissionsController” calls the “getUser” method in the “app – model - BackendCalls” with the current user’s email.

As we have discussed in 5.7, Line 35will get the current information in the database.

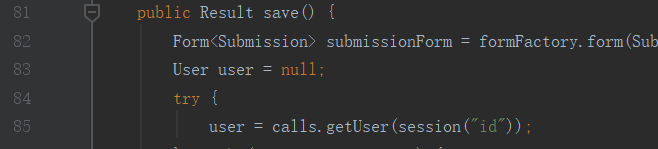
Line 38 jumps to “app – view - createSubmission.scala.html”.



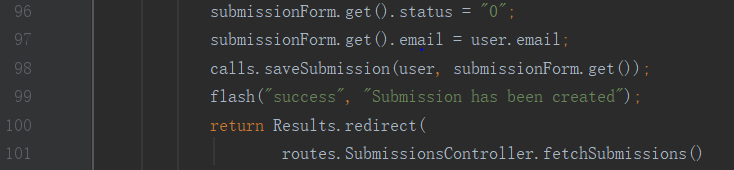
In “createSubmission.scala.html”, the authority of user will be checked. This will be talked about later in 5.12.



In “createSubmission.scala.html”, it calls a method “save” in “app – controllers - SubmissionsContoller”.

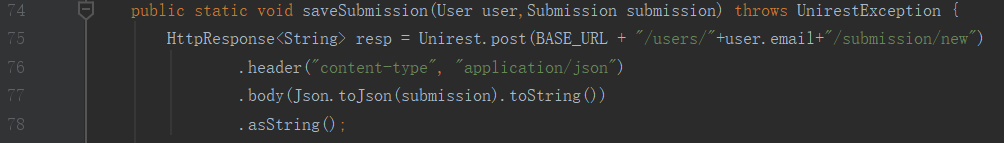


In “SubmissionsContoller”, Line 85 gets the object of current user by email address.



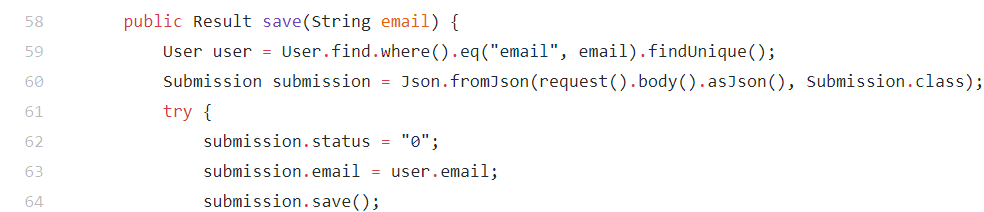
Line 96-97 adds the state (pending) and submitted email address to the submission.

Line 98 calls “saveSubmission” method in the “BackendCalls”.



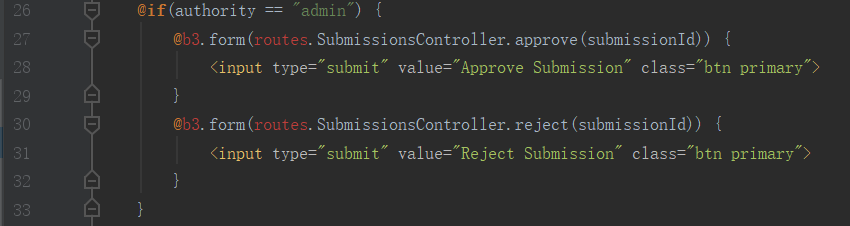
Line 75 in “BackendCalls” shows that it will call a back-end method via sending HTTP POST to the URL “http://localhost:9001/users/[email]/submission/new” using Unirest.

In back-end project, this URL means to call the “save” method in “app – controllers - SubmissionsController”.



Line 63-64 show that the submission will be save in the table “submission” in database with it’s owner’s email address.

## Edit a submission



“changeSubmission.sacla.html” shows administrator can change the status of a submission, which will be talked about later in 5.13.



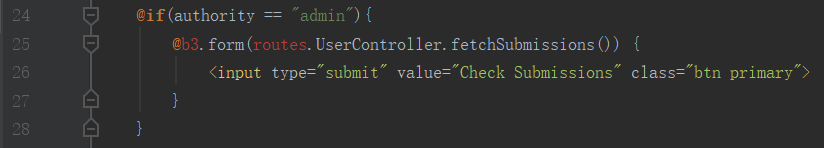
The “update” method in “app – controllers - SubmissionsController” is used to update the content of a submission. This is very similar to what we discussed in 5.7 and 5.9, so we will not discuss it again.

## Admin login

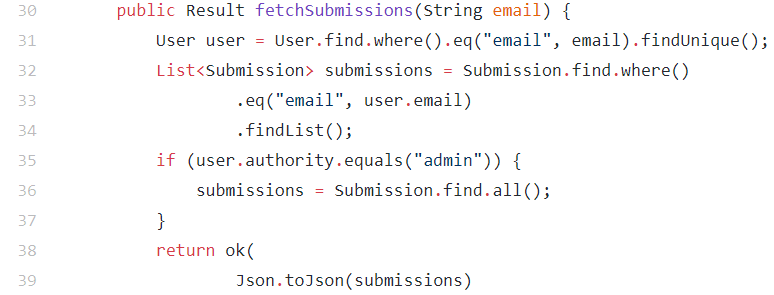
We predefined an account as administrator in the database, and if a user logs in with this account, the user will be seen as the administrator.

## Admin view submissions

Ordinary user’s authority is “user”, while only the administrator has a “admin” authority.



In front-end project, as we have mentioned in 5.9, the administrator can see a button to check all the submissions in the database.

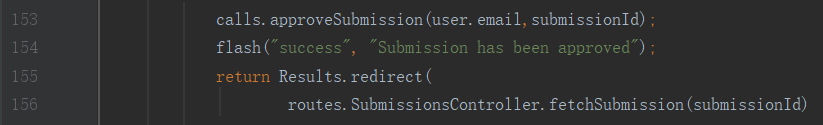


As we have described in 5.8, the only difference is in Line 35 in “fetchSubmissions” method in back-end project.

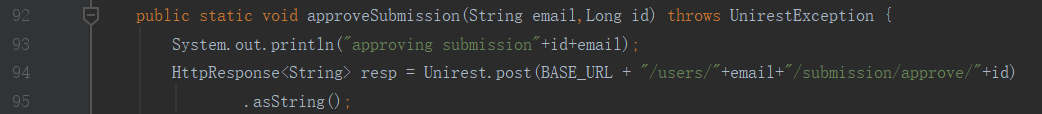
If the authority is “admin”, this method will return all submissions in the table “submission” in database using “app – models - Submission”, instead of only the corresponding user’s (administrator’s) own submissions.

## Approve or reject submission

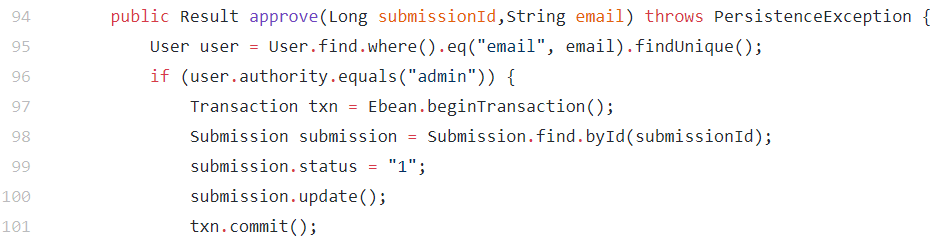
As we have described in 5.10, administrator can change the status of a submission.



In front-end project, Line 153 in “approve” method in “app – controller - SubmissionsController” calls the “approveSubmission” method in “BackendCalls”.



Line 94 in “BackendCalls” sends HTTP POST to the URL “http://localhost:9001/users/[email]/submission/approve/[id]” using Unirest.



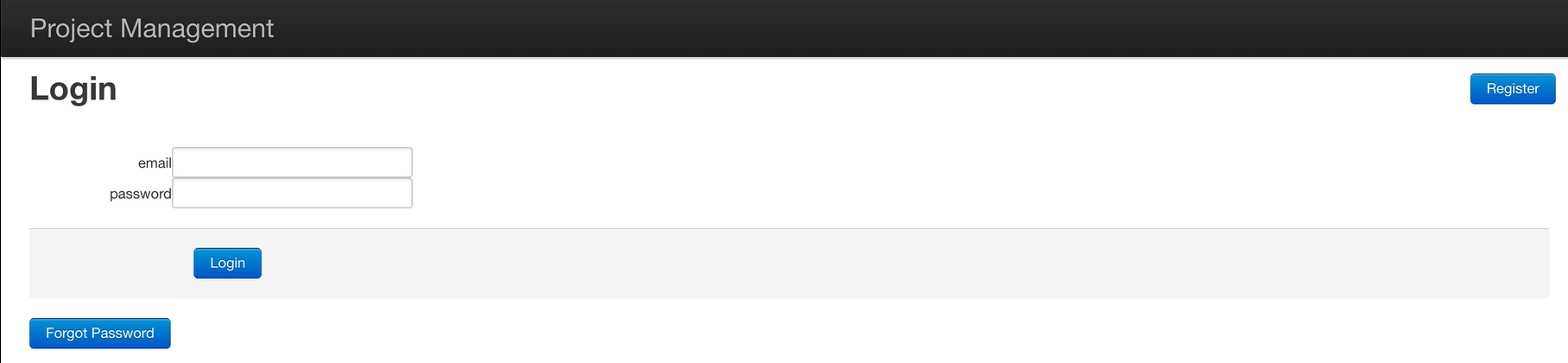
In back-end project, this URL means to call the “approve” method in “app – controllers - SubmissionsController” in back-end project.

Line 98 gets the target submission in the database.

Line 100 update the “submission” entry with the new status “1” in database.

Reject a submission is almost the same. The only different is the status change to “-1” in “reject” method in “app – controllers - SubmissionsController” in back-end project.

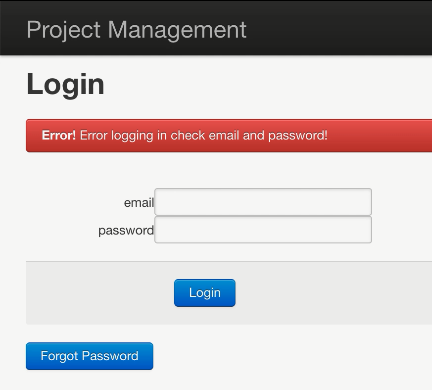
# Analysis and discussions



**Figure 15 Start page**

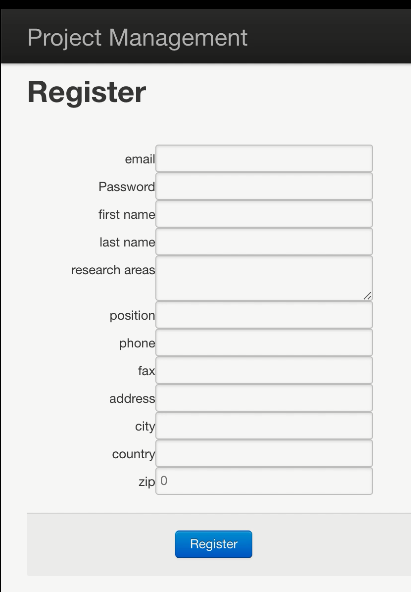
Figure 15 shows the home page of our decision-making system. Anyone who accesses this page can choose to login or register. And a registered user can click a button called “Forgot Password” to reset a new password via his or her email address, which is also the username.

If a registered user chooses to login, but inputs the wrong username or password, this page show an error alert message as Figure 16.



**Figure 16 login**

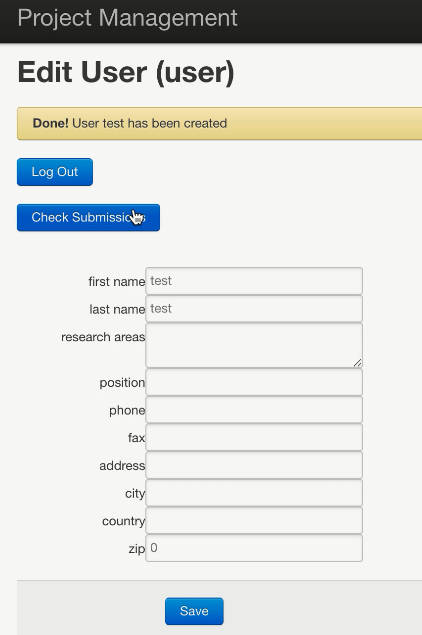
If a user chooses to register, this page will jump to the other page as Figure 17.



**Figure 17 register**

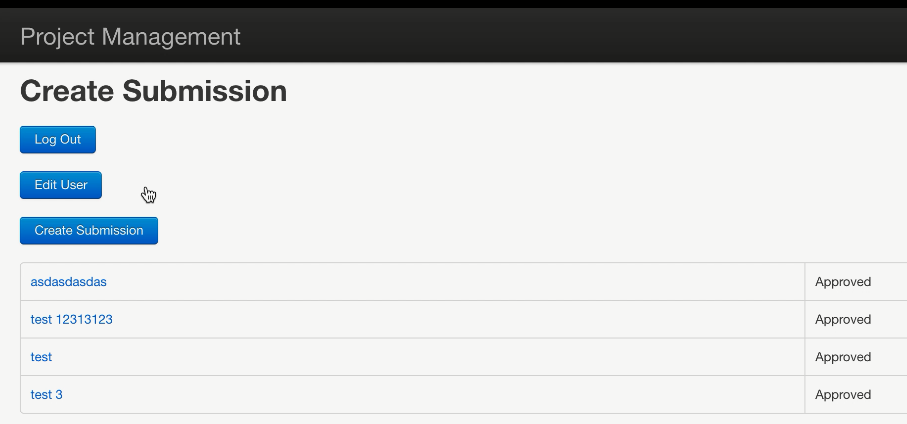
Figure 17 shows the register page. A new user can register into our system with his or her email and password. And the user can input some personal information like first name, last name, research areas, position, phone, fax, address, city, country and zip number to increase the possibility of the submissions to be approved.

After login or successful register, the user can see his or her personal home page as Figure 18.



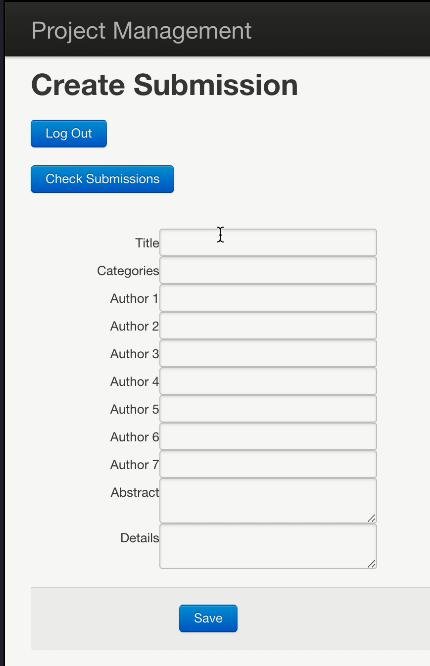
**Figure 18 edit user**

The user can update the personal information in this page and can also choose to jump into the submissions page by clicking “Check Submissions” button. And if the user chooses “Log Out” button, our web system will jump into the home page as Figure 15.



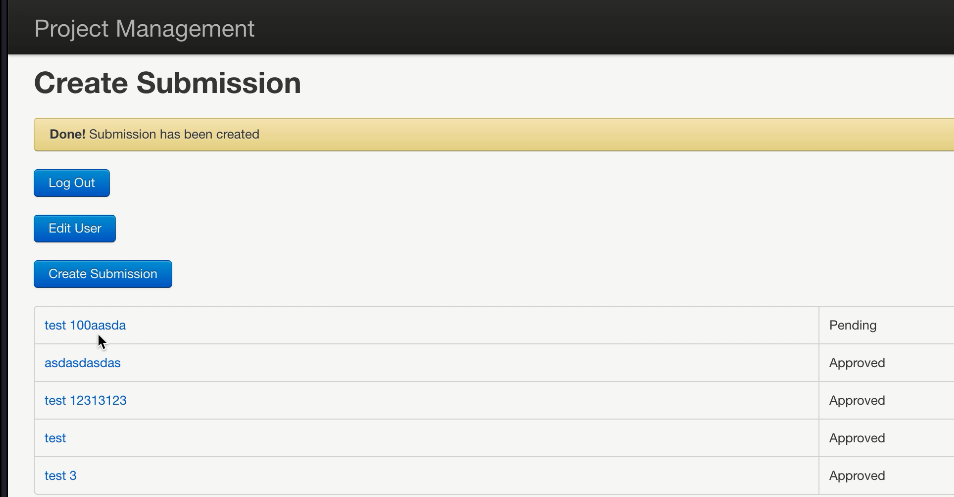
**Figure 19 submissions**

Figure 19 shows the view-submissions page. The user can also logout in this page. And if the user clicks “Edit User” button, our system will jump into Figure 12. The user can view and edit all his or her submitted capabilities by click the title of a submission. Besides, the user can create a new capability by clicking the “Create Submission” button.



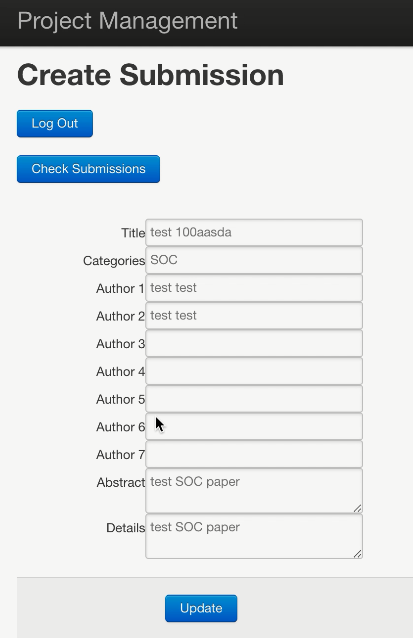
**Figure 20 create**

Figure 20 shows the create-submission page. A user can create a new capability into system by given its title, categories, authors, the abstract of the capability and some detail description about the capability. The user is not necessary to input all the information above, but the quality and quantity of the user’s given information will influence the decision about whether to approve by the administrator. After the user save a submission, the submission will be shown as a submitted capability, and the user will jump into the list of submissions page as Figure 21.



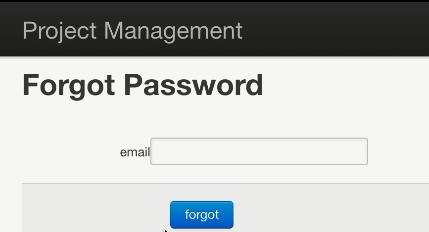
**Figure 21 submissions**

Figure 21 is also the view-submissions page, but with a message to show the state of whether the submission has been added into system successfully. A user can click any of the submitted capability to edit the details.



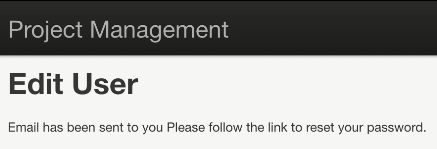
**Figure 22 edit submission**

Figure 22 shows how a user can change his or her submitted capabilities. If the user clicks “Check Submissions” button, the contents of submission will not be changed; and if the user clicks “update” button, the new details will re-write the old version in our system. Both two of these buttons will jump into view-submissions page.

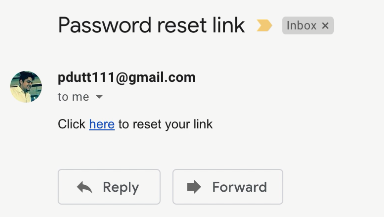


**Figure 23 forget**

As Figure 15 shows, a user can click “Forgot Password” button to reset a new password via his or her email address. Figure 23 shows the interface after clicking that button. After the user clicks “forgot” button in this page, an email will be sent to the input email address like Figure 23 and a message will be shown to display the state of sending email as Figure 24.

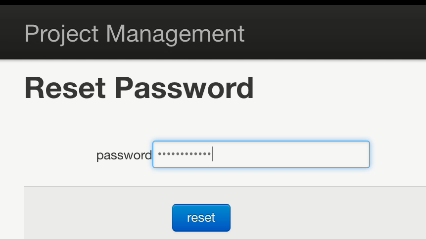


**Figure 24 send email**



**Figure 25 email**

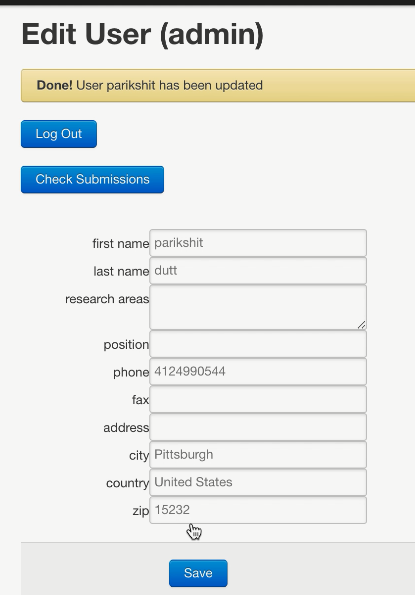
Figure 25 shows a user can click the link in the email to reset a new password.



**Figure 26 reset**

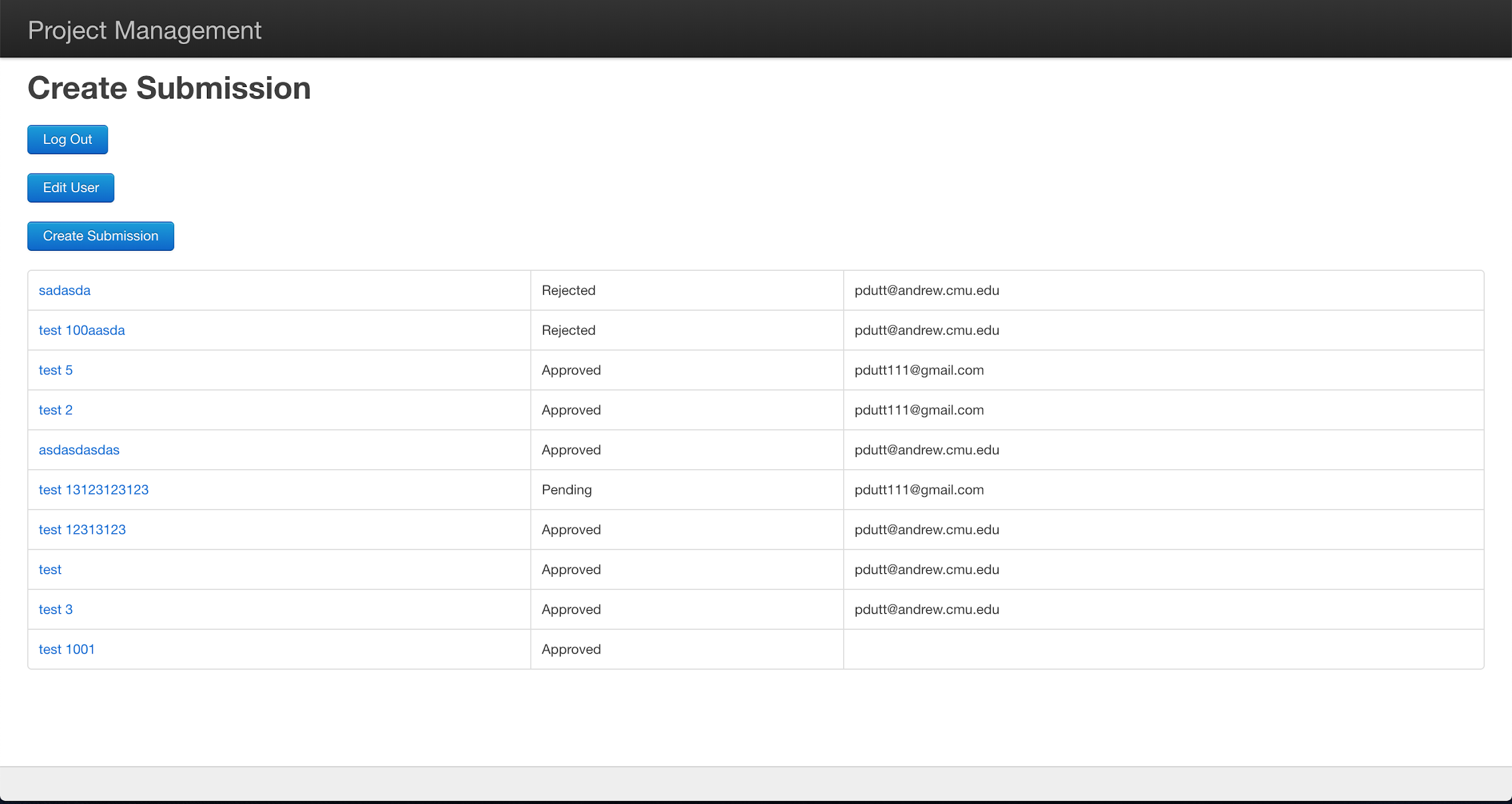
After open the link, the user can input the new password, and this password will be used to update the password in the system.

Above figures show the interfaces designed for ordinary users. Then, we will discuss the interfaces designed for the administrator.



**Figure 27 admin**

As a special user, the administrator can also do anything that an ordinary user can do. The only difference is there is a tag of “admin” to show this is for administrator mode as Figure 27.



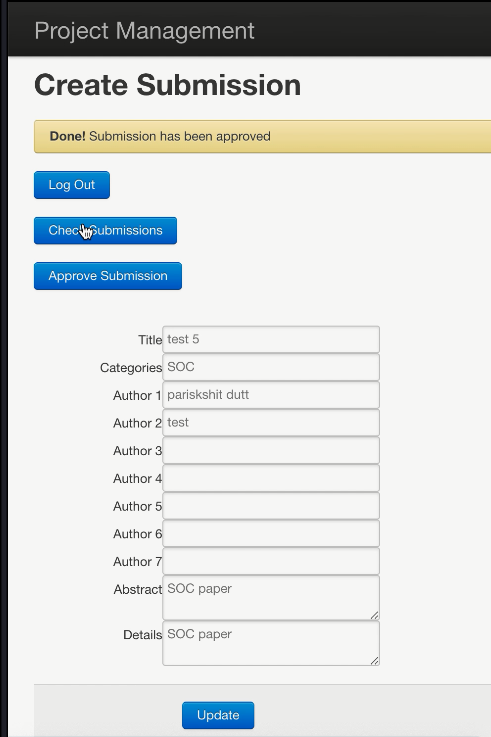
**Figure 28 completed submissions**

Different with ordinary users, the administrator can view all the submitted capabilities in the system with the title, status and submitting account as Figure 28. There are three statuses in total, which are “Pending” for not decide submission, “Approved” and “Reject”.



**Figure 29 details on submission**

Figure 29 shows the interface if the administrator clicks a submission. The administrator can choose to approve or reject the submission by clicking the corresponding button or just click “Check Submissions” button to do nothing with the status. After choosing the new status for a submission, it will show a message with the state of action as Figure 30.



**Figure 30 approve**

# Conclusions and future work

## Conclusion

All the requirements have been met by this decision-making system. And all known bugs and issues have been fixed.

Since our team has only two members, we are required to finish the 40 percent of the whole 14 required features, i.e., 6 features. For the completed logic of our project, we have done 6 completed features and 3 partial features in total, which are feature 1, 2, 3, 4, 5, 13, and part of feature 6, 7, 8. And we are not required to have user group related functions.

We have separate our system into a front-end project and a back-end project, so that we can run them in different ports.

## Future work

### Cloud deployment

This system is deployed on our own computer, but it needs us to keep the personal computer on power to maintain the service. In future, we can deploy the system on some cloud platform like Amazon Web Services to keep the stability of our service.

### Highly scalable system

This system supports limited traffic for now. We only test our project with small number of access. If there’s hundreds of thousands of users access to our system in the same time, the system may crash. In future, we can expand the system to support large scale of traffic.

### Angular.js

Angular.js is a popular javascript framework that allows the system to work using REST api calls. So, if the system were to use Angular.js then the dependency on the front-end server is removed and the whole system can function just using the backend system. This system distributes the processing over the the client side and allows for the server to be simplified. This is important when considering the scalability of the server. Server which does minimal processing is easily scalable versus a server which does a lot of processing. So, it’s always a good idea to reduce the processing on the server and offset some of the processing to the frontend instead.

### Stress test

Stress testing a system is crucial to have a good idea of the load the system can take and stress testing will provide a lot of information of where the system might be suffering from a bottleneck. JMeter or artillery can be used to benchmark each tier separately and this will help us in establishing the bottlenecks of the systems. once we have benchmarks of one server than its easier to calculate how many users the system will accomodate and to establish the growth path and at which point its necessary to add more servers.