FilmedIn

Team 8 - Design Document

Shreya Shankar, Bhavana Kakubal, Parth Kenkere, Prashast Vaidya, Manu Aggarwal

Table of Contents

PURPOSE

There is a need for a coordinated service in Purdue for students interested in film to connect. Students who produce films, are unable to find talent based on their requirements. A member of our team (a film minor) was requested by the film club (FVSO) to build a website. The film club wants to manage a website with which students who are developing a production can approach other students who are enthusiastic and qualified to participate. This site will also help the film club notify the students about the upcoming festivals and events of all film related organizations. Backstage.com will be a reference to our project. This website is a medium for artists all over the world to be able to share profiles, experiences and interests. Using this information, producers and directors can match their production needs, and possibly employ some artists.The goal of our website is similar, but is smaller in scale. (audience is restricted to students of Purdue, at least initially, before we may open it up to other universities). We also want to regulate management of inventory to facilitate smooth rental of film equipment by students. The business model of our group is different, and so we will not need to charge students to be able to use this website.

Functional Requirements

Logging in and creating a profile

1) A user should be able to login using username and password, and there should be an option for the user to be able to reset his password if he has forgotten it.
2) A user should be able to create a profile and enter required details and at some point, if need be, he must be able to change it.

Posting a job opportunity

1) A user must be able to post a job opportunity and provide a brief description of the role that is required, and the production.
2) A user should be able to search through all the jobs/opportunities posted based on parameters such as time period of production, position of job, prerequisites, etc.

Searching for profiles of candidates who are interested in participating in productions

1) A user should be able to search for candidates based on criterions such as age, gender and name.

News Feed

1) A user should be able to see news feed of the events that are going to be happening in Purdue soon.
2) A user who is an admin should be able to change the events that are posted.

Admin Rights

1) An admin should be able to add and remove accounts made in the website should the need arise.
2) An admin should be able to add and remove posts from the News feed if reported by other users of the website.

Non Functional Requirements

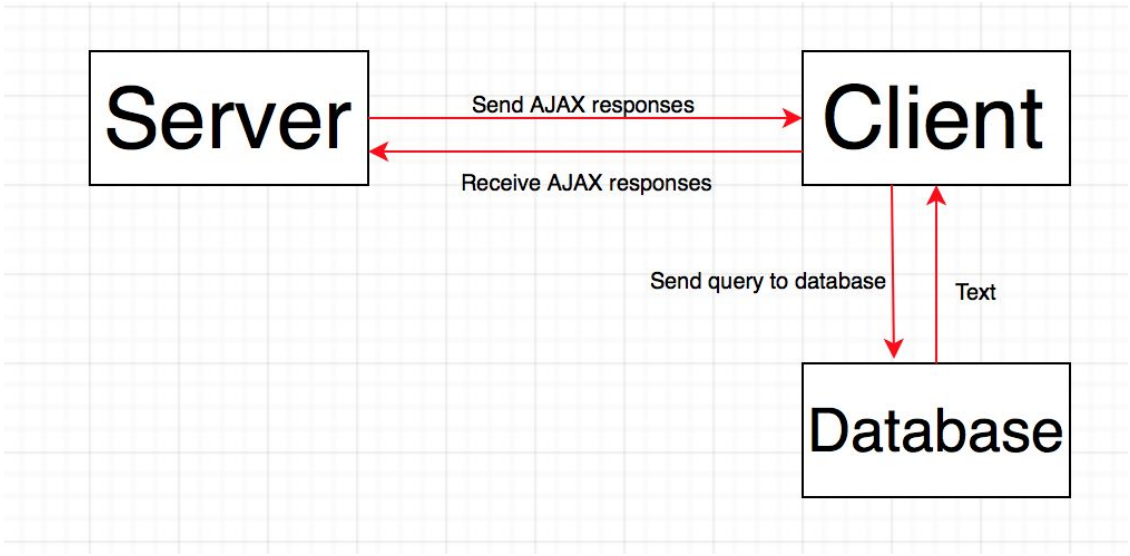
1) When a user enters his password, it should not be displayed to him.
2) A user would want his password to be safely stored using encryption.

3) A user would like the server to be able to perform account verification during login for greater security.
4) A user would like powerful search schemas and facilities that make looking for profiles or jobs easier

DESIGN OUTLINE

Our project is a web application that will use the client server model. Our clients (students) will use our project to search for job opportunities or for student profiles. These queries will be parsed to the server using Angular JS. On receiving these the server will validate these requests, use the database to search for these queries and send the responses using Angular JS.

Our model is designed based on the following model:



Client

- Client sends AJAX requests to our server API.
- Client receives a response from the server.
- The response is parsed to the user using Angular JS

Server

- Receives and responds to the client calls.
- Uses the database (MongoDB) to query the requests and find responses to be sent to the client.

Database

- Our database (built using MongoDB) will be the repository of any information inputted by the user. It stores the profiles of all users, the jobs posted by the users, and the list of all news items posted onto the website. Our database will also serve as an efficient library management system because it will have dynamic information about the inventory that is available in the gear room.

Interactions between Individual System Components

When a user opens the web app, he will be directed to the login screen where he can enter his details. Once he does this, he (the client) requires a response from the server to be validate and allowed entry into the page.

The server queries the database to look for the login credentials that the user entered. If the server gets a valid response, then the user is allowed further entry. If not, the user is prompted to sign up and become a user, and again the server uses the database to save the newly created profile.
Once the user is logged in, he will be able to view the different components of the website.
The database will be used by the server to pull up the data. (Ex the different job opportunities available, the list of available equipment in the gear room, etc).

DESIGN ISSUES

Issue 1) : Choice of database:

We had exploring two possible options:

1) MySQL
2) **MongoDB**

Both MySQL and MongoDB are open source database management systems that have extremely powerful features, and that simplify the process of information storage. After considerable analysis, we chose MongoDB because it has a rich data model and a dynamic schema that MySQL doesn't offer. This means that the database can evolve with different business requirements and this flexibility seems attractive in a large scale project like ours. MongoDB also scales pretty easily irrespective of the volume of data and the throughput.

Issue 2) : Choice of front end language base:

We had two possible options:

1) React JS
2) **Angular JS**

AngularJS is an all encompassing framework that can be used over all platforms easily. ReactJS on the other hand not a full scale framework and hence might cause a problem if we needed to change our platform sometime during the project.
AngularJS is also part of the MEAN framework, which we want to use for the different components of the project, and this is why we choose to use AngularJS.

Issue 3) : Choice of back end language base:

We had two possible options:

1) Django
2) **NodeJS**

We decided to go with NodeJs due to its high prevalence in the world of web development. Another reason for choosing NodeJs over Django is the skill set of the team. Most of the team is decently familiar with NodeJs and not so much with Django. This would reduce the time spent in the learning curve. Functionally, NodeJs is known to have much faster startup time than Django.

Issue 4) How to handle the 'Reset Password' functionality

We had two possible options:

1) **Send an email to the user with a link to reset his password**
2) Send a new random password to the user for him to reactivate his account.
3) Ask the user some security questions before allowing him to reset his password.

The first option seemed to be the most user friendly option, because it is easy to understand. It is sometimes a hassle for users to need to remember the security questions that they had entered earlier while creating their profile. The second option also seems to be more cumbersome for the user. Hence the first option helps us achieve our goal of making a user friendly interface.

Issue 5) What type of architecture are we going to use?

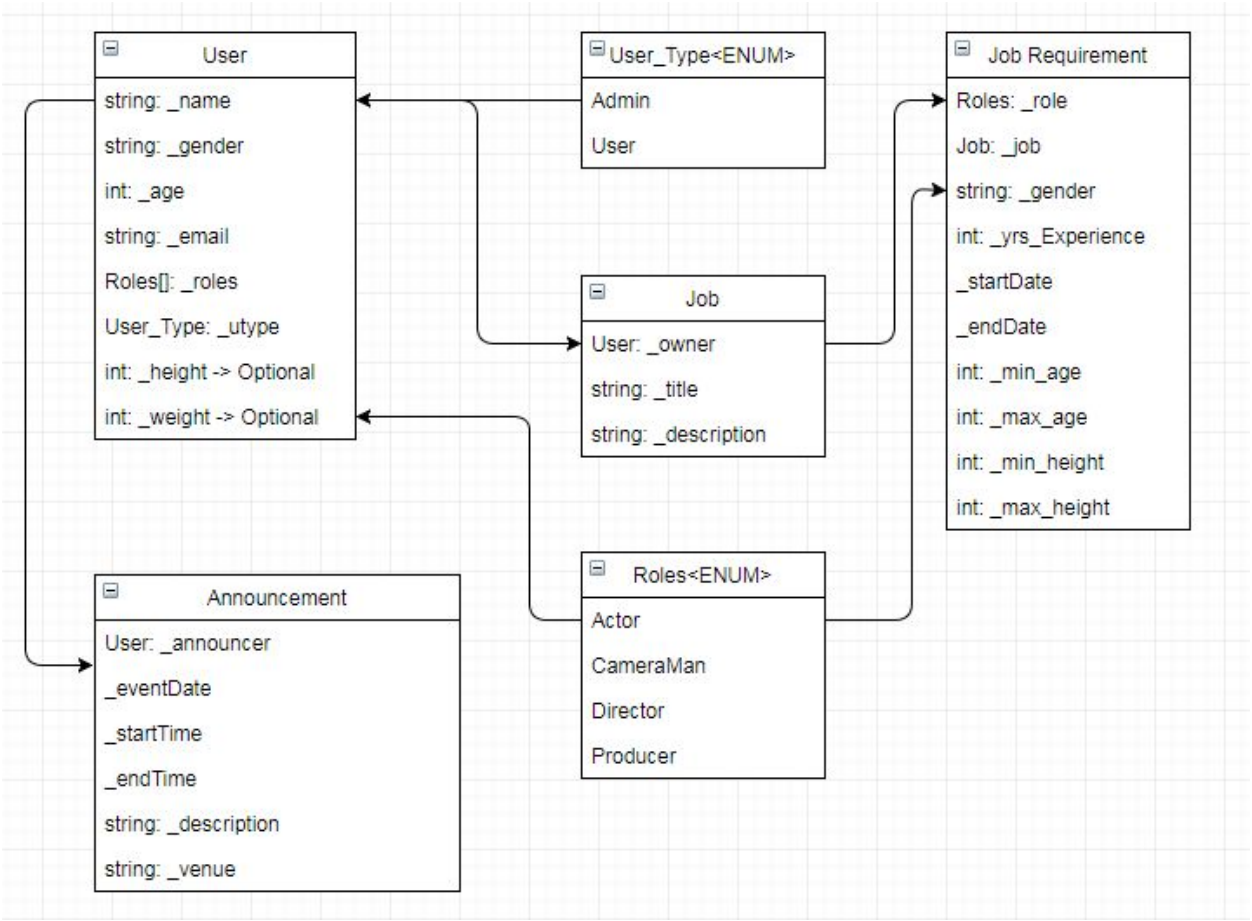
We had two possible options:

1) **Client Server Model**
2) Model View Controller

The interactions between the components of our webapp is based more on a client server system. The client program can access one or more servers to obtain services. This architecture allows for both the client and the server to be designed separately. This is why we chose to use the Client Server Model.

DESIGN DETAILS

Data Class Level - Design



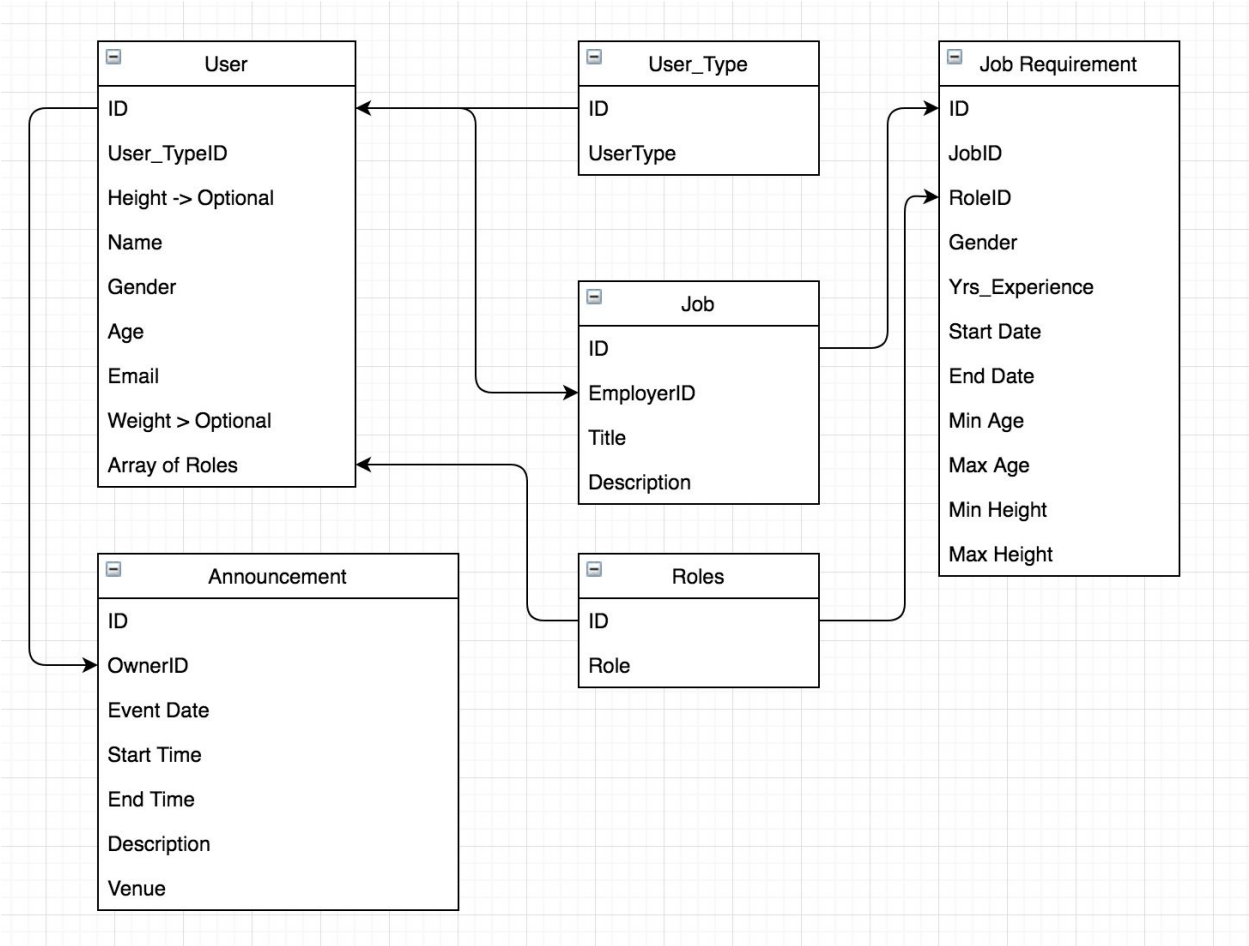
Description of Class Interactions

Our class design is based on our understanding of the different important elements of our project and as such is similar to the database diagram. The classes are also structured according to how they may be stored in our database for data storage.

- **User**:
  - This is the class storing all the user data.
  - It is created when a new user creates an account.
  - A user of type admin will have access to more control of the website such as removing accounts, removing posts, changing the layout, adding posts, etc. This access is granted to the board members of the club.
  - Other users consist of the general public.
- **Announcement**:
  - This class defines the structure for any events posted on the site.
  - Each event must contain the following criteria in order to define the time, location, description and owner of the post.
  - Announcements can only be done by admins regarding club activities.
- **User_Type**:
  - This defines the kind of privileges given to a user.
  - It is an enum class.
- **Job**:
  - This class defines the structure of a typical job posting holding the job title and the job description
  - This class is inherited by the Job Requirement class which links the job type to the requirements
- **Job Requirement:**
  - This class defines all the essential details of a job posting like the preferred age, gender, years of experience and the time frame of work, and the role of the employee.
- **Roles:**
  - This class defines the roles of different users like actor, cameraman, director, etc.
  - It is an enum class.

Sequence Diagrams

Database Design



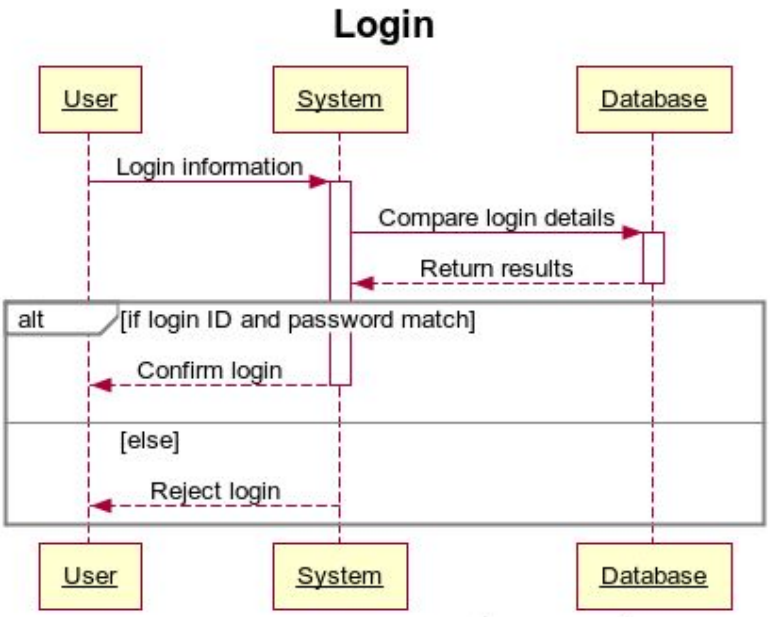
‘

API Routes

Route	Supported HTTP Methods
/casting	GET
/talentSearch	GET
/feed	GET
/login	GET
/signup	POST
/profile={id}	GET
/profiles	GET

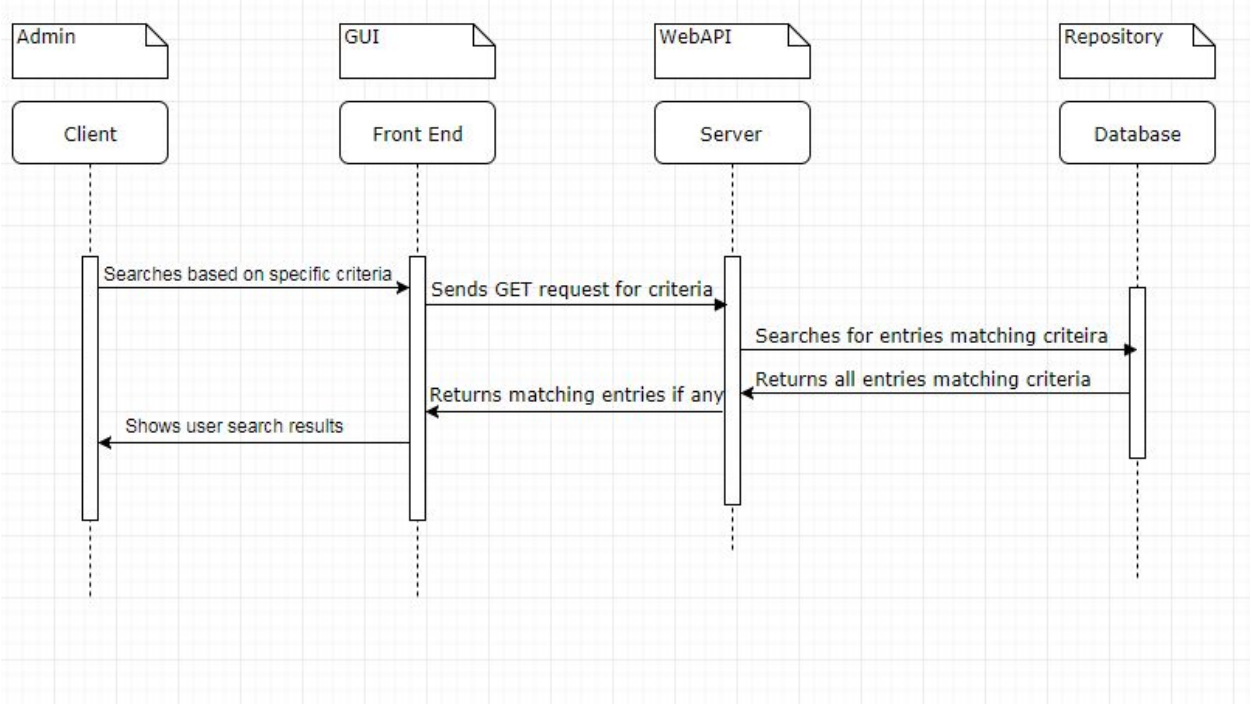
Login Sequence

Sequence of events when user tries to Login



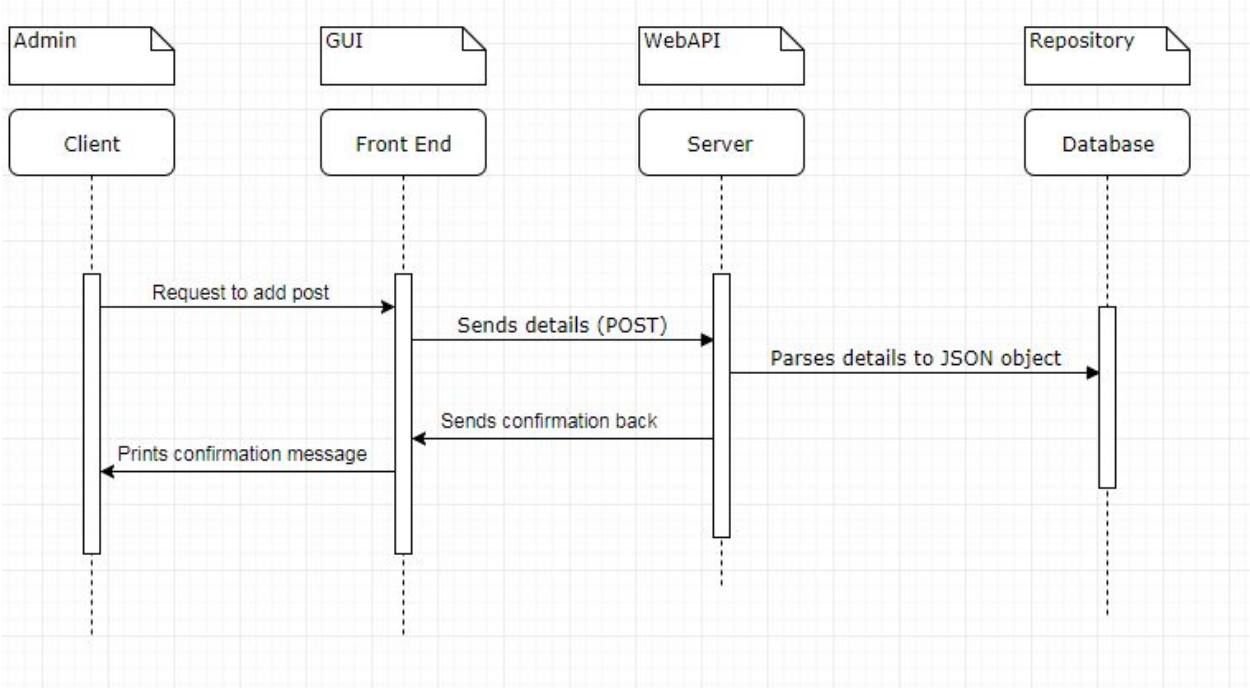
Searching Sequence

Sequence of events when the user searches for a criteria



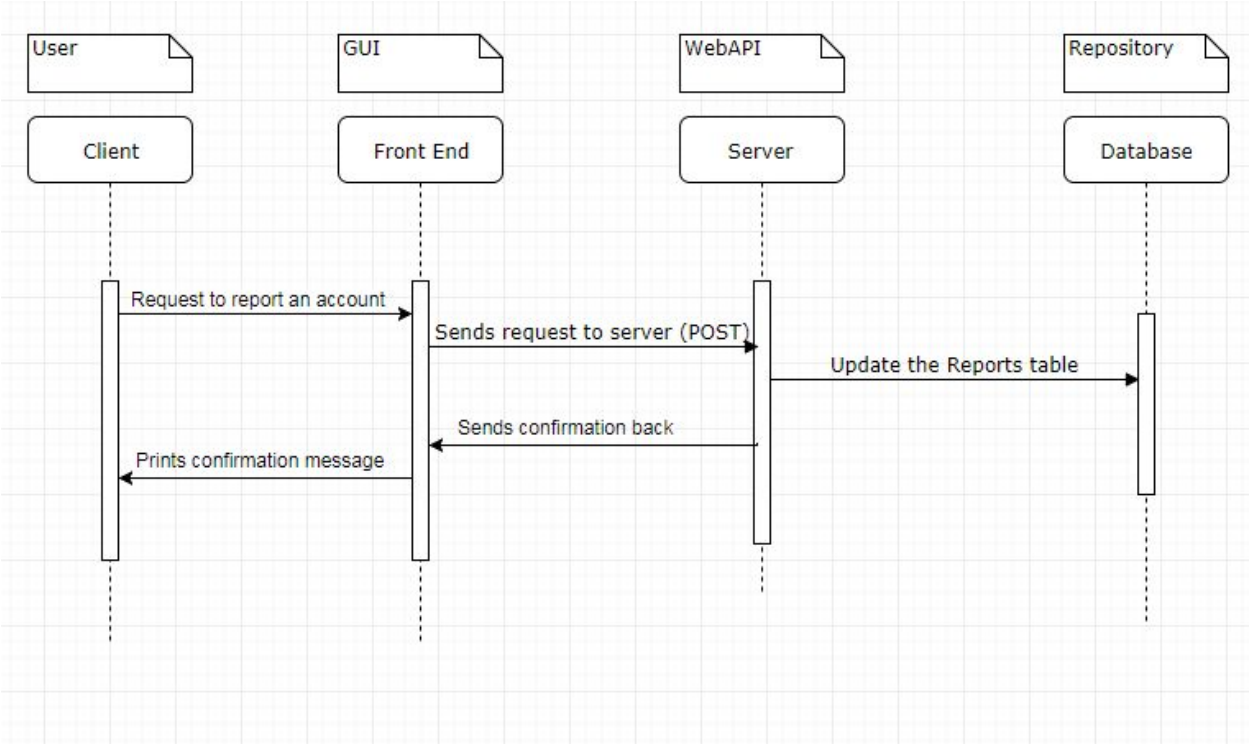
Posting Sequence

Sequence of events when user posts a job



Reporting Sequence

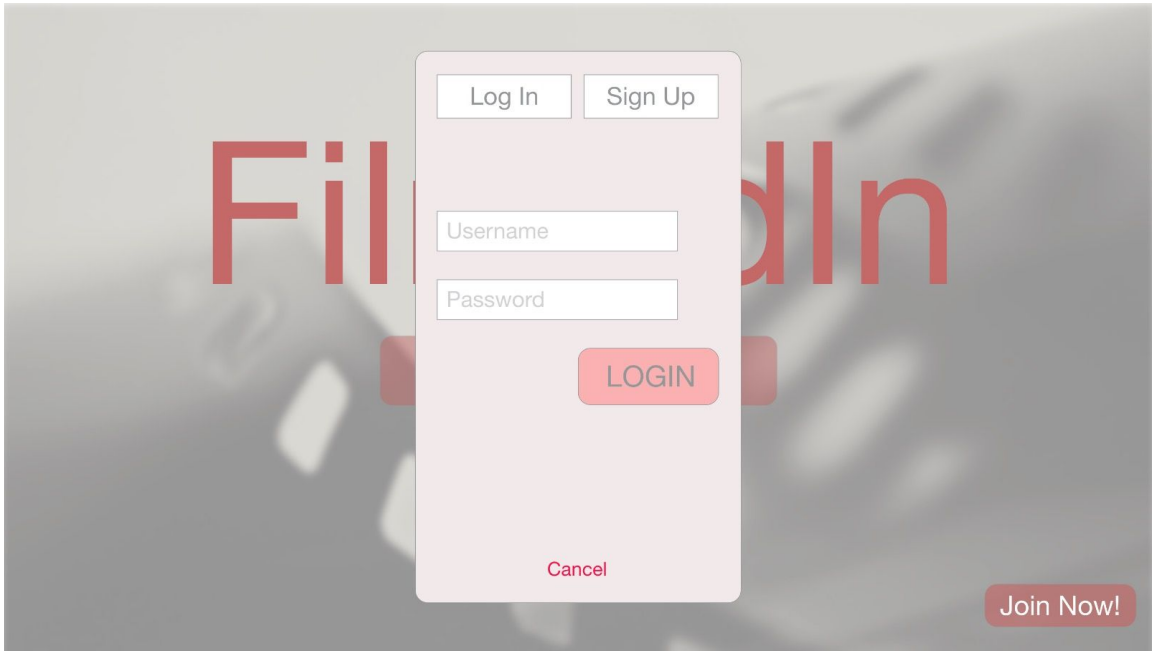
Sequence of events when the user tries to report a activity



UI Mockups

The landing screen:

The login/sign up screen:



Screen to search user candidates: