San Francisco State University

SW Engineering CSC648/848

Section 1

Easy Connect

Section 1 Team 6

Team Lead - Kurtis Chan
Scrum Master - Ethan Vocal
Frontend Lead - Mekonnen Tesfazien
Backend Lead - Hann Zhao
Backend Lead - Ahmar Huda
Git Master - David Lemming

Milestone 2

10/11/2023

Version	Date
1	10/11/2023

I. Data Definitions V2

- **user** Entity - An entity that stores information about registered user accounts
    - firstName - First name of user
    - lastName - Last name of user
    - email - Email address of user. Used for login
    - password - Password of user. Used for login, and will be hashed
    - userCountry - The country in which the user resides or would like to work
    - userState - The state in which the user resides or would like to work
    - userCity - The city in which the user resides or would like to work
    - userJobinterests - The job categories that user is interested in
    - userRemote - Whether or not if the user is interested in remote only jobs
    - userResume - The user’s resume if they uploaded one
    - userJobs - The jobs the user has bookmarked
- **job** Entity - An entity that stores information about job listings
    - jobTitle - The title of the job
    - jobCountry - The country where the job is located
    - jobState - The state where the job is located
    - jobCity - The city where the job is located
    - jobCompany - The company who is advertising the job
    - jobDescription - The description of the job
    - jobRemote - Whether or not the job is remote
    - jobHourly - Whether the job is hourly
    - jobSalary - Whether the job is salary
    - jobPay - How much the job pays

    - jobInternship - Whether or not the job is an internship

- **company** Entity - An entity that stores information about registered company accounts
    - companyName - The name of the company
    - companyLocation - The location where the company is located
    - companyEmail - The email address of the company. Used for login
    - companyPassword - The password for the company. Used for login and will be hashed
    - companyWebsite - The company’s own website if applicable
    - companyJobs - The jobs the company has posted on the website
- **team** Entity - An entity that stores information about the team members of team 6
    - teamName - The first and last name of the team member
    - teamTitle - Role in team
    - teamPicture - Stock photo of team member’s favorite animal
    - teamEmail - School email of team member
    - teamGithub - Github username of team member
    - teamFavshow - Favorite show or movie of team member

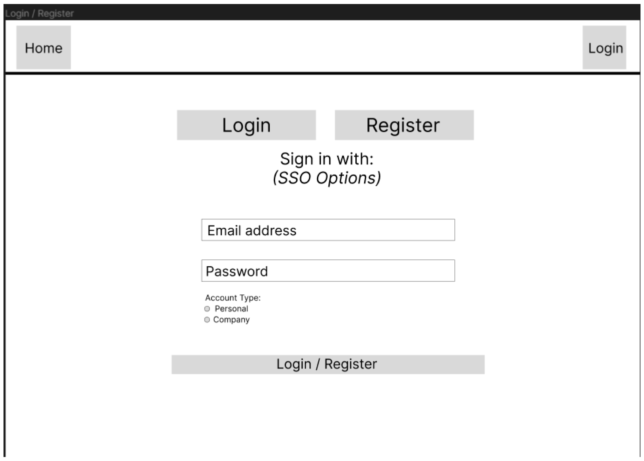
II. Functional Requirements V2

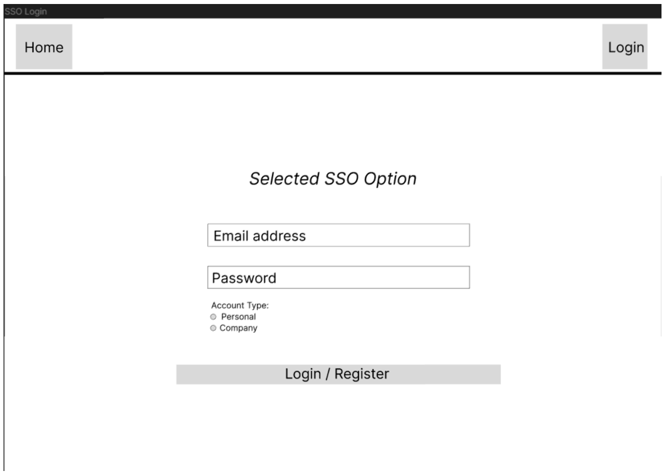
ID	Functional Requirement	Details	Prioritize
1	Users can register an account	1.1) A user can sign up for a new account using an email. 1.2) Account registration requires a password with a capital letter, a symbol, and at least 8 characters long.	1
2	User Login	2.1) Users can log in with their Google account or email. 2.2)The page should display an error when an incorrect password or email is used.	1
3	Search Bar for Job	3.1)Users can search for jobs based on job titles, keywords, or locations.	1
4	Job Post	4.1) The website will connect to JSearch from Rapid API to get new job post information. 4.2) Registered companies can add job posts to the website.	1
5	User Profile Page	5.1) The user will have a profile page. 5.2) The user can update information on the profile page. 5.2) The user can fill out a resume form on the profile page.	1
6	Save Job Posts to Bookmark	6.1) The user can save job posts to bookmark in their account for later use. 6.2) The user must log in	1
7	Apply to Jobs	7.1) Users can apply for jobs using an external link provided by the JSearch API.	1
11	Navigation Bar	11.1) The user can use the navigation bar to navigate the website.	1
12	Remove Job Posts from the Bookmark	12.1) The user can remove job posts from the bookmark in their account. 12.2) The user must log in	1

13	Account type personal account or company account	13.1) During account registration, the user can choose between a personal account or a company account.	2
14	AI for user and job compatibility	14.1) The website will create a job compatible score generated by AI based on a user's personal profile and resume when the user visits a job post.	2
15	AI for improving user resume	15.1) Use AI to give feedback for improving a user's resume.	2
8	Add Job Post by Company	8.1) Companies can create new job posts on our website.	3
9	Company Profile	9.1) The company will have a profile page. 9.2) The company can update company information on the profile page.	3
10	Job recommendation	10.1) Users will receive job recommendation based on user education, major, and experience on their resume.	3
16	Remove Job Post by Company	16.1) Companies can remove job posts on our website if the job was posted by the company account.	3
17	Register an account using Google Account	17.1) Users can create an account using google login.	3
18	Job Search Filter	18.1) Users can filter job posts based on education and experience requirements.	3

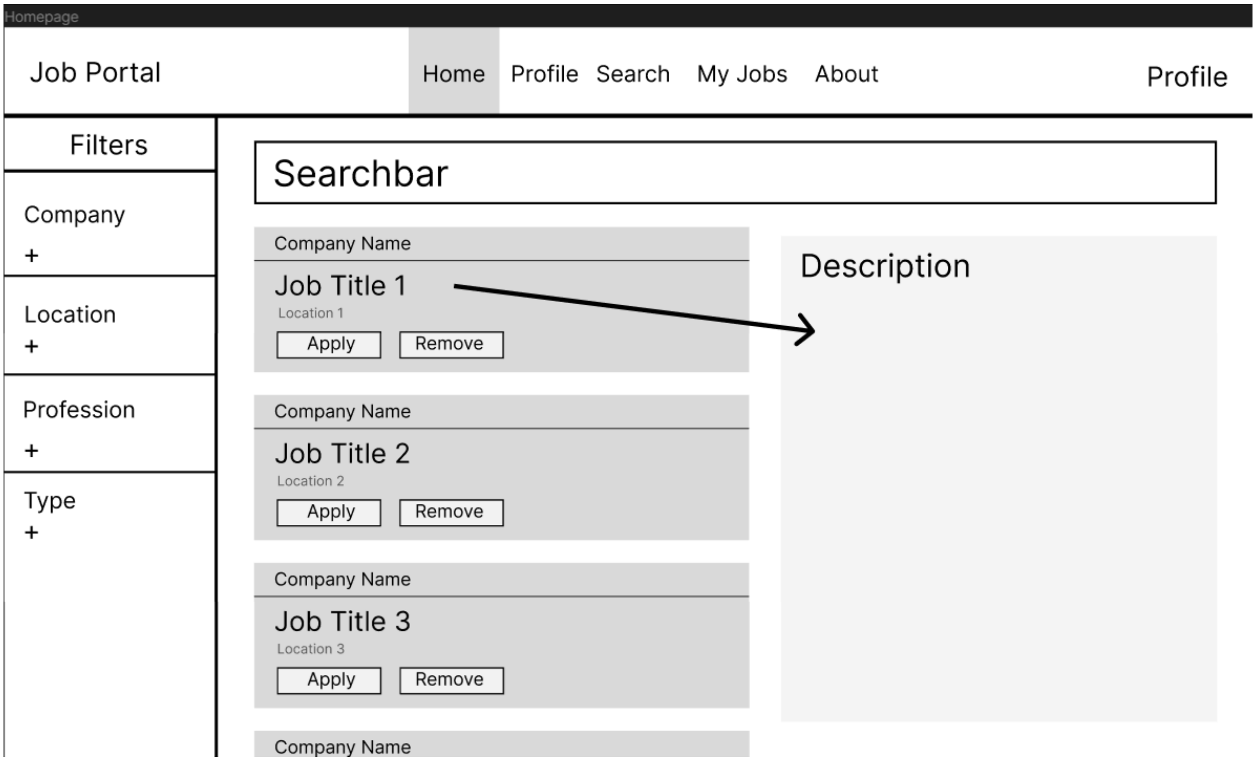
III. UI Mockups and Storyboards

1. Login / Register (with SSO)

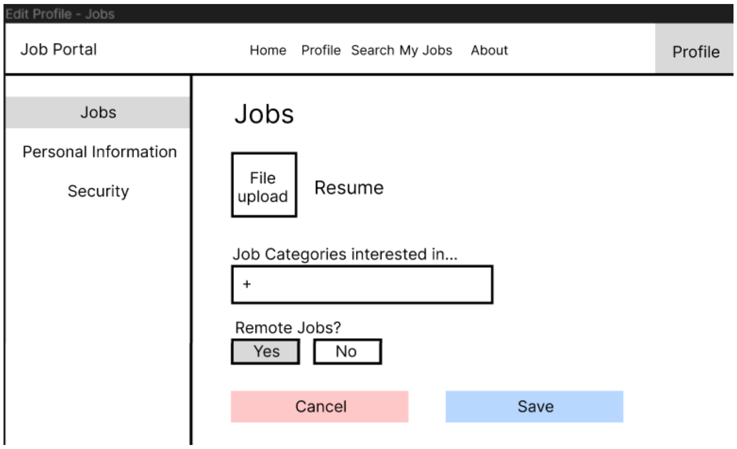


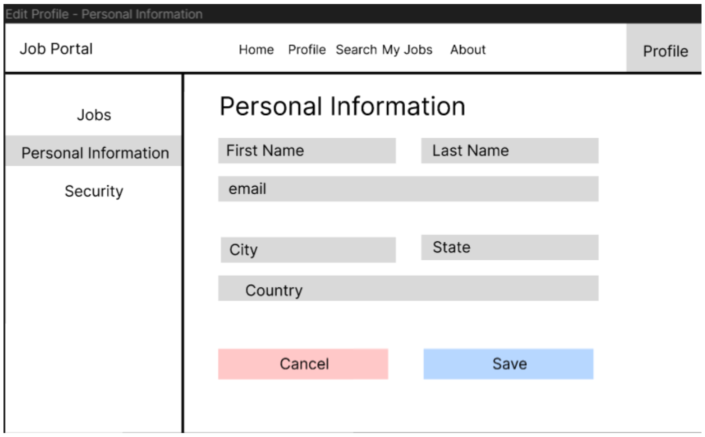


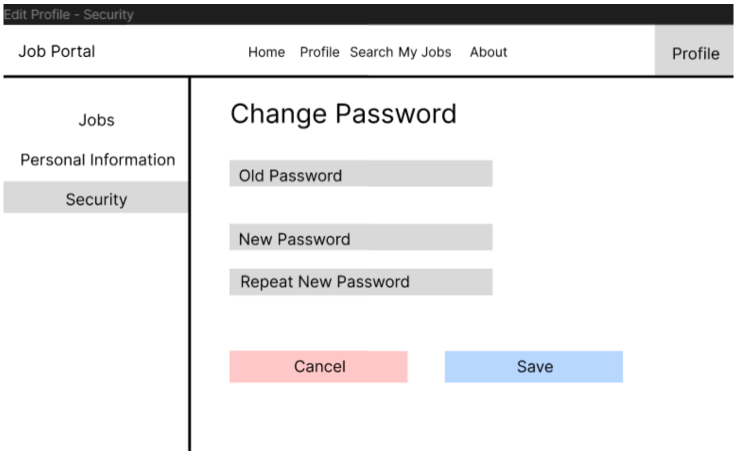
2. Homepage



3. Edit Profile-Page







UX FLow

The general UX flow for all job-search related user stories looks similar. It comprises the three major components shown above. Firstly, when visiting the website, a user wants to login with his existing account, or register a new one. They can do so either manually or through one of the SSO options.

The user is then able to access the website's main functionality through the homepage. Here, a user can utilize the search bar to search for jobs, set filters, and look at the details of a specific job, by clicking on it.

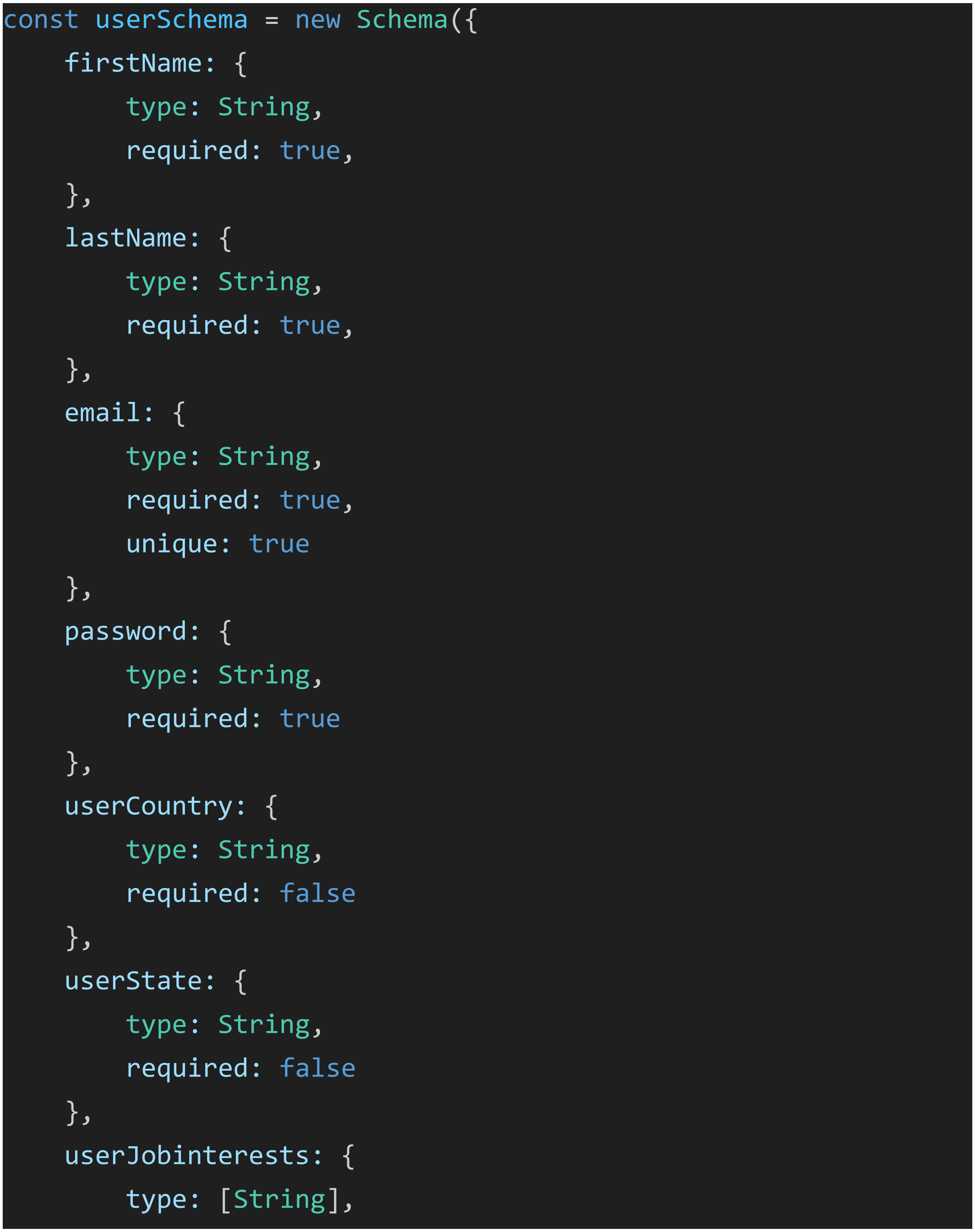
To configure their account, a user can access the profile section, where a variety of options are offered. The tab “jobs" includes uploading a resume or specifying job-search parameters. Under “personal information”, the user can change personal information, such as name, birthday or location. Under “security”, a users can change his password.

Review

Upon reviewing the developed prototype, we consider all the key UX principles, namely usefulness, usability, desirability, accessibility, findability, and credibility to be effectively met. The prototype offers a user-friendly and intuitive interface, ensuring ease of navigation and interaction. The login/registration page is simple, yet contains all the necessary components. Information and job listings are readily findable after logging in through clear organization. Lastly, the platform conveys trust and confidence in its users, demonstrating high credibility. This confirms that the prototype aligns with all essential UX principles, contributing to an excellent user experience.

## *IV.	High-level Architecture, Database Organization*

User Document Schema

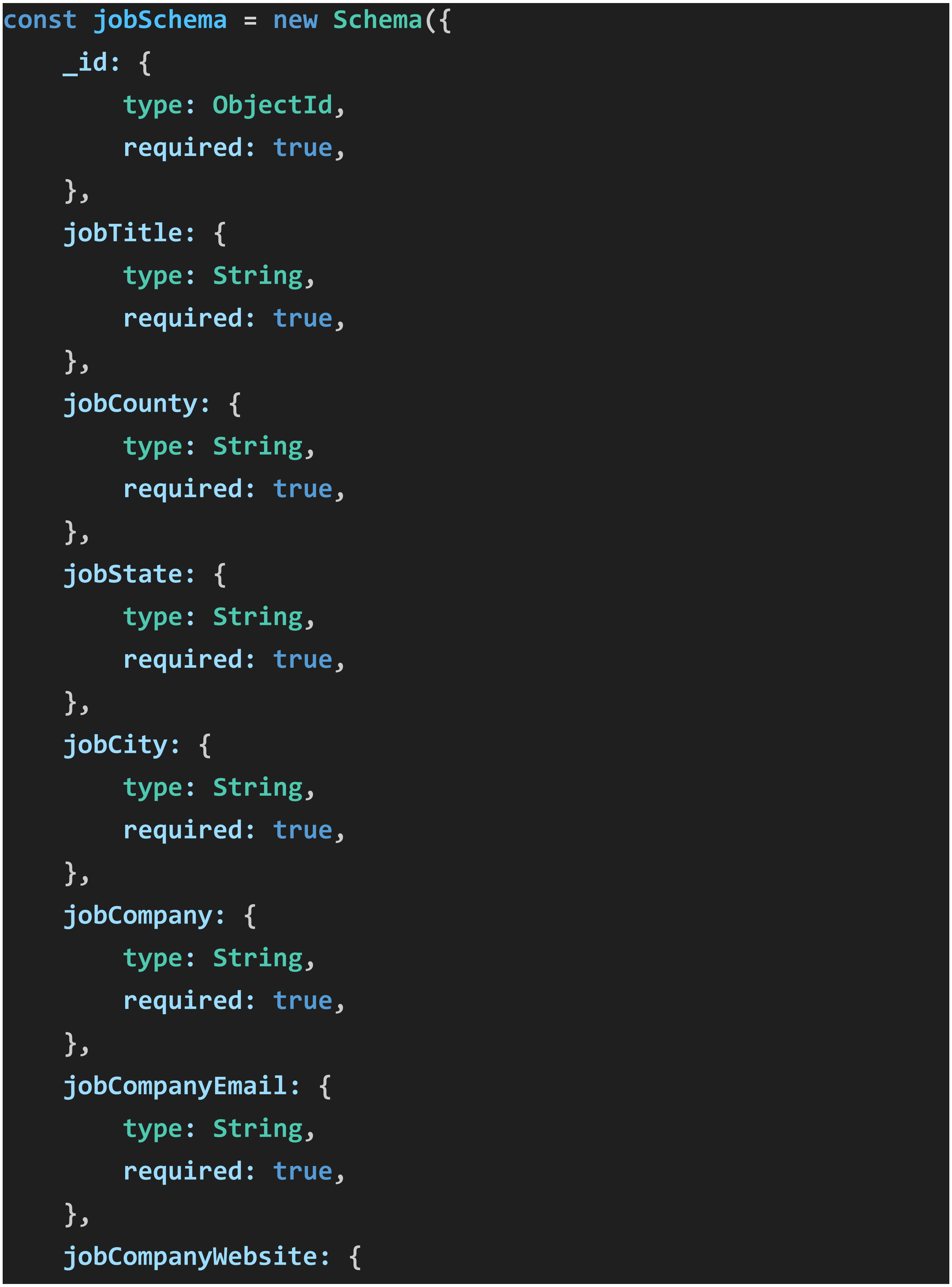


```
const userSchema = new Schema({
    firstName: {
        type: String,
        required: true,
    },
    lastName: {
        type: String,
        required: true,
    },
    email: {
        type: String,
        required: true,
        unique: true
    },
    password: {
        type: String,
        required: true
    },
    userCountry: {
        type: String,
        required: false
    },
    userState: {
        type: String,
        required: false
    },
    userJobinterests: {
        type: [String],
        required: false
    },
    userRemote: {
        type: Boolean,
        required: false
    },
    userResume: {
        type: String,
        required: false
    },
    userJobs: {
        type: [String],
        required: false
    }
})
```

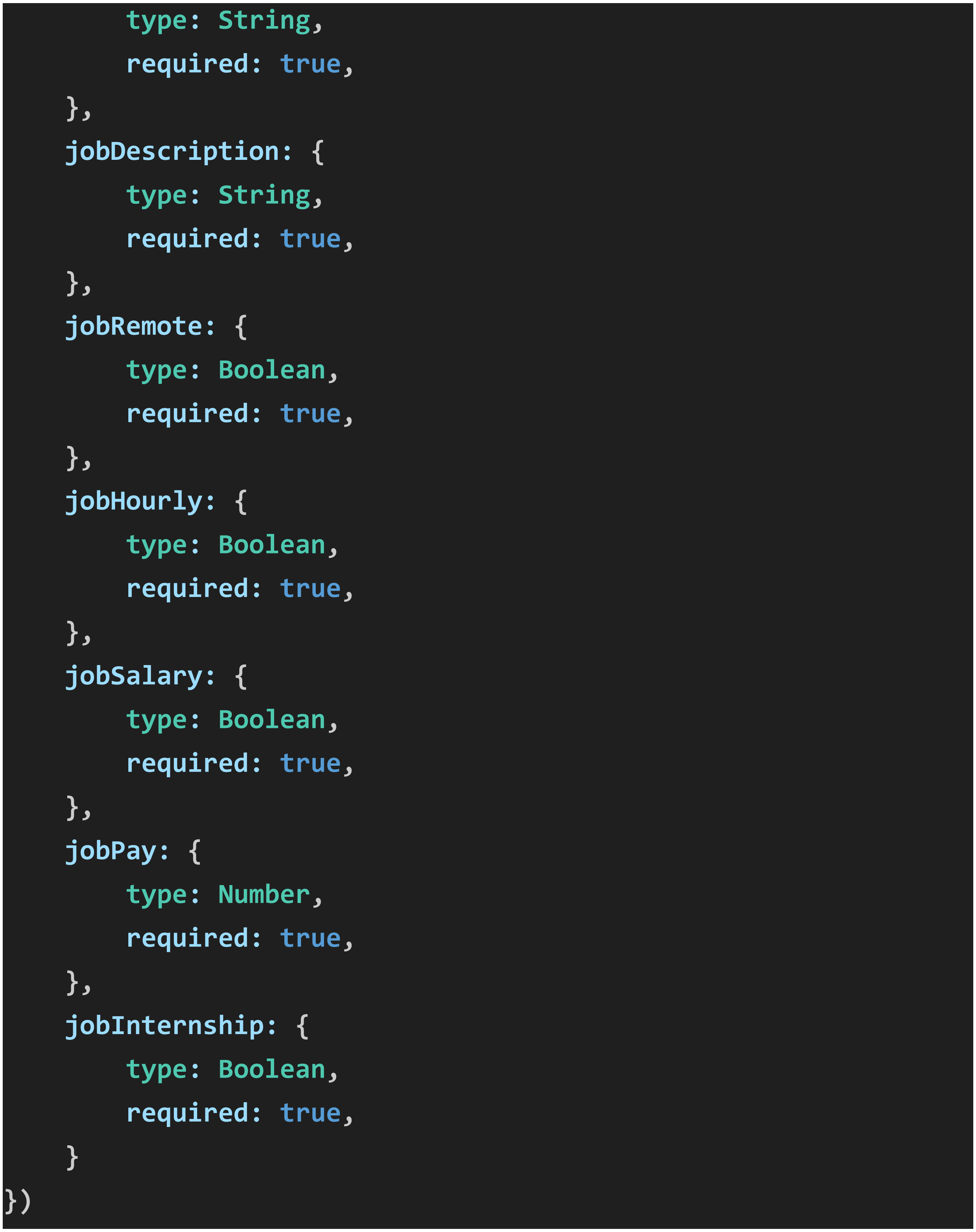
Company Document Schema

```
const companySchema = new Schema({
    companyName: {
        type: String,
        required: true,
    },
    companyCounty: {
        type: String,
        required: true,
    },
    companyState: {
        type: String,
        required: true,
    },
    companyCity: {
        type: String,
        required: true,
    },
    companyEmail: {
        type: String,
        required: true,
        unique: true
    },
    companyPassword: {
        type: String,
        required: true
    },
    companyWebsite: {
        type: String,
        required: false
    },
    companyJobs: {
        type: [String],
        required: false
    }
})
```

Job Document Schema



```
const jobSchema = new Schema({
    _id: {
        type: ObjectId,
        required: true,
    },
    jobTitle: {
        type: String,
        required: true,
    },
    jobCounty: {
        type: String,
        required: true,
    },
    jobState: {
        type: String,
        required: true,
    },
    jobCity: {
        type: String,
        required: true,
    },
    jobCompany: {
        type: String,
        required: true,
    },
    jobCompanyEmail: {
        type: String,
        required: true,
    },
    jobCompanyWebsite: {
```



```
        type: String,
        required: true,
    },
    jobDescription: {
        type: String,
        required: true,
    },
    jobRemote: {
        type: Boolean,
        required: true,
    },
    jobHourly: {
        type: Boolean,
        required: true,
    },
    jobSalary: {
        type: Boolean,
        required: true,
    },
    jobPay: {
        type: Number,
        required: true,
    },
    jobInternship: {
        type: Boolean,
        required: true,
    }
})
```

Team Document Schema

```
const teamSchema = new Schema({
    _id: {
        type: ObjectId
    },
    teamName: {
        type: String
    },
    teamTitle: {
        type: String
    },
    teamEmail: {
        type: String
    },
    teamPicture: {
        type: String
    },
    teamGitHub: {
        type: String
    },
    teamFavshow: {
        type: String
    },
}, { collection: 'teaminfo'});
```

Add Delete Search Architecture	Functional Requirement
add / delete / search / display for Users	When users register

Search / display for job Post	When user search for job post
Add / delete / Search / display for job bookmark	When users save or unsave job posts.
Add / delete when posting new job	When companies post new jobs.

Database Information

Since we are using MongoDB which is a NoSQL database, we can store data in JSON documents as key value pairs and the schema of these documents are flexible so it is very easy to change or update data. We are also utilizing MongoDB Atlas which is a cloud database service provided by MongoDB which helps us to visualize our data more easily and clearly than if we only had to rely on the terminal. In our database, there are three entities currently which are **User**, **Company**, and **Job**. Company accounts will be allowed to post their own jobs to our job board whose data will be stored in the Job entity. However User accounts will not be able to post their own jobs.

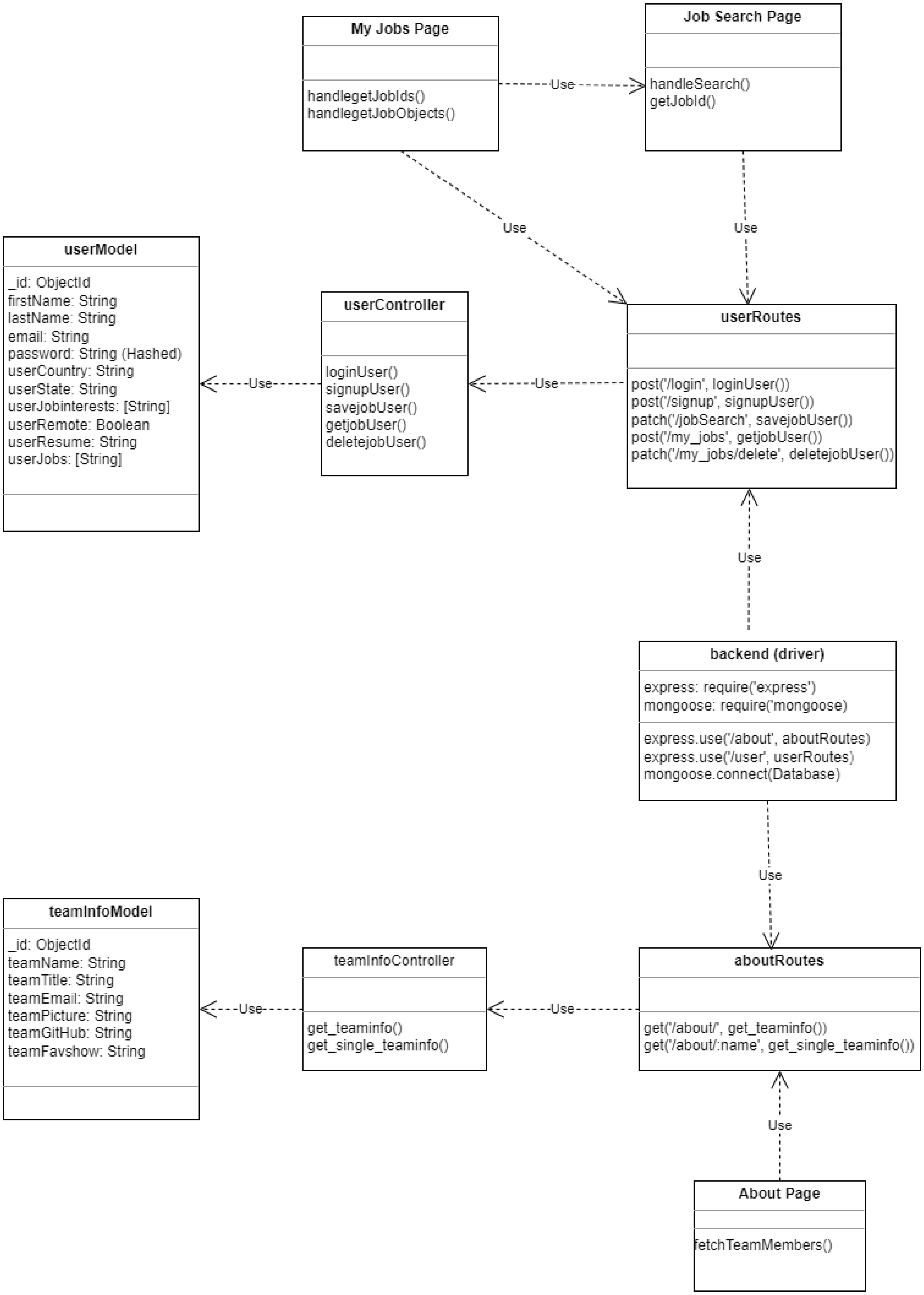
APIs

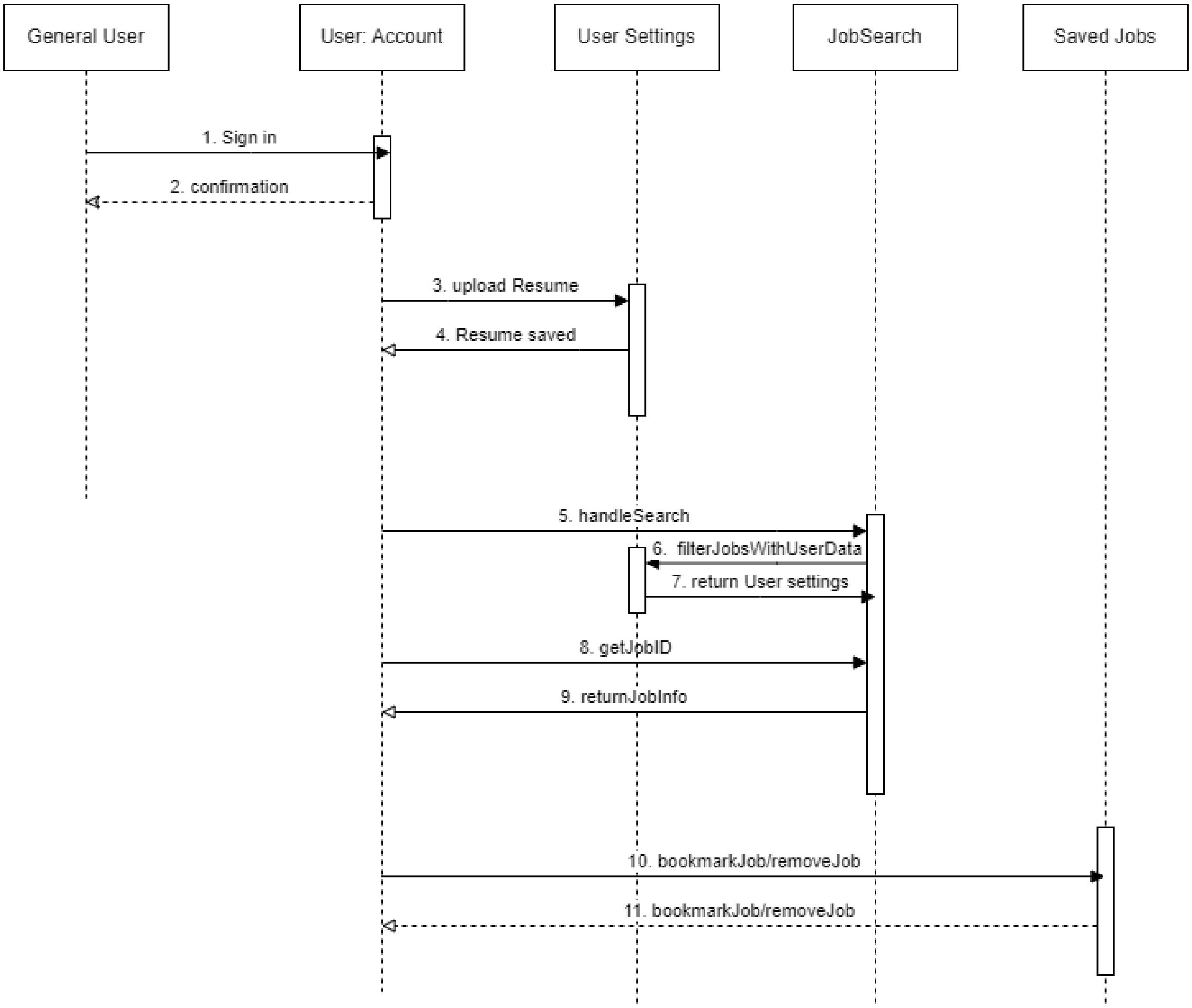
Our backend API is created using the Express Node.js framework which will handle the incoming frontend requests that will be made using the Axios Javascript library. The combination of these will be enough to handle all the operations listed above.

3rd Party APIs

We are currently using a 3rd party API called JSearch from Rapid API. JSearch allows us to search for real job posts with keywords. JSearch also allows searching job posts using a job id. We are also using OpenAI API for comparing user and job compatibility by displaying a compatibility score and list of things that can be improved on a resume.

V. High-level UML Diagram




General User
User: Account
User Settings
JobSearch
Saved Jobs
1. Sign in
2. confirmation
3. upload Resume
4. Resume saved
5. handleSearch
6. filterJobsWithUserData
7. return User settings
8. getJobID
9. returnJobInfo
10. bookmarkJob/removeJob
11. bookmarkJob/removeJob

VI. Key Risks

- **Skills risk & mitigation plan**
    - If studying a certain topic for our app takes longer than expected, production will get delayed and may delay implementation
        - To resolve:
            - Study a little everyday
            - Start early
            - Share resources/guides in the resources channel on discord
    - Uncontrollable factors like family emergencies, work from other classes, and jobs may get in the way of their study plans and app production.
        - To resolve:
            - communicate to at least one person on the team about their situation
            - start early
    - If a team member encounters a severe bug or is stuck on a certain concept that prevents the progress of their task
        - To resolve
            - share problems with the team in the discord channel and consult other members in the designated forntend/backend roles
            - Bring up during team meetings
- **Schedule Risks**
    - If team member believes they cannot finish a feature of the app by the deadline and doesn't let other team members know, it can lead to confusion and may interfere with others' work.
        - To resolve:
            - communicate if they believe they cannot finish their task by the deadline set
            - have another team member assist them with concepts or remaining code
            - make sure deadlines are clear to everybody

    - If team lead does not make clear requirements for the functionality of the app and deadlines for tasks can lead to confusion for the group and further delay progress
        - To resolve:
            - suggest a team meeting to go over the requirements and specifics of the functionality of the app or feature or ask when the deadline for a certain task is

            - message team lead to verify what functionality needs to be implemented and by what time should it be finished

- **Teamwork risks**
    - Two team members end up working on the same task and end up with two versions of the same function which delays other tasks
        - To resolve:
            - communicate what we are working on during meetings
            - use Jira and only implement the task you take
            -
    - Backend and frontend do not communicate how each other’s code work for example backend api and what frontend will need to work can lead to unorganized code and confuse others trying to go over their code
        - To resolve:
            - hold regular meetings with team members
            - declare how backend api should work and inform other team members on how to use it
            - review code every deadline to have everyone on the same page

VII. Project Management

In our team of six members, we have one front-end and two back-end lead developers leaving the remaining three members helping out with whatever our developers need assistance with. We decided that we have two back-end lead developers instead of two front-end because we all have minimal experience with topics pertaining towards back-end. For this Milestone, our back-end lead developers aimed to complete: the "My Jobs" page, "Search Job" page, and to update the database schemas . On the front-end side, our lead front-end developer aimed to complete the "Registration" page.

Based on our schedule and availability, we agreed on to hold two meetings per week: Tuesdays from 2-3pm, and on Thursdays from 1-2pm. For our first meeting- Thursday, September 28, we had a brief discussion and reviewed our requirements for Milestone 2. Our second meeting- Tuesday, October 2, we divided the Milestone 2 tasks up between us team members as well as discussing what state we aim to get our website to by the submission date. We as well split the remaining three members up to help with the front-end and back-end side of the website.

Using the online-agenda *Jira*, as well as using *Discord*, we found that communicating about our tasks for Milestone 2 was a breeze. Setting priorities, splitting the Milestone 2 tasks, and planning ahead is something we all benefited from. In the future, we will continue using *Jira* to keep tasks organized as well as track, and *Discord* as the main source of communications.