SENG2021

Deliverable 2

Group: AT3K

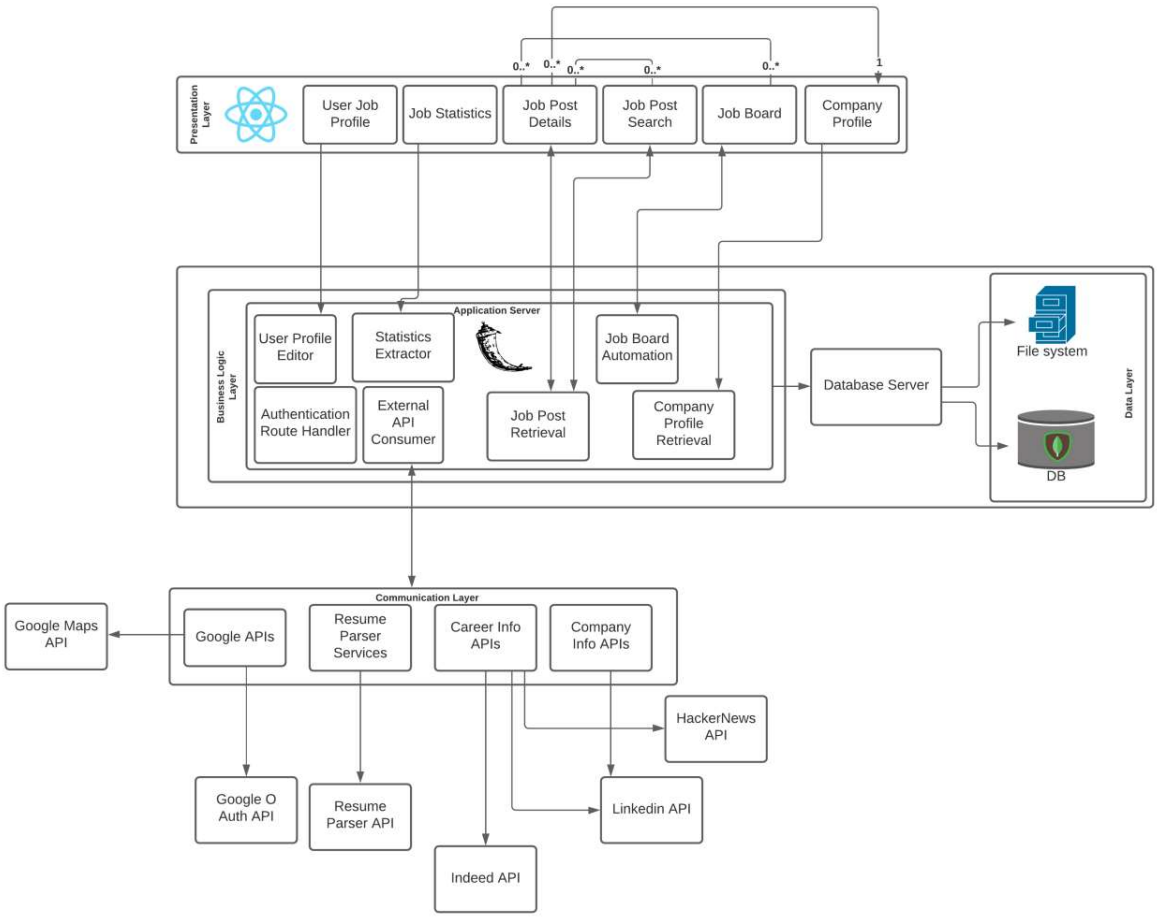
12th March 2021

Summary

The objective of this report is to provide a clear justification of the components and technologies that comprise our application alongside providing diagrams which illustrate the integration of our design. Our application aims to provide an intuitive platform for users to manage job applications from various sources. The main factors when choosing the technology for each 'level' of our application was ease of use, ease of integration within the system and how suitable it is for our purpose. The chosen technology for our frontend was React JS along with Material UI for its library of components. This front end will communicate with a Flask server via the REST API as it is a system that all team members are familiar with. The function modules will communicate with the MongoDB database along with the Google, CareerJet, HackerNews and OpenCorporates APIs. The main design elements of the application are evident through the sequence, UML class and ER diagrams which are used to illustrate our use cases, frontend interactions and data model respectively.

Part 1: Software Architecture

System/Architectural Diagram:



Tech Stack Choice and Justification
Frontend:

Language choice and Front-end framework/library

Traditionally, the web was made up of the 3 technologies HTML, CSS, and JavaScript, which is powerful enough to build the front end of any modern website. However, as web applications grew in complexity, tools were created which addressed issues with building websites using native HTML/CSS and JavaScript. One such tool is TypeScript, a superset of JavaScript which adds static type definition to the language as one huge issue with JavaScript is its lack of type declarations, which could result in common bugs. With typescript, testing and validating code becomes easier, as the typescript compiler does type checks and having static type declarations means better documentation of the code. However, JavaScript was chosen as the language of choice for the frontend due to its simplicity, as additional tools are not required to be set up for the project, such as the typescript compiler (tsc) or babel, and most of the team is familiar with JavaScript rather than TypeScript. Using JavaScript also provides the flexibility to eventually migrate over to TypeScript gradually as the project scales, as JavaScript is valid TypeScript and TypeScript supports gradual adoption.

With the migration of JavaScript from browsers into backend runtime environments in the form of Node.js, it becomes easier to leverage the power of libraries and frameworks through NPM or Yarn to assist with web front end development. To assist with building an efficient and well-organized front end, a component-based UI framework/library can be used, which provides reusable and performant UI code. Such tools include React, Angular and Vue.

React is a frontend JavaScript library which uses JSX, a declarative programming language which is a superset of HTML, allowing injection of JavaScript into it. React uses a virtual DOM, which is a copy the website's DOM, which is faster to manipulate than the actual DOM. React by itself is minimal in size, as many features are not built into React by default such as routing. Angular is a TypeScript based framework, a successor to AngularJS. It comes with a powerful command line tool called ng to handle tasks including building, testing, and serving angular applications. Vue is a progressive JavaScript framework which allows the flexibility to start simple, and progressively add in tools and features to scale to more complex web apps. Vue also makes use of a virtual DOM for better performance.

While both Vue and Angular are viable options, we decided to use React because of the following factors:

- Performant because of using techniques such as the virtual DOM
- its huge community, which offsets its weakness of not having many basic features built in, as a huge ecosystem of 3rd party libraries, packages, tools, and extensions exists, with routing handled by the react-router-dom library, state management with Redux, a huge amount of component libraries such as material UI,
- and because most team members are familiar with React due to its popularity.

Front-end State management

With more complex React applications, some sort of state management becomes necessary to avoid issues like prop drilling, which involves passing properties down multiple levels, through components that do not need the property. Redux is a third-party state management library which allows a global state that can be shared between react components without needing to pass data top-down. While Context API is a first party feature built into react which also allows for global state. For this project, Context API is more suited as it is

simpler to use, and better for smaller applications, while Redux's use case is more geared towards applications which have a medium to large-size codebase and worked on by many people (specified in Redux's documentation).

Styling Tools

When it comes to creating UI components, there exist libraries which provides many reusable components, such as buttons, navigation bars, modals, etc. The most used component libraries for React are bootstrap, a mature web component and layout framework made by Twitter and Material UI, which is based on Google's Material Design which supports the best practices of user interface design. Both of which comes with a huge amount of different UI components and a grid system for layouts. We decided to use Material UI since it was made for React, while Bootstrap was originally a html/CSS library, although it does come with support for react through libraries such as react-strap. However, Material UI comes with a larger number of components, and a huge number of themes which can be bootstrapped, which is what this project did, and templates which can be combined to form a starter. Allowing us to save a lot of time without having to write as much boilerplate code.

Backend

Database

For the database, **MongoDB** is chosen in this project. Initially, there were two common database options to consider – PostgreSQL and MongoDB. PostgreSQL is a relational database, which supports most of the SQL standards, stores the data in tables, and uses schema; MongoDB is a non-relational database that is schemeless and provides support for JSON-like storage. After doing some research, we find that MongoDB provides more flexibility since different types of data can be saved in a separate JSON document. Furthermore, MongoDB is more friendly to us. While using PostgreSQL requires the basic knowledge of SQL and may be time-consuming to develop the schema, MongoDB allows us to simply construct a JSON-format document and store the data, which has a smoother learning curve and is suitable for rapid development. Nevertheless, MongoDB has its limitation. For example, it has high memory usage due to the lack of functionality of joins; the maximum size of BSON document (binary representation of JSON documents), where the data records are stored, is 16 megabytes. However, the limitation of MongoDB is acceptable since the scale of this project is small, and it will not involve a large amount of data. Hence, MongoDB becomes our final choice for the database.

Object Document Mapper

Since non-relational databased has been selected, we would like to use **ODM (Object Document Mapper)** framework. It is a similar technology to ORM (Object Relational Mapper), but it is developed for document databases. Although running queries in the backend server can also achieve the development goal, implementing ODM is more efficient. ODM maps a document database system to objects, making data access more abstract and portable, eliminating repetitive code, and allows the implementation of object-oriented design patterns. On the other hand, there are lots of useful libraries that support ODM, such as Mongokit, Pymongo, Mangoose for Python, which largely increases development effectiveness. Although using ODM may reduce the performance with a huge volume of data, this drawback is acceptable since this project will not involve a large amount of data.

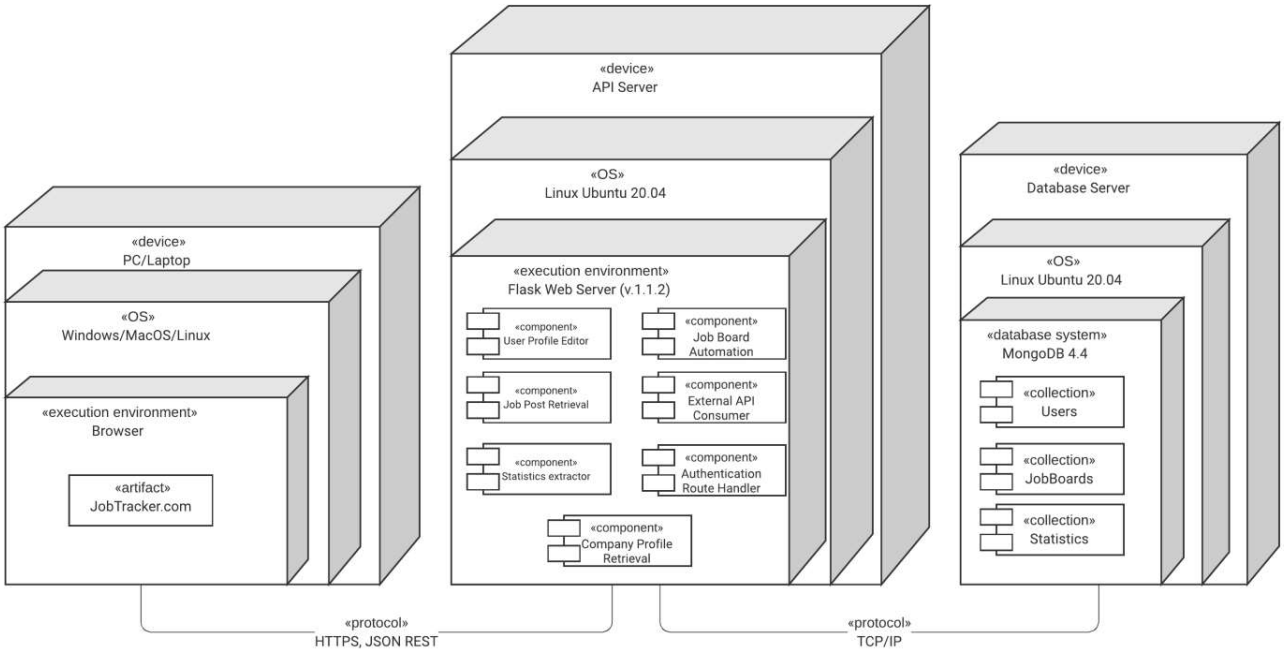
API Management

To communicate with the backend, API management is important. As popular API management tools, Rest API and GraphQL were considered. Our final decision is to implement **Rest API** because all team members understand how it works and how to implement the framework, boosting our efficiency. Although GraphQL is more flexible than Rest API since it allows receiving a lot of different information on the same URL, Rest API is considered more adequate for this project by considering the limited time.

Web Server

Lastly, for the backend server, the **Flask** framework that written in Python will be implemented. Firstly, Python is considered appropriate to use in this project, since all the team members have learnt Python. Then, we had two options to develop the web server in Python – Django and Flask. Django provides more built-in tools than Flask, but it is considered too "heavy" for a simple project, which may increase the size of the application, loading times, and complexity. Conversely, Flask is light and quick, although it does not have many built-in tools, the same functionalities can be achieved by importing modules or libraries. Additionally, a big advantage of Flask is that it has a useful extension called Flask-RESTX. Flask-RESTX provides a coherent collection of decorators and tools to document the API used in the project, so that Swagger API documentation can be automatically generated from code, saving time and increasing effectiveness.

Deployment:



The backend API server will be deployed separately on a VPS from a cloud provider such as AWS. The MongoDB database server will be run locally on the same environment as the backend API server. The frontend build will be deployed independently from the backend API server on Netlify's hosting platform.

Netlify additionally provides a configurable CI/CD pipeline and automatically deploys builds of the project from the project's remote repository and provides additional features such as version control, site previews and configurable environment variables.

The project will use Docker, an OS-level virtualisation tool for convenient and portable deployment. 'Containers' are essentially abstractions at the application layer that package code and dependencies together. The choice to use Docker for this project is based on the time and resources that can saved by eliminating compatibility issues of deploying the API server on different environments and operating systems. With Docker, it is convenient to set up and configure an identical development environment for the whole development team. Docker and other container technologies use much less resources than deploying projects on virtual machines, which was previously the standard method of hosting applications on different systems. Unlike virtual machines, Docker containers share the same OS kernel and exist concurrently with other running container instances. Despite the steeper learning curve for the development team, the comprehensive documentation and massive community around Docker makes it a viable technology to include in the DevOps toolchain.

Deploying the application will also involve setting up an NGINX reverse proxy server, a lightweight open-source web server that can handle load balancing, caching and other functionalities. Having an intermediate reverse proxy server handle web requests before it reaches the API server provides a useful level of abstraction for controlling network traffic, cache frequently accessed static content and handle SSL encryption so that the development team can focus on building the business logic layer. Furthermore, the development team has had prior experience setting up NGINX as a reverse proxy so this would allow the team to avoid spending resources on learning how to set up and configure other alternative web servers, such as Apache.

External Data Sources:

APIs We Will Use

Careerjet

Careerjet's API offers their API in many languages, but we will be using either the Java or Python language. The API has an MIT licence which allows us to use this API without many restrictions. However, it is difficult to find documentation for the Java format and documentation for the Python format is found on an unofficial Careerjet API version. The API also has limited calls available for use. Despite these disadvantages, the information that Careerjet provides is immense and doesn't require an API key to access the API. Hence, we will be using the API.

HackerNews

HackerNews' API has extensive documentation and is well-supported. The API returns results in JSON format and there is also an unofficial Python wrapper available if required. The MIT licence also allows us to use the API with little restrictions, but the information the API provides is limited compared to Careerjet. But since HackerNews API does not require a publisher ID or company to grant access, it would be easy to extract information. Thus, we will be using the API in conjunction with Careerjet's API to supply us with the job search data.

OpenCorporates

The OpenCorporates API provides us with company details in JSON or XML format which we require for the "Company" aspect of our Job Tracker application. The API can be used with or without a key, although there is a restriction on the number of API calls we can make if we decide to use the API without a key.

However, it costs money to buy a subscription to increase the number of calls, so as a result, we will most likely be using it with a basic and free API key, or without a key. The API is also under an Open Database Licence so we can freely use the information that we extract.

Resume Parser

A feature of our application compares the user's current knowledge to the required knowledge for each job and this information is only available through the user's resume, thus requiring us to use a resume parser API. The API parses the user's resume into JSON text that our backend can then extract when required. The API is well-documented and has a GNU licence allowing free use, however, it does have a restricted number of calls that could affect our application.

Google Maps

Google offers many APIs including an API for Google Maps. We may require this API as the job searches can be sorted based on the user's location. As it is a Google API, it comes with detailed documentation and both Python and Java wrappers, so it is well-supported for our application. Company or organisation data is not required to generate an API key and combined with an Apache 2.0 licence, we can use the API with little restrictions.

Google O Auth

We will be using Google O Auth API for the authentication aspect of our application. Similar to the Google Maps' API, the Google O Auth API is well-supported with Python and Java wrappers, has extensive documentation and includes an Apache 2.0 licence. Similar to Google Maps API, Google O Auth API is easily accessible through a key generation. Our application requires this API so that external apps can access each user's data without revealing the user's sensitive information, such as their password.

APIs We Will Consider Using

Github Jobs

Github Jobs API is a public API and is very easy to use. It allows job searching and viewing using HTTP with the option to return results in JSON. However, the filters available with Github Jobs is quite limited which could pose a hindrance to our application. The documentation is straightforward and simple. Like the previous APIs, Github Jobs API will only be considered if we need the extra data or if we have enough time to implement it.

StackOverflow

Like the other APIs, StackOverflow's API offers a similar range of data for job searching and has ample documentation. It also has an MIT licence, however, similar to Workable, we will only use the StackOverflow API if necessary or if we have more time, as the benefits of implementing another career API currently does not outweigh the time taken to implement it. Also, a possible advantage of using StackOverflow instead of Workable is that it does not require company details to generate an API key. However, it does contain restrictions on the number of requests that can be made.

APIs We Will Not Use

Indeed

The Indeed Job Search API returns the response in JSON or Python. From initial inspection, the Indeed API has extensive documentation and an open-source licence which would work well in conjunction with our Job Tracker app. Indeed's job searching also returns results from a wide variety of companies and job types, and

hence, can supply with majority of the data we need. Despite the API being exactly what we need, a publisher ID is mandatory for API usage which we cannot apply for as our project does not satisfy the requirements.

Linkedin

Linkedin's APIs have lengthy documentations and are based on REST and JSON. The licence for the API states that we are able to use the content accessed through the APIs within our application. Linkedin also provides information about company pages and this will be useful for our company-related aspects of the Job Tracker app, which we currently do not have. However, similar to the Indeed API, Linkedin requires a company and company link on the site to grant access to the API. Thus, while Linkedin's API would a huge asset for us, we cannot use the API.

Adzuna

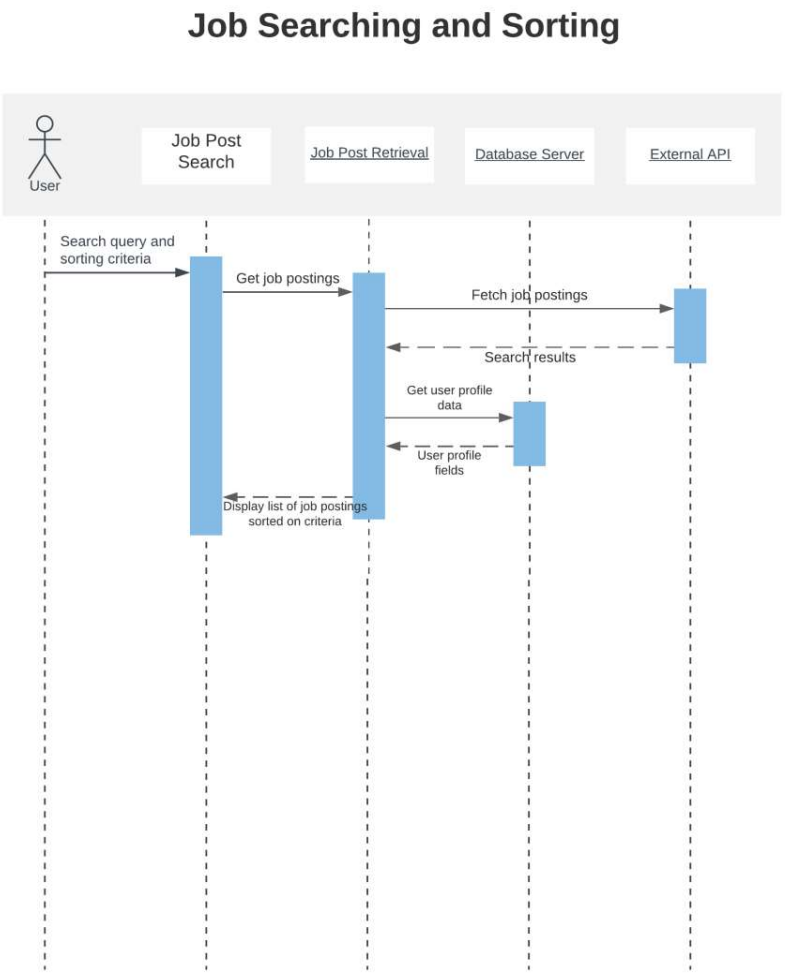
Adzuna's API offers a range of features that previous APIs already include, however, the API can also get a salary estimate for specific jobs. While this feature seems quite useful, it is likely that the time constraint on the project stops us from implementing it. The salary estimate would also need to be converted into AUD which would require more work as we would need to use a currency converter. Aside from this feature, the API does not offer anything new. The API has adequate documentation and has an Apache 2.0 licence, but requires company details to be given access to the API, thus we will not be able to use the API.

Workable

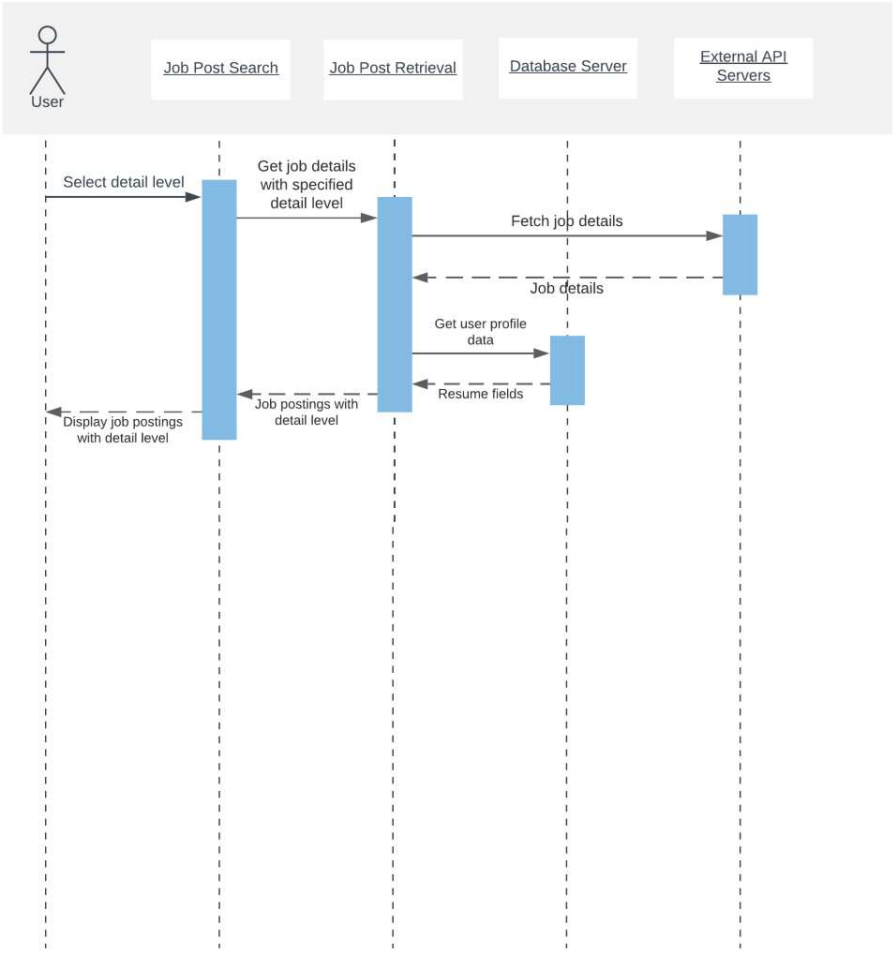
Workable's API has a considerable amount of documentation which would be highly useful to us and is well-supported, but to access the API, we are required to provide company details which we do not have as this is a small project. So similar to Linkedin and Indeed, we will be unable to use this API despite it providing majority of what we require.

Part 2: Software Design

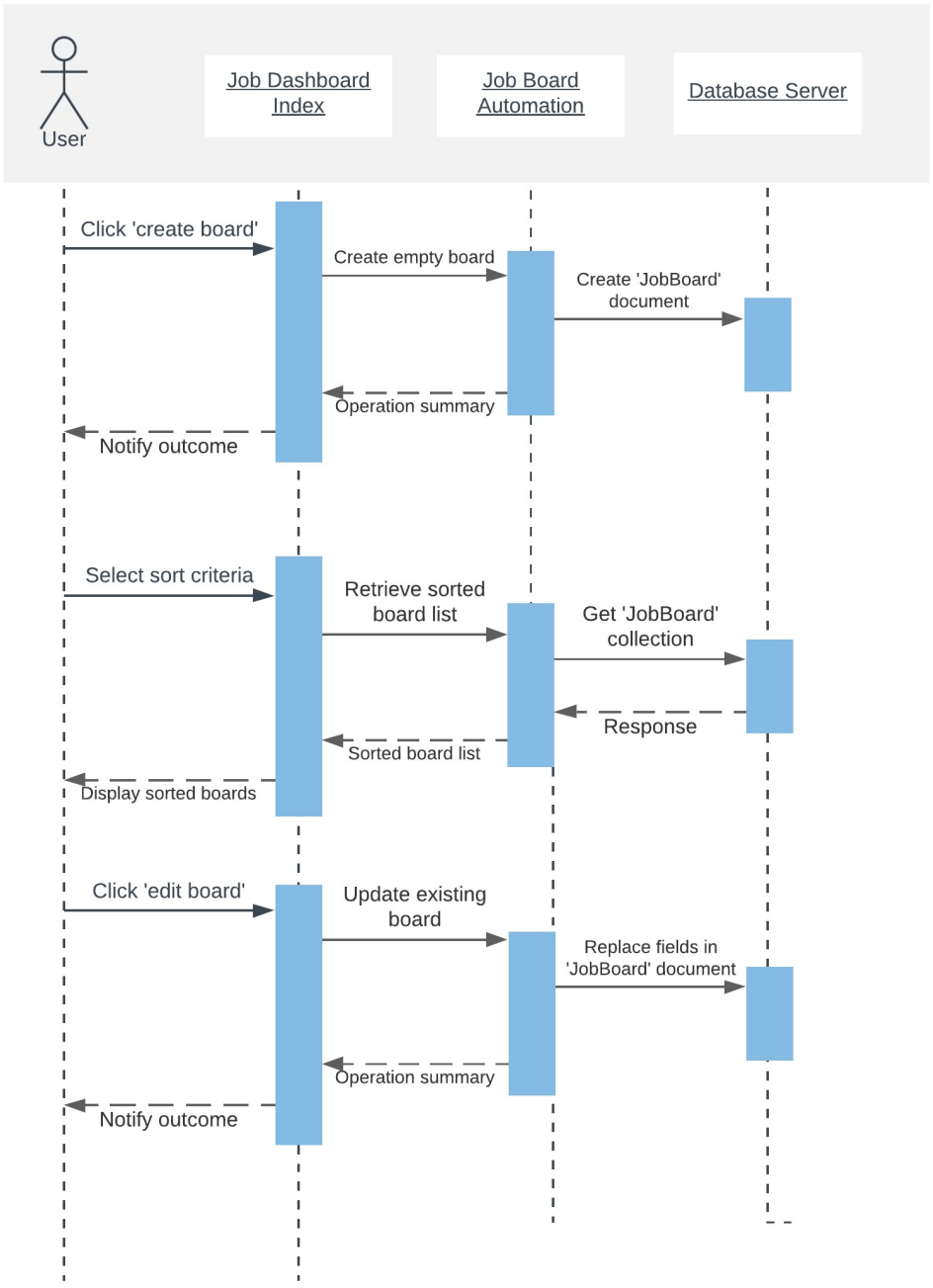
Use Case / Sequence / Interaction Diagrams:



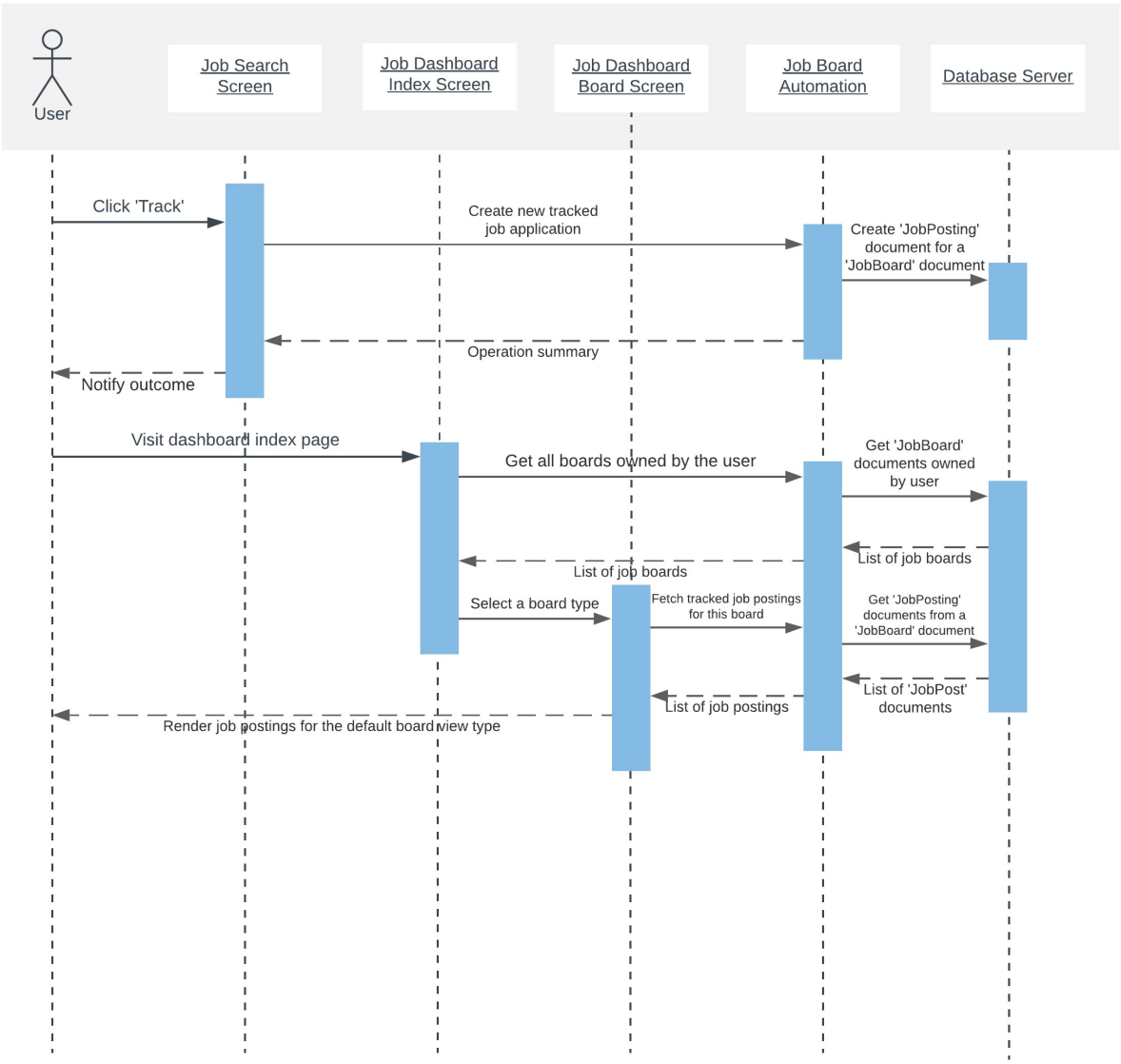
Toggleable level of detail



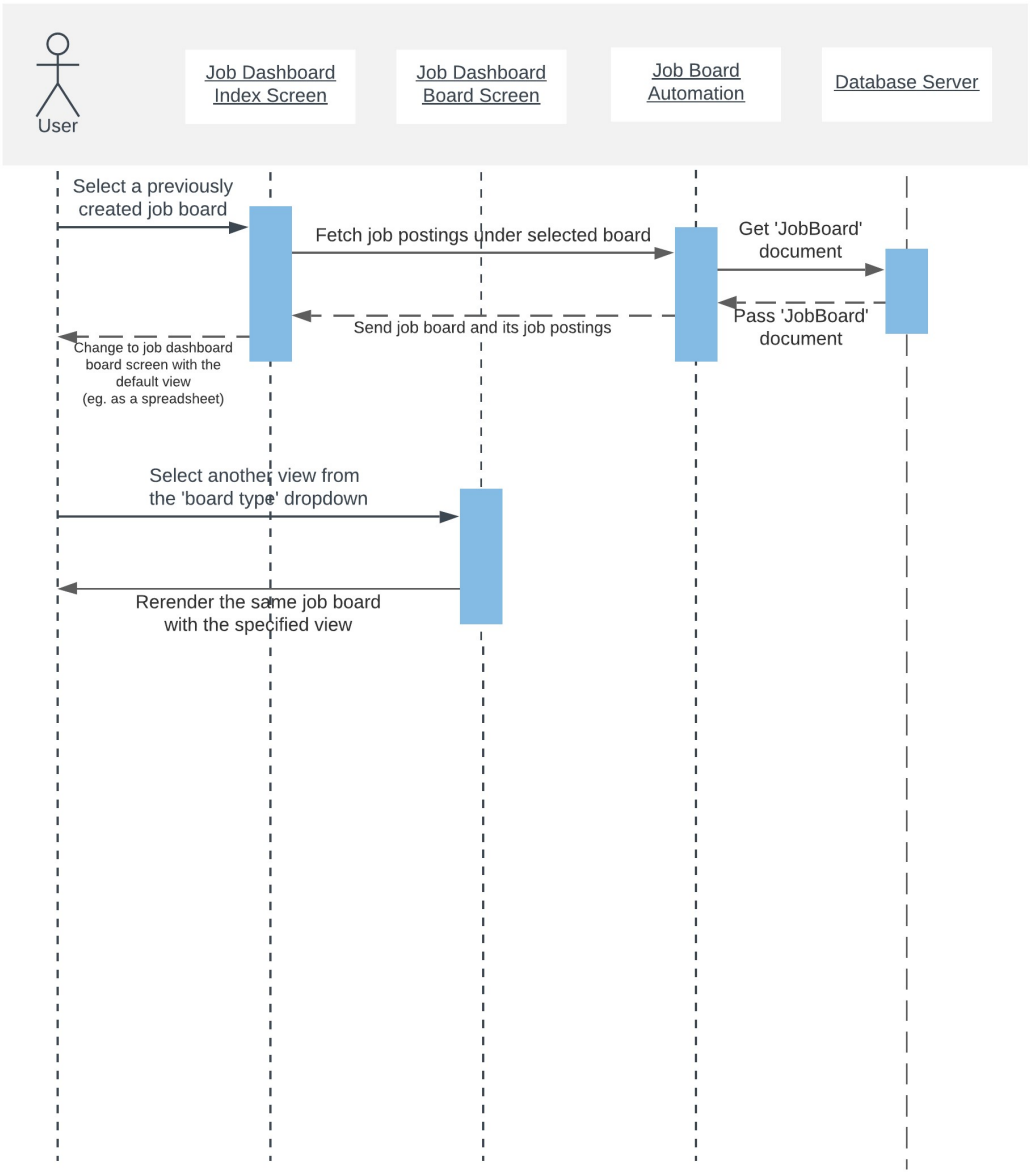
Board Management



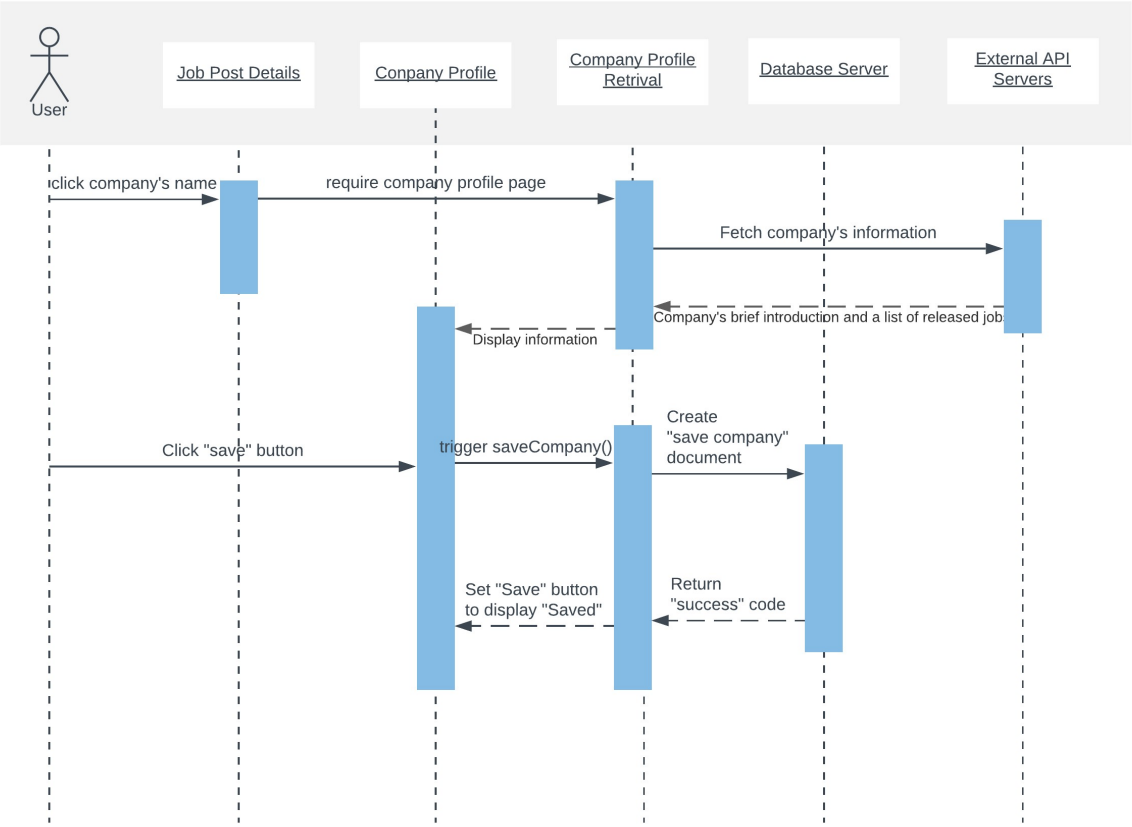
Automated Job Tracking



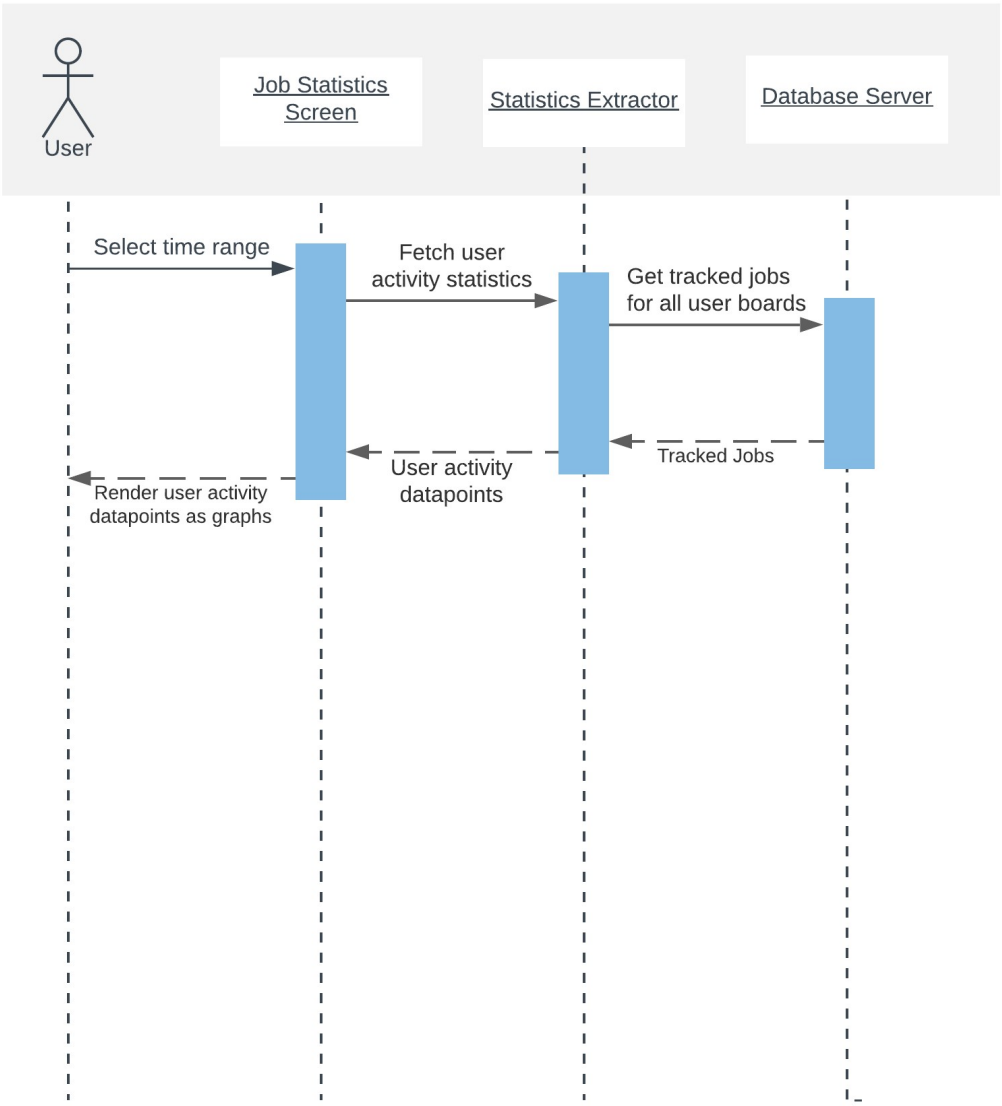
Toggleable Board View



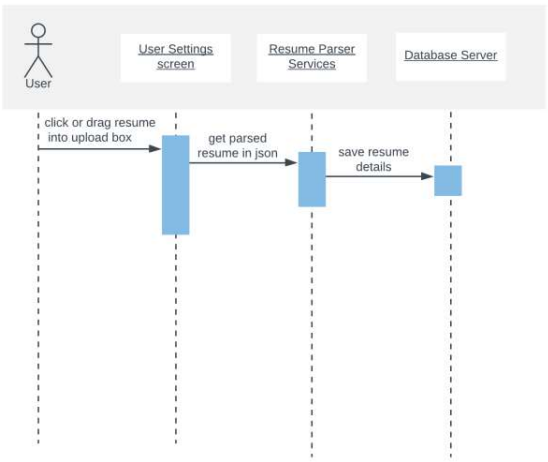
Saving favourite companies



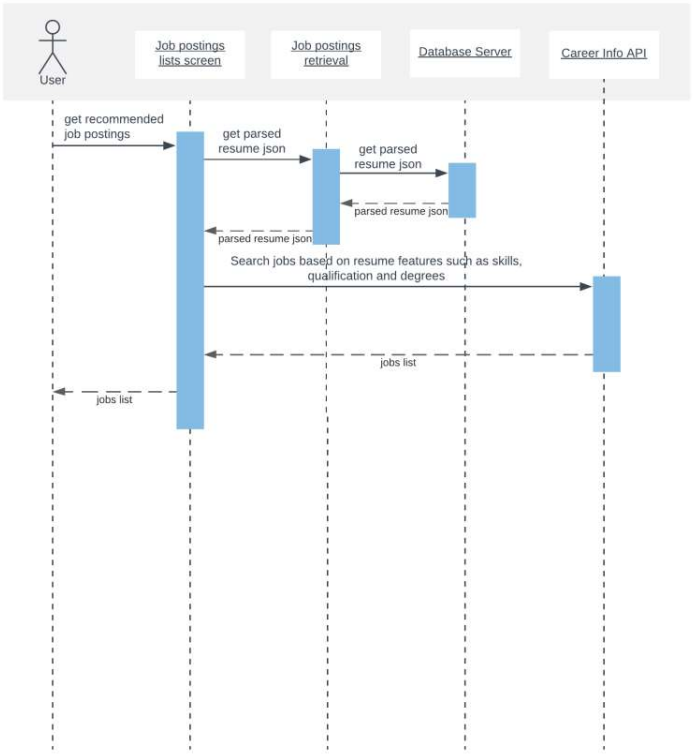
User Analytics Visualisation



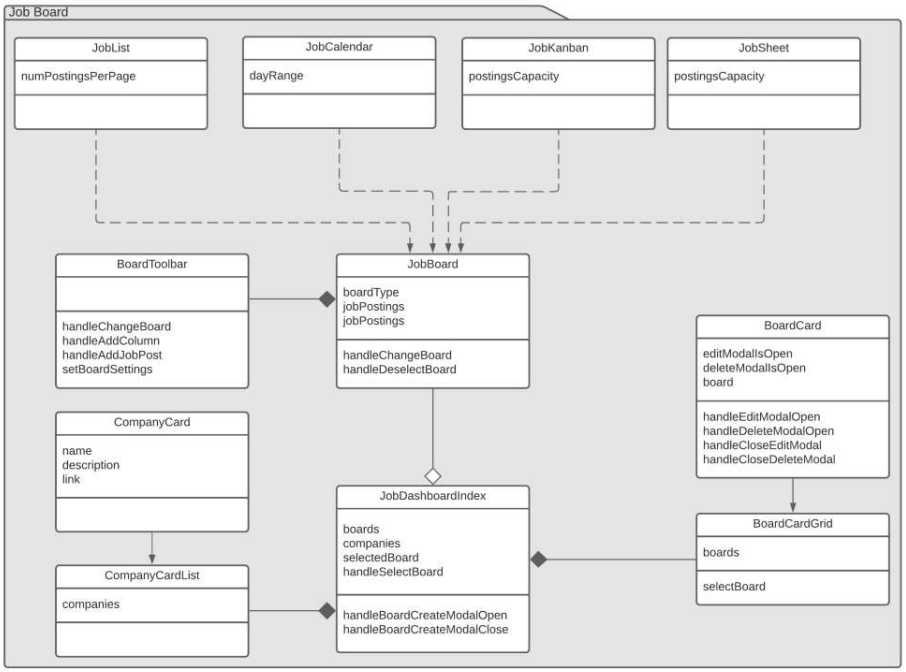
Tailored Job Recommendations (Uploading resume)

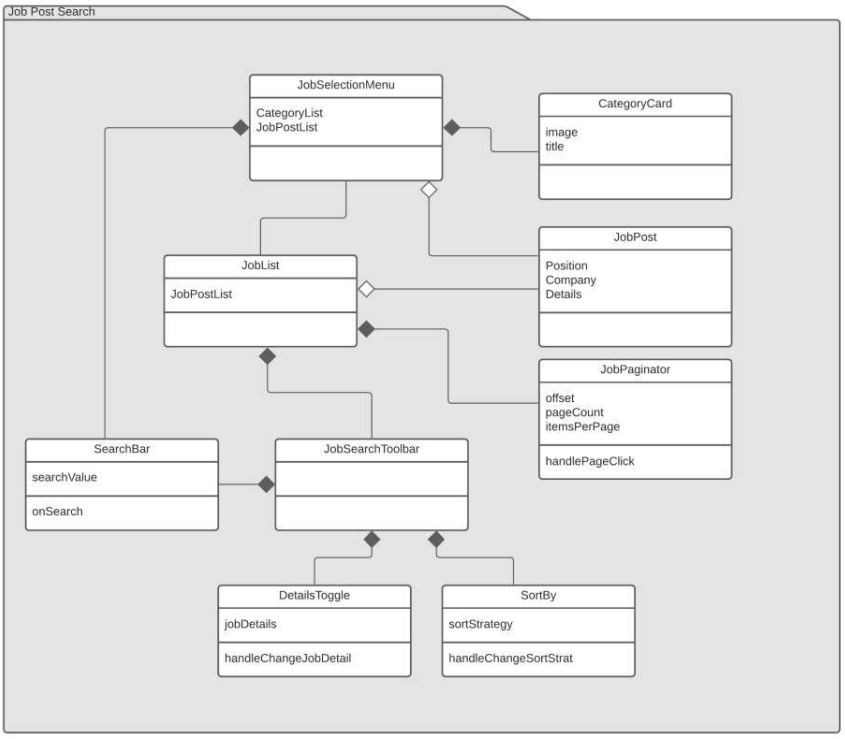


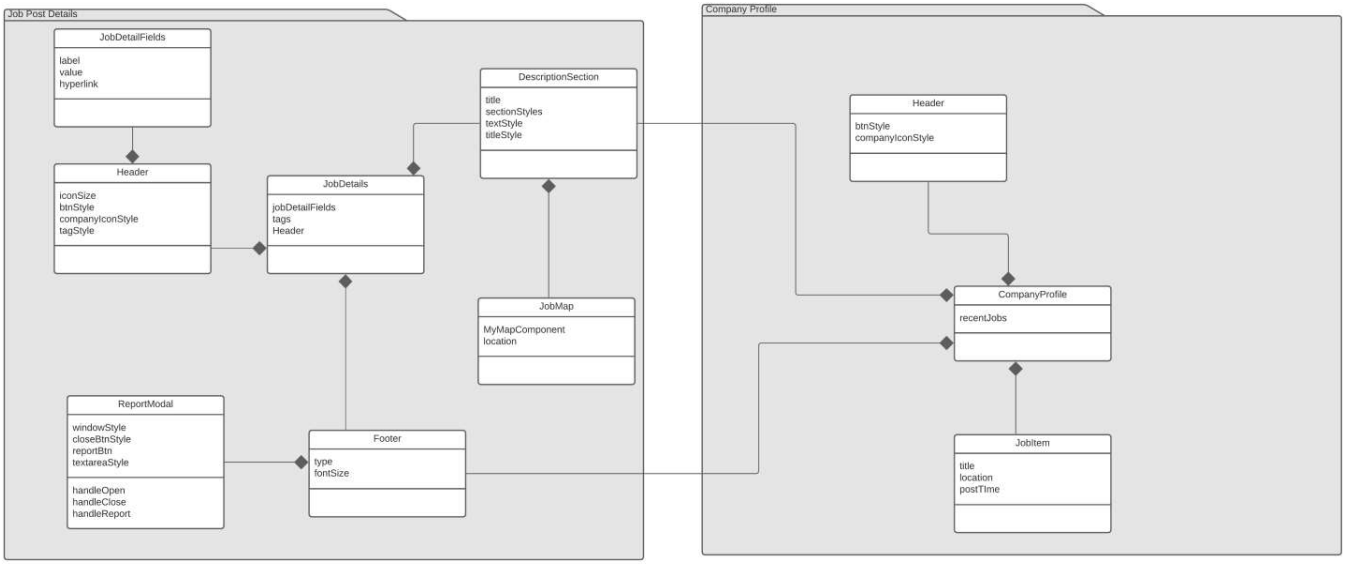
Tailored Job Recommendations (After upload)

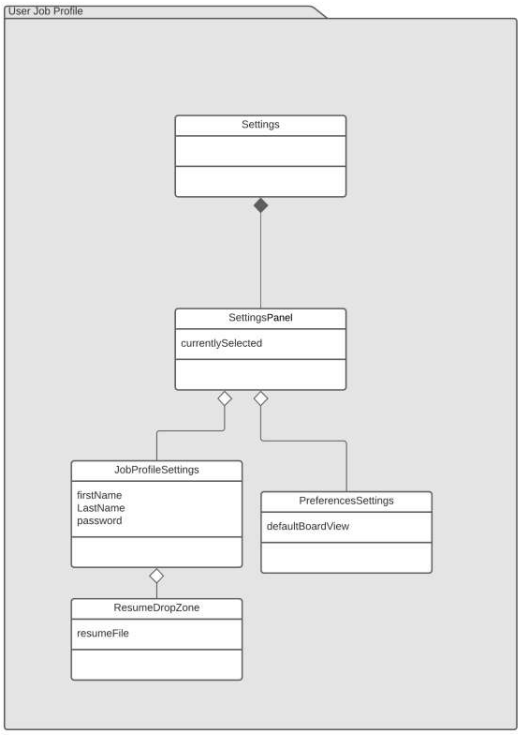


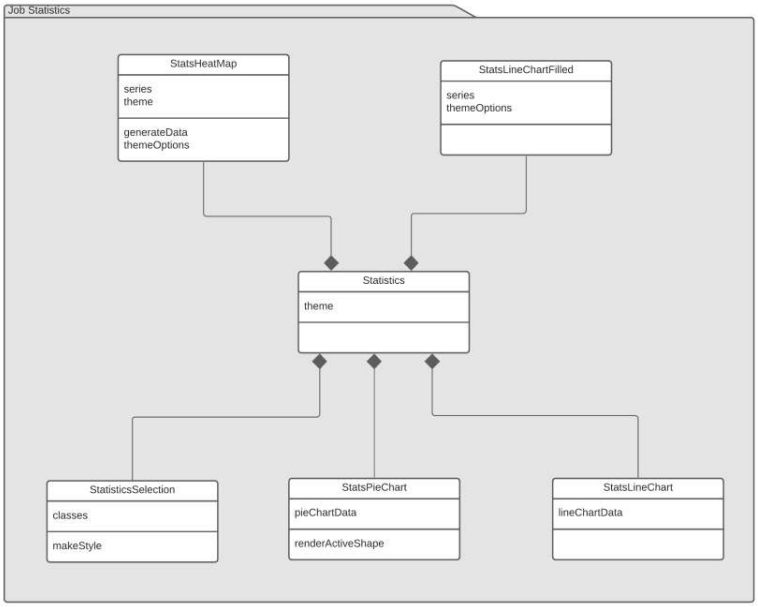
Frontend UML Class Diagram:






Job Post Details
JobDetailFields
label
value
hyperlink
Header
iconSize
btnStyle
companyIconStyle
tagStyle
JobDetails
jobDetailFields
tags
Header
DescriptionSection
title
sectionStyles
textStyle
titleStyle
JobMap
MyMapComponent
location
ReportModal
windowStyle
closeBtnStyle
reportBtn
textareaStyle
handleOpen
handleClose
handleReport
Footer
type
fontSize
Company Profile
Header
btnStyle
companyIconStyle
CompanyProfile
recentJobs
JobItem
title
location
postTime


User Job Profile
Settings
SettingsPanel
currentlySelected
JobProfileSettings
firstName
LastName
password
PreferencesSettings
defaultBoardView
ResumeDropZone
resumeFile


Job Statistics
StatsHeatMap
series
theme
generateData
themeOptions
StatsLineChartFilled
series
themeOptions
Statistics
theme
StatisticsSelection
classes
makeStyle
StatsPieChart
pieChartData
renderActiveShape
StatsLineChart
lineChartData

Database ER Diagram: