Capstone Project Final Report

GlassCV

Group	
Group Members	Phạm Anh Tuấn – SE05114 Nguyễn Duy Hải – SE04546 Nguyễn Thanh Hiếu – SE05129 Nguyễn Hải Long – SE04527
Supervisor	Mr. Nguyễn Tất Trung
Capstone Project code	GLASSCV

Hanoi, November 1st, 2018

This page is intentionally left blank

Table of contents

This page is intentionally left blank

Acknowledgements

We wish to express our deepest gratitude to our supervisor, Mr. Nguyen Tat Trung, for his continuously sharing and motivating throughout the project. Following strictly Mr. Trung instructions, the GlassCV team has always felt confident in dealing with problems along the way. Taking aside all the technologies and methodologies, Mr. Trung induced us with the idea of having the right attitude & resolution towards tackling obstacles. That was the most important factor that has led us to the completion of this project.

Finally, we truly appreciate the instructors at FPT University for all the lectures & knowledge shared to us. We hope you will find this project as a reflection of the knowledge and experiences you have given us during this period of three years.

This page is intentionally left blank

Definitions and Acronyms

Acronym	Definition	Note
API	Application Programming Interface	
JSON	JavaScript Object Notation	
CI	Continuous Integration	
GUI	Graphic User Interface	
URL	Uniform Resource Locator	
SDK	Software Development Kit	
CD	Continuous Delivery	

This page is intentionally left blank

Chapter 1 : Introduction

1.1 Purpose

This chapter provides an overview of the project include background information, a literature review of the existing system and raising a proposal for ideas of improvement.

1.2 Project Information

- Project name: **GlassCV**
- Project code: **GLASSCV**
- Project group name**: CrystalCV**
- Product type: **Web Application**
- Timeline: **From 10th Sep 2018 to 28th Dec 2018**

1.3 The people

1.3.1 Supervisors

	Full name	**Phone**	**E-Mail**	**Title**
Supervisor	Nguyễn Tất Trung	0904399139	trungnt@fpt.edu.vn	Lecturer

Table 1-1: Supervisor's information

1.3.2 Team members

	Full name	**Student code**	**Phone**	**E-mail**	**Role in Group**
1	Nguyễn Hải Long	SE04527	0903214643	longnhse04527@fpt.edu.vn	Member
2	Nguyễn Thanh Hiếu	SE05129	0392672397	hieuntse05129@fpt.edu.vn	Member
3	Phạm Anh Tuấn	SE05114	0989813652	tuanpase05114@fpt.edu.vn	Member
4	Nguyễn Duy Hải	SE04546	0347286982	haindse04546@fpt.edu.vn	Member

Table 1-2: Team member's information

1.4 Background

For many undergraduates & graduates, finding a job is difficult. They either find it hard to get a job that suits their skills, or they are not confident in their skills. Many websites for job seekers & employers have been in existence for such reasons. However, most of them do not assist those job seekers much in finding a suitable job because the seekers do not find the suggested jobs very clear in requirements, in other words, the descriptions are not detailed enough, and the information about the company is not very objective. This often leads to a high number of unemployed undergraduates in recent years. These are a few headlines alerting us about some alarming statistics about the Vietnam workforce status.

237 nghìn cử nhân thất nghiệp trong quý III năm 2017

Thứ Ba, 26/12/2017, 02:30:53 A Font Size: − | + Print

Bảng 5. Số lao động trong độ tuổi thất nghiệp theo giới tính, thành thị/nông thôn và nhóm tuổi

Đơn vị: nghìn người

	2016		2017		
	Q3	Q4	Q1	Q2	Q3
Chung	**1.117,7**	**1.110,0**	**1.101,7**	**1.081,6**	**1.074,8**
Nam	619,4	598,7	654,8	641,7	579,3
Nữ	498,4	511,3	446,9	439,9	495,5
Thành thị	515,7	520,3	518,3	510,5	505,0
Nông thôn	602,0	589,7	583,4	571,1	569,8
Thanh niên (15-24)	642,6	586,7	548,5	575,1	610,9
Người lớn (≥25)	475,1	523,3	553,3	506,6	463,9

Nguồn: TCTK (2016, 2017), Điều tra LĐ-VL hằng quý.

Figure 1-1: Numbers of unemployed bachelors in Q1 2017[1]

Giật mình 60 % cử nhân làm trái nghề, quá nhiều người chọn việc chạy Grab, Uber để kiếm sống

SaoStar 14/12/17 07:00 GMT+7 Gốc

Việt Nam có 24 triệu thanh niên, chiếm khoảng 44% lực lượng lao động tuy nhiên, tỷ lệ thất nghiệp ở nhóm này lại cao gấp 3 lần mức chung cả nước. Thống kê mới nhất cũng cho thấy, 60% sinh viên đang làm trái nghề và tập trung nhiều ở lĩnh vực chạy 'xe ôm công nghệ' cho các hãng Grab, Uber.

Hàng triệu thanh niên thất nghiệp

Trong cuộc đối thoại với các đại biểu tham dự Đại hội Đoàn toàn quốc lần thứ 11 với chủ đề: "Tuổi trẻ với khởi nghiệp, việc làm" mới đây, Bộ lao động Thương binh Xã hội đã đưa ra con số khiến nhiều người giật mình: cả nước hiện có khoảng 24 triệu thanh niên (chiếm khoảng 44%) lực lượng lao động), tuy nhiên, tỷ lệ thất nghiệp ở nhóm này lại cao gấp 3 lần mức chung của cả nước.

Thất nghiệp của thanh niên trong độ tuổi 15 đến 24 tuổi chiếm 51,3% tổng số thất nghiệp. Tỷ lệ thất nghiệp thanh niên cả nước là 7,67% và đặc biệt cao ở khu vực thành thị với mức 11,95%. Nghĩa là cứ 100 thanh niên trong lực lượng lao động thì có 12 người thất nghiệp.

Figure 1-2: Bachelors working in jobs not matching their major[2]

[1] http://www.nhandan.com.vn/xahoi/tin-tuc/item/35112802-237-nghin-cu-nhan-that-nghiep-trong-quy-iii-nam-2017.html

[2] https://baomoi.com/giat-minh-60-cu-nhan-lam-trai-nghe-qua-nhieu-nguoi-chon-viec-chay-grab-uber-de-kiem-song/c/24286282.epi

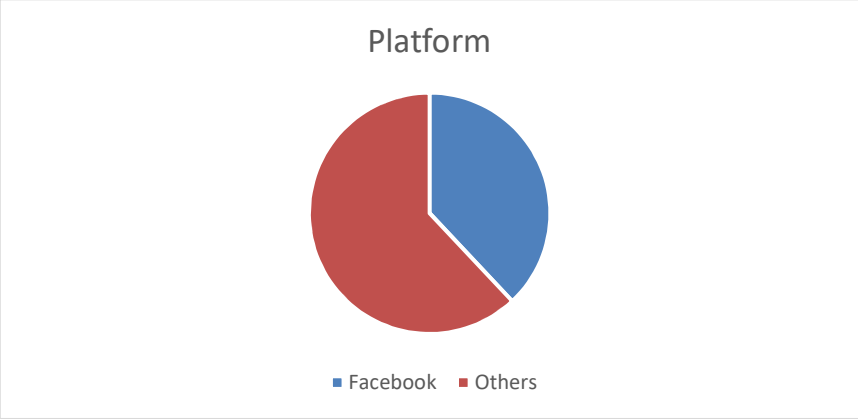


Figure 1-3: Students finding jobs via Facebook[3]

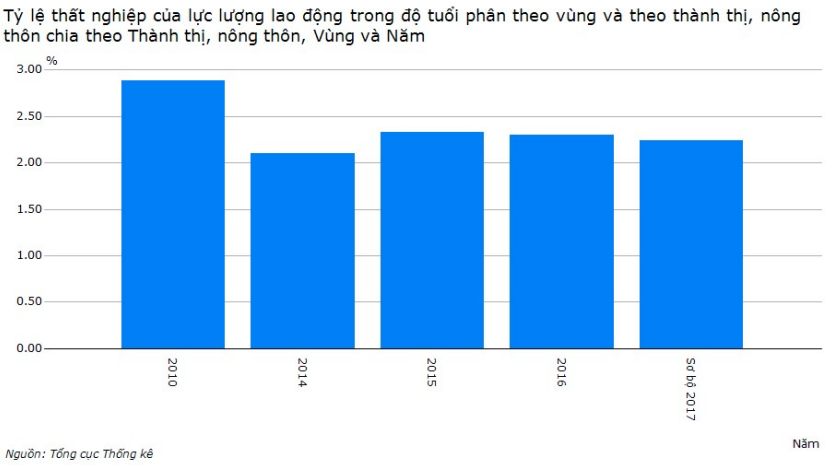


Figure 1-4: Unemployment rate by age group[4]

[3] https://qandme.net/vi/baibaocao/38-phan-tram-sinh-vien-Viet-Nam-tim-cong-viec-qua-Facebook.html
[4] Tổng cục thống kê, https://www.gso.gov.vn/

Cử nhân thất nghiệp lại tăng

16:18 31/12/2017

Số người thất nghiệp có trình độ đại học trở lên là 237 nghìn người, tăng 53,9 nghìn người so với quý II/2017. Tỷ lệ thất nghiệp của nhóm này là 4,51%.

Bản tin thị trường lao động Việt Nam số 15, quý III năm 2017 mà Bộ Lao động Thương binh Xã hội vừa công bố cho thấy tỷ lệ thanh niên thất nghiệp giảm nhẹ, tuy nhiên thất nghiệp của nhóm có trình độ đại học và trên đại học lại tăng so với quý II.

Cụ thể, số người thất nghiệp có trình độ đại học trở lên là 237 nghìn người, tăng 53,9 nghìn người so với quý II/2017. Tỷ lệ thất nghiệp của nhóm này là 4,51% (quý trước là 3,63%).

Figure 1-5: Unemployment rate increases[5]

The unemployment rate is consistently of 2-3% is normal, but the number of 237.000 is quite a large number, burdening the economy since it might slow down the increasing trend of the Vietnam economy.

We have conducted a small survey in FPT university and our study finds out that these are the most common obstacles explaining why undergraduates find it quite hard to get themselves a suitable job:

- They are not very confident and quite inactive when finding a job.
- They do not have good networking skills.
- They are not very professional writing about themselves (strength, weaknesses).
- They dream of having an easy job with a high salary or joining a big company with extensively professional cultures.
- They do not consider the job interview a serious chance for themselves.
- They do not have experiences in doing real work.

1.4.1 The current job finding & review systems are not comprehensive

Our study also finds out that currently, the job finding systems are heavily based on the job seekers alone. For example, Vietnamworks.com and TopCV.com are the two big websites in Vietnam, however, they do not provide comprehensive public feedbacks & reviews systems for normal guest users to weigh the companies. The need of public feedbacks & reviews for the employers & jobseekers is visible, since providing the information without someone verifying it shows a sign of biased. The weaknesses of those websites will be eliminated after the creation of this capstone project.

1.4.2 TopCV advantages & disadvantages

1.4.2.1 Advantages

[5] https://news.zing.vn/cu-nhan-that-nghiep-lai-tang-post808169.html

TopCV main features:

For job seekers

- Create CV based on hundreds of ready-designed templates.
- View/Share CV by link.
- Professional assistance for serious job seeker by experts in the field of writing great CVs.
- CV management page.
- User can see basic information/description of a job.
- User can see basic information about the company in a job post.
- Jobs posted by a company.
- MBTI, MCQ test: help users to understand more about their characters better, therefore find a suitable job.

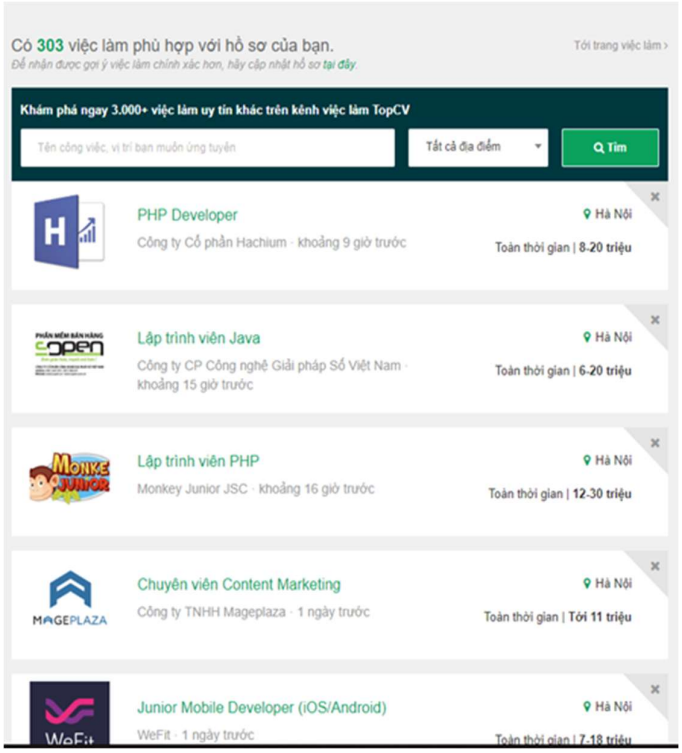


Figure 1-6: Suitable jobs

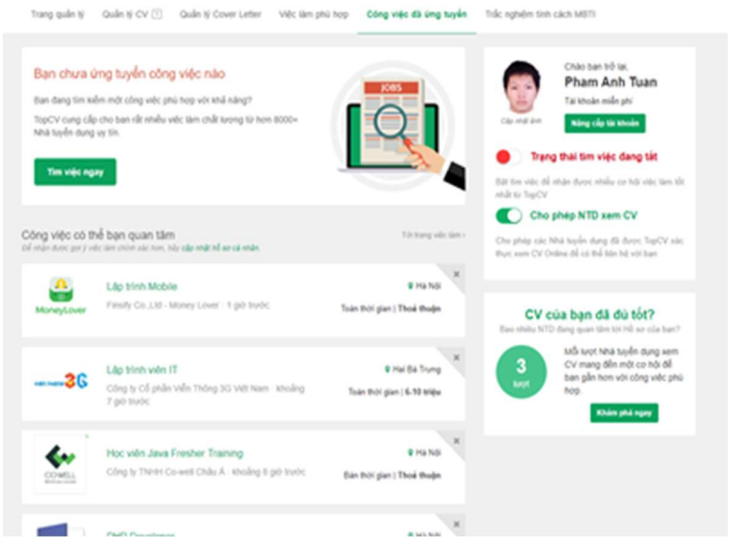


Figure 1-7: Applied jobs

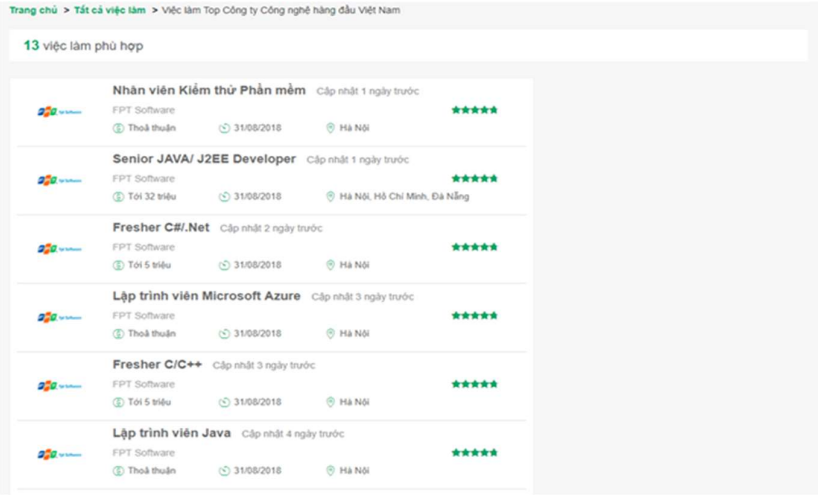


Figure 1-8: List jobs by 4 main categories: Startups, English Centers, Banks, IT Companies

For employers:

- ❖ Look for employees who match the required qualities.
- ❖ Advanced search: Allow user to search based on specific requirements (age, gender, state of job seekers, educational background, languages used, company, location, job position, optional keywords, required keywords, must be omitted keywords).

1.4.2.2 Disadvantages

- ❖ No job recommender system for job seekers.
- ❖ No detailed public review/cross evaluation for job seekers.

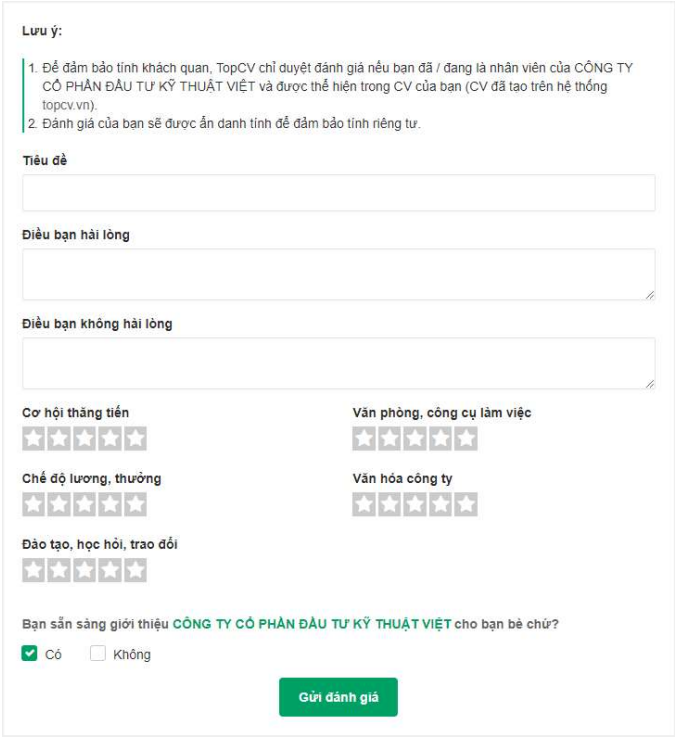


Figure 1-9: The review section provides very little information

- Details about any company is very little.

1.5 Proposal of system

1.5.1 Our proposal system

After reviewing all properties of the current system as well as our target customers being undergraduates in Vietnam, we have come to a decision to choose which features and functions we will provide in our system.

1.5.1.1 System functions

- Allow users to register/sign in with email addresses.
- Allow users to create/edit their own user profiles, which contain details about their education, experiences & skills.
- Allow users to search for jobs with keywords. They can search for jobs, company.
- Allow users to quick apply for a job.
- Allow users to receive notification emails.
- Allow users to export their profiles to CV.
- Allow users to upload their pre-created CV.
- Allow users to create reviews (rate and comment) on any company's jobs and benefits.
- Allow job seekers to view all any company's reviews on jobs, benefits.
- Allow job seekers to turn on their up-for-job status.
- Allow users to request to become an employer.

- ❖ Allow administrators to have their own dashboards to manage users, companies, reviews on companies and employer requests.
- ❖ Allow employers to create job posts.

1.5.1.2 The GlassCV user process

Figure 1-10: The job seeker process on GlassCV

Figure 1-11: The employer process on GlassCV

1.5.1.3 Out of scope functions

- × Managing CVs.
- × Viewing CV templates.
- × Viewing suitable candidates for the job posted.

1.5.1.4 Special approaches

- ➢ Using Microservices architecture for application's deployment.
- ➢ Using Docker to compose the project into containers to deploy on Google Cloud.
- ➢ Using Google Cloud Kubernetes Engine as deployment host for scaling and load balancing.
- ➢ Use Git as source code version control, hosted on Github.
- ➢ Having a separate background job service to handle heavy tasks.
- ➢ Using Google Pub Sub for communicating between microservices.
- ➢ Using Google Cloud Storage for image and CV storage.
- ➢ For web frontend system:
  - o Using Angular 7 for web component rendering.
  - o Using Angular Material to create all web form components.

Chapter 2 : Project Plan

2.1 Purpose

This chapter provides an overview of the project plan includes project organization and project management plan.

2.2 Project Organization

2.2.1 Software Process Model

Figure 2-1: Iterative and Incremental Software Process Model[6]

GlassCV project uses the Iterative and Incremental Software Process Model as shown in the above, which describes the overall lifecycle process.

The Iterative and Incremental Software Process Model is mostly used when the scope of the project is big, the major requirements are defined clearly, some more details will be added later, and for the "newbie" group in software development. By using this software process model, we break down the developing system task into series of smaller tasks which will be completed separately, evaluated, and subsequently re-worked until the system performs adequately. In addition, the iterative model is easier than other models when the issues are discovered. The feedbacks are immediately given, and solutions are proposed right on the spot.

[6] https://www.topsinfosolutions.com/methodology/

2.2.2 Roles and Responsibilities

2.2.3 Organization Structure

Role	Responsibility
Project Manager	Planning, developing schedules, coordinating communication, generally responsible for keeping the team's focus on the main goal.
Developer	Involve to code the product and review code of other developers.
Designer	Involve to design product's user interface.
Tester	Involve testing the product.
Business Analyst	Analyzes an organization or business domain and documents its business or processes or systems.
Technical Leader	Responsible for choosing and deciding what technologies should be used, as well as for overseeing the work being done by other developers.

Table 2-1: Project Structure

2.2.4 Project Team Member

Team Member	Role
LongNH	Developer, Tester, Designer
HieuNT	Developer, Tester, Designer
TuanPA	Project Manager, Developer, Business Analyst
HaiND	Developer, Tester, Business Analyst, Technical Leader, Test Leader

Table 2-2: Project Team Member

2.2.5 Tools and Techniques

Programming languages	Python 3, JavaScript
Framework	Flask, Angular 7, Angular Material
Software architecture	MVC, Microservices
Version control	Git
IDEs/Editors	Visual Studio Code
UML tools	Astah Professional 7.0
Web server	Nginx
DBMS	MySQL 8
Deployment server	Google Cloud
Project management tool	Microsoft Project 2017

Process model	Iterative and Incremental Software Process Model
Development process	Behavior-driven development, Test-driven development, Continuous integration, Continuous delivery
CI, CD Tools	TravisCI, Docker 2.0

Table 2-3: Project Team Member

2.3 Project Management Plan

2.3.1 Tasks

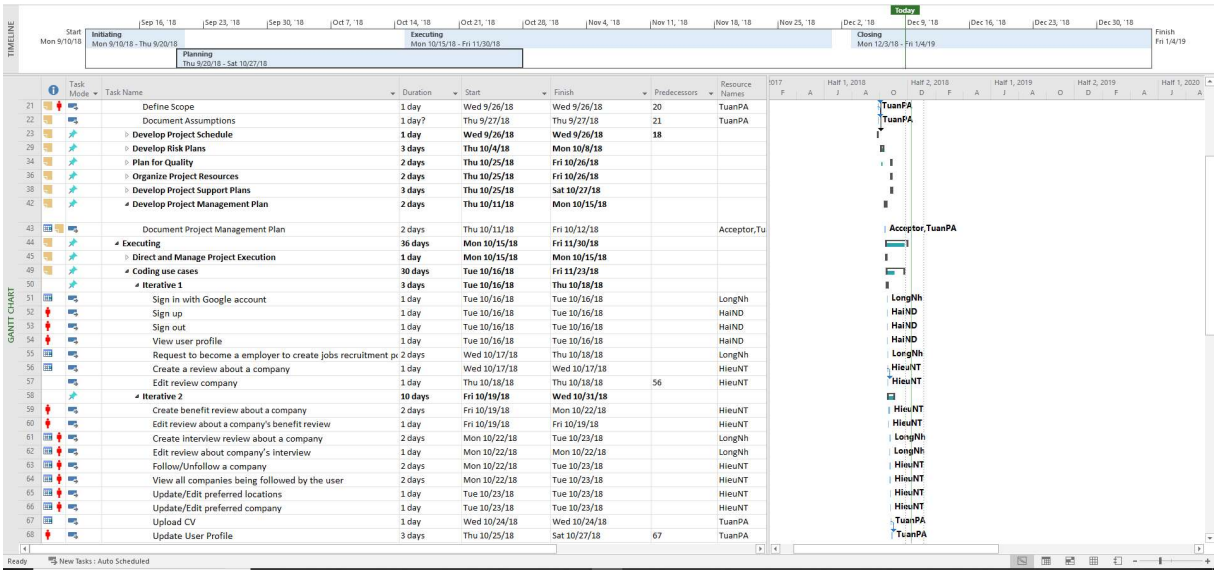


Figure 2-2: GlassCV Project Management file

2.3.2 Meeting Minutes

All meeting minutes will be written following this template:

Meeting/Project Name:	*GlassCV*		
Date of Meeting:	*15/09/2018*	***Time: (Type)***	*4 hours (Face-to-face)*
Meeting Called by:	*LongNH*	***Location:***	*FPT University, Room 302L*
Note Taker:	*TuanPA*	***Time Keeper:***	*HieuNT*
1. Meeting Objective			
- Choose names, ideas for project			
2. Attendance			
Name	***Roles***	***E-mail***	***Phone***

Meeting/Project Name:	*GlassCV*		
Date of Meeting:	*15/09/2018*	***Time: (Type)***	*4 hours (Face-to-face)*
Meeting Called by:	*LongNH*	***Location:***	*FPT University, Room 302L*
Note Taker:	*TuanPA*	***Time Keeper:***	*HieuNT*
Pham Anh Tuan	Project Manager Developer Business Analyst	tuanpase05114@fpt.edu.vn	0989813652
Nguyen Thanh Hieu	Developer Designer	hieuntse05129@fpt.edu.vn	0392672397
Nguyen Hai Long	Developer Tester	longnhse04527@fpt.edu.vn	0903214643
Nguyen Duy Hai	Developer Technical Leader	haindse04546@fpt.edu.vn	0347286982
3. Content			
-			
4. Note			
-			

Table 2-4: Meeting Minutes Template

2.3.3 Coding Conventions

We strictly follow Google JavaScript Style Guide as the coding convention for the project. Specific convention rules can be found on the page: https://google.github.io/styleguide/jsguide.html

Google JavaScript Style Guide

Table of Contents

Figure 2-3: Google JavaScript Style Guide

2.3.4 Risk Management Plan

No	Name	Prevention	Correction	Status
R1	**Internet interruptions**	All developers must set up the isolated development environment and have an offline copy of the documentation. Using various online communication tools and creating face-to-face meeting at any possible time.	Always make sure that there are alternative ways to connect to the internet, such as: using VPN or 3G.	Closed
R2	**Illness or absence of team members**	Member must notice to the team about absence period and the plan of how to keep up with the work process.	Ensure that the absence of a member won't affect others and always have plans to deal with this problem.	Closed
R3	**Business problem**	Any ideas are welcome, but members must discuss with others and always focus on the reality and possibility.	Make sure the business logic of any ideas is carefully analyzed.	Closed
R4	**Requirement changed**	Any requirements listed must be analyzed and understood by team members.	If there is a "must be changed" requirement, all team members must join the meeting to decide whether it should be implemented or not.	Closed

R5	**Miscommunication**	Anything raised has to be documented or filed. Any team member should attempt to express his/her opinion clearly.	Miscommunication must be resolved early.	Closed
R6	**New technology**	Choosing technology based on member's qualification. Any member must know how to do self-instructing.	The technology choice should be well explained.	Closed

Table 2-5: Risk Management

2.3.5 Communication Plan

***Weekly meeting schedule**:* We follow the Iterative and Incremental Process Model, and we separate the project into two main team, back-end API team and front-end web app team. Each team will do the tasks the task assigned to team members by the Team Leader and depending on difficulty the Technical Leader will assign deadlines for each task. We have a meeting every Thursday to update all team members about what has been done during last week.

***Daily meeting schedule**:* Each sub team has one development team with different schedule and deadlines. Before the daily meetings, each member will be telling:

- What he has completed.
- What he is working on.
- When he will finish.
- What issues he is encountering, or if he needs assistance.
- What he will be doing after finishing the task.

***Unscheduled meeting**:* Assuming someone has encountered a serious problem that he wants to solve immediately, we will have a meeting via some online channel: Skype, or Phone. Face to face meeting in any emergency cases.

***Communication channel**:* Our main communication channel is Skype and Facebook group. We use TeamViewer for assisting team member when he meets technical issues.

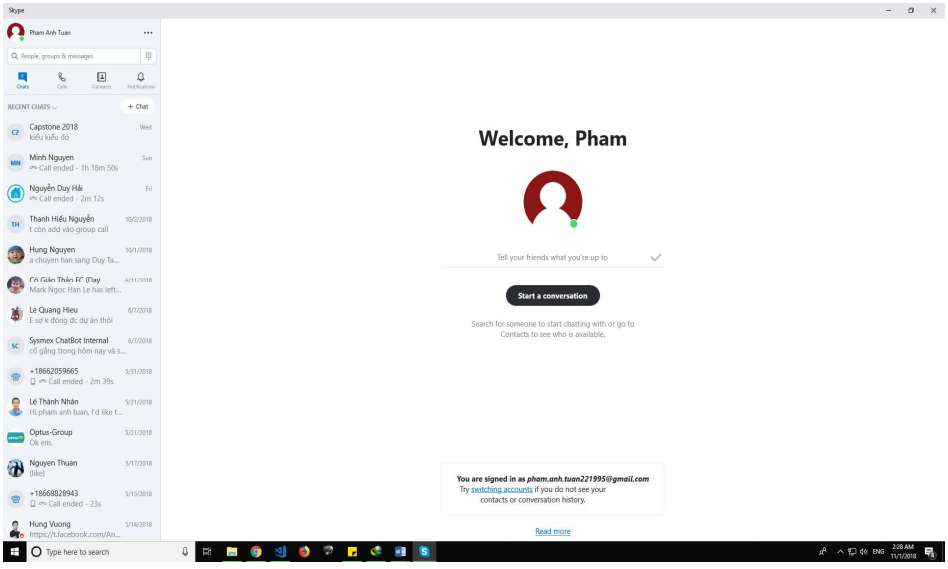


Figure 2-4: Communication via Skype

Chapter 3 : Software Requirement Specification

3.1 Purpose

This chapter contains details about functional and non-functional requirements the website. Constraints and detailed requirements are specifically described for developers to follow when completing the tasks.

3.2 Functional Requirements

3.2.1 Use Case Diagram

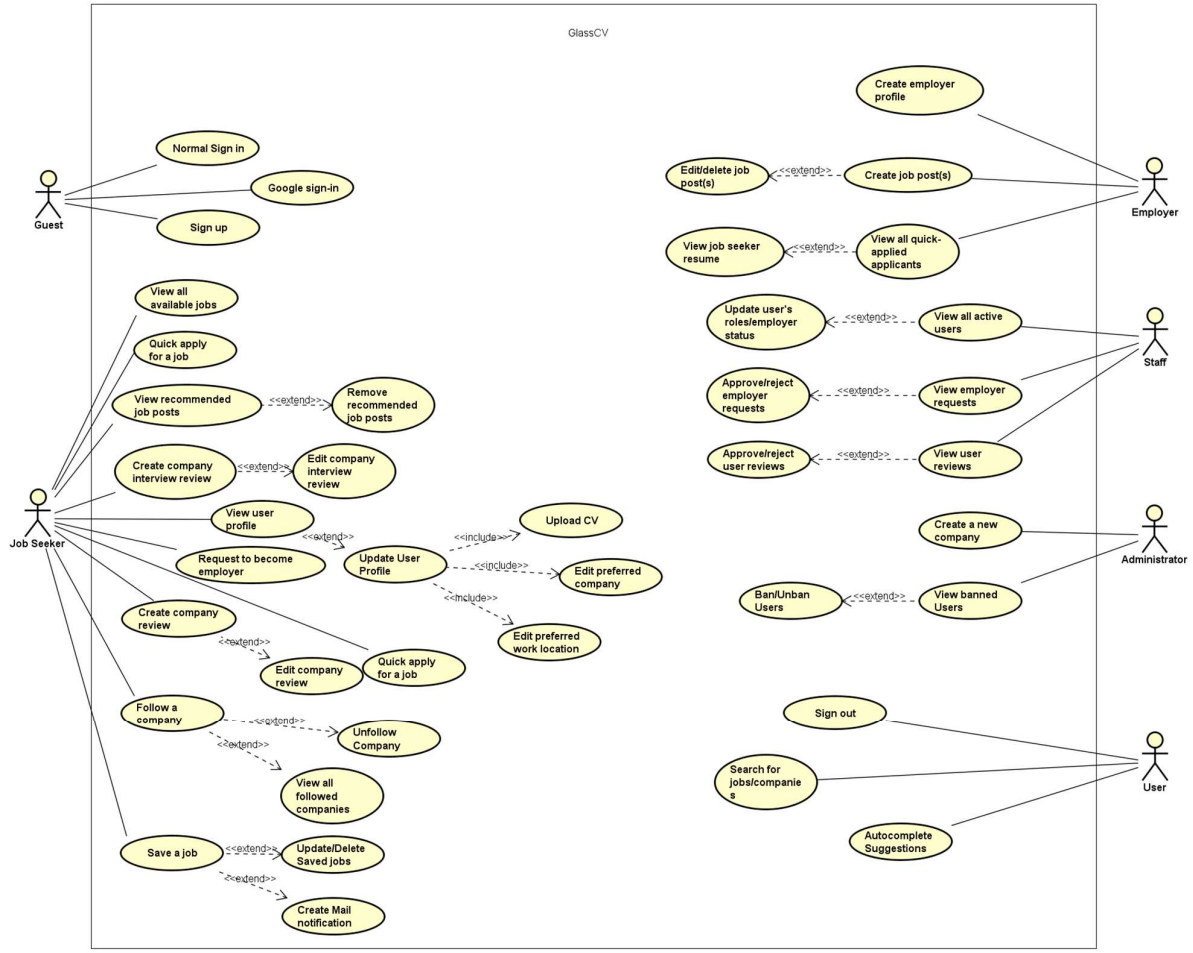


Figure 3-1: Use case diagram

3.2.2 Business Rules

No	Description
B1	The email must not be empty and valid.
B2	The confirm password must be matched with the new password.
B3	User's first name and last name must not be empty.
B4	User's first name and last name must not be longer than 255 characters.
B5	Phone number must be a numeric string only.

B6	Image file's mime type must be: "image/png" or "image/jpeg".
B7	Every user has one of the following roles: Normal User, Moderator, or Administrator.
B8	The root Administrator has the highest privilege and cannot be changed by other Administrators.
B9	An Administrator can: ban/unban user, update user's information, accept/reject requests.
B10	User can request to become employer.
B11	User must be an employer to post a job.
B12	Any review post has 3 states: "PENDING", "REJECTED", "ACCEPTED".
B13	A banned user cannot create/edit reviews, follow companies, save jobs, apply for jobs or view recommendations.
B14	New reviews are assigned the "PENDING" state and must be accepted by staff before they are visible to other users.
B15	Maximum file size allowed is 10 MB.
B16	A Moderator can approve user reviews and employer requests.
B17	Each user can only request to be employer of one company
B18	Uploaded CV file must be of the type ".pdf"
B19	A Job seeker can follow many companies.
B20	When job seeker follows a company, he/she is automatically registered to receive email about the company's new job post.
B21	When user signs out, his/her current access token is blacklisted.
	User can tick choices of preferred locations
B23	User can tick as many choices as he/she prefers
B24	Each job seeker can upload one CV, uploading another will overwrite the previous one.
B25	User profile must be in view mode so employer can only view an user's information
B26	

Table 3-1: Business rules

3.2.3 Use Cases

Actor	Description
Guest	Anyone who visits the website, including those who are seeking for jobs, or someone who is looking for a suitable person to do their jobs.
Job Seeker	Those who are trying to get a job matching their skills, or one who is finding a company that has a culture that has values that the job seeker is looking for.
Employer	Those who are seeking for skillful people with relevant knowledge and available for being hired.
Administrator	A person who has the highest permission level and is responsible for managing the entire website.
Moderator	Staff member who can approve/reject user reviews and employer requests.

User	Any normal user of these following types: guest, job seeker, employer.
Staff	Users that are either Administrator or Moderator

Table 3-2: Actor description

ID	Actor	Name
UC-1.0	Guest	Normal sign in
UC-2.0	Guest	Sign in with Google account
UC-3.0	Guest	Sign up
UC-4.0	User	Sign out
UC-5.0	User	Autocomplete suggestions
UC-6.0	User	Search for jobs/companies
UC-7.0	Job Seeker	View user profile
UC-8.0	Job Seeker	Request to become an employer to create jobs recruitment posts
UC-9.0	Job Seeker	Create a review about a company
UC-10.0	Job Seeker	Edit a company review
UC-11.0	Job Seeker	Create interview review about a company
UC-12.0	Job Seeker	Edit review about the company's interview
UC-13.0	Job Seeker	Follow/Unfollow a company
UC-14.0	Job Seeker	View all companies being followed by the user
UC-15.0	Job Seeker	Edit preferred locations
UC-16.0	Job Seeker	Edit preferred company
UC-17.0	Job Seeker	Update User Profile
UC-18.0	Job Seeker	Upload CV
UC-19.0	Job Seeker	Save a job
UC-20.0	Job Seeker	Create mail notification
UC-21.0	Job Seeker	Update/Delete saved jobs
UC-22.0	Job Seeker	Quick apply for a job
UC-23.0	Job Seeker	View all available jobs
UC-24.0	Job Seeker	View recommended job posts
UC-25.0	Job Seeker	Remove a recommended job post
UC-26.0	Employer	Create a job post
UC-27.0	Employer	Edit/Delete created job posts
UC-28.0	Employer	Create employer profile
UC-29.0	Employer	View a job seeker resume
UC-30.0	Employer	View all applicants in quick-apply

UC-31.0	Staff	View all active users
UC-32.0	Administrator	View banned users
UC-33.0	Staff	Update a user's role or employer status
UC-34.0	Administrator	Ban/Unban a user
UC-35.0	Administrator	View all companies
UC-36.0	Administrator	Create a new company
UC-37.0	Administrator	Update/delete company
UC-38.0	Staff	View employer requests
UC-49.0	Staff	Approve/Reject employer requests
UC-40.0	Staff	View user reviews
UC-41.0	Staff	Approve/Reject user reviews

Table 3-3: Use Case list

3.2.3.1 Guest

3.2.3.1.1 Normal sign in

Use Case Specification

UC ID and Name:	**UC-1.0 Normal sign in**		
Created by:	**HaiND**	Date Created:	**16/09/2018**
Primary Actor:	**Guest**	Secondary Actors:	
Trigger:	N/A		
Description:	Use a Google account to sign in to the website.		
Preconditions:	**PRE-1.1.** User has registered an account on the website. **PRE-1.2.** User has not logged into the website.		
Post conditions:	**POST-1.1.** User is logged in successfully. **POST-1.2.** User icon is displayed in the top bar.		
Normal Flow:	**1.0 Normal sign in** 1. User visits website. 2. System displays Welcome page. 3. User clicks "Login" button at the top of the web page. 4. User fills in credentials. 5. User clicks "Login" button. 6. System directs to home page.		
Alternative Flows:	N/A		
Exceptions:	**1.0-E1 – Cannot communicate with API server**		

	1. Web page displays error message. **1.0-E2 – Invalid credentials** 1. Web page displays error message.
Priority:	High
Frequency of Use:	High
Business Rules:	N/A
Other Information:	
Assumptions:	N/A

3.2.3.1.2 Sign in with Google account

Use Case Specification

UC ID and Name:	**UC-2.0 Sign in with Google account**		
Created by:	**HaiND**	Date Created:	**16/09/2018**
Primary Actor:	**Guest**	Secondary Actors:	
Trigger:	N/A		
Description:	Use a Google account to sign in to the website.		
Preconditions:	**PRE-2.1.** User has a Google account. **PRE-2.2.** User has not logged into the website.		
Post conditions:	**POST-2.1.** User is logged in successfully. **POST-2.2.** User icon is displayed in the top bar.		
Normal Flow:	**2.0 Sign in with FPT University email account** 1. User visits website. 2. System displays Welcome page. 3. User clicks "Login" button at the top of the web page. 4. User clicks "Sign in with Google" button on the login form. 5. Browser displays another window that contains Google sign in form. 6. User enters his/her Google account credentials. 7. Google verifies login and closes the popup. 8. System directs to home page.		
Alternative Flows:	**2.1 User has already signed into Google with one or more accounts. (Step 6)** 1. User chooses desired Google account. **2.2 User has already signed into Google with one or more accounts. (Step 3)** 1. Web page displays small popup at the top right corner, asking if use want to login with a Google account. 2. User chooses desired Google account from the list.		

	3. Google verifies login and closes the popup. 4. User is logged into the website in-place, no redirects happen.
Exceptions:	**2.0-E1 – Cannot communicate with API server** 1. Web page displays error message.
Priority:	High
Frequency of Use:	Medium
Business Rules:	N/A
Other Information:	
Assumptions:	N/A

3.2.3.1.3 Sign up

Use Case Specification

UC ID and Name:	**UC-3.0 Sign up**		
Created by:	**HieuNT**	Date Created:	**16/09/2018**
Primary Actor:	**Guest**	Secondary Actors:	
Trigger:	N/A		
Description:	Register as a user of the website.		
Preconditions:	**PRE-3.1.** There is no user access token in browser's local storage.		
Post conditions:	**POST-3.1.** User is registered in the system as an unverified account. **POST-3.2.** An email with an activation link is sent to verify user's email address.		
Normal Flow:	**UC-3.0 Sign up** 1. User visits website. 2. User clicks the "Login" button on the navigation bar. 3. Browser displays login page. 4. User clicks "Register" button. 5. Browser directs to registration page. 6. User fills in required details. 7. User clicks "Register" button. 8. User receives a verification email. 9. User clicks "Confirm email" in the verification email. 10. System verifies the email address and redirect to home page.		
Alternative Flows:	N/A		
Exceptions:	**3.0-E1 – Cannot communicate with API server** 1. Web page displays error message. **3.0-E2 – Missing or invalid data**		

	1. Web page displays error message.
Priority:	High
Frequency of Use:	Medium
Business Rules:	**B1**, **B2**, **B3**, **B4**
Other Information:	
Assumptions:	N/A

3.2.3.2 User

3.2.3.2.1 Sign out

Use Case Specification

UC ID and Name:	**UC-4.0 Sign out**		
Created by:	**HieuNT**	Date Created:	**16/09/2018**
Primary Actor:	**User**	Secondary Actors:	
Trigger:	N/A		
Description:	Log a user out.		
Preconditions:	**PRE-4.1.** User is registered. **PRE-4.2.** User is logged in		
Post conditions:	**POST-4.1.** User is logged out **POST-4.2.** Welcome page is displayed **POST-4.3.** User access token is deleted		
Normal Flow:	**4.0 Sign out** 1. Job seeker visits website. 2. Job seeker clicks the "User" icon on the navigation bar. 3. Job seeker clicks "Sign out". 4. Welcome page is displayed.		
Alternative Flows:	N/A		
Exceptions:	**4.0-E1 – Cannot communicate with API server** 1. Web page displays error message.		
Priority:	High		
Frequency of Use:	Medium		
Business Rules:	**B21**		
Other Information:			
Assumptions:	N/A		

3.2.3.2.2 Autocomplete suggestions

Use Case Specification

UC ID and Name:	**UC-5.0 Autocomplete suggestions**		
Created by:	**HaiND**	Date Created:	**16/09/2018**
Primary Actor:	**User**	Secondary Actors:	
Trigger:	N/A		
Description:	Suggestions during search terms input.		
Preconditions:			
Post conditions:	**POST-5.1.** Suggestions are shown		
Normal Flow:	**5.0 Autocomplete suggestions** 1. User visits website. 2. User input search terms into the search bar. 3. Relevant suggestions are displayed under the search bar.		
Alternative Flows:	N/A		
Exceptions:	**5.0-E1 – Cannot communicate with API server** 1. Web page displays error message.		
Priority:	High		
Frequency of Use:	Medium		
Business Rules:	N/A		
Other Information:			
Assumptions:	N/A		

3.2.3.2.3 Search for jobs/companies

Use Case Specification

UC ID and Name:	**UC-6.0 Search for jobs/companies**		
Created by:	**TuanPA**	Date Created:	**16/09/2018**
Primary Actor:	**Job Seeker**	Secondary Actors:	
Trigger:	N/A.		
Description:	User wants to search for all available jobs/companies by keywords.		
Preconditions:			
Post conditions:	**POST-6.1.** A list of job/companies matching the search terms is displayed.		
Normal Flow:	**6.0 View all available jobs** 1. User visits the website.		

	2. User type keywords into the search bar at the top of the page. 3. User clicks “Search” or presses Enter. 4. System displays the list of available jobs/companies.
Alternative Flows:	N/A
Exceptions:	**6.0-E1 – Cannot communicate with API server.** 1. GlassCV system displays error message.
Priority:	High
Frequency of Use:	High
Business Rules:	N/A
Other Information:	
Assumptions:	N/A

3.2.3.3 Job seeker

3.2.3.3.1 View user profile

Use Case Specification

UC ID and Name:	**UC-7.0 View user profile**		
Created by:	**HieuNT**	Date Created:	**16/09/2018**
Primary Actor:	**Job seeker**	Secondary Actors:	
Trigger:	N/A		
Description:	Job seeker wants to see his/her own profile.		
Preconditions:	**PRE-7.1.** Job seeker is logged in. **PRE-7.2.** Job seeker is not banned.		
Post conditions:	**POST-7.1.** Profile page is displayed.		
Normal Flow:	**7.0 View user profile** 1. Job seeker visits website. 2. Job seeker clicks the “User” icon on the navigation bar. 3. Job seeker clicks “Profile”. 4. Profile page is displayed.		
Alternative Flows:	N/A		
Exceptions:	**7.0-E1 – Cannot communicate with API server** 1. Web page displays error message.		
Priority:	High		
Frequency of Use:	Medium		
Business Rules:	N/A		

Other Information:	
Assumptions:	N/A

3.2.3.3.2 Request to become an employer

Use Case Specification

UC ID and Name:	**UC-8.0 Request to become an employer**		
Created by:	**HieuNT**	Date Created:	**16/09/2018**
Primary Actor:	**Job seeker**	Secondary Actors:	
Trigger:	N/A		
Description:	Job seeker wants to submit a request for an employer account.		
Preconditions:	**PRE-8.1.** Job seeker is logged in. **PRE-8.2.** Job seeker is not banned.		
Post conditions:	**POST-8.1.** Account request is submitted. **POST-8.2** Success message is displayed.		
Normal Flow:	**8.0 Request to become an employer** 1. Job seeker visits website. 2. Job seeker clicks "Become an Employer" on the navigation bar. 3. Request form is displayed. 4. Job seeker fills the form and click "Submit". 5. Success message is displayed.		
Alternative Flows:	N/A		
Exceptions:	**8.0-E1 Request form validation fails** 1. An error message is displayed. 2. The invalid form fields glow red. **8.0-E2 Cannot communicate with API server** 1. An error message is displayed.		
Priority:	Low		
Frequency of Use:	Low		
Business Rules:	**B17**		
Other Information:			
Assumptions:	N/A		

3.2.3.3.3 Create a review about a company

Use Case Specification

UC ID and Name:	**UC-9.0 Create a review about a company**		
Created by:	**HieuNT**	Date Created:	**16/09/2018**
Primary Actor:	**Job seeker**	Secondary Actors:	
Trigger:	N/A		
Description:	Job seeker wants to make a review of a company.		
Preconditions:	**PRE-9.1.** Job seeker is logged in. **PRE-9.2.** Job seeker is not banned.		
Post conditions:	**POST-9.1.** Company review is submitted. **POST-9.2** Success message is displayed.		
Normal Flow:	**9.0 Create a review of a company** 1. Job seeker visits website. 2. Job seeker clicks on a company. 3. Company profile page is displayed. 4. Job seeker clicks "Create review" button. 5. Review form is displayed. 6. Job seeker fills the form and click "Submit". 7. Success message is displayed.		
Alternative Flows:	N/A		
Exceptions:	**9.0-E1 Request form validation fails** 1. An error message is displayed. 2. The invalid form fields glow red. **9.0-E2 Cannot communicate with API server** 1. An error message is displayed.		
Priority:	High		
Frequency of Use:	Medium		
Business Rules:	**B14**		
Other Information:			
Assumptions:	N/A		

3.2.3.3.4 Edit a company review

Use Case Specification

UC ID and Name:	**UC-10.0 Edit a company review**		
Created by:	**HieuNT**	Date Created:	**16/09/2018**
Primary Actor:	**Job seeker**	Secondary Actors:	

Trigger:	N/A
Description:	Job seeker wants to edit his/her review of a company.
Preconditions:	**PRE-10.1.** Job seeker is logged in. **PRE-10.2** Job seeker has got a company review. **PRE-10.3.** Job seeker is not banned.
Post conditions:	**POST-10.1.** Company review is updated. **POST-10.2** Success message is displayed.
Normal Flow:	**10.0 Edit a company review** 1. Job seeker visits website. 2. Job seeker clicks "Profile". 3. Profile page is displayed. 4. Job seeker clicks "My Review". 5. Review page is displayed. 6. Job seeker clicks "Company". 7. Job seeker choose a company review and clicks the Edit button. 8. Review form is displayed. 9. Job seeker updates the form and click "Update". 10. Success message is displayed.
Alternative Flows:	N/A
Exceptions:	**10.0-E1 Request form validation fails** 1. An error message is displayed. 2. The invalid form fields glow red. **10.0-E2 Cannot communicate with API server** 1. An error message is displayed.
Priority:	Medium
Frequency of Use:	Low
Business Rules:	N/A
Other Information:	
Assumptions:	N/A

3.2.3.3.5 Create an interview review

Use Case Specification

UC ID and Name:	**UC-11.0 Create an interview review**		
Created by:	**HieuNT**	Date Created:	**16/09/2018**
Primary Actor:	**Job seeker**	Secondary Actors:	
Trigger:	N/A		

Description:	Job seeker wants to make a review of a company's interview process.
Preconditions:	**PRE-11.1.** Job seeker is logged in. **PRE-11.2.** Job seeker is not banned.
Post conditions:	**POST-11.1.** Interview review is submitted. **POST-11.2** Success message is displayed.
Normal Flow:	**11.0 Create an interview review** 1. Job seeker visits website. 2. Job seeker clicks on a company. 3. Company profile page is displayed. 4. Job seeker clicks on "Interview" tab. 5. Job seeker clicks "Create review". 6. Review form is displayed. 7. Job seeker fills the form and click "Submit". 8. Success message is displayed.
Alternative Flows:	N/A
Exceptions:	**11.0-E1 Request form validation fails** 1. An error message is displayed. 2. The invalid form fields glow red. **11.0-E2 Cannot communicate with API server** 1. An error message is displayed.
Priority:	High
Frequency of Use:	Medium
Business Rules:	**B14**
Other Information:	
Assumptions:	N/A

3.2.3.3.6 Edit an interview review

Use Case Specification

UC ID and Name:	**UC-12.0 Edit an interview review**		
Created by:	**HieuNT**	Date Created:	**16/09/2018**
Primary Actor:	**Job seeker**	Secondary Actors:	
Trigger:	N/A		
Description:	Job seeker wants to edit his/her review of a company's interview process.		
Preconditions:	**PRE-12.1.** Job seeker is logged in. **PRE-12.2.** Job seeker has got an interview review. **PRE-12.3.** Job seeker is not banned.		

Post conditions:	**POST-12.1.** Interview review is updated. **POST-12.2** Success message is displayed.
Normal Flow:	**12.0 Edit an interview review** 1. Job seeker visits website. 2. Job seeker clicks "Profile". 3. Profile page is displayed. 4. Job seeker clicks "My Review". 5. Review page is displayed. 6. Job seeker clicks "Interview". 7. Job seeker choose an interview review and clicks the Edit button. 8. Review form is displayed. 9. Job seeker updates the form and click "Update". 10. Success message is displayed.
Alternative Flows:	N/A
Exceptions:	**12.0-E1 Request form validation fails** 1. An error message is displayed. 2. The invalid form fields glow red. **12.0-E2 Cannot communicate with API server** 1. An error message is displayed.
Priority:	Medium
Frequency of Use:	Low
Business Rules:	**B13**
Other Information:	
Assumptions:	N/A

3.2.3.3.7 Follow/Unfollow a company

Use Case Specification

UC ID and Name:	**UC-13.0 Follow/Unfollow a company**		
Created by:	**HieuNT**	Date Created:	**16/09/2018**
Primary Actor:	**Job seeker**	Secondary Actors:	
Trigger:	N/A.		
Description:	Job seeker wants to subscribe to a company's notifications.		
Preconditions:	**PRE-13.1.** Job seeker is logged in. **PRE-13.2.** Job seeker is not banned.		
Post conditions:	**POST-13.1.** Follow status is updated.		
Normal Flow:	**13.0 Follow a company**		

	1. Job seeker visits website. 2. Job seeker clicks on a company. 3. Company profile page is displayed. 4. Job seeker clicks “Follow”. 5. “Follow” button changes to “Following” with a tick mark. 6. Job seeker clicks on “Follow” again to “Unfollow”. **13.1 Unfollow a company** 1. Job seeker visits website. 2. Job seeker clicks on a company. 3. Company profile page is displayed. 4. Job seeker clicks “Following”. “Following” button changes to “Follow”.
Alternative Flows:	**13.2 Follow a company after unfollowing (at “My Follow” page)** 1. Job seeker visits website. 2. Job seeker clicks “My Follow”. 3. Follow page is displayed. 4. Job seeker chooses a company and click “Unfollow”. 5. “Following” button changes to “Follow”. 6. Job seeker clicks “Follow”. 7. “Follow” button changes to “Following” with a tick mark. **13.3 Unfollow a company at “My Follow” page** 1. Job seeker visits website. 2. Job seeker clicks “My Follow”. 3. Follow page is displayed. 4. Job seeker chooses a company and click “Unfollow”. “Following” button changes to “Follow”.
Exceptions:	**13.0-E1 Cannot communicate with API server** 1. An error message is displayed.
Priority:	Medium
Frequency of Use:	Medium
Business Rules:	**B19**
Other Information:	
Assumptions:	N/A

3.2.3.3.8 View following companies

Use Case Specification

UC ID and Name:	**UC-14.0 View all companies followed by the user**		
Created by:	**HieuNT**	Date Created:	**16/09/2018**
Primary Actor:	**Job seeker**	Secondary Actors:	
Trigger:	N/A		

Description:	Job seeker wants to see all companies being followed by him/her.
Preconditions:	**PRE-14.1.** Job seeker is logged in. **PRE-14.2.** Job seeker is not banned.
Post conditions:	**POST-14.1.** Follow page is displayed.
Normal Flow:	**14.0 View following companies** 1. Job seeker visits website. 2. Job seeker clicks "Profile". 3. Profile page is displayed. 4. Job seeker clicks "My Follow". 5. Follow page is displayed.
Alternative Flows:	N/A
Exceptions:	**14.0-E1 – Cannot communicate with API server** 1. An error message is displayed.
Priority:	Medium
Frequency of Use:	Medium
Business Rules:	N/A
Other Information:	
Assumptions:	N/A

3.2.3.3.9 Update preferred locations

Use Case Specification

UC ID and Name:	**UC-15.0 Update preferred locations**		
Created by:	**HieuNT**	Date Created:	**16/09/2018**
Primary Actor:	**Job seeker**	Secondary Actors:	
Trigger:	N/A		
Description:	Job seeker wants to update his/her preferred working locations.		
Preconditions:	**PRE-15.1.** Job seeker is logged in. **PRE-15.2.** Job seeker is not banned.		
Post conditions:	**POST-15.1.** Preferred locations are updated.		
Normal Flow:	**15.0 Update preferred locations** 1. Job seeker visits website. 2. Job seeker clicks "Profile". 3. Profile page is displayed. 4. Job seeker clicks on "Preferences" tab. 5. Job seeker updates his/her location preferences and click "Save". 6. Success message is displayed.		

Alternative Flows:	N/A
Exceptions:	**15.0-E1 Cannot communicate with API server** 1. An error message is displayed.
Priority:	Medium
Frequency of Use:	Medium
Business Rules:	**B23**
Other Information:	
Assumptions:	N/A.

3.2.3.3.10 Update preferred companies

Use Case Specification

UC ID and Name:	**UC-16.0 Update ideal companies**		
Created by:	**HieuNT**	Date Created:	**16/09/2018**
Primary Actor:	**Job seeker**	Secondary Actors:	
Trigger:	N/A		
Description:	Job seeker wants to update his/her ideal companies.		
Preconditions:	**PRE-16.1.** Job seeker is logged in. **PRE-16.2.** Job seeker is not banned.		
Post conditions:	**POST-16.1.** Preferred companies are updated.		
Normal Flow:	**16.0 Update preferred companies** 1. Job seeker visits website. 2. Job seeker clicks “Profile”. 3. Profile page is displayed. 4. Job seeker clicks on “Preferences” tab. 5. Job seeker updates his/her company preferences and click “Save”. 6. Success message is displayed.		
Alternative Flows:	**B22**		
Exceptions:	**16.0-E1 Cannot communicate with API server** 1. An error message is displayed.		
Priority:	Medium		
Frequency of Use:	Medium		
Business Rules:	N/A		
Other Information:			

Assumptions:	N/A

3.2.3.3.11 Update user profile

Use Case Specification

UC ID and Name:	**UC-17.0 Update user profile**		
Created by:	**HaiND**	Date Created:	**16/09/2018**
Primary Actor:	**Job seeker**	Secondary Actors:	
Trigger:	N/A		
Description:	Update user profile, including avatar, skills, experience, education, etc.		
Preconditions:	**PRE-17.1.** Job seeker is logged in. **PRE-17.2.** Job seeker is not banned.		
Post conditions:	N/A		
Normal Flow:	**17.0 Update user profile** 1. Job seeker clicks “User” icon on the navigation bar. 2. Job seeker clicks “Profile” to open profile page. 3. Job seeker click “Add”/”Edit”/”Delete” button in profile sections. 4. Job seeker make changes to the information. 5. Job seeker clicks “Save changes” to save the changed information. 6. System saves changes to database. 7. Browser displays success message.		
Alternative Flows:	N/A		
Exceptions:	**17.0-E1 – Cannot communicate with API server.** 1. An error message is displayed.		
Priority:	Medium		
Frequency of Use:	Low		
Business Rules:	**B6**		
Other Information:			
Assumptions:	N/A		

3.2.3.3.12 Upload CV

Use Case Specification

UC ID and Name:	**UC-18.0 Upload CV**		
Created by:	**HaiND**	Date Created:	**16/09/2018**
Primary Actor:	**Job seeker**	Secondary Actors:	
Trigger:	N/A		
Description:	User wants to change his/her profile avatar.		

Preconditions:	**PRE-18.1.** Job seeker is logged in. **PRE-18.2.** Job seeker is not banned.
Post conditions:	**POST-18.1.** Job seeker's CV is uploaded and listed in the profile page.
Normal Flow:	**18.0 Upload CV** 1. Job seeker clicks "User" icon on the navigation bar. 2. Job seeker clicks "Profile" to open the profile page. 3. Job seeker clicks "Upload CV". 4. Job seeker chooses CV file from the file chooser dialog. 5. The file is saved in cloud storage. 6. Browser list the uploaded CV under CV list.
Alternative Flows:	N/A.
Exceptions:	**18.0-E1 – Cannot communicate with API server.** 1. An error message is displayed. **18.0-E2 – Invalid file.** 1. An error message is displayed.
Priority:	Medium
Frequency of Use:	Medium
Business Rules:	**B15, B18, B24**
Other Information:	
Assumptions:	N/A

3.2.3.3.13 Save a job

Use Case Specification

UC ID and Name:	**UC-19.0 Upload CV**		
Created by:	**HaiND**	Date Created:	**16/09/2018**
Primary Actor:	**Job seeker**	Secondary Actors:	
Trigger:	N/A		
Description:	User wants to save a job.		
Preconditions:	**PRE-19.1.** Job seeker is logged in. **PRE-19.2.** Job seeker is not banned.		
Post conditions:	**POST-19.1.** Job seeker successfully manage to save a job for personal viewing.		
Normal Flow:	**19.0 Save a job** 1. Job seeker clicks "Jobs" icon on the navigation bar. 2. Job seeker clicks on a job post on the list to view the job description. 3. Job seeker clicks on the heart button next to a preferred job 4. Job is saved for the logged in job seeker		

Alternative Flows:	**19.1 Save a job** 1. Job seeker enter keywords into the search bar and click "Search" 2. Job seeker clicks on a job post on the list to view the job description. 3. Job seeker clicks on the heart button next to a preferred job 4. Job is saved for the logged in job seeker
Exceptions:	**19.0-E1 – Cannot communicate with API server.** 1. An error message is displayed.
Priority:	Medium
Frequency of Use:	Medium
Business Rules:	
Other Information:	
Assumptions:	N/A

3.2.3.3.14 Create mail notification

Use Case Specification

UC ID and Name:	**UC-20.0 Create mail notification**		
Created by:	**HaiND**	Date Created:	**16/09/2018**
Primary Actor:	**Job seeker**	Secondary Actors:	
Trigger:	N/A		
Description:	User wants to create mail notification for a company's activities.		
Preconditions:	**PRE-20.1.** Job seeker is logged in. **PRE-20.2.** Job seeker is not banned.		
Post conditions:	**POST-20.1.** Job seeker's CV is uploaded and listed in the profile page.		
Normal Flow:	**20.0 Create mail notification** 1. Job seeker enter keywords into the search bar and click "Search" 2. Job seeker clicks on "Follow" 3. Job seeker is now in the mailing list for new jobs from this company		
Alternative Flows:	N/A.		
Exceptions:	**20.0-E1 – Cannot communicate with API server.** 1. An error message is displayed.		
Priority:	Medium		
Frequency of Use:	Medium		
Business Rules:	**B20**		
Other Information:			

Assumptions:	N/A

3.2.3.3.15 Update/Delete saved jobs

Use Case Specification

UC ID and Name:	**UC-21.0 Update/delete saved jobs**		
Created by:	**HaiND**	Date Created:	**16/09/2018**
Primary Actor:	**Job seeker**	Secondary Actors:	
Trigger:	N/A		
Description:	User wants to change his/her profile avatar.		
Preconditions:	**PRE-21.1.** Job seeker is logged in. **PRE-21.2.** Job seeker is not banned. **PRE-21.3.** Job seeker has at least one saved job(s).		
Post conditions:	**POST-21.1.** Job seeker manage to delete a saved job.		
Normal Flow:	**21.0 Update/delete saved jobs** 1. Job seeker clicks "Saved Jobs" icon on the navigation bar. 2. Job seeker clicks on a job post on the list to view the job description. 3. Job seeker clicks on the heart button next to a preferred job 4. Saved job is now deleted from the list.		
Alternative Flows:	N/A.		
Exceptions:	**21.0-E1 – Cannot communicate with API server.** 1. An error message is displayed.		
Priority:	Medium		
Frequency of Use:	Medium		
Business Rules:			
Other Information:			
Assumptions:	N/A		

3.2.3.3.16 Quick apply for a job

Use Case Specification

UC ID and Name:	**UC-22.0 Quick apply for a job**		
Created by:	**TuanPA**	Date Created:	**16/09/2018**
Primary Actor:	**Job Seeker**	Secondary Actors:	
Trigger:	N/A		
Description:	User wants to use [Quick Apply] when seeing a job post.		

Preconditions:	**PRE-22.1.** User is logged into GlassCV. **PRE-22.2.** User is not banned. **PRE-22.3.** User is viewing a job post.
Post conditions:	**POST-22.1.** User's application is saved in the database.
Normal Flow:	**22.0 Quick apply for a job** 1. User visits a job post. 2. User clicks on the job post. 3. User reads the description about the job and is interested in the job. 4. User clicks on [Quick Apply] button on the job post. 5. User is notified about successfully applying for the job.
Alternative Flows:	N/A
Exceptions:	**22.0-E1 – Cannot communicate with API server.** 1. GlassCV system displays error message.
Priority:	High
Frequency of Use:	High
Business Rules:	N/A
Other Information:	
Assumptions:	N/A

3.2.3.3.17 View all available jobs

Use Case Specification

UC ID and Name:	**UC-23.0 View all available jobs**		
Created by:	**TuanPA**	Date Created:	**16/09/2018**
Primary Actor:	**Job Seeker**	Secondary Actors:	
Trigger:	N/A.		
Description:	User wants to view all available jobs of a company.		
Preconditions:	**PRE-23.1.** User is logged into GlassCV. **PRE-23.2.** User is not banned. **PRE-23.3.** User is watching the company profile page.		
Post conditions:	**POST-23.1.** Browser displays a list of available jobs.		
Normal Flow:	**23.0 View all available jobs** 1. User clicks on "Job" tab on company profile. 2. System displays the list of the jobs posted by that company.		

Alternative Flows:	N/A
Exceptions:	**23.0-E1 – Cannot communicate with API server.** 1. GlassCV system displays error message.
Priority:	Medium
Frequency of Use:	High
Business Rules:	N/A
Other Information:	
Assumptions:	N/A

3.2.3.3.18 View recommended job posts

Use Case Specification

UC ID and Name:	**UC-24.0 View recommended job posts**		
Created by:	**TuanPA**	Date Created:	**16/09/2018**
Primary Actor:	**Job Seeker**	Secondary Actors:	
Trigger:	N/A		
Description:	User wants to view jobs that may interest her/him.		
Preconditions:	**PRE-24.1.** User is logged into GlassCV. **PRE-24.2.** User is not banned.		
Post conditions:	**POST-24.1.** A list of recommended job posts is displayed		
Normal Flow:	**24.0 View recommended job posts** 1. User clicks on [Recommendations] on the navigation bar. 2. A list of suggested job is displayed.		
Alternative Flows:	N/A		
Exceptions:	**24.0-E1 – Cannot communicate with API server.** 1. GlassCV system displays error message.		
Priority:	High		
Frequency of Use:	High		
Business Rules:	N/A		
Other Information:			
Assumptions:	N/A		

3.2.3.3.19 Remove a recommended job post

Use Case Specification

UC ID and Name:	**UC-25.0 Remove a recommended job post**		
Created by:	**HaiND**	Date Created:	**16/09/2018**
Primary Actor:	**Job Seeker**	Secondary Actors:	
Trigger:	N/A		
Description:	User wants to remove a job post recommended to him/her.		
Preconditions:	**PRE-25.1.** User is logged into GlassCV. **PRE-25.2.** User is not banned.		
Post conditions:	**POST-25.1.** The specified job post is removed from recommendations		
Normal Flow:	**25.0 Remove a recommended job post** 1. User clicks [Recommendations] on the navigation bar. 2. A list of suggested job is displayed. 3. User clicks the menu icon on a recommended job post. 4. User clicks [Not interested] button. 5. The job post is removed from recommendations.		
Alternative Flows:	N/A		
Exceptions:	**25.0-E1 – Cannot communicate with API server.** 1. GlassCV system displays error message.		
Priority:	High		
Frequency of Use:	High		
Business Rules:	N/A		
Other Information:			
Assumptions:	N/A		

3.2.3.4 Employer

3.2.3.4.1 Create a job post

Use Case Specification

UC ID and Name:	**UC-26.0 Create a job post**		
Created by:	**LongNH**	Date Created:	**16/09/2018**
Primary Actor:	**Employer**	Secondary Actors:	
Trigger:	N/A		
Description:	An employer creates a job recruitment post.		
Preconditions:	**PRE-26.1.** User is logged into GlassCV as an employer**.** **PRE-26.2.** User is not banned.		
Post conditions:	**POST-26.1.** User is notified of successful submission.		

	POST-26.2. Job post is submitted, pending approval from administrator. **POST-26.3.** User is redirected to job post list.
Normal Flow:	**26.0. Create job post** 1. User visits website. 2. User clicks on "Employer dashboard". 3. User clicks on "Add Job Post". 4. System displays add post form. 5. User inputs: "Title", "Location", "Length of contract", "Description". 6. User choose job type. 7. User clicks on "Submit".
Alternative Flows:	N/A
Exceptions:	**26.0-E1 – Cannot communicate with API server.** 1. GlassCV system displays an error message.
Priority:	High
Frequency of Use:	Medium
Business Rules:	N/A
Other Information:	
Assumptions:	N/A

3.2.3.4.2 Edit/Delete created job posts

Use Case Specification

UC ID and Name:	**UC-27.0 Edit/Delete created job posts**		
Created by:	**LongNH**	Date Created:	**16/09/2018**
Primary Actor:	**Employer**	Secondary Actors:	
Trigger:	N/A		
Description:	An employer updates or delete a job recruitment post.		
Preconditions:	**PRE-27.1.** User is logged into GlassCV as an employer user. **PRE-27.2.** User is not banned.		
Post conditions:	**POST-27.1.** User is notified of successful submission. **POST-27.2.** Job post is updated. **POST-27.3.** User is redirected to job post list.		
Normal Flow:	**27.0. Edit created job posts** 1. User visits website. 2. User clicks on "Employer dashboard". 3. User clicks on "Job posts". 4. System redirects to job posts page. 5. User chooses desired job post and click on the post's respective "Update" button.		

	6. System redirects to update post page. 7. User updates the input fields as desired. 8. User clicks on “Submit”.
Alternative Flows:	**27.0. Delete Job post** **(**continued from step 6 of normal flow) 7. User clicks on “Delete” button 8. System shows confirmation dialog 9. User clicks on “Yes” 10. Job post is deleted 11. User is redirected to job posts page
Exceptions:	**27.0-E1 – Cannot communicate with API server.** 1. GlassCV system displays error message.
Priority:	High
Frequency of Use:	Medium
Business Rules:	N/A
Other Information:	
Assumptions:	N/A

3.2.3.4.3 Create employer profile

Use Case Specification

UC ID and Name:	**UC-28.0 Create employer profile**		
Created by:	**LongNH**	Date Created:	**16/09/2018**
Primary Actor:	**Employer**	Secondary Actors:	
Trigger:	N/A		
Description:	Use completes employer profile.		
Preconditions:	**PRE-28.1.** User is logged into GlassCV. **PRE-28.2.** User is not banned.		
Post conditions:	**POST-28.1.** User’s employer profile is updated.		
Normal Flow:	**28.0. Create employer profile** 1. User visits website. 2. User clicks on “Create employer profile”. 3. System redirects user to Employer Profile form. 4. User inputs into respective fields: “Company”, “Position”, “Location”. 5. User clicks on “Submit”. 6. Employer profile is created. 9. System redirects user to Employer Dashboard page.		
Alternative Flows:	N/A		

Exceptions:	**28.0-E1 – Cannot communicate with API server.** 1. GlassCV system displays error message.
Priority:	Medium
Frequency of Use:	Medium
Business Rules:	N/A
Other Information:	
Assumptions:	N/A

3.2.3.4.4 View a job seeker resume

Use Case Specification

UC ID and Name:	**UC-29.0 View a job seeker resume**		
Created by:	**LongNH**	Date Created:	**16/09/2018**
Primary Actor:	**Employer**	Secondary Actors:	
Trigger:	N/A		
Description:	An employer views a potential applicant's profile		
Preconditions:	**PRE-29.1.** User is logged into GlassCV. **PRE-29.2.** User is not banned. **PRE-29.3.** There are applicant(s) applied for a job(s) posted by the user using the system.		
Post conditions:	**POST-29.1.** The job seeker's resume is displayed.		
Normal Flow:	**29.0. View a job seeker resume** 1. User visits website. 2. User clicks on "Employer dashboard" 3. User clicks on "View applicants" 4. System redirects user to Applicants page 5. User clicks on desired applicant 6. System redirects user to applicant's profile page		
Alternative Flows:	N/A		
Exceptions:	**29.0-E1 – Cannot communicate with API server.** 1. GlassCV system displays error message.		
Priority:	High		
Frequency of Use:	High		
Business Rules:	**B25**		
Other Information:			
Assumptions:	N/A		

3.2.3.4.5 View all applicants in quick-apply

Use Case Specification

UC ID and Name:	**UC-30.0 View all applicants in quick-apply**		
Created by:	**LongNH**	Date Created:	**16/09/2018**
Primary Actor:	**Employer**	Secondary Actors:	
Trigger:	N/A		
Description:	An employer views all applicants who quick-apply for a job post		
Preconditions:	**PRE-30.1.** User is logged into GlassCV. **PRE-30.2.** User is not banned. **PRE-30.3.** User has unlocked Employer Account privileges **PRE-30.4.** There are applicant(s) applied for a job(s) posted by the user using the system.		
Post conditions:	**POST-30.1.** The job post's applicants are displayed.		
Normal Flow:	**30.0. View all applicants in quick-apply** 1. User visits website. 2. User clicks on "Employer dashboard" 3. User clicks on preferred job 4. Applicants for the job are listed under "Applicants" list		
Alternative Flows:	N/A		
Exceptions:	**30.0-E1 – Cannot communicate with API server.** 1. GlassCV system displays error message.		
Priority:	High		
Frequency of Use:	High		
Business Rules:	N/A		
Other Information:			
Assumptions:	N/A		

3.2.3.5 Staff

3.2.3.5.1 View all active users

Use Case Specification

UC ID and Name:	**UC-31.0 View all active users**		
Created by:	**HaiND**	Date Created:	**16/09/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		

Description:	View all active users, including administrators, moderators and normal users.
Preconditions:	**PRE-31.1.** User logged in as Administrator or Moderator. **PRE-31.2.** User is not banned.
Post conditions:	**POST-31.1.** A list of currently active users is displayed.
Normal Flow:	**31.0. View all active users** 1. Staff visits the admin site. 2. Staff clicks "Users" on the left side bar. 3. A list of active users is displayed.
Alternative Flows:	N/A
Exceptions:	**31.0-E1 – Cannot communicate with API server.** 1. An error message is displayed.
Priority:	Medium
Frequency of Use:	Medium
Business Rules:	N/A
Other Information:	
Assumptions:	N/A

3.2.3.5.2 View banned users

Use Case Specification

UC ID and Name:	**UC-32.0 View banned users**		
Created by:	**HaiND**	Date Created:	**16/09/2018**
Primary Actor:	**Administrator**	Secondary Actors:	
Trigger:	N/A		
Description:	View all banned users.		
Preconditions:	**PRE-32.1.** User logged in as Administrator. **PRE-32.2.** User is not banned.		
Post conditions:	**POST-32.1.** A list of banned users is displayed.		
Normal Flow:	**32.0. View banned users** 1. Administrator visits the admin site. 2. Administrator clicks "Users" on the left side bar. 3. Administrator clicks "Banned" tab. 4. A list of banned users is displayed.		
Alternative Flows:	N/A		

Exceptions:	**32.0-E1 – Cannot communicate with API server.** 1. An error message is displayed.
Priority:	Medium
Frequency of Use:	Medium
Business Rules:	N/A
Other Information:	
Assumptions:	N/A

3.2.3.5.3 Update a user's role or employer status

Use Case Specification

UC ID and Name:	**UC-33.0 Update a user's role or employer status**		
Created by:	**HaiND**	Date Created:	**16/09/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Update the role or employer status of a user. Possible roles are: Normal, Moderator, Administrator.		
Preconditions:	**PRE-33.1.** User logged in as Administrator or Moderator. **PRE-33.2.** User is not banned.		
Post conditions:	**POST-33.1.** The specified user has new role and/or employer status.		
Normal Flow:	**33.0. Update a user's role** 1. User visits the admin site. 2. User clicks "User" on the left side bar. 3. User list is displayed. 4. User clicks "Edit" button on a user row. 5. The table cell displays "Edit mode", user chooses a new role from the role select input. 6. User clicks "Done" button. 7. A toast notification displays, informing successful update of the user's role.		
Alternative Flows:	**33.1. Update a user's employer status** (continued from step 4) 5. The table cell displays "Edit mode", Staff checks/unchecks the employer checkbox input. 6. A toast notification displays, informing successful update of the user's employer status.		
Exceptions:	**33.0-E1 – Cannot communicate with API server.** 1. An error message is displayed. **33.0-E2 – The user whose role will be updated is the root admin.**		

	1. An error message is displayed. **33.0-E3 – The currently logged in Moderator trying to update role of an Administrator.** 1. An error message is displayed. **33.0-E4 – The currently logged in Staff trying to update his/her own role.** 1. An error message is displayed.
Priority:	Medium
Frequency of Use:	Medium
Business Rules:	N/A
Other Information:	
Assumptions:	N/A

3.2.3.5.4 Ban/Unban a user

Use Case Specification

UC ID and Name:	**UC-34.0 Ban/Unban a user**		
Created by:	**HaiND**	Date Created:	**16/09/2018**
Primary Actor:	**Administrator**	Secondary Actors:	
Trigger:	N/A		
Description:	Ban or unban a user.		
Preconditions:	**PRE-34.1.** User logged in as Administrator. **PRE-34.2.** User is not banned.		
Post conditions:	**POST-34.1.** The specified user is banned.		
Normal Flow:	**34.0. Ban a user** 1. Administrator visits the admin site. 2. Administrator clicks “Users” on the left side bar. 3. Administrator clicks the “Ban” button on a user row. 4. A toast notification displays, informing successful banning of the user.		
Alternative Flows:	**34.1. Unban a user** (continued from step 2 of normal flow) 3. Administrator clicks “Banned” tab. 4. Administrator clicks the “Unban” button on a user row. 5. A toast notification displays, informing successful unbanning of the user.		
Exceptions:	**34.0-E1 – Cannot communicate with API server.** 1. An error message is displayed.		

	34.0-E2 – The user who will be banned is the root admin. 1. An error message is displayed. **34.0-E3 – The currently logged in Moderator trying to ban an Administrator.** 1. An error message is displayed. **34.0-E4 – The currently logged in Staff trying to ban herself/himself.** 1. An error message is displayed.
Priority:	Medium
Frequency of Use:	Medium
Business Rules:	N/A
Other Information:	
Assumptions:	N/A

3.2.3.5.5 View all companies

Use Case Specification

UC ID and Name:	**UC-35.0 View all companies**		
Created by:	**HaiND**	Date Created:	**16/09/2018**
Primary Actor:	**Administrator**	Secondary Actors:	
Trigger:	N/A		
Description:	View all companies.		
Preconditions:	**PRE-35.1.** User logged in as Administrator or Moderator. **PRE-35.2.** User is not banned.		
Post conditions:	**POST-35.1.** A list of all companies is displayed.		
Normal Flow:	**35.0. View all companies** 1. User visits the admin site. 2. Administrator clicks “Companies” on the left side bar. 3. A list of all companies is displayed.		
Alternative Flows:	N/A		
Exceptions:	N/A		
Priority:	Medium		
Frequency of Use:	Medium		
Business Rules:	N/A		
Other Information:			
Assumptions:	N/A		

3.2.3.5.6 Create a new company

Use Case Specification

UC ID and Name:	**UC-36.0 Create a new company**		
Created by:	**HaiND**	Date Created:	**16/09/2018**
Primary Actor:	**Administrator**	Secondary Actors:	
Trigger:	N/A		
Description:	Create a new company		
Preconditions:	**PRE-36.1.** User logged in as Administrator or Moderator. **PRE-36.2.** User is not banned.		
Post conditions:	**POST-37.1.** The new company is created.		
Normal Flow:	**36.0. Create a new company** 1. User visits the admin site. 2. Administrator clicks "Companies" on the left side bar. 3. Staff clicks "Add" button. 4. Staff fills in company information. 5. Staff clicks "Done". 6. A toast notification pops up confirming successful creation of the new company.		
Alternative Flows:	N/A		
Exceptions:	N/A		
Priority:	Medium		
Frequency of Use:	Medium		
Business Rules:	N/A		
Other Information:			
Assumptions:	N/A		

3.2.3.5.7 Update/Delete a company

Use Case Specification

UC ID and Name:	**UC-37.0 Update/Delete a company**		
Created by:	**HaiND**	Date Created:	**16/09/2018**
Primary Actor:	**Administrator**	Secondary Actors:	
Trigger:	N/A		
Description:	Create a new company		
Preconditions:	**PRE-37.1.** User logged in as Administrator or Moderator. **PRE-37.2.** User is not banned.		

Post conditions:	**POST-37.1.** The specified company is updated/deleted.
Normal Flow:	**37.0. Update a company** 1. User visits the admin site. 2. Administrator clicks “Companies” on the left side bar. 3. Administrator clicks “Edit” on a company row. 4. Administrator fills in company information. 5. Administrator clicks “Done”. 6. A toast notification pops up confirming successful update of the company.
Alternative Flows:	**37.1. Delete a company** (continued from step 2 of normal flow) 3. Administrator clicks “Delete” on a company row. 4. A toast notification pops up confirming successful deletion of the company.
Exceptions:	N/A
Priority:	Medium
Frequency of Use:	Medium
Business Rules:	N/A
Other Information:	
Assumptions:	N/A

3.2.3.5.8 View employer requests

Use Case Specification

UC ID and Name:	**UC-38.0 View employer requests**		
Created by:	**HaiND**	Date Created:	**16/09/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	View requests from users who want to unlock employer account.		
Preconditions:	**PRE-38.1.** User logged in as Administrator or Moderator. **PRE-38.2.** User is not banned.		
Post conditions:	**POST-38.1.** A list of employer requests is displayed.		
Normal Flow:	**38.0. View employer requests** 1. User visits the admin site. 2. Staff clicks “Requests” from the left side bar to expand sub menus. 3. Staff clicks “Employer requests” from the sub menu. 4. Staff clicks the Pending/Approved/Rejected tab.		

	5. A list of pending/approved/rejected employer requests is displayed accordingly.
Alternative Flows:	N/A
Exceptions:	N/A
Priority:	Medium
Frequency of Use:	Medium
Business Rules:	N/A
Other Information:	
Assumptions:	N/A

3.2.3.5.9 Approve/Reject employer requests

Use Case Specification

UC ID and Name:	**UC-39.0 Approve/Reject employer requests**		
Created by:	**HaiND**	Date Created:	**16/09/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Approve/Reject an employer request		
Preconditions:	**PRE-39.1.** User logged in as Administrator or Moderator. **PRE-39.2.** User is not banned.		
Post conditions:	**POST-39.1.** The employer request is approved/rejected.		
Normal Flow:	**39.0. Approve/Reject an employer request** 1. User visits the admin site. 2. Staff clicks “Requests” from the left side bar to expand sub menus. 3. Staff clicks “Employer requests” from the sub menu. 4. Staff clicks “Approve” or “Reject” on an employer request row from the list. 5. A toast notification pops up confirming successful approval/rejection of the request.		
Alternative Flows:	N/A		
Exceptions:	N/A		
Priority:	Medium		
Frequency of Use:	Medium		
Business Rules:	N/A		
Other Information:			

Assumptions:	N/A

3.2.3.5.10 View user reviews

Use Case Specification

UC ID and Name:	**UC-40.0 View user reviews**		
Created by:	**HaiND**	Date Created:	**16/09/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	View reviews made by users that need to be approved first.		
Preconditions:	**PRE-40.1.** User logged in as Administrator or Moderator. **PRE-40.2.** User is not banned.		
Post conditions:	**POST-40.1.** A list of user reviews is displayed.		
Normal Flow:	**40.0. View user reviews** 1. User visits the admin site. 2. Staff clicks "Requests" from the left side bar to expand sub menus. 3. Staff clicks "User reviews" from the sub menu. 4. Staff clicks the Pending/Approved/Rejected tab. 5. A list of pending/approved/rejected user reviews is displayed accordingly.		
Alternative Flows:	N/A		
Exceptions:	N/A		
Priority:	Medium		
Frequency of Use:	Medium		
Business Rules:	N/A		
Other Information:			
Assumptions:	N/A		

3.2.3.5.11 Approve/Reject user reviews

Use Case Specification

UC ID and Name:	**UC-41.0 Approve/Reject user reviews**		
Created by:	**HaiND**	Date Created:	**16/09/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Approve/Reject a user review		

Preconditions:	**PRE-41.1.** User logged in as Administrator or Moderator. **PRE-41.2.** User is not banned.
Post conditions:	**POST-41.1.** The user review is approved/rejected.
Normal Flow:	**41.0. Approve/Reject a user review** 1. User visits the admin site. 2. Staff clicks "Requests" from the left side bar to expand sub menus. 3. Staff clicks "User reviews" from the sub menu. 4. Staff clicks "Approve" or "Reject" on a user review row from the list. 5. A toast notification pops up confirming successful approval/rejection of the review.
Alternative Flows:	N/A
Exceptions:	N/A
Priority:	Medium
Frequency of Use:	Medium
Business Rules:	N/A
Other Information:	
Assumptions:	N/A

3.3 Non-functional Requirement

3.3.1 Security

- ✓ Token-Based Authentication, relies on a signed token that is sent to the server on each request.
- ✓ Because we use Token-Based Authentication, so all request must be made on the correct frontend website otherwise it will be rejected, which made Cross-Site Request Forgery (CSRF) is impossible.
- ✓ All query made to SQL must be escaped so no SQL Injection.
- ✓ Apply SSL encryption to restrict user access and prevent man-in-the-middle attacking.
- ✓ All passwords are encrypted by bcrypt algorithm with salt.
- ✓ The security matrix is as the following table:

Function	Guest	Job seeker	Employer	Moderator	Administrator
Normal sign in	✓	✓	✓	✓	✓
Sign in with Google account	✓	✓	✓	✓	✓
Sign up	✓	✓	✓	✓	✓
Sign out		✓	✓	✓	✓
Autocomplete suggestions	✓	✓	✓	✓	✓
Search for jobs/companies	✓	✓	✓	✓	✓

Function	Guest	Job seeker	Employer	Moderator	Administrator
View user profile		✓	✓	✓	✓
Request to become an employer to create jobs recruitment posts		✓	✓	✓	✓
Create a review about a company		✓	✓	✓	✓
Edit a company review		✓	✓	✓	✓
Create interview review about a company		✓	✓	✓	✓
Edit review about the company's interview		✓	✓	✓	✓
Follow/Unfollow a company		✓	✓	✓	✓
View all companies being followed by the user		✓	✓	✓	✓
Edit preferred locations		✓	✓	✓	✓
Edit preferred company		✓	✓	✓	✓
Update User Profile		✓	✓	✓	✓
Upload CV		✓	✓	✓	✓
Save a job		✓	✓	✓	✓
Create mail notification		✓	✓	✓	✓
Update/Delete saved jobs		✓	✓	✓	✓
Quick apply for a job		✓	✓	✓	✓
View all available jobs		✓	✓	✓	✓
View recommended job posts		✓	✓	✓	✓
Remove a recommended job posts		✓	✓	✓	✓
Create a job post			✓		
Edit/Delete created job posts			✓		
Create employer profile			✓		
View a job seeker resume			✓		
View all applicants in quick-apply			✓		
View all active users				✓	✓
View banned users					✓
Update a user's role/employer status				✓	✓
Ban/Unban a user					✓
View all companies					✓

Function	Guest	Job seeker	Employer	Moderator	Administrator
Create a new company					✓
Update/Delete a company					✓
View employer requests				✓	✓
Approve/Reject employer requests				✓	✓
View user reviews				✓	✓
Approve/Reject user reviews				✓	✓

Table 3-2: Security matrix

3.3.2 Maintainability & Extensibility

- Microservices architecture for both backend & frontend.
- Automation test including Unit Tests and Integration Tests are written using Karma and unittest.
- Using TSLint for a consistent coding convention.
- Maintainable source code and easy deployment since we follow a single coding convention & a highly maintainable architecture design.
- Object Oriented Programming paradigm is applied in order to ensure scalability & maintainability.
- Front-end project follows Angular 7 standard MVC structure to ensure maintainability.

3.3.3 Availability and Scalability

- Google Cloud service for automatic horizontal scalability.
- ELK Stack helps monitor all the services through smart logs & bugs report.
- Third-party services for convenient development:
  - SendGrid for sending emails.
  - Google storage for storing images.

3.3.4 Performance

- ✓ Web front-end is a Single Page Application with the goal of providing a more fluid user experience similar to a desktop application.
- ✓ Backend system uses background job for external requests, heavy processing requests such as:
  - Sending email
  - Training recommendations model
- ✓ Use Elasticsearch for high performance searching.
- ✓ Add indexes to MySQL, which make query incurring filtering and sorting run much faster.

3.3.5 Usability

- ✓ The interface should be elegant and simple.
- ✓ Alt attributes are provided for non-text elements, such as images and maps.
- ✓ Links, buttons and checkboxes are easily clickable, for example a user can select a checkbox by clicking the text, not just the checkbox.
- ✓ Search box is wide enough, so that users can see what they've typed.
- ✓ Search is available on every page, not just the homepage.

- ✓ Links are easily recognizable. They look clickable. Items that aren't links don't look clickable, for example underlining text is avoided.
- ✓ Important commands are displayed as buttons, not links. For example, "Apply" or "Save" is a button, not a link.

Chapter 4 : Software Design

4.1 Purpose

This chapter is to give the developer team a of what the system's architecture is, and how they should be implemented. This chapter consists of:

- ✓ Architecture overview
- ✓ Component diagram
- ✓ Detailed design
- ✓ Detailed description of components
- ✓ Database design

4.2 Architecture Overview

4.2.1 System Architecture

4.2.1.1 Diagram

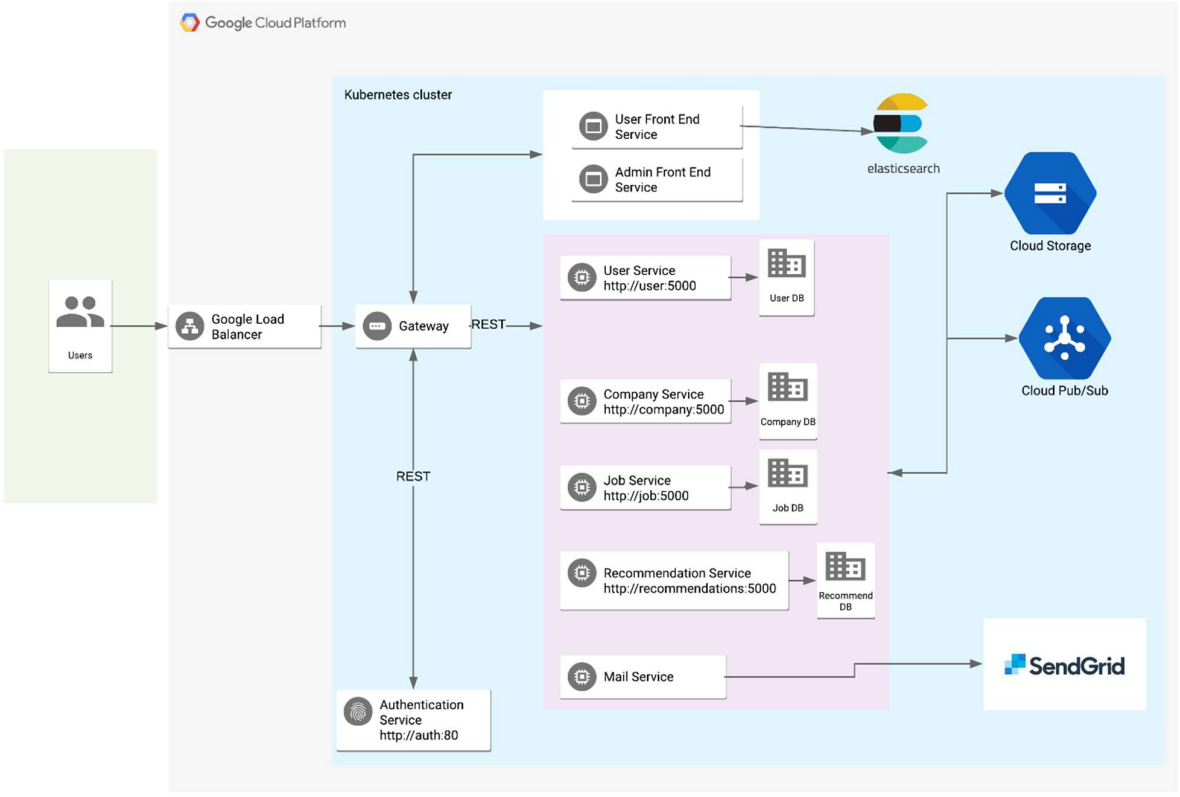


Figure 4-1: GlassCV System Architecture

4.2.2 System Architecture Explanation

The entire project will be deployed on Digital Ocean. We aim at delivering a secured, responsive, and highly available system. In the following section, we will explain the function and mechanism of each unit in the system architecture design.

4.2.2.1 Google Cloud Storage

Figure 4-2: Google Cloud Storage

Google Cloud Storage is a web-services interface data storage & retrieval. We use Google Cloud Storage for **storing images and CVs uploaded by users**.

4.2.2.2 Google Cloud Pub/Sub

Figure 4-3: Google Cloud Pub/Sub

Cloud Pub/Sub is a fully-managed real-time messaging service that allows you to send and receive messages between independent applications. We use Cloud Pub/Sub to handle communication among microservices.

4.2.2.3 Docker

Figure 4-4: Docker

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. We use Docker to build and push images to Docker Hub for deployment and testing.

4.2.2.4 Google Cloud

Figure 4-5: Google Cloud

Google Cloud Platform is a suite of cloud computing services that runs on the same infrastructure that Google uses internally for its end-user products. Alongside a set of management tools, it provides a series of modular cloud services including computing, data storage, data analytics and machine learning. Apart from **Cloud Storage** and **Pub/Sub** mentioned above, the main part of operation lies in the usage of the **Kubernetes Engine** that supports easy deployment and scaling from docker images.

4.2.2.5 Nginx

Figure 4-6: Nginx

NGINX is a free, open-source, high-performance HTTP server and reverse proxy, as well as an IMAP/POP3 proxy server. NGINX is known for its high performance, stability, rich feature set, simple configuration, and low resource consumption. We use Nginx to serve the frontend service of GlassCV.

4.2.2.6 MySQL

Figure 4-7: MySQL

MySQL is an open-source relational database management system (RDBMS). The MySQL development project has made its source code available under the terms of the GNU General Public License, as well as under a variety of proprietary agreements. We use MySQL for **transactional data storage & retrieval.**

4.2.2.7 Elasticsearch

Figure 4-8: Elasticsearch

Elasticsearch is a search engine based on Lucene. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. Elasticsearch is considered to be a NoSQL Database according to this article https://www.elastic.co/blog/found-elasticsearch-as-nosql.

4.2.2.8 Angular 7

Figure 4-9: Angular 7

Angular 7 is the latest created library of Angular, a framework created by Google that supports Flat & Material design website. Supporting Progressive Web Application, this framework has become very popular nowadays.

4.2.2.9 Angular Material

Figure 4-10: Angular Material

Angular Material is a great JavaScript framework for creating themed UI components that ignite good user experience.

4.2.2.10 Python 3

Figure 4-11: Python 3

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library. We use Python 3 to write back-end codes.

4.2.2.11 Flask

Figure 4-12: Flask

Flask is a microframework for Python. We use Flask to write back-end API services.

4.2.2.12 SendGrid

Figure 4-13: SendGrid

SendGrid is a platform for transactional and marketing email. We use SendGrid to send emails to users for verification and notification.

4.3 Design of GlassCV System

4.3.1 Architecture Layers Design

4.3.1.1 API Layer Design

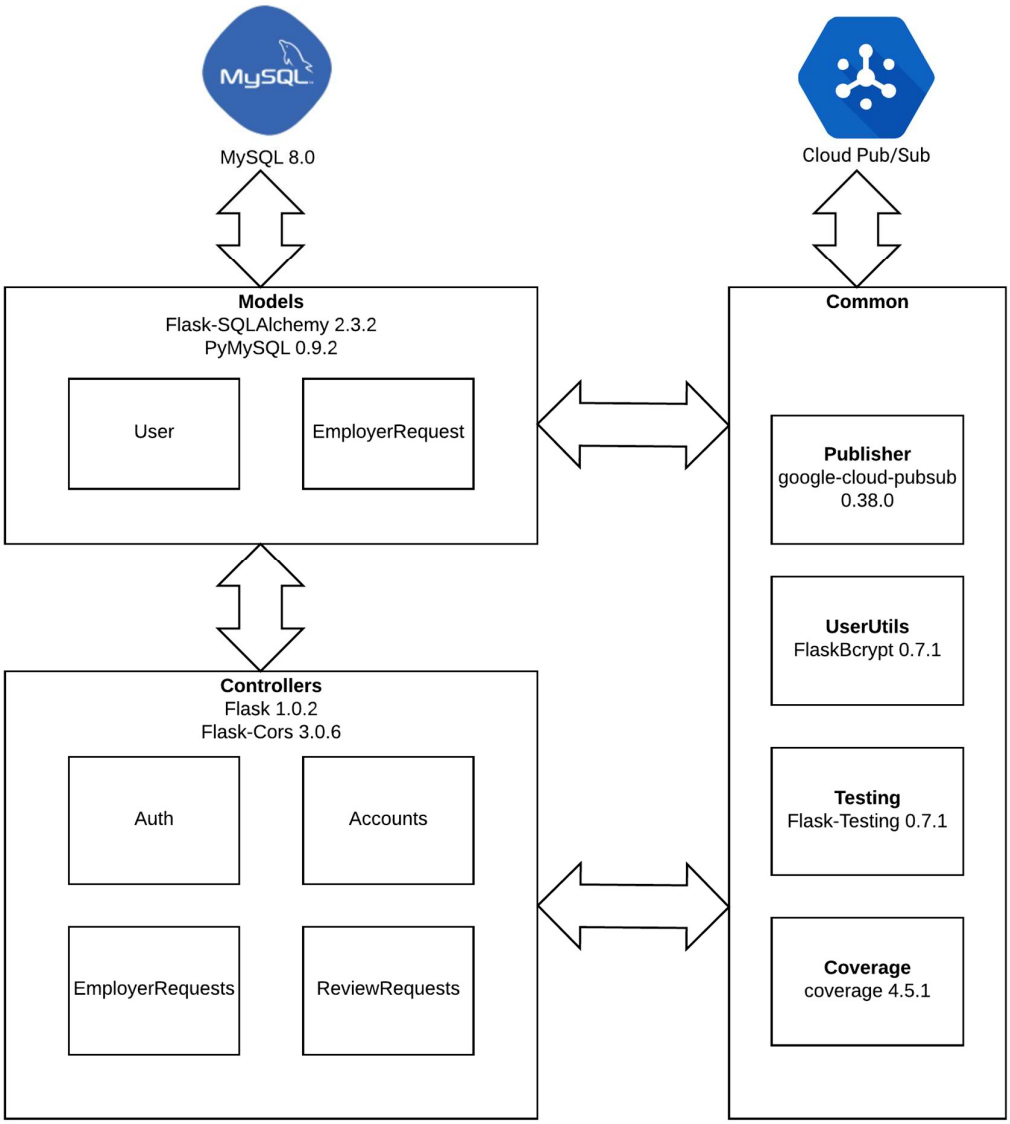


Figure 4-14: Auth Service Layer Design

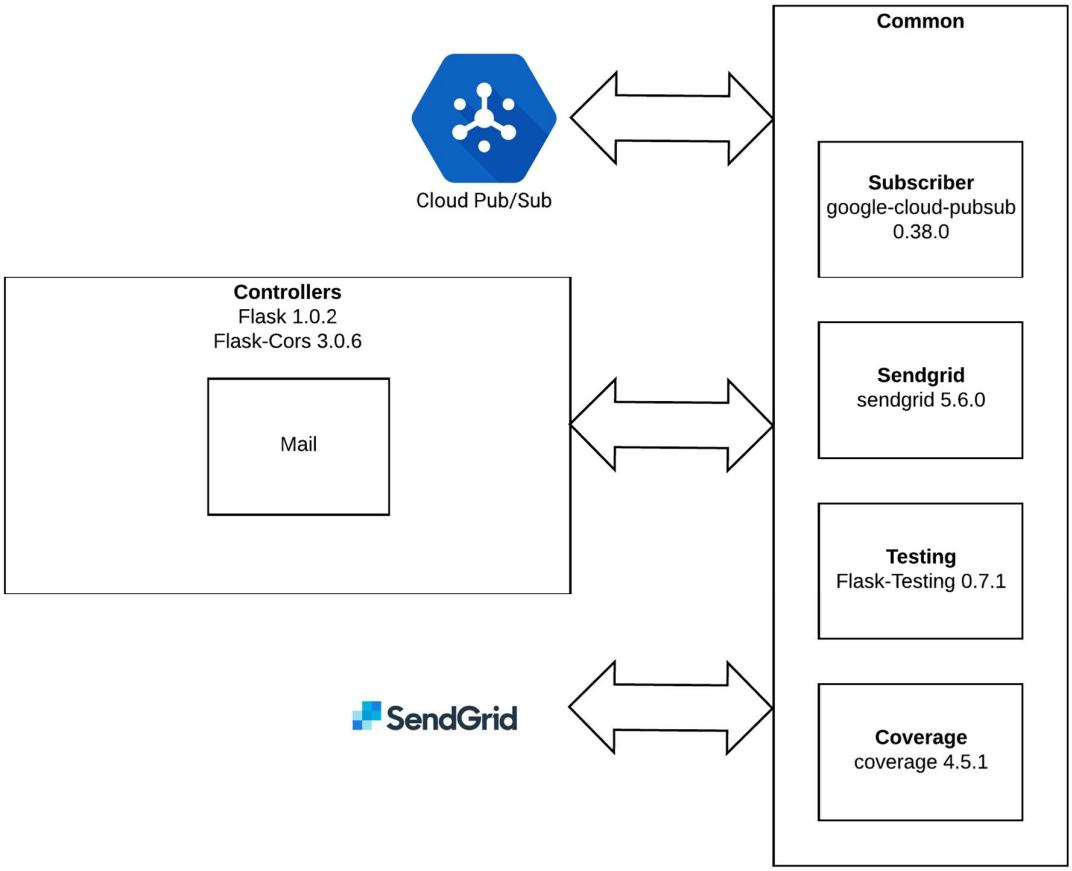


Figure 4-15: Mail Service Layer Design

No	Directory/File	Description	Convention
1	.dockerignore	Contains files and folders that should be excluded during Docker image build.	
2	.editorconfig	Configs for automated format of line endings, tab size, etc.	
3	Dockerfile	Contains instructions for Docker to build and run image.	
4	entrypoint.sh	Entry point of the service execution	
5	manage.py	Root script to run cli commands.	
6	requirements.txt	Contains dependency libraries to install with pip.	
7	project/	Contains all source code of the service.	
8	project/api/	Contains source code for API controllers.	

9	project/scripts/	Contains source code for miscellaneous scripts such as worker or cron job.	
10	project/tests/	Contains unit tests of the service.	
11	project/__init__.py	Factory file to create app context.	
12	project/config.py	Contains code to create configuration variables for different runtime environment.	
13	env.*	Shell script to populate current shell with environment variables for local debugging.	env.*

4.3.1.2 Frontend Layer Design

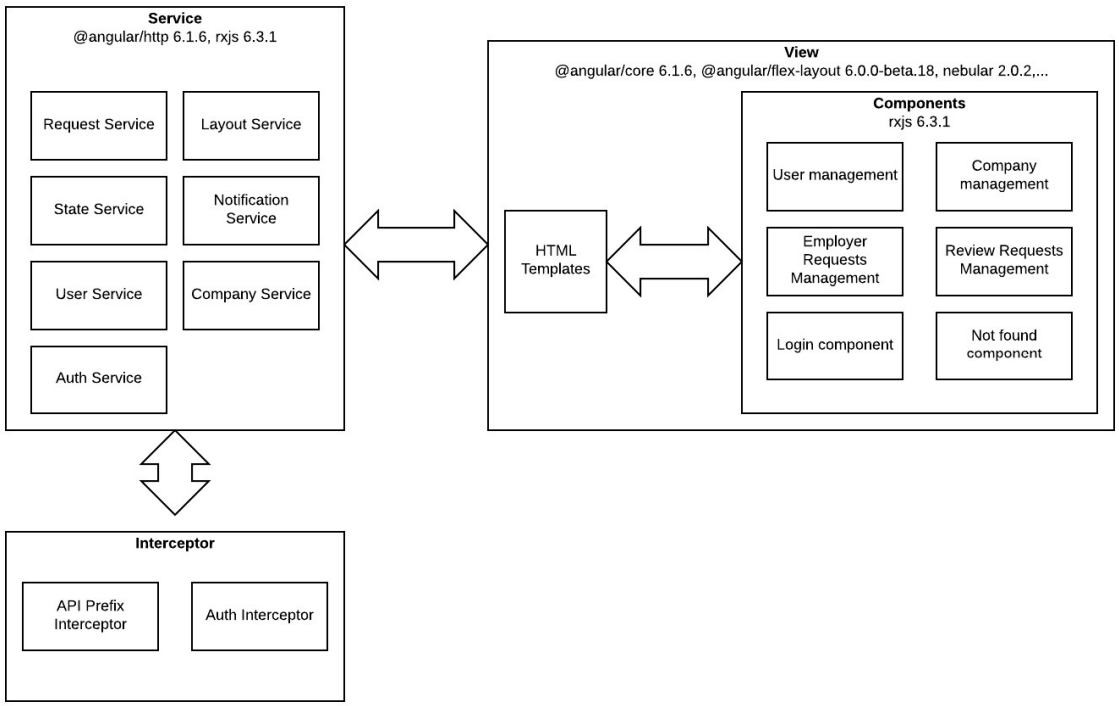


Figure 4-16: Admin Frontend Layer Design

No	Directory/File	Description	Convention
1	.editorconfig	Configs for automated format of line endings, tab size, etc.	
2	.gitignore	Contains files and folders to be excluded from being tracked by git.	
3	.stylelintrc.json	Contains rules for css/scss file style linting.	
4	angular.json	Angular project configurations	

5	karma.conf.js	Configurations for running unit tests.	
6	package.json	Contains dependencies to be installed using npm	
7	tsconfig.json	TypeScript configuration for running and building the project.	
8	tslint.json	Configurations for TypeScript file linting.	
9	src/	Contains all source files of the angular project.	
10	src/app/	Contains code to render the application UI.	
11	src/assets/	Contains asset files.	
12	src/environment	Contains environment variables for development or production runtime.	
13	src/index.html	Root container of all HTML elements.	
14	src/main.ts	Entry point of the application.	
15	src/polyfills.ts	Includes polyfills needed by Angular and is loaded before the app.	
16	src/test.ts	File required by karma.conf.js to load recursively all the .spec and framework files.	
17	src/*.routing.module.ts	Contains route definition for endpoints.	<component>-routing.module.ts
18	src/*.module.ts	Contains code to load all modules needed by a component.	<component>.module.ts
19	src/*.component.ts	Contains component class.	<component>.component.ts
20	src/*.service.ts	Contains service class used by components.	*.service.ts
21	src/*.spec.ts	Files for unit testing.	*.spec.ts
22	src/*.component.html	Contains HTML template for a component.	<component>.component.html
23	src/*.component.scss	Contains styles for a component.	<component>.component.scss
24	dist/	Contains results of project build process.	

4.3.2 Database Design

4.3.2.1 Explanation of database decision

4.3.2.1.1 Operational Database – MySQL

MySQL is our first choice for storing operational data because it has powerful features that suit our needs:

- Support storing schema-less data by providing data type as **JSON**.
- Reliability, Performance & Ease of Use - MySQL is proven as the world's most popular open source database.
- Database Development, Design and Administration - MySQL Workbench provides an integrated development, design and administration environment to make developers and DBAs more productive.

4.3.2.1.2 Elasticsearch - NoSQL

4.3.2.1.2.1 What is Elasticsearch

Elasticsearch is a search engine based on Lucene[7]. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. Elasticsearch is considered to be a NoSQL Database according to this article https://www.elastic.co/blog/found-elasticsearch-as-nosql.

4.3.2.1.2.2 The problems with RDBMS

When searching occured, we have to query all details & data of an entity including:

- Job: title, location, date added, type, length of contract, description
- Company data: name, website, size, type, etc.

Retrieving all the above data requires **joining tables** in a query, hence it is **inefficient in performance**.

4.3.2.1.2.3 How is Elasticsearch helping

We use Elasticsearch on top of MySQL to store full job and company information. This approach not only eliminates above problems but also brings us many benefits:

- **Speed:** Elasticsearch is able to execute complex queries extremely fast
- **Scalability**: can easily scale horizontally, providing the ability to extend resources

[7] https://en.wikipedia.org/wiki/Lucene

- **Lots of search options**: Searching is now supported by many features of Elasticsearch such as:
    - Typo correction by fuzzy query[8]
    - Powerful full-text search
    - Search-as-you-type autocomplete suggestion

4.3.2.1.2.4 Database synchronization

We have set up Elasticsearch, but it has no data to search since all data are saved in MySQL. So, one of the main concern is the synchronization of data from MySQL to Elasticsearch periodically. Any INSERT, UPDATE or DELETE event have to be replicated to Elasticsearch, which will be discussed more detail in **section 4.3.3.4** below**.**

4.3.2.2 Database Diagram

4.3.2.2.1 MySQL Database Diagram

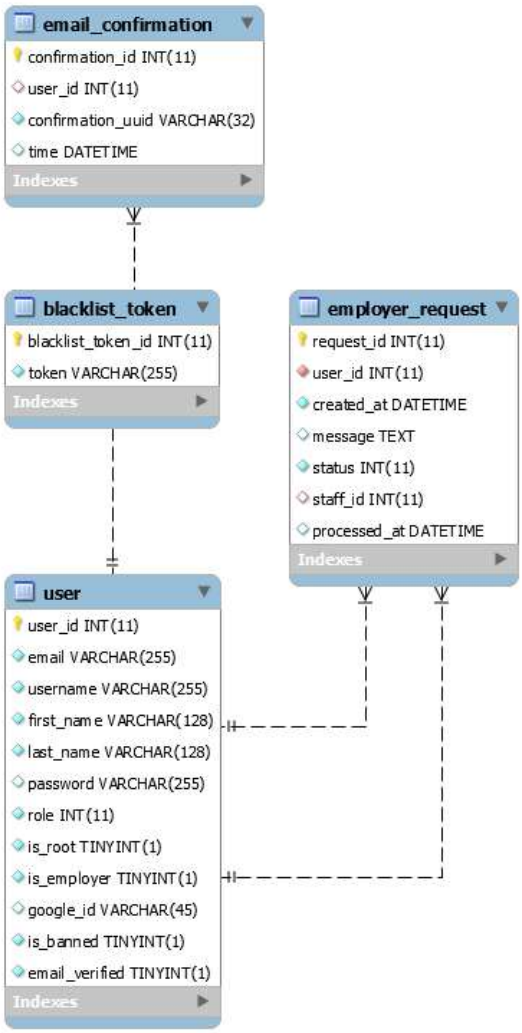


Figure 4-17: Auth Database Design

[8] https://www.elastic.co/guide/en/elasticsearch/guide/current/fuzzy-query.html

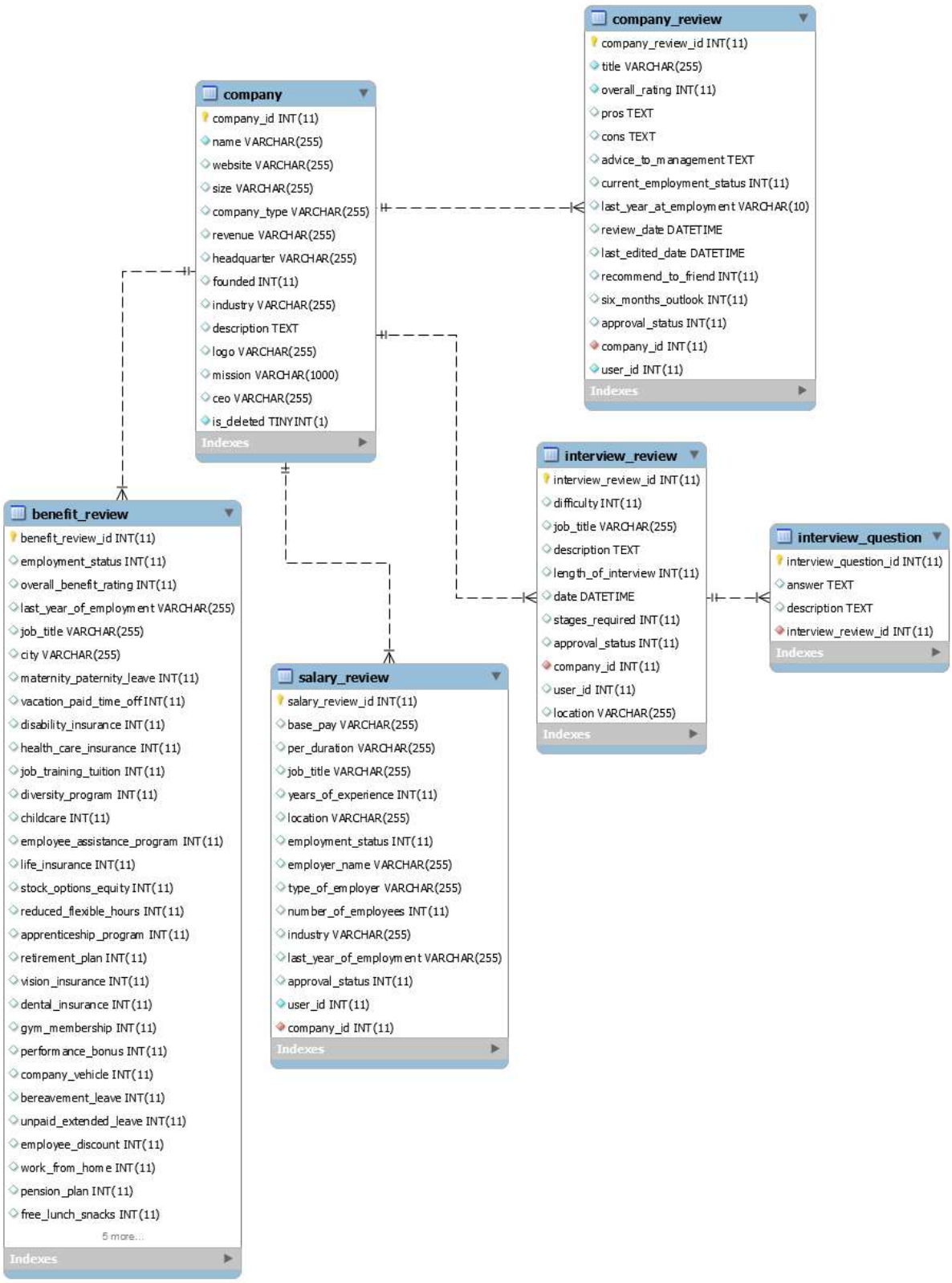


Figure 4-18: Company Database Design

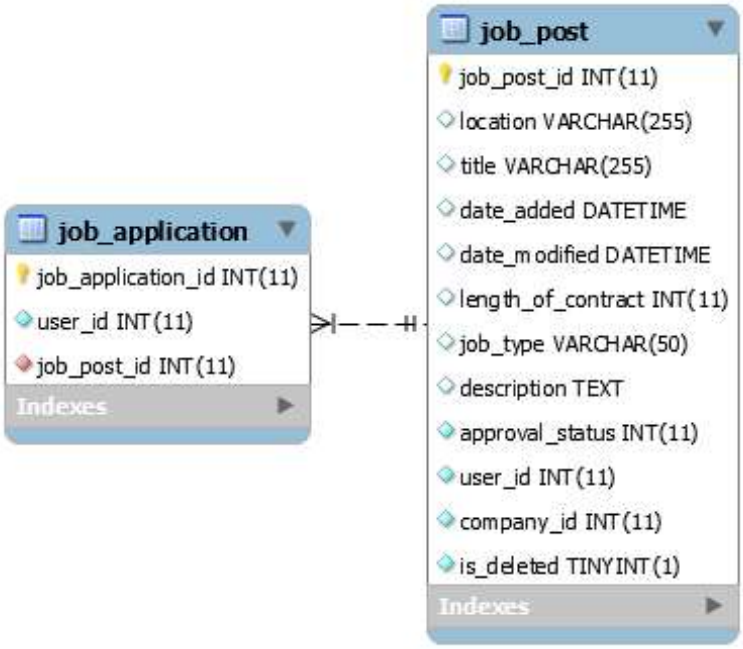


Figure 4-19: Job Database Design

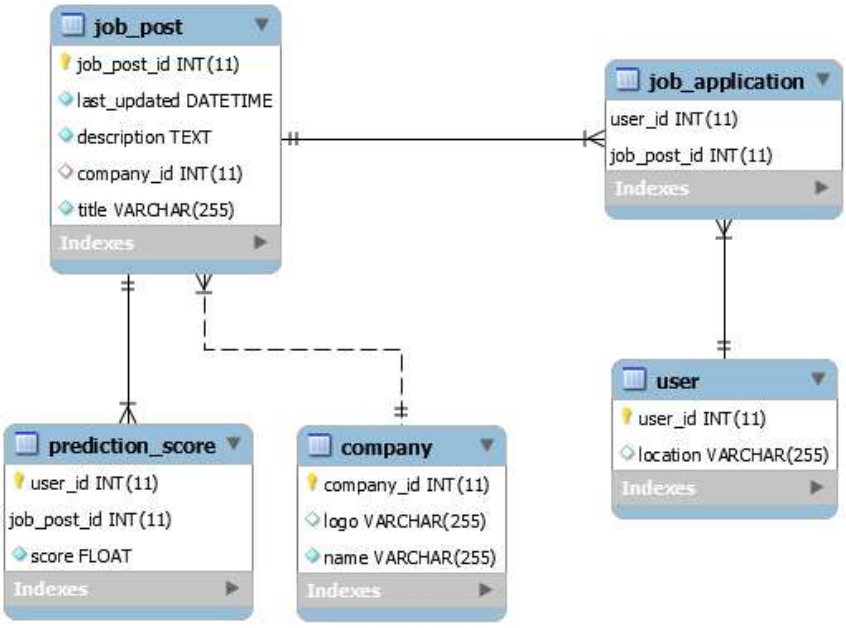


Figure 4-20: Recommendation Database Design

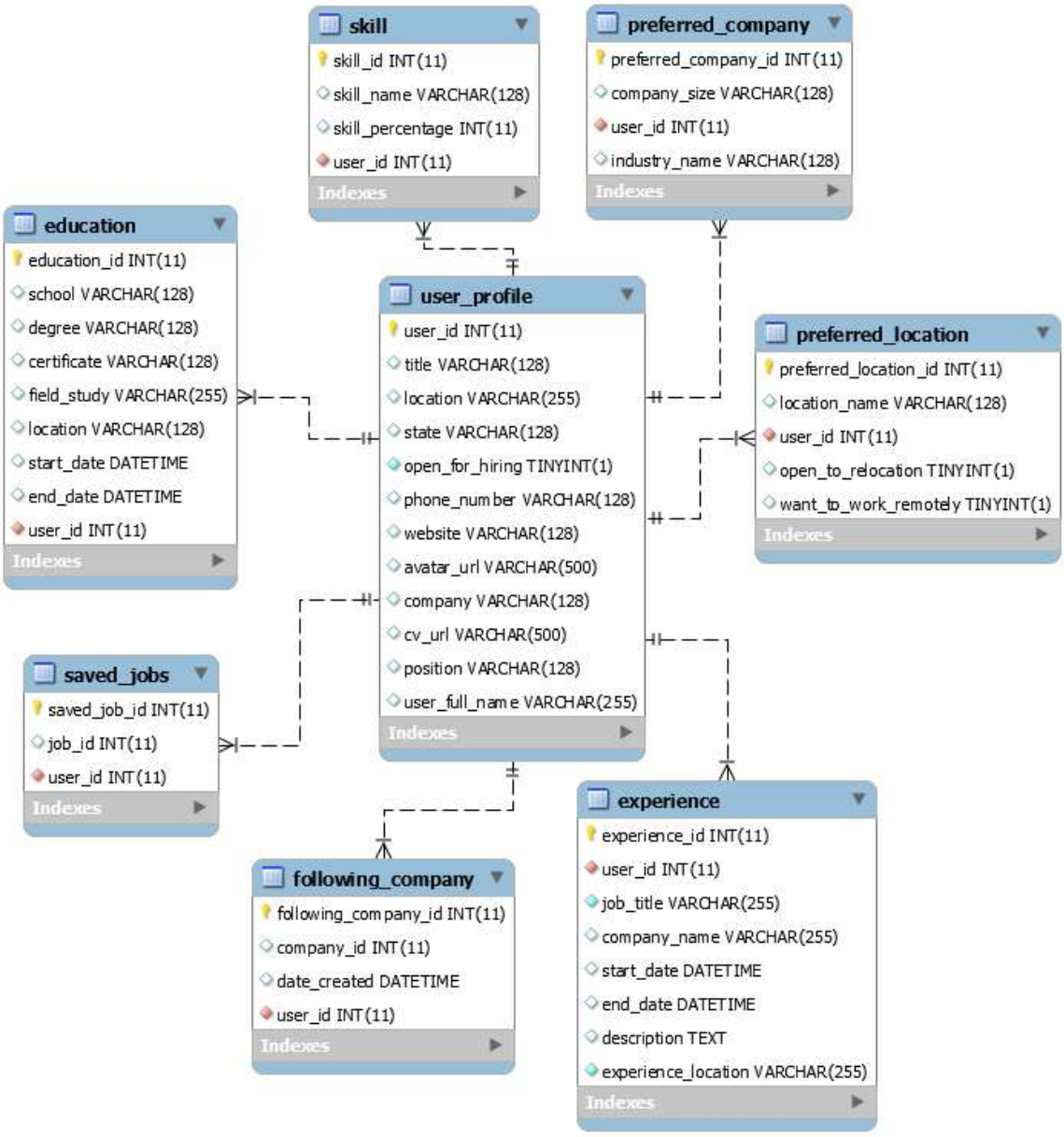


Figure 4-21: User Database Design

4.3.2.3 Data Dictionary

4.3.2.3.1 GlassCV – Operation Database

4.3.2.3.1.1 Entities

No	Entity Name	Description	Alias	Occurrence
1	user	General term describing a user that is registered		Each user has one user_profile Each user has one email_confirmation

				Each user has one employer_request Each user has many company_review Each user has many interview_review Each user has many benefit_review Each user many job_applications Each user has many job_post
2	blacklist_token	General term describing a user access token that should no longer be used		
3	email_confirmation	General term describing an email confirmation request for a registering user		Each email_confirmation belongs to one user
4	employer_request	General term describing an Employer account request from a user		Each employer_request belongs to one user
5	user_profile	General term describing a user profile		Each user_profile belongs to one user Each user_profile has many educations Each user_profile has many skills Each user_profile has many experience Each user_profile has many following_companies Each user_profile has many saved_jobs Each user_profile has many preferred_companies Each user_profile has many preferred_locations
6	education	General term describing a user's education history on his/her profile		Each education belongs to one user_profile

7	skill	General term describing a user's set of skills on his/her profile		Each skill belongs to one user_profile
8	following_company	General term describing the companies that a user is following		Each following_company belongs to one user_profile Each following_company belongs to one company
9	saved_jobs	General term describing job posts that a user has saved		Each saved_jobs belong to one user_profile Each saved_jobs belongs to one job_post
10	preferred_location	General term describing a user's preferred work locations		Each preferred_location belongs to one user_profile
11	preferred_company	General term describing a user's preferred company		Each preferred_company belongs to one user_profile
12	experience	General term describing a user's work experience		Each education belongs to one user_profile
13	company	General term describing a company		Each company has many company_review Each company has many interview_review Each company has many benefit_review Each company has many one following_company Each company has many job_post
14	company_review	General term describing a review of a company		Each company_review belongs to one user Each company_review belongs to one company
15	benefit_review	General term describing a review of the benefits of a company		Each benefit_review belongs to one user Each benefit_review belongs to one company
16	interview_review	General term describing a review of a company's interviews		Each interview_review belongs to one user Each interview_review belongs to one company

				Each interview_review has many interview_question
17	interview_question	General term describing the question(s) of an interview review		Each interview_question belongs to one interview_review
18	job_post	General term describing a job recruitment post		Each job_post belongs to one user Each job_post has many job_application
19	job_application	General term describing applications for a job post via the GlassCV's quick-apply feature		Each job_application belongs to one user Each job_application belongs to one job_post

4.3.2.3.1.2 Attributes

No	Entities Name	Field Name	Data Type & Length	Description	Not Null
1	user	user_id	integer		Yes
		first_name	character varying (45)		Yes
		last_name	character varying (45)		Yes
		email	character varying (45)		Yes
		password	character varying (45)		Yes
		role	tinyint	User role	Yes
		is_employer	boolean	Whether user is an employer	Yes
		google_id	character varying (45)		No
		email_verified	boolean	Whether user's email is verified	Yes

		is_root	boolean	Whether user is a root user	Yes
		is_banned	boolean	Whether user is banned	Yes
2	email_confirmation	email_confirmation_id	integer		Yes
		confirmation_uuid	character varying (32)	Custom confirmation UUID string	Yes
		time	datetime		Yes
		user_id	integer		Yes
3	employer_request	employer_request_id	integer		Yes
		created_at	datetime		Yes
		message	text		No
		status	integer		Yes
		processed_at	datetime		No
		user_id	integer		Yes
		staff_id	integer		No
4	user_profile	user_profile_id	integer		Yes
		title	character varying (255)		Yes
		location	character varying (255)		No
		state	character varying (255)		No
		open_for_hiring	boolean	Whether user is open for hiring at the moment	Yes
		user_full_name	character varying (255)		Yes
		phone_number	character varying (255)		No

		position	character varying (255)	Current job position (if applicable)	No
		company	character varying (255)	Company name (if applicable)	No
		website	character varying (255)	Website URL	No
		avatar_url	character varying (255)		No
		cv_url	character varying (255)		No
5	education	education_id	integer		Yes
		school	character varying (255)	School user went to	Yes
		degree	character varying (255)		Yes
		certificate	character varying (255)		No
		field_study	character varying (255)		No
		location	character varying (255)		Yes
		start_date	date		Yes
		end_date	date		Yes
		user_id	integer		Yes
6	experience	experience_id	integer		Yes
		job_title	character varying (255)		Yes

		company_name	character varying (255)		Yes
		company_location	timestamp with time zone		Yes
		start_date	date		Yes
		end_date	date		Yes
		description	text		Yes
		user_id	integer		Yes
7	skill	skill_id	integer		Yes
		skill_name	character varying (255)	Skill name	Yes
		skill_percentage	integer	Level of expertise	Yes
		user_id	integer		Yes
8	preferred_location	preferred_location_id	integer		Yes
		location_name	character varying (255)		Yes
		open_to_relocation	boolean	Whether user is open to relocation	Yes
		want_to_work_remotely	boolean	Whether user wants to work remotely	Yes
		user_id	integer		Yes
9	preferred_company	preferred_company_id	integer		Yes
		company_name	timestamp with time zone		Yes
		company_size	timestamp with time zone		Yes
		user_id	integer		Yes
10	following_company	following_company_id	integer		Yes

		date_created	datetime		Yes
		company_id	integer		Yes
		user_id	integer		Yes
11	saved_jobs	saved_job_id	integer		Yes
		job_post_id	integer		Yes
		user_id	integer		No
12	company	company_id	integer		Yes
		name	character varying (255)		Yes
		website	character varying (255)		Yes
		size	int	Number of employees	Yes
		company_type	character varying (255)	Company type, such as private, public, limited	Yes
		revenue	int	Revenue per annum	Yes
		headquarters	character varying (255)	Location of headquarters	Yes
		founded	date	When company was founded	Yes
		industry	character varying (255)		Yes
		description	text		Yes
		logo	character varying (255)	Logo URL	Yes
		mission	character varying (255)		Yes
		ceo	character varying (255)	Name of company CEO	Yes

		is_deleted	boolean		Yes
13	company_review	company_review_id	integer		Yes
		title	character varying (255)	Email of User	Yes
		overall_rating	integer	Overall rating over 5	Yes
		pros	text	Pros of working in said company	Yes
		cons	text	Cons of working in said company	Yes
		advice_to_management	text		No
		current_employment_status	boolean		No
		last_year_at_employment	integer		No
		review_date	datetime		Yes
		last_edited_date	datetime		No
		recommend_to_friend	integer		No
		approval_status	integer		Yes
		company_id	integer		Yes
		user_id	integer		Yes
14	benefit_review	benefit_review_id	integer		Yes
		employment_status	tinyint		Yes
		overall_benefit_rating	integer		Yes
		last_year_at_employment	character varying (45)		No
		job_title	character varying (45)		Yes
		city	character varying (45)		Yes
		maternity_paternity_leave	integer		No
		vacation_paid_time_off	integer		No

		disability_insurance	integer		No
		healthcare_insurance	integer		No
		job_training_tuition	integer		No
		diversity_program	integer		No
		childcare	integer		No
		employee_assistance_program	integer		No
		life_insurance	integer		No
		stock_options_equity	integer		No
		reduced_flexible_hours	integer		No
		apprenticeship_program	integer		No
		retirement_plan	integer		No
		vision_insurance	integer		No
		dental_insurance	integer		No
		gym_membership	integer		No
		performance_bonus	integer		No
		company_vehicle	integer		No
		bereavement_leave	integer		No
		unpaid_extended_leave	integer		No
		employee_discount	integer		No
		work_from_home	integer		No
		pension_plan	integer		No
		free_lunch_snacks	integer		No
		sick_leave	integer		No
		approval_status	integer		No
		company_id	integer		Yes
		user_id	integer		Yes
15	interview_review	interview_review_id	integer		Yes
		difficulty	integer		Yes

		job_title	character varying (45)		Yes
		description	text		Yes
		length_of_interview	integer	Length of interview (minutes)	Yes
		date	date		Yes
		stages_required	tinyint	Number of interview stages	No
		approval_status	tinyint		Yes
		company_id	integer		Yes
		user_id	integer		Yes
16	interview_question	interview_question_id	integer		Yes
		answer	text	Answer to question	No
		description	text	Description of question	No
		interview_review_id	integer		Yes
17	job_post	job_post_id	integer		Yes
		location	character varying (255)		Yes
		title	character varying (255)		Yes
		length_of_contract	integer	Length of contract (in months)	Yes
		job_type	integer	etc. on-site, off-site	Yes
		description	text		Yes
		approval_status	integer		Yes
		date_added	datetime		Yes
		date_modified	datetime		No
		is_deleted	boolean		Yes

		user_id	integer		Yes
		company_id	integer		Yes
18	job_application	job_application_id	integer		Yes
		job_post_id	integer		Yes
		user_id	integer		Yes

4.3.2.3.1.3 Database Index and Data Constraint

No	Entities Name	Index name	Column	Note
1	user	pkey_user	user_id	PK
2	email_confirmation	pkey_email_confirmation	email_confirmation_id	PK
		fk_email_confirmation_user1	(user_id) ref user (user_id)	FK
3	employer_request	pkey_employer_request	id	PK
		fk_email_confirmation_user1	(user_id) ref user (user_id)	FK
		fk_email_confirmation_user2	(staff_id) ref user (user_id)	FK
4	user_profile	pkey_user_profile	user_profile_id	PK
		fk_user_profile_user	(user_id) ref user (user_id)	FK
5	education	pkey_education	education_id	PK
		fk_education_user_profile1	(user_id) ref user_profile (user_id)	FK
6	skill	pkey_skill	skill_id	PK
		fk_skill_user_profile1	(user_id) ref user_profile (user_id)	FK
7	experience	pkey_experience	experience_id	PK
		fk_experience_user_profile1	(user_id) ref user_profile (user_id)	FK
8	preferred_location	pkey_preferred_location	preferred_location_id	PK
		fk_preferred_location_user_profile1	(user_id) ref user_profile (user_id)	FK
9		pkey_preferred_company	preferred_company_id	PK

	preferred_company	fk_preferred_company_user_profile1	(user_id) ref user_profile (user_id)	FK
10	saved_jobs	pkey_saved_jobs	saved_jobs_id	PK
		fk_saved_jobs_user_profile1	(user_id) ref user_profile (user_id)	FK
		fk_saved_jobs_job_post1	(job_post_id) ref job_post (job_post_id)	FK
11	following_company	pkey_following_company	following_company_id	PK
		fk_following_company_user_profile1	(user_id) ref user_profile (user_id)	FK
		fk_following_company_company1	(company_id) ref company (company_id)	FK
12	company	pkey_company	company_id	PK
13	company_review	pkey_company_review	company_review_id	PK
		fk_company_review_company1	(company_id) ref company (company_id)	FK
		fk_company_review_user1	(user_id) ref user (user_id)	FK
14	benefit_review	pkey_benefit_review	benefit_review_id	PK
		fk_company_review_company1	(company_id) ref company (company_id)	FK
		fk_company_review_user1	(user_id) ref user (user_id)	FK
15	interview_review	pkey_interview_review	interview_review_id	PK
		fk_company_review_company1	(company_id) ref company (company_id)	FK
		fk_company_review_user1	(user_id) ref user (user_id)	FK
16	interview_question	pkey_interview_question	interview_question_id	PK
		fk_interview_question_interview_review1	(interview_review_id) ref interview_review (interview_review_id)	FK
17	job_post	pkey_job_post	job_post_id	PK
		fk_job_post_user1	(user_id) ref user (user_id)	FK

		fk_job_post_company1	(company_id) ref company (company_id)	FK
18	job_application	pkey_job_application	job_application_id	PK
		fk_job_application_job_post1	(job_post_id) ref job_post (job_post_id)	FK
		fk_job_application_user1	(user_id) ref user (user_id)	FK

4.3.2.3.2 Elasticsearch database

4.3.2.3.2.1 Job index mapping

No	Field	Config value	Explanation
1	title	type: text fields: edgengram: type: text analyzer: edge_ngram_analyzer search_analyzer: edge_ngram_search_analyzer	Title for the job post. Edge Ngram analyzer is included for full-text searches and autocompletes
2	location	type: text fields: edgengram: type: text analyzer: edge_ngram_analyzer search_analyzer: edge_ngram_search_analyzer	Location of the job post. Edge Ngram analyzer is included for full-text searches and autocompletes
3	Is_deleted	type: boolean	Since JDBC plugin cannot catch MySQL's DELETE events, we will use is_deleted flag for filtering

4.3.2.3.2.2 Company index mapping

No	Field	Config value	Explanation

1	name	type: text fields: edgengram: type: text analyzer: edge_ngram_analyzer search_analyzer: edge_ngram_search_analyzer	Name of the company. Edge Ngram analyzer is included for full-text searches and autocompletes
2	location	type: text fields: edgengram: type: text analyzer: edge_ngram_analyzer search_analyzer: edge_ngram_search_analyzer	Location of the company. Edge Ngram analyzer is included for full-text searches and autocompletes
3	Is_deleted	type: boolean	Since JDBC plugin cannot catch MySQL's DELETE events, we will use is_deleted flag for filtering

4.3.3 Common Design

4.3.3.1 Microservices architecture

4.3.3.1.1 What is microservices?

Microservices is a software architecture style in which complex applications are composed of small, independent processes communicating with each other using language-agnostic APIs. These services are small, highly decoupled and focus on doing a small task, facilitating a modular approach to system-building.

4.3.3.1.2 Philosophy

- Philosophy of Microservices architecture essentially equals the Unix philosophy of "Do one thing and do it well". It is described as follows
- The services are small - fine-grained to perform a single function.
- The organization culture should embrace automation of deployment and testing. This eases the burden on management and operations.
- The culture and design principles should embrace failure and faults, similar to anti-fragile systems.
- Each service is elastic, resilient, composable, minimal, and complete

4.3.3.1.3 Microservices in GlassCV

In GlassCV, we implement Microservices as the following things:

- The overall system is divided into small microservices according to their domain of operation.
- Deployment of services is done by using Kubernetes.
- Continuous delivery using Travis CI.
- Use third-party services:
    - SendGrid for sending mail.
    - Google Cloud Storage for storing images

4.3.3.2 Recommender system

4.3.3.2.1 What is a recommender system?

A recommender system or a recommendation system (sometimes replacing "system" with a synonym such as platform or engine) is a subclass of "information filtering system" that seeks to predict the "rating" or "preference" a user would give to an item.[9]

4.3.3.2.2 Types of recommender system

There are three basic types of recommender system:

- **Content-based filtering**: Recommendations are made based on a comparison between the content of the items and a user profile. Systems of this type try to recommend items which are similar to those a user has interacted positively in the past such as reading an article, purchasing a product or applying to a job, etc.
- **Collaborative filtering**: Recommendations are calculated based on the similarities among users' behaviors. If two people both like several items, it can be determined that they are somehow similar in taste. Therefore, it would be of high confidence that apart from the several items in common mentioned before, if we recommend an item that the first person like, the other person is likely to also like it.
- **Hybrid** recommender system: Combines both content-based filtering and collaborative filtering.

4.3.3.2.3 Recommendation in GlassCV

4.3.3.2.3.1 Algorithms

Content-based filtering and collaborative filtering each has both pros and cons. Content-based depends purely on metadata, so recommendations are limited to the taste of a single user only, i.e. it does not benefit from data from other users. Eventually, as there are more user interactions every day, collaborative filtering comes to beat content-based filtering. However, collaborative filtering requires a large amount of data, rendering it performing poorly for user cold-start.

GlassCV goes for the **hybrid approach** of building recommendations to address the above problem. In this way, we can make use of both job properties like title or description, as well as user behaviors. This is especially effective in the cold-start scenario where the number of user interactions are scarce.

[9] Francesco Ricci and Lior Rokach and Bracha Shapira, Introduction to Recommender Systems Handbook, Recommender Systems Handbook, Springer, 2011, pp. 1-35

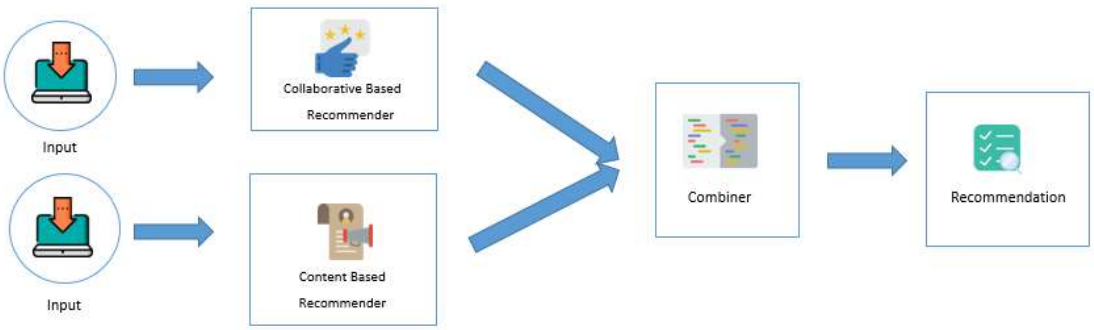


Figure 4-22: GlassCV Hybrid Recommender

For implementation, GlassCV uses the LightFM[10] library developed by Maciej Kula, which is based on **matrix factorization**. Matrix factorization starts with a matrix representing users' interactions with each item and then, as the name already suggests, decomposes it into user and item feature matrices.

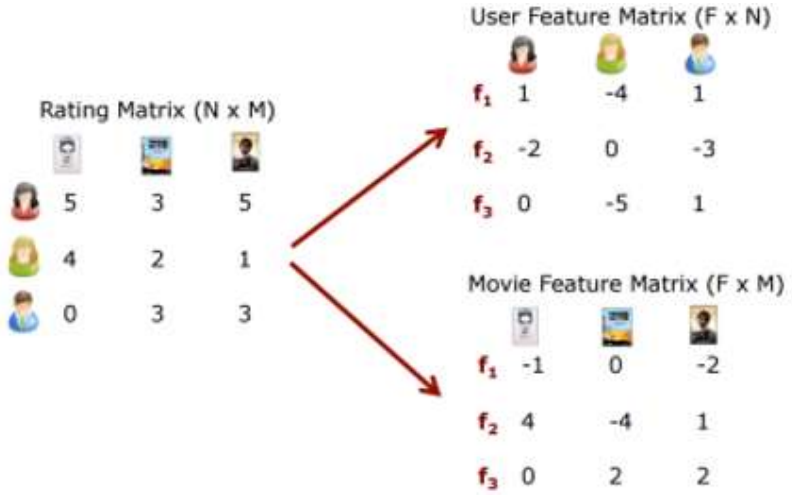


Figure 4-23: Matrix Factorization[11]

In order to make the content-based filtering part possible with matrix factorization, LightFM represents items and users as linear combinations of their content features. Let's assume that each content attribute a has its own latent vector s_a. If x_i is the original user vector, N(i) represents the set of attributes of item i, then the total item vector is

$$x_i + \sum_{a \in N(i)} s_a$$

The same idea holds for user vectors.

[10] LightFM, https://github.com/lyst/lightfm
[11] Solving Business Usecases by Recommender System Using LightFM, https://towardsdatascience.com/solving-business-usecases-by-recommender-system-using-lightfm-4ba7b3ac8e62

4.3.3.2.3.2 Data preparation

To start building our model, we Wuzzuf[12]'s "Wuzzuf Job Post" dataset published on Kaggle. This data is accessible from the URL https://www.kaggle.com/WUZZUF/wuzzuf-job-posts.

The dataset contains two CSV files, with the following corresponding columns:

File	Column	Description
Wuzzuf_Application_Sample.csv (1854190 entries)	id	Application identifier
	user_id	Applicant ID
	job_id	Job ID
	app_date	Date the application was recorded
Wuzzuf_Job_Posts_Sample.csv (21850 entries)	id	Job post identifier
	city_name	City of the job
	job_title	Title of the job
	job_category_1	Categories of the job
	job_category_2	
	job_category_3	
	job_industry_1	Industries of the job
	job_industry_2	
	job_industry_3	
	salary_minimum	Salary limits
	salary_maximum	
	num_vacancies	Number of vacancies for the job
	experience_years	Number of years of experience
	post_date	Date the job was published
	views	View count for the job
	job_description	Detaled description for the job
	job_requirements	Requirements for the job
	payment_period	Salary payment interval

[12] Wuzzuf is a technology firm founded in 2009 and one of the very few companies in the MENA region specialized in developing Innovative Online Recruitment Solutions for top enterprises and organizations

	currency	Salary currency

Table 4-1: Wuzzuf Job Post dataset structure

In the scope of this project, we only use job title and job description (combination of job_description and job_requirements fields). A quick look at the data shows that job titles are quire clean already.

	job_title
0	Sales & Marketing Agent
1	German Training Coordinator
2	Junior Software Developer
3	Application Support Engineer
4	Electrical Maintenance Engineer
5	IT Adminstrator
6	e-payments System Administrator
7	PROCESS ENGINEER
8	Senior Software Engineer
9	OPERATION ENGINEER
10	Graphic Designer
11	Service & Maintenance Engineer

Figure 4-24: Dataset job titles

Job descriptions are in HTML, so preprocessing is necessary. Here we use MeaningCloud Topics Extraction API to extract an entity list and concept list from the text.

Result

Entry Type	Relevance	Form	Type
Entity	100	Information technology	Institute
Entity	50	IT service	ProfessionalService
Entity	50	data protection	LawRule
Entity	50	Knowledge of Multi-threading & Parallel	Top
Entity	50	SQL	Top
Entity	50	Ajax	SportsTeam

Figure 4-25: MeaningCloud Topics Extraction result

4.3.3.2.3.3 Training flow

4.3.3.2.3.3.1 Offline training

Initially, a training script is run to create a model based on all historical data up to the point of running the script. As training the whole model requires a substantial amount of time, this script is scheduled to run at the end of each day to reflect changes in user/item features[13]. Upon completion, user and item embeddings, along with biases are saved to files on disk for serving recommendation requests.

4.3.3.2.3.3.2 Online training

In addition to offline training, GlassCV also learns user activities on the run, making live recommendations possible. Whenever a user interacts with a particular item such as applying for a job or disliking a job, an event with corresponding changes is sent to the training service. These changes are then incorporated into the already trained model from offline training; new embeddings and biases are also saved to disk. As the process only involves new data that comes in small batches every time, it can be completed almost right away. The next time the user views recommended job posts, it is safe to assume that the recommendations are fresh and correctly reflect his/her previous actions.

4.3.3.2.3.4 Evaluation

A commonly used evaluation metric for recommendation problems is the area under the Receiver Operating Characteristic (ROC) curve. A higher value for the area under curve (AUC) means we are recommending items that a user would interact with near the top of his/her recommendation list. LightFM provides a handy **`lightfm.evaluation.auc_score`** method to calculate the AUC metric for a model.

We tested our model for up to 14 thousand jobs from the dataset, and the AUC score gets on the test set gets higher as we feed more data to the model.

[13] https://github.com/lyst/lightfm/issues/322

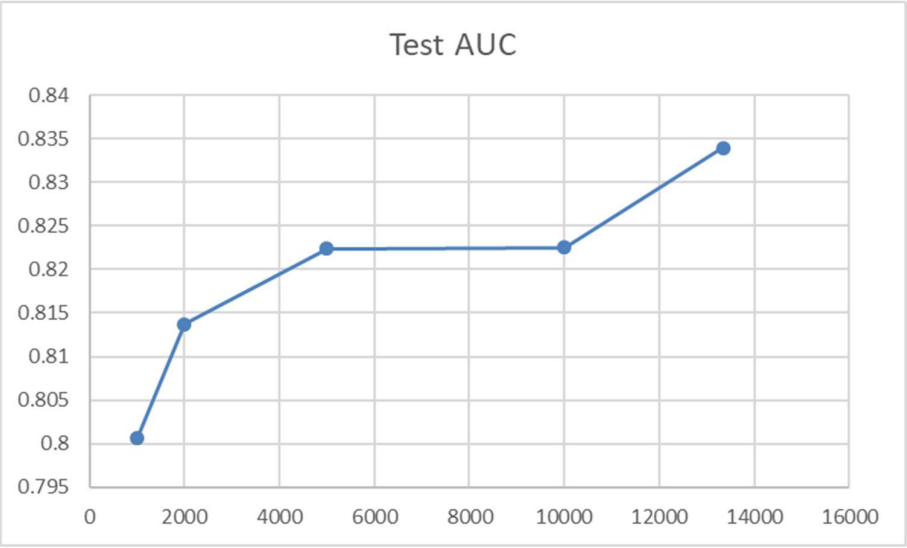


Figure 4-26: AUC score for test set by number of jobs

4.3.3.3 Authentication and authorization via API Gateway

As GlassCV is divided into small services, there needs to be a central point for request authentication and authorization. This is accomplished with the help of Ambassador, an open-source API gateway for Kubernetes.

Figure 4-27: Ambassador API Gateway

By using ambassador, every incoming request is leveraged to the authentication service for authentication and authorization before it is forwarded to the target services.

4.3.3.4 MySQL and Elasticsearch database synchronization

To take advantages of Elasticsearch, we have to synchronize GlassCV's MySQL database with Elasticsearch's NoSQL database periodically (every minute, in this case) using Logstash.

The diagram below describes the migration procedure:

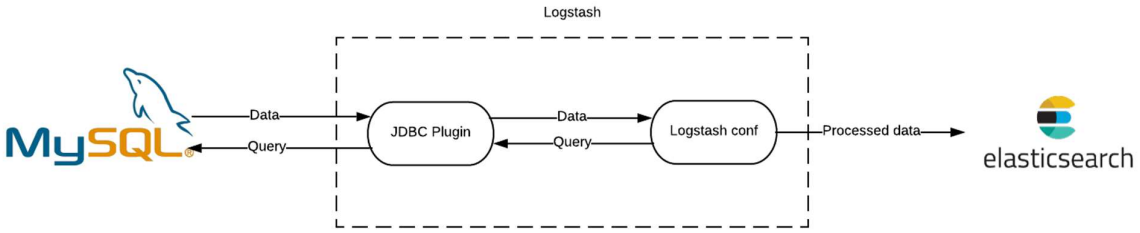


Figure 4-28: Database synchronization activity diagram

In the diagram we have Logstash running the configuration file (logstash.conf) which fires the predefined query we have set to JDBC plugin. JDBC plugin then passes it to MySQL and collects the data, which it will hand over to Logstash. We can then process the data and put it in the desired form, before indexing it to Elasticsearch.

```
input {
  jdbc {
    jdbc_driver_library => "/usr/share/logstash/mysql-connector-java-8.0.13.jar"
    jdbc_driver_class => "com.mysql.jdbc.Driver"
    jdbc_connection_string => "jdbc:mysql://db:3306/company"
    jdbc_user => "root"
    jdbc_password => "1234321"
    statement => "SELECT * FROM `company`"
    tags => "company"
    schedule => "* * * * *"
  }
  jdbc {
    jdbc_driver_library => "/usr/share/logstash/mysql-connector-java-8.0.13.jar"
    jdbc_driver_class => "com.mysql.jdbc.Driver"
    jdbc_connection_string => "jdbc:mysql://db:3306/job"
    jdbc_user => "root"
    jdbc_password => "1234321"
    statement => "SELECT * FROM `job_post`"
    tags => "job"
    schedule => "* * * * *"
  }
}
```

Figure 4-29: Configurations for collecting data from MySQL, using JDBC plugin. Description for each attribute is in the table below

No	Attribute name	Description
1	Jdbc_driver_library	Path to JDBC MySQL connector
2	Jdbc_connection_string	Details of JDBC connection, including database address, port and database name
3	Jdbc_user, jdbc_password	MySQL credentials
4	Statement	Predefined query

5	Tags	Add tag to help with processing later
6	Schedule	Scheduler for running the task. "* * * * *" = interval of 1 minute

```
output {
  if "company" in [tags] {
    elasticsearch {
    "hosts" => "elasticsearch:9200"
    "index" => "company"
    "document_type" => "details"
    "document_id" => "%{[company_id]}"
    }
  } else {
    elasticsearch {
      "hosts" => "elasticsearch:9200"
      "index" => "job"
      "document_type" => "details"
      "document_id" => "%{[job_post_id]}"
    }
  }
}
```

Figure 4-30: Configurations for indexing the data to Elasticsearch. Description for each attribute is in the table below

No	Attribute name	Description
1	Host, index, document_type	Elasticsearch host and index information
2	Document_id	Given as one of the unique column values for each row. This prevents the duplication of Elasticsearch documents.

4.3.3.5 Searching and auto-completion with Elasticsearch

The searching process of GlassCV actually does not involve querying MySQL database for job and company data. Instead, it will query through Elasticsearch for much better performance (as mentioned in 4.3.2)

In 4.3.3.4, we configured Logstash to parse MySQL data and send it to Elasticsearch. But we want to tune Elasticsearch to store that data in a more efficient way. Benefits including reducing memory requirements, especially for complex queries (Edge NGram, fuzzy), and reducing storage requirements. We do this with the help of mapping templates.

```
"template": "job*",
"version": 50001,
"settings": {
    "index.refresh_interval": "5s",
    "index": {
        "analysis": {
            "filter": {},
            "analyzer": {
                "edge_ngram_analyzer": {
                    "filter": [
                        "lowercase"
                    ],
                    "tokenizer": "edge_ngram_tokenizer"
                },
                "edge_ngram_search_analyzer": {
                    "tokenizer": "lowercase"
                }
            },
            "tokenizer": {
                "edge_ngram_tokenizer": {
                    "type": "edge_ngram",
                    "min_gram": 2,
                    "max_gram": 15,
                    "token_chars": [
                        "letter"
                    ]
                }
            }
        }
    }
},
```

Figure 4-31: Custom settings for Job index, where Edge NGram analyzer and tokenizer is specified

```
"title": {
    "type": "text",
    "fields": {
      "edgengram": {
          "type": "text",
          "analyzer": "edge_ngram_analyzer",
          "search_analyzer": "edge_ngram_search_analyzer"
       }
    },
    "analyzer": "standard"
},
```

*Figure 4-32: Custom mapping for field **title** of **Job** index, where we use a special type of mapping called edgengram to enable Edge NGram searching*

We will use following functions of Elasticsearch for searching/auto-completion:

Full-text search:

```
{
    "from": 0,
    "size": 10,
    "query": {
        "bool": {
            "must": [
                {
                    "match": {
                        "name.edgengram": {
                            "query": "compani",
                            "fuzziness": "1"
                        }
                    }
                },
                {
                    "match_phrase": {
                        "is_deleted": {
                            "query": false
                        }
                    }
                }
            ]
        }
    }
}
```

Figure 4-33: A sample full-text search request body

Query/Option	Value/Type	Explanation
match	Boolean	The text provided is analyzed and the analysis process constructs a boolean query from the provided text
fuzziness	*1*	Allow inexact fuzzy matching. Value is interpreted as a Levenshtein Edit Distance[14]. Mainly used to catch spelling errors.
bool	An object containing: *must/ should/ filter* query	A query that matches documents matching boolean combinations of other queries
must	An array of clauses (queries)	The clause (query) must appear in matching documents and will contribute to the score
fields	*"title.edgengram", "name.edgengram"*	The fields to be queried. In this case we use the edgegram field to enable matching query in the middle of the text.

[14] https://en.wikipedia.org/wiki/Levenshtein_distance

from	Integer	Defines the offset from the first result we want to fetch (page index). Used for paging.
size	Integer	Defines the maximum amount of hits to be returned (page size). Used for paging.

Auto-complete suggestion:

```
{
    "from": 0,
    "size": 10,
    "query": {
        "multi_match": {
            "query": "compan",
            "type": "phrase_prefix",
            "fields": [
                "title.edgengram",
                "name.edgengram"
            ]
        }
    }
}
```

Figure 4-34: A sample auto-complete request body

Query/Option	**Value/Type**	**Explanation**
multi_match	An object contains *multi_match* query attributes	Builds on the match query to allow multi-field queries
type	*phrase_prefix*	A type of multi_match query. Runs a match_phrase_prefix query on each field and combines the _score from each field
fields	*"title.edgengram", "name.edgengram"*	The fields to be queried. In this case we use the edgegram field to enable matching query in the middle of the text.

4.3.3.6 Using Object-Relational Mapping (ORM) with SQLAlchemy

SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL. In GlassCV, we use the Flask-SQLAlchemy package to use SQLAlchemy with Flask application architecture.

4.3.4 Detail Design

4.3.4.1 Guest signs in with credentials

Screen Design

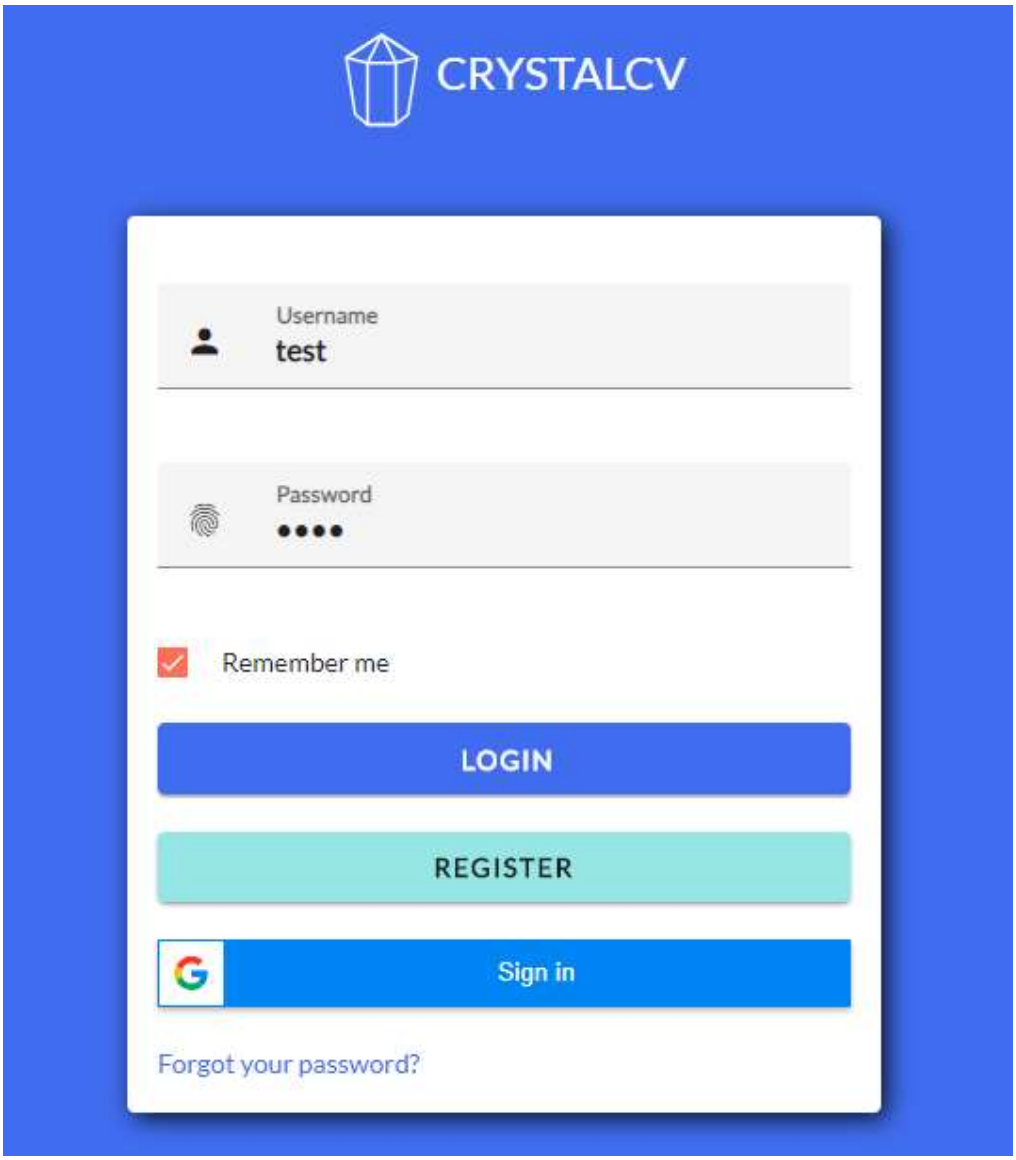


Figure 4-35: Guest signs in with credentials screen design

Class Diagram

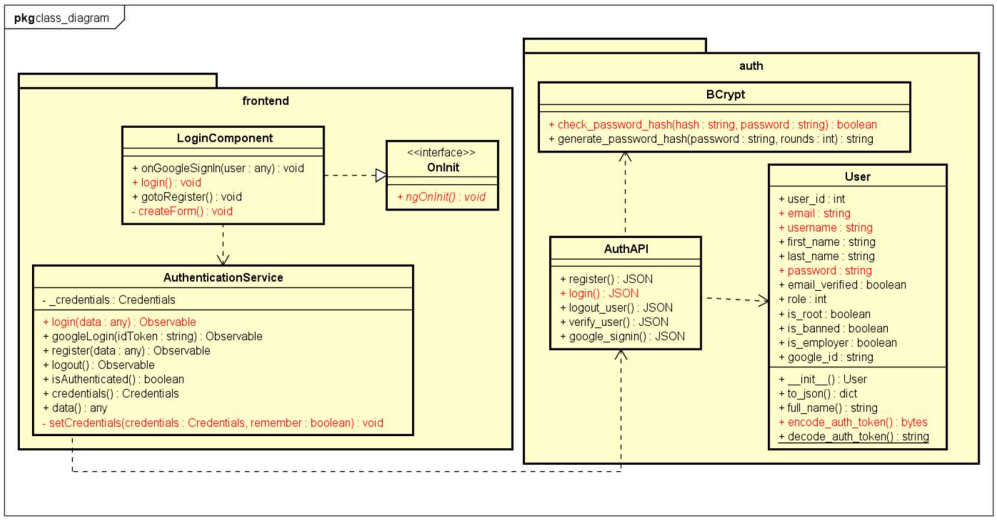


Figure 4-36: Guest signs in with credentials class diagram

Class Specification

LoginComponent

LoginComponent			
Physical address	/services/frontend/src/app/login/login.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
login	Log the user in		
Return Type	void		
Parameters	Name	Type	Description
createForm	Initialize form fields		
Return Type	void		
Parameters	Name	Type	Description
ngOnInit	Initialize component		
Return Type	void		
Parameters	Name	Type	Description

AuthenticationService

AuthenticationService			
Physical address	/services/frontend/src/app/core/authentication/authentication.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description

1	_credentials	Credentials	Credentials object, containing access token and username
Operation			
login	Send login request to the API		
Return Type	Observable		
Parameters	Name	Type	Description
	data	any	The login data containing username and password
setCredentials	Save the credentials to browser local storage		
Return Type	void		
Parameters	Name	Type	Description
	credentials	Credentials	The credentials to be saved

AuthAPI

AuthAPI			
Physical address	/services/auth/project/api/auth.py		
Base class	Blueprint		
Attributes			
No	Name	Type	Description
Operation			
login	Validate user credentials and return access token		
Return Type	JSON		
Parameters	Name	Type	Description

User

User			
Physical address	/services/auth/project/api/models.py		
Base class	Model		
Attributes			
No	Name	Type	Description
1	email	string	Email of the user
2	username	string	Username of the user
3	password	string	Hashed password of the user
Operation			
encode_auth_token	Create encoded access token from user information		
Return Type	bytes		
Parameters	Name	Type	Description

BCrypt

BCrypt			
Physical address	bcrypt		
Base class	Class		
Attributes			
No	Name	Type	Description

Operation			
check_password_hash	Check if the plain-text password matches the hashed password		
Return Type	boolean		
Parameters	Name	Type	Description
	hash	string	Hashed password
	password	string	Plain-text password

Sequence Diagram

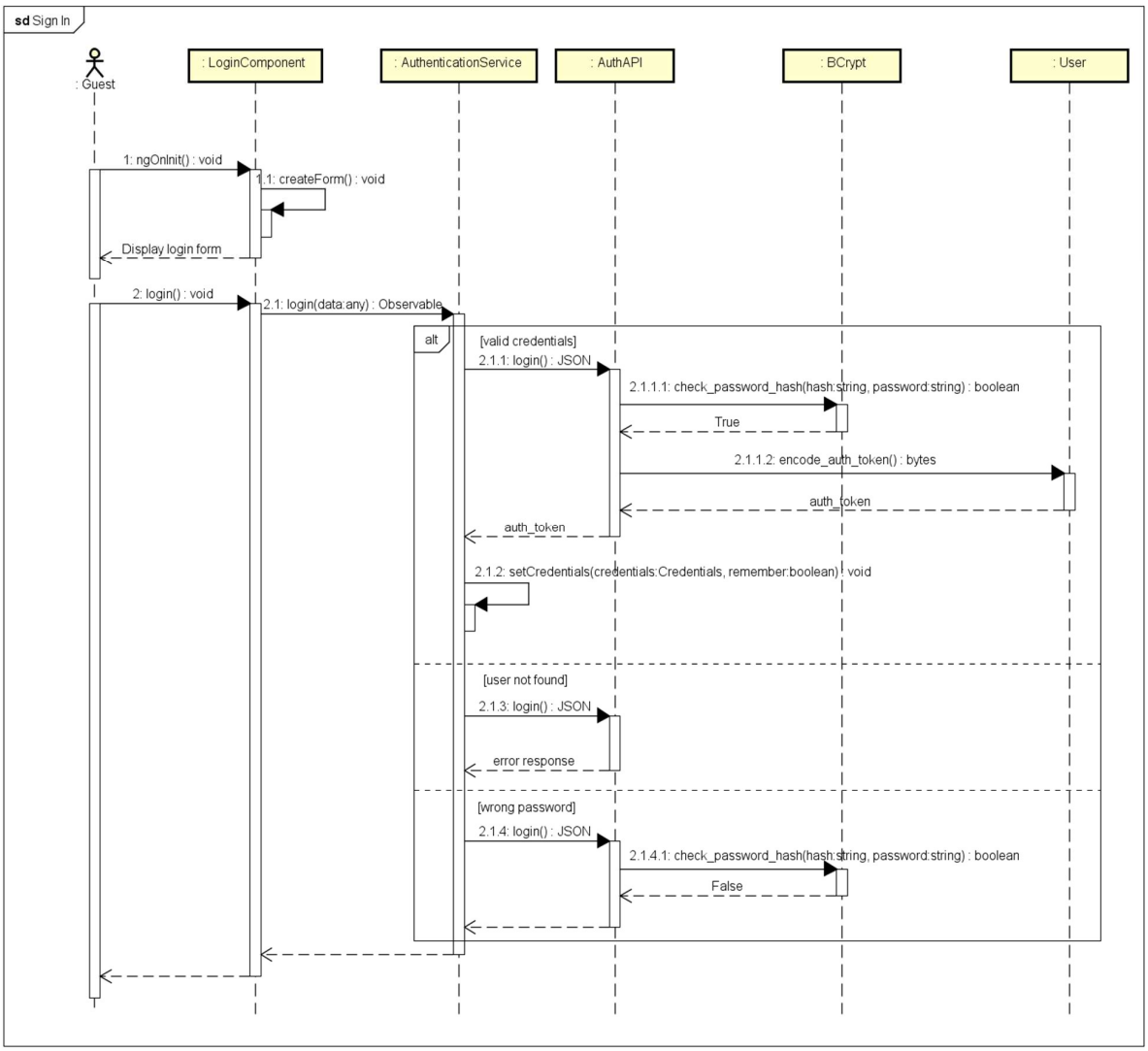


Figure 4-37: Guest signs in with credentials sequence diagram

4.3.4.2 Guest signs in with Google account

Screen Design

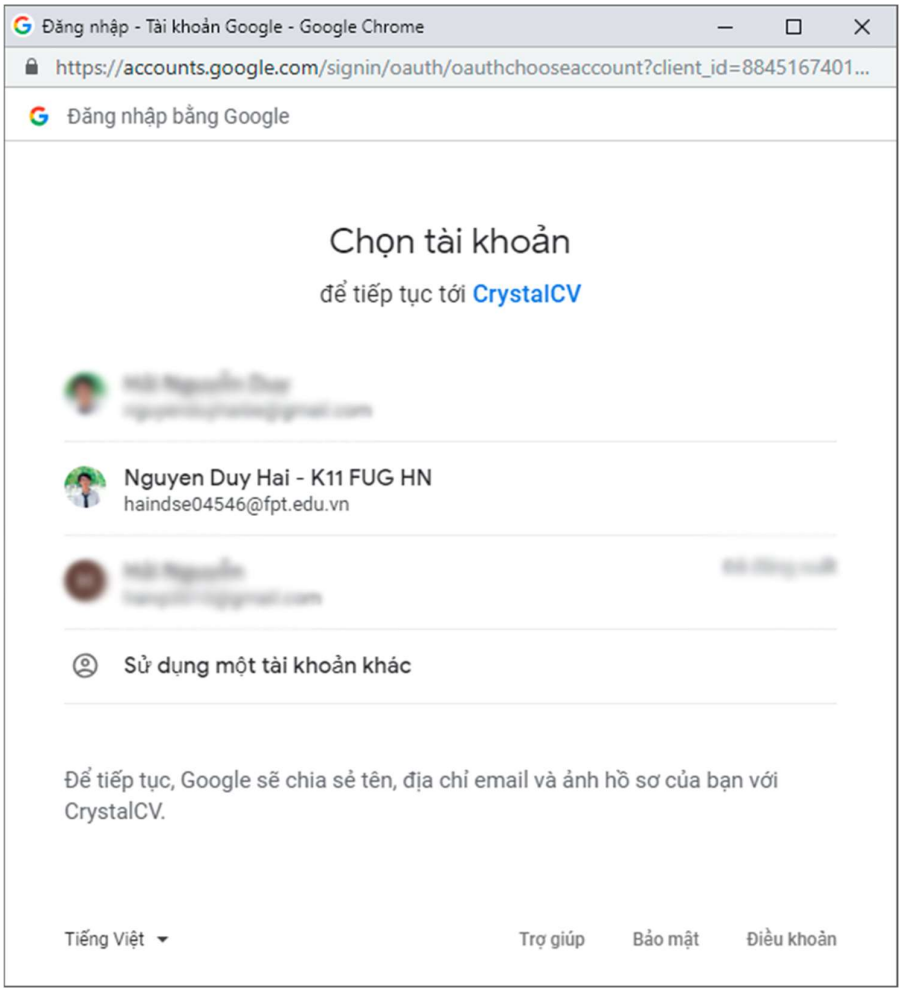


Figure 4-38: Guest signs in with Google screen design

Class Diagram

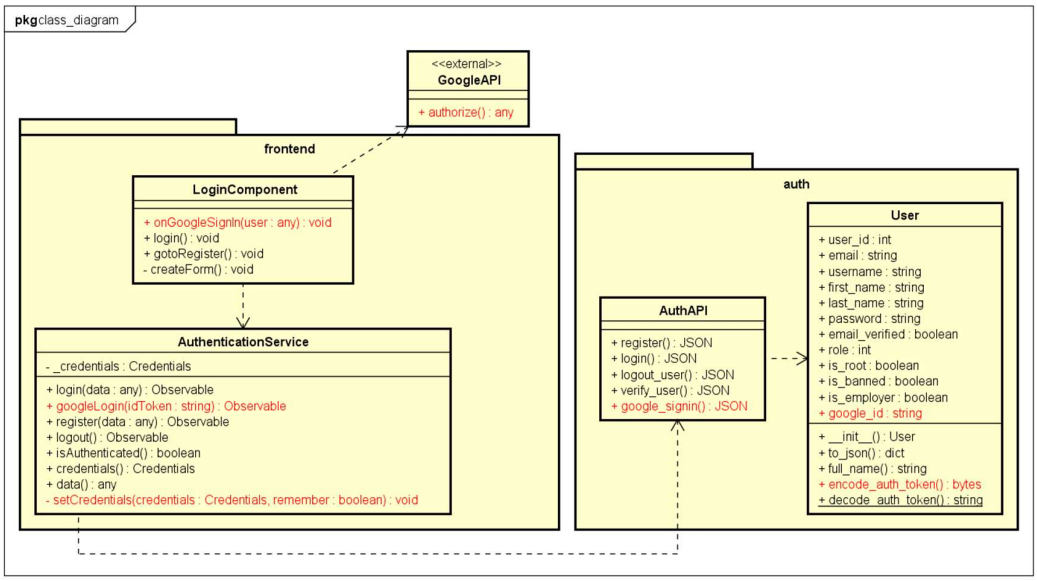


Figure 4-39: Guest signs in with Google account class diagram

Class Specification

LoginComponent

LoginComponent			
Physical address	/services/frontend/src/app/login/login.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
onGoogleSigin	Callback for handling google sign in credentials		
Return Type	void		
Parameters	Name	Type	Description
	user	any	User profile object returned from Google API

AuthenticationService

AuthenticationService			
Physical address	/services/frontend/src/app/core/authentication/authentication.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
1	_credentials	Credentials	Credentials object, containing access token and username
Operation			
googleLogin	Send the ID token to backend for validation and account creation		
Return Type	Observable		

Parameters	Name	Type	Description
	idToken	string	The ID token in the profile object returned from Google API
setCredentials	Save the credentials to browser local storage		
Return Type	void		
Parameters	Name	Type	Description
	credentials	Credentials	The credentials to be saved

AuthAPI

AuthAPI			
Physical address	/services/auth/project/api/auth.py		
Base class	Blueprint		
Attributes			
No	Name	Type	Description
Operation			
googleSignin	Sign the user in using Google credentials, or create a new account first if the user does not already exist		
Return Type	JSON		
Parameters	Name	Type	Description

User

User			
Physical address	/services/auth/project/api/models.py		
Base class	Model		
Attributes			
No	Name	Type	Description
1	google_id	string	Google ID of the user
Operation			
encode_auth_token	Create encoded access token from user information		
Return Type	bytes		
Parameters	Name	Type	Description

Sequence Diagram

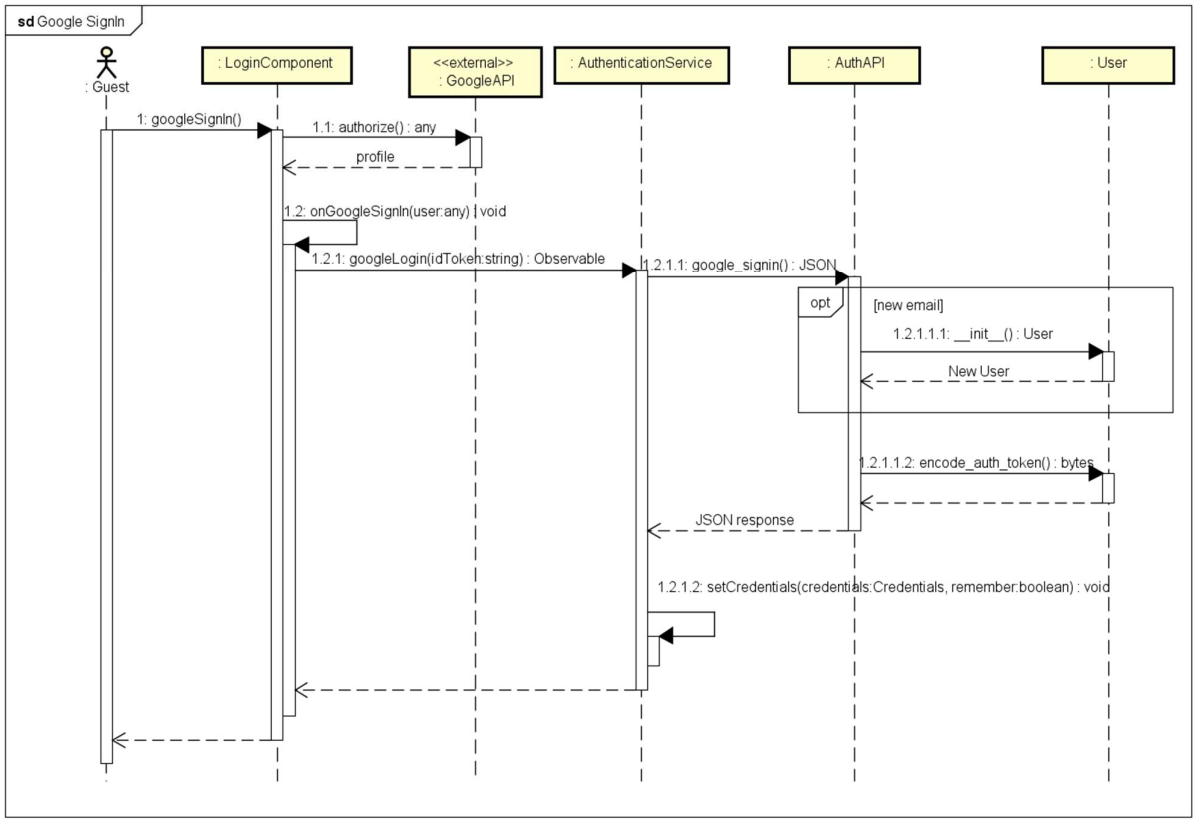


Figure 4-40: Guest signs in with Google account sequence diagram

4.3.4.3 Guest signs up

Screen Design

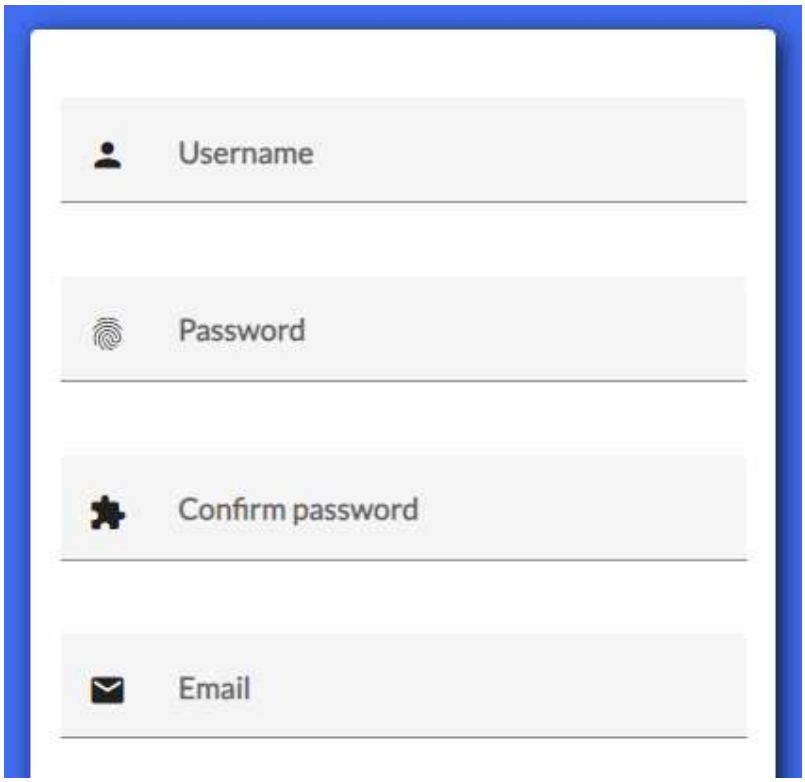


Figure 4-41: Guest signs up screen design

Class Diagram

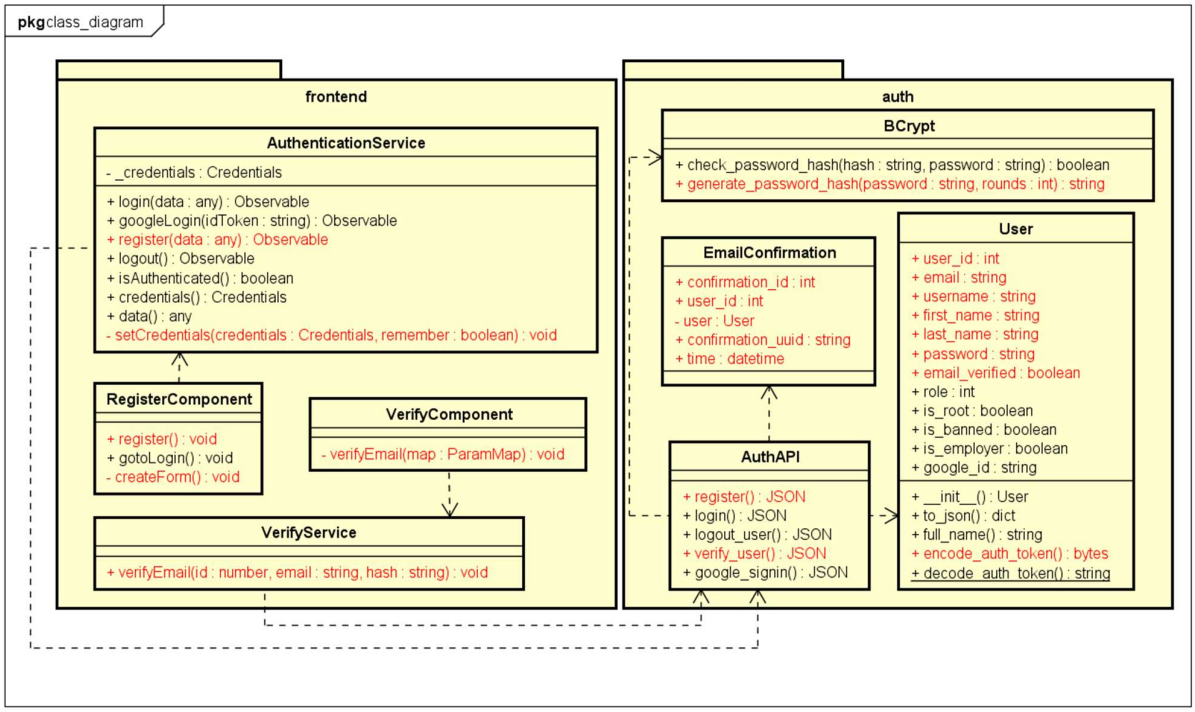


Figure 4-42: Guest signs up class diagram

Class Specification

RegisterComponent

RegisterComponent			
Physical address	/services/frontend/src/app/register/register.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
register	Register a new user		
Return Type	void		
Parameters	Name	Type	Description
createForm	Initialize form fields		
Return Type	Void		
Parameters	Name	Type	Description

VerifyComponent

VerifyComponent			
Physical address	/services/frontend/src/app/verify/verify.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
verifyEmail	Verify email address		
Return Type	void		
Parameters	Name	Type	Description
	map	ParamMap	Object containing GET parameters from current route

AuthenticationService

AuthenticationService			
Physical address	/services/frontend/src/app/core/authentication/authentication.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
1	_credentials	Credentials	Credentials object, containing access token and username
Operation			
register	Send registration request to the API		
Return Type	Observable		
Parameters	Name	Type	Description
	data	any	The login data containing username and password

setCredentials	Save the credentials to browser local storage		
Return Type	void		
Parameters	Name	Type	Description
	credentials	Credentials	The credentials to be saved

VerifyService

VerifyService			
Physical address	/services/frontend/src/app/verify/verify.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
verifyEmail	Send verification request to API		
Return Type	Observable		
Parameters	Name	Type	Description
	id	number	ID of the EmailConfirmation instance
	email	string	Email of the user
	hash	string	The confirmation_uuid of the EmailConfirmation instance

AuthAPI

AuthAPI			
Physical address	/services/auth/project/api/auth.py		
Base class	Blueprint		
Attributes			
No	Name	Type	Description
Operation			
verify_user	Verify the user's email address		
Return Type	JSON		
Parameters	Name	Type	Description

User

User			
Physical address	/services/auth/project/api/models.py		
Base class	Model		
Attributes			
No	Name	Type	Description
1	user_id	int	ID of the user
2	email	string	Email of the user
3	username	string	Username of the user
4	first_name	string	First name of the user
5	last_name	string	Last name of the user
6	password	string	Hashed password of the user
7	email_verified	boolean	Indicates whether user has verified his/her email address or not
Operation			
encode_auth_token	Create encoded access token from user information		

Return Type	bytes		
Parameters	Name	Type	Description

EmailConfirmation

User			
Physical address	/services/auth/project/api/models.py		
Base class	Model		
Attributes			
No	Name	Type	Description
1	confirmation_id	int	ID of the confirmation
2	user_id	int	User ID of the confirmation
3	user	User	User instance related to the confirmation
4	confirmation_uuid	string	A random uuid for each confirmation
5	time	datetime	The time of the confirmation creation
Operation			
encode_auth_token	Create encoded access token from user information		
Return Type	bytes		
Parameters	Name	Type	Description

BCrypt

BCrypt			
Physical address	Bcrypt		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
generate_password_hash	Generate hash from plain-text password		
Return Type	string		
Parameters	Name	Type	Description
	password	string	Plain-text password
	rounds	int	Rounds parameter of the bcrypt algorithm

Sequence Diagram

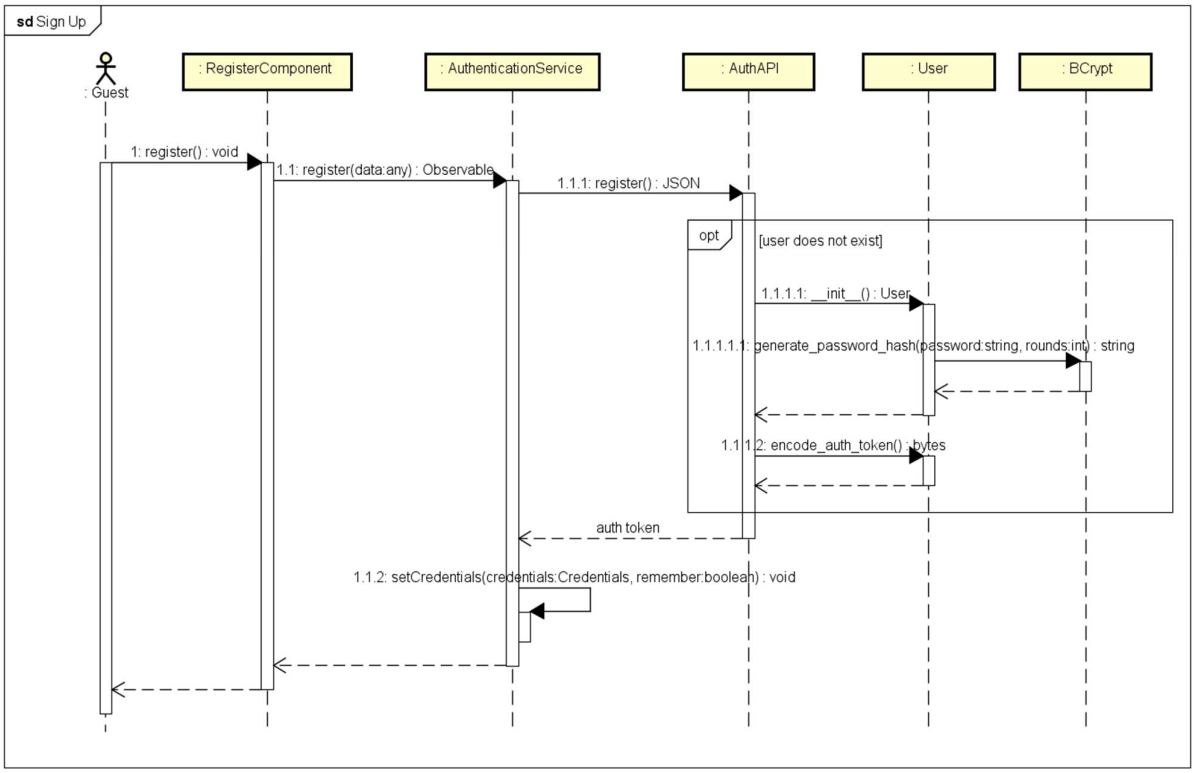


Figure 4-43: Step 1 – Guest signs up sequence diagram

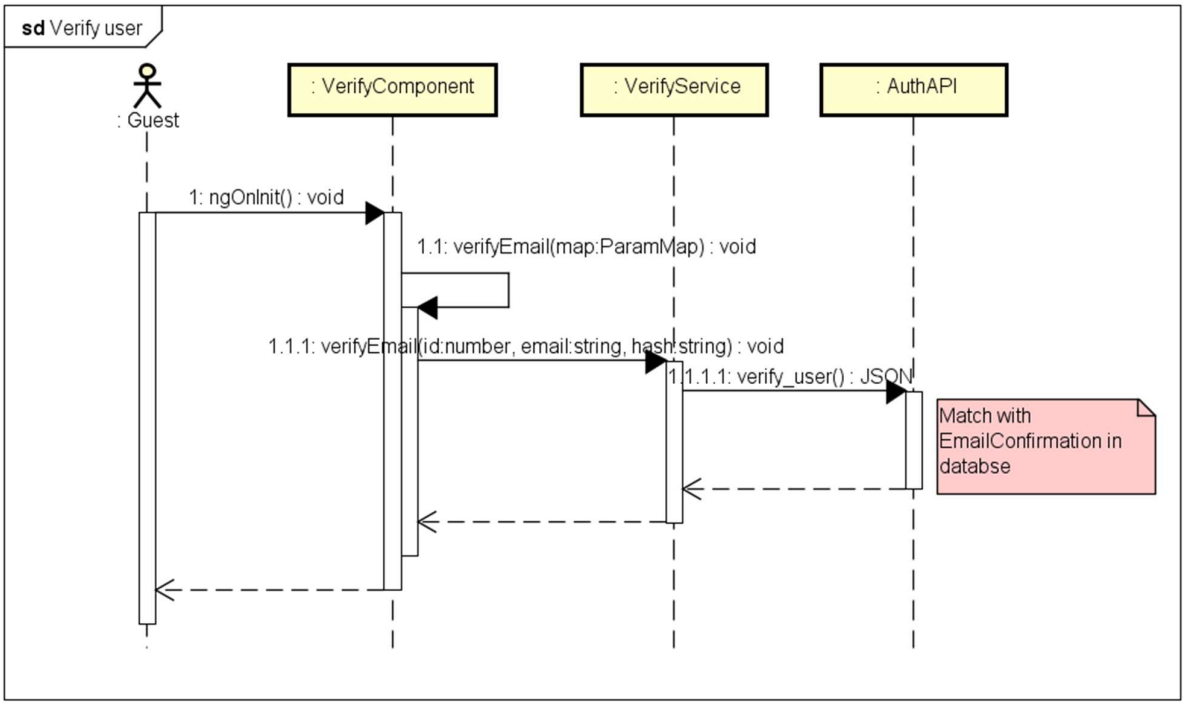


Figure 4-44: Step 2 – Guest verifies email address

4.3.4.4 User signs out

Screen Design

Figure 4-45: User signs out screen design

Class Diagram

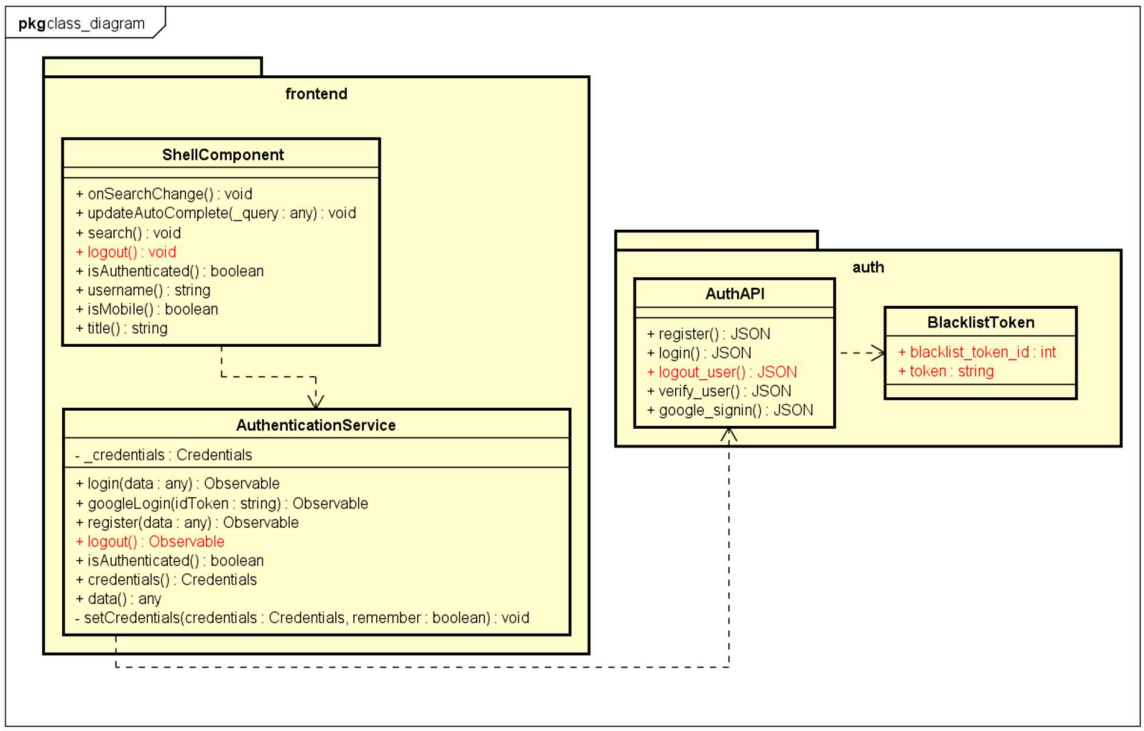


Figure 4-46: User signs out class diagram

Class Specification

ShellComponent

ShellComponent	
Physical address	/services/frontend/src/app/shell/shell.component.ts
Base class	Model
Attributes	

No	Name	Type	Description
Operation			
logout	Log the user out		
Return Type	void		
Parameters	Name	Type	Description

AuthenticationService

AuthenticationService			
Physical address	/services/frontend/src/app/core/authentication/authentication.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
1	_credentials	Credentials	Credentials object, containing access token and username
Operation			
logout	Send logout request to the API		
Return Type	Observable		
Parameters	Name	Type	Description

AuthAPI

AuthAPI			
Physical address	/services/auth/project/api/auth.py		
Base class	Blueprint		
Attributes			
No	Name	Type	Description
Operation			
logout_user	Log the user out and blacklist user's access token		
Return Type	JSON		
Parameters	Name	Type	Description

BlacklistToken

BlacklistToken			
Physical address	/services/auth/project/api/models.py		
Base class	Model		
Attributes			
No	Name	Type	Description
1	blacklist_token_id	int	ID of the blacklist token
2	token	string	User's access token to be blacklisted

Sequence Diagram

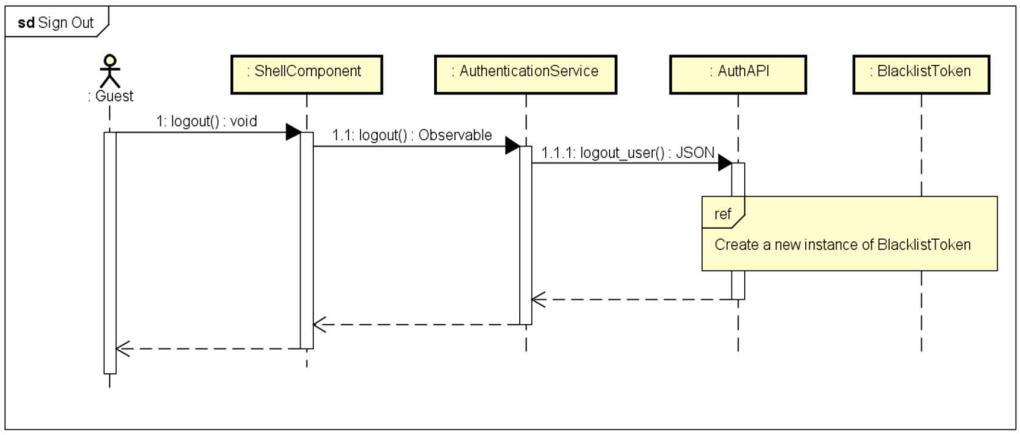


Figure 4-47: User signs out sequence diagram

4.3.4.5 User gets autocomplete suggestions

Screen Design

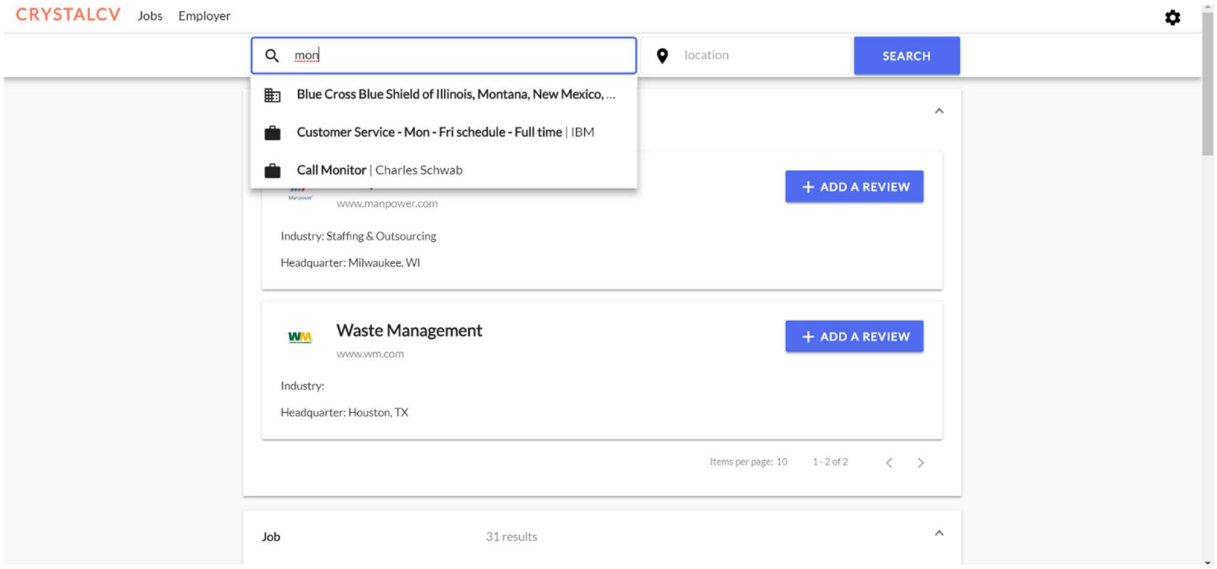


Figure 4-48: Autocomplete suggestion screen design

Screen Explanation

Element	Type	Description
Name input	Text input	Input for job title/company name
Location input	Text input	Input for location of job/company
Search	Button	Submit search form
Suggestion list	List	List of suggestions

Class Diagram

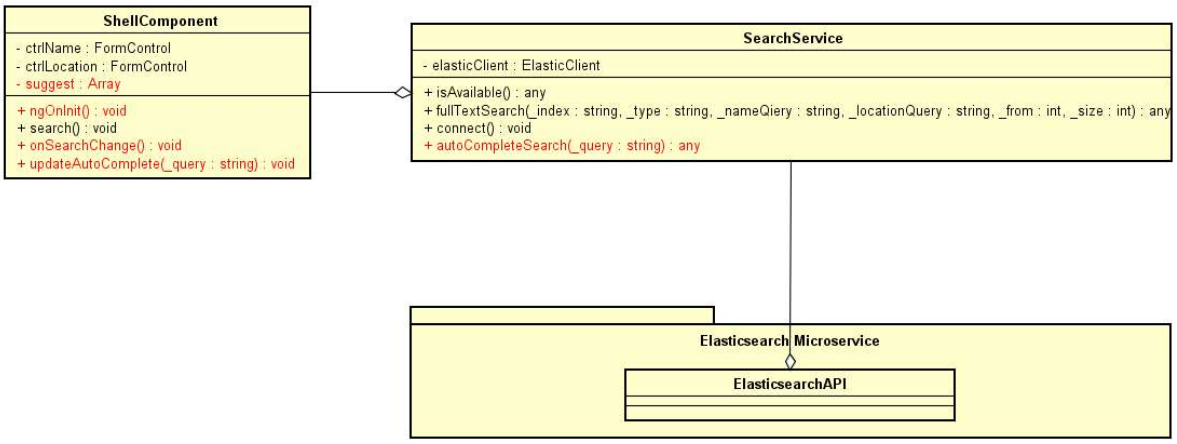


Figure 4-49: Autocomplete suggestion class diagram

Class Specification

ShellComponent

ShellComponent			
Physical address	/services/frontend/src/app/shell/shell.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
1	ctrlName	FormControl	Form control for name query
2	ctrlLocation	FormControl	Form control for location query
3	suggest	Array	List of suggestions
Operation			
onSearchChange	Call update method whenever search input is changed		
Return Type	void		
Parameters	Name	Type	Description
updateAutoComplete	Call search method from service, then display return results		
Return Type	void		
Parameters	Name	Type	Description
	_query	String	

SearchService

SearchService			
Physical address	/services/frontend/src/app/search/search.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
1	elasticClient	ElasticClient	Elasticsearch client
Operation			

isAvailable	Check connection to Elasticsearch microservice		
Return Type	any		
Parameters	Name	Type	Description
connect	Connect to Elastic microservice		
Return Type	void		
Parameters	Name	Type	Description
autoCompleteSearch	Query elasticsearch for suggestions		
Return Type	any		
Parameters	Name	Type	Description
	_query	String	

Implementation

- User searches for job/company by typing in the search bar (part of navbar – ShellComponent). As he/she is typing, provide he/she with a list of suggestions (job and company) related to what's in the search box.

Sequence Diagram

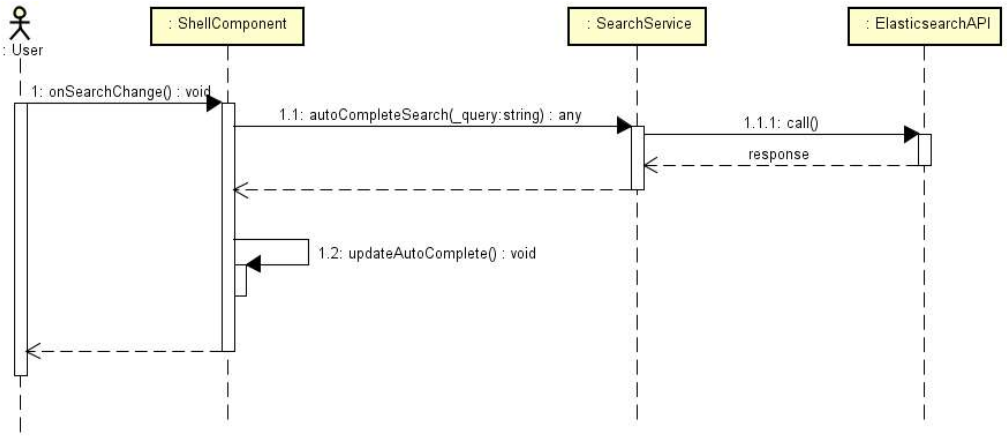
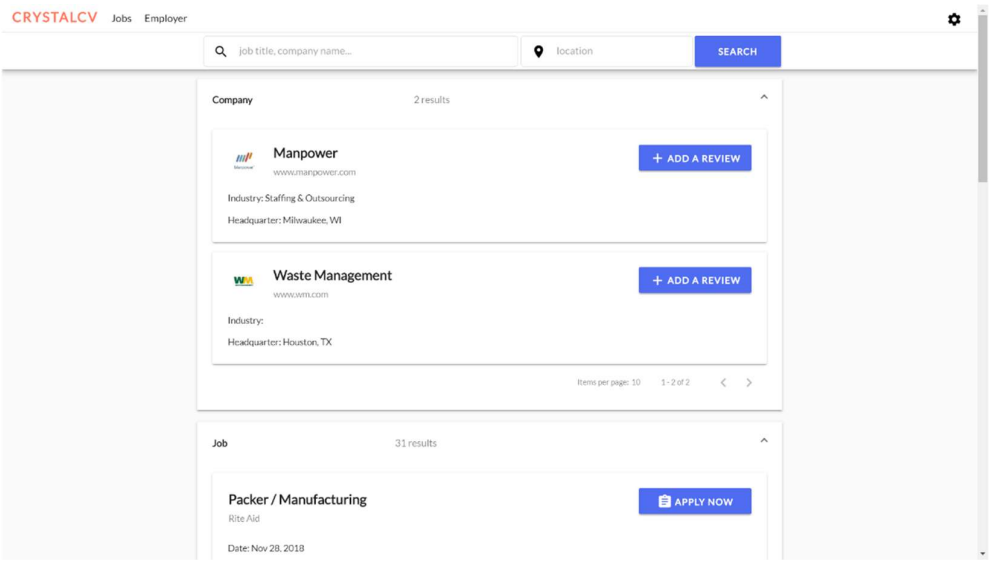


Figure 4-50: Autocomplete suggestion sequence diagram

4.3.4.6 User searches for jobs/companies

Screen Design



Screen Explanation

Element	Type	Description
Name input	Text input	Input for job title/company name
Location input	Text input	Input for location of job/company
Search	Button	Submit search form
Company list	List	List of found companies, with paging
Company card	Card	Description of each found company
Add a review	Button	Quick action for each found company. User can choose between types of reviews that they want to add.
Job list	List	List of found job posts, with paging
Job card	Card	Description of each found job
Apply now	Button	Quick action for each found job.

Class Diagram

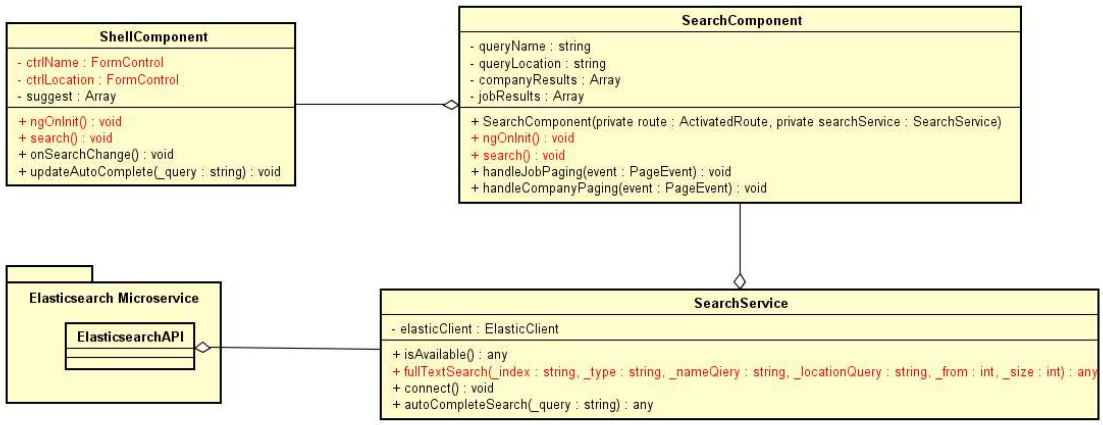


Figure 4-51: User searches for jobs/companies class diagram

Class Specification

ShellComponent

ShellComponent			
Physical address	/services/frontend/src/app/shell/shell.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
1	ctrlName	FormControl	Form control for name query
2	ctrlLocation	FormControl	Form control for location query
Operation			
search	Redirect to search results page, with name and location query as params		
Return Type			
Parameters	Name	Type	Description

SearchComponent

SearchComponent			
Physical address	/services/frontend/src/app/search/search.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
1	queryName	string	Query for name
2	queryLocation	string	Query for location
3	companyResults	Array	List of found companies
4	jobResults	Array	List of found jobs
Operation			
SearchComponent	Constructor		
Return Type			

Parameters	Name	Type	Description
	route	ActivatedRoute	Contains the information about a route, including query params
	searchService	SearchService	Instance of SearchService
search	Call search method from service		
Return Type			
Parameters	Name	Type	Description
handleJobPaging	Reload job results		
Return Type	void		
Parameters	Name	Type	Description
	event	PageEvent	Contains page-related information, such as page index, page size…
handleCompanyPaging	Reload company results		
Return Type	void		
Parameters	Name	Type	Description
	event	PageEvent	Contains page-related information, such as page index, page size…

SearchService

SearchService			
Physical address	/services/frontend/src/app/search/search.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
1	elasticClient	ElasticClient	Elasticsearch client
Operation			
isAvailable	Check connection to Elasticsearch microservice		
Return Type	any		
Parameters	Name	Type	Description
connect	Connect to Elastic microservice		
Return Type	void		
Parameters	Name	Type	Description
fullTextSearch	Initiate search		
Return Type	any		
Parameters	Name	Type	Description
	_index	String	Elastic index name
	_type	String	Elastic type name
	_nameQuery	String	Name query
	_locationQuery	String	Location query
	_from	Int	Offset from the first result

	_size	Int	Maximum amount of results to be returned (page size)

Implementation

- User searches for job/company by typing in the search bar (part of navbar – ShellComponent). When he/she clicks Search, redirect to search result page (SearchComponent) with name and location query as parameters. SearchComponent calls methods from SearchService, which calls Elasticsearch API.
- Support pagination by putting *size* and *from* in query object, *size* is number of results returned per page, and *from* is the offset from the first result

Sequence Diagram

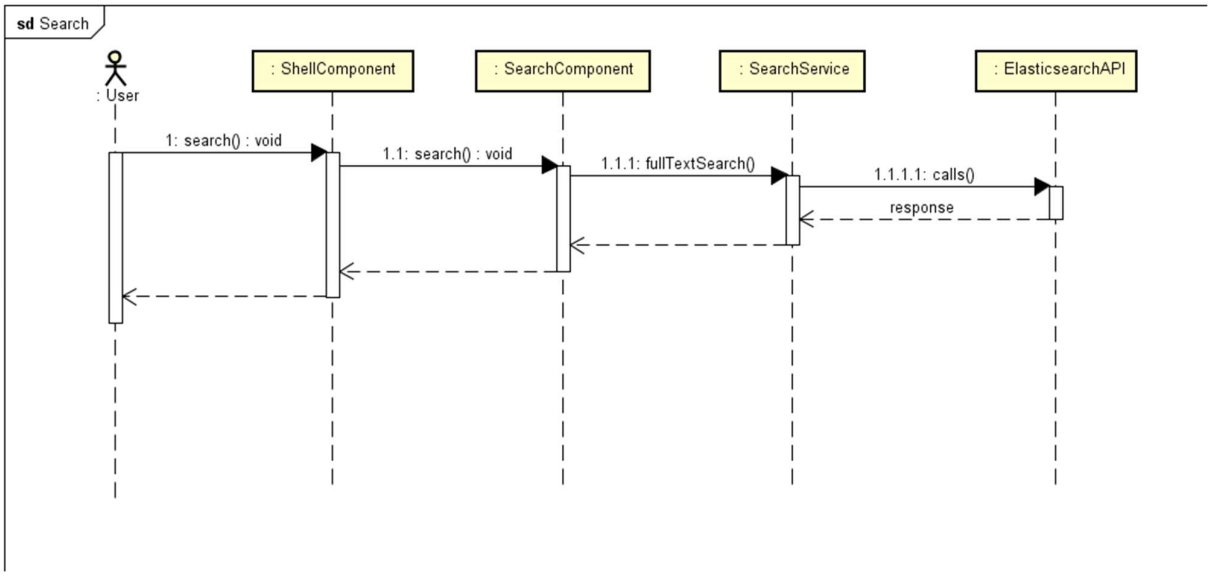


Figure 4-52: User searches for jobs/companies sequence diagram

4.3.4.7 Job seeker views user profile

Screen Design

Class Diagram

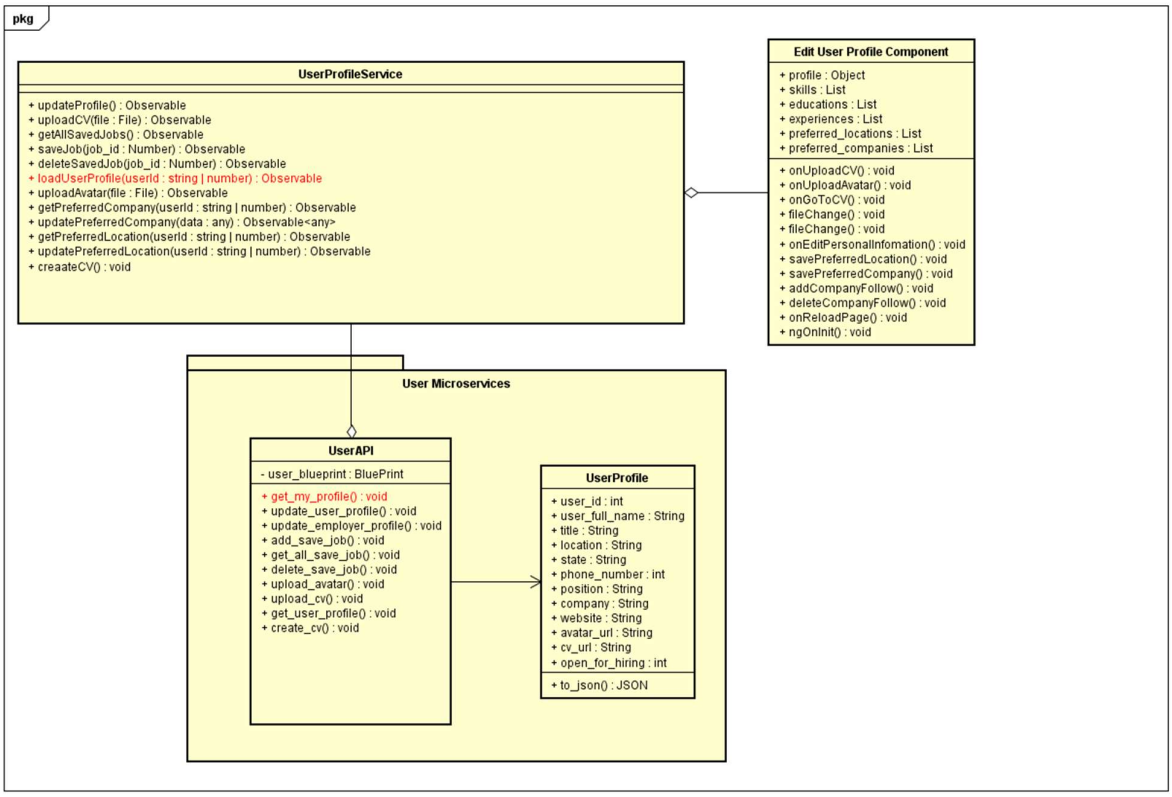


Figure 4-53: Job seeker views user profile class diagram

Class Specification

UserProfileService

UserProfileService			
Physical address	services\frontend\src\app\user-profile\user-profile.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
UserProfileService	Constructor		
Return Type			
Parameters	Name	Type	Description
	httpClient	HttpClient	instance of HttpClient
	authService	Authenticat-onService	instance of AuthenticationSer-vice
loadUserProfile	Update user's personal details		
Return Type	Observable		
Parameters	Name	Type	Description
	userId	String	User ID

User API

User API			
Physical address	services\user\project\api\users.py		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
get_my_profile	Get user profile details		
Return Type	JSON		
Parameters	Name	Type	Description
	profile	JSON	Profile details

EditUserProfileComponent

EditUserProfileComponent			
Physical address	services\frontend\src\app\user-profile\user-profile.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
onInitalLoadUserProfile	Load user personal details		
Return Type	void		
Parameters	Name	Type	Description

UserProfile

UserProfile			
Physical address	services\user\project\api\models.py		
Base class	Class		
Attributes			
No	Name	Type	Description
1	user_id	int	User ID
2	location	string	User location
3	title	string	User title
4	state	string	User's state
5	phone_number	string	User's phone number
6	position	string	User's job position
7	company	string	User's company
8	website	string	User's website
9	avatar_url	string	User's avatar URL
10	cv_url	string	User's CV URL
11	open_for_hiring	int	User's availability for hiring
Operation			
to_json	return JSON of model		
Return Type	JSON		
Parameters	Name	Type	Description

Sequence Diagram

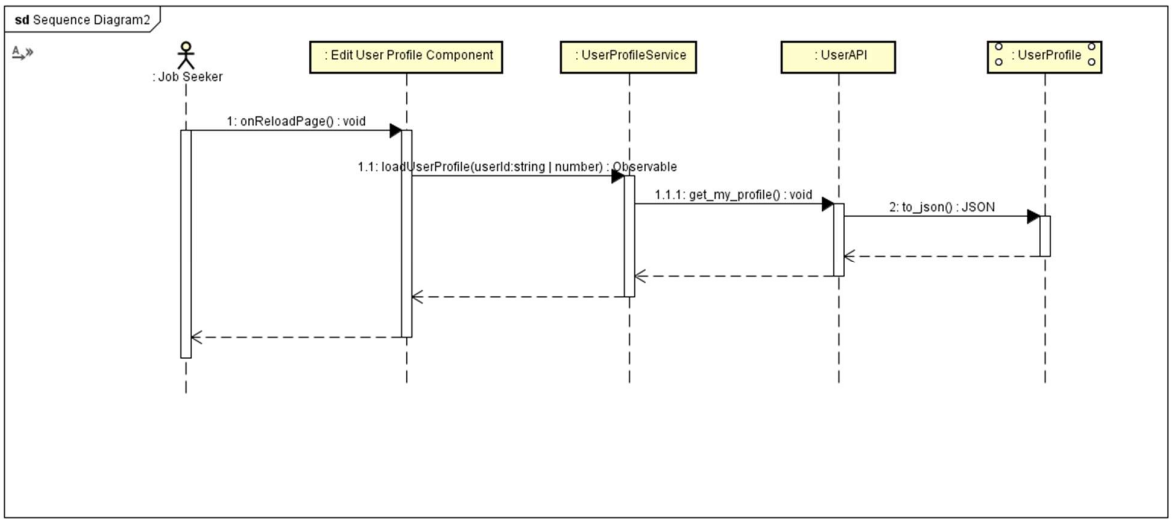


Figure 4-54: Job seeker views user profile sequence diagram

4.3.4.8 Job seeker requests to become an employer

4.3.4.9 Job seeker creates a review about a company

Screen Design

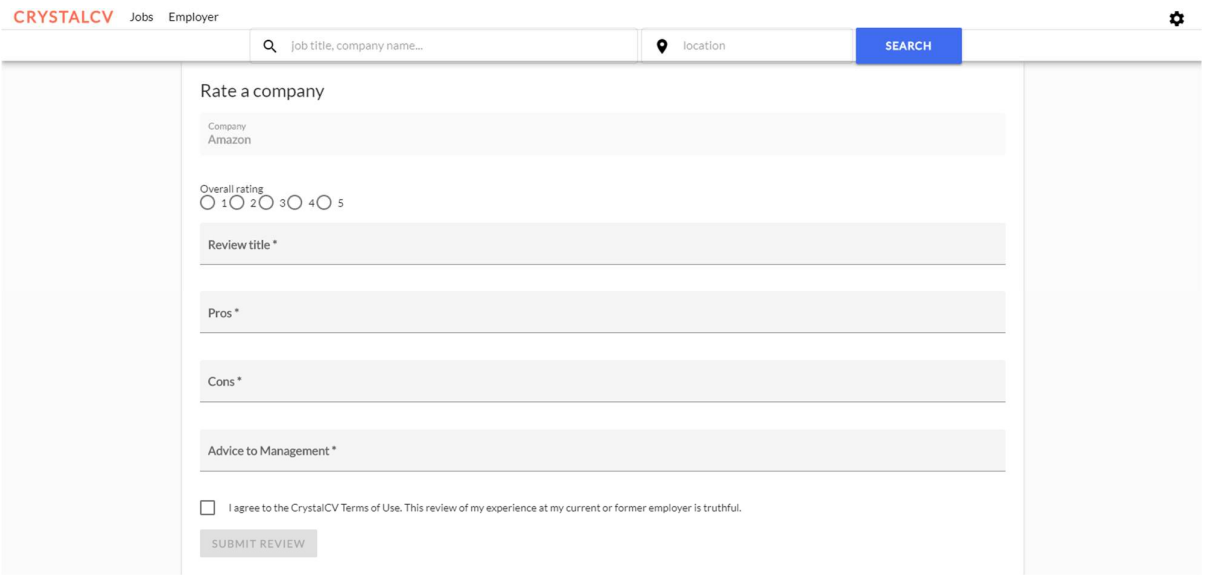


Figure 4-55: Job seeker adds a new company review

Screen Explanation

Element	Type	Description
Company	Disabled text box	Shows the company name for the company being reviewed
Overall rating	Radio buttons	Overall rating over 5 for the reviewed company

Pros	Text area	Pros of working in company
Cons	Text area	Cons of working in company
Advice to management	Text area	Advice to company management
Terms agreement	Checkbox	Whether user agrees to website's Terms of Use. Required to submit

Class Diagram

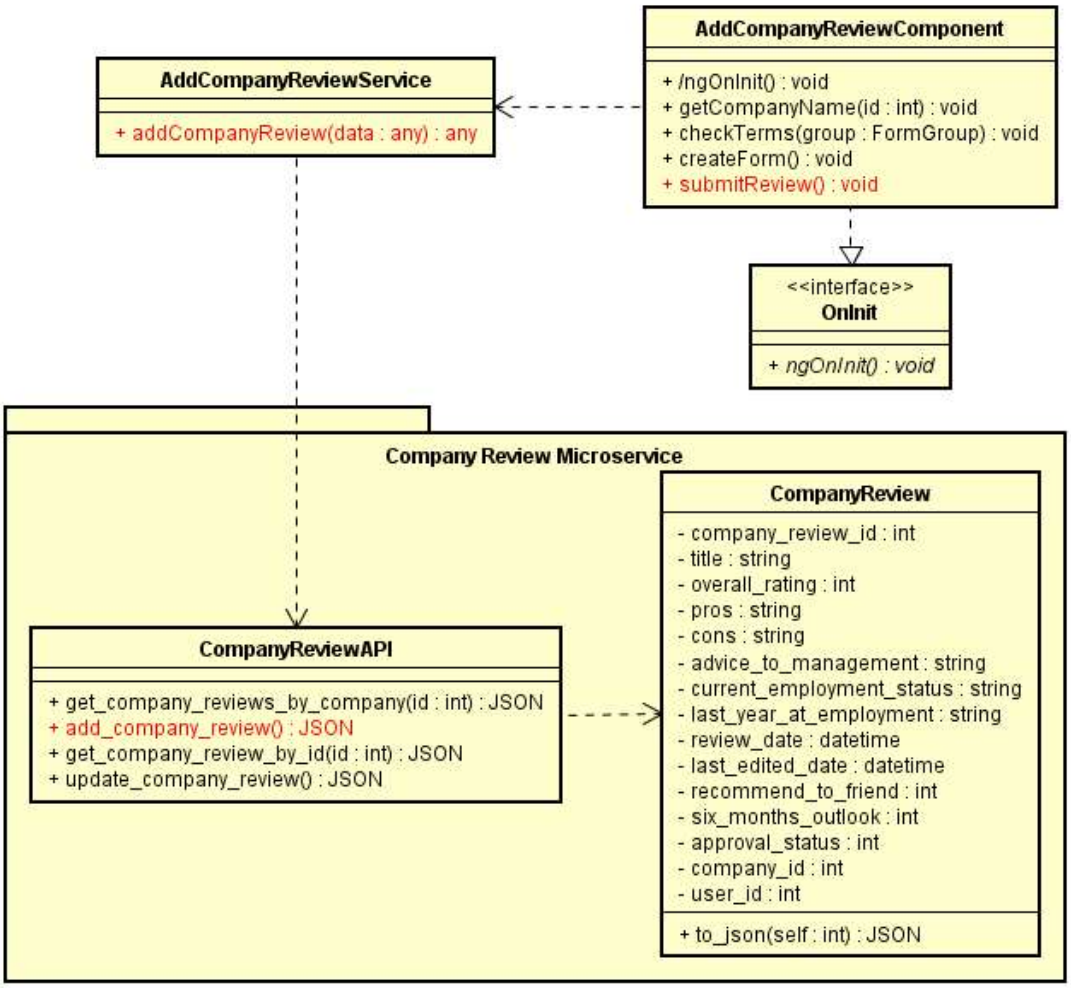


Figure 4-56: Job seeker adds a company review class diagram

Class Specification

AddCompanyReviewService

AddCompanyReviewService	
Physical address	/services/frontend/src/app/company/add-company-review/add-company-review.service.ts

Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
addCompanyReview	Send company review data to API to process		
Return Type	Observable		
Parameters	Name	Type	Description
	data	any	review data

AddCompanyReviewComponent

AddCompanyReviewComponent			
Physical address	/services/frontend/src/app/ add-company-review/add-company-review.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
ngOnInit	Page initialization, used to populate company data		
Return Type			
Parameters	Name	Type	Description
checkTerms	Check if user agrees to Terms and Conditions		
Return Type			
Parameters	Name	Type	Description
submitReview	Submit review form to add new review		
Return Type			
Parameters	Name	Type	Description

CompanyReviewAPI

CompanyReviewAPI			
Physical address	/services/company/project/api/companies.py		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
add_company_review	Update company review from sent data		
Return Type	JSON	Return Type	JSON
Parameters	Name	Parameters	Name

CompanyReview

Company	
Physical address	/services/company/project/api/models.py
Base class	Class
Attributes	

No	Name	Type	Description
1	company_review_id	int	Company review ID
2	title	string	Review title
3	overall_rating	int	Overall rating
4	pros	string	Pros of company
5	cons	string	Cons of company
6	advice_to_management	string	Advice to management
7	current_employment_status	string	Current employment status
8	last_year_at_employment	string	Last year at employment (if unemployed)
9	review_date	datetime	When review was made
10	last_edited_date	datetime	Last edited date
11	recommend_to_friend	int	URL to company logo image
12	six_month_outlook	int	6-month outlook of company
13	approval_status	int	Approval status
14	company_id	int	Company ID
15	user_id	int	Reviewing user ID
Operation			
to_json	return dict of model in json key/value format		
Return Type	dict		
Parameters	Name	Type	Description
	self	CompanyR-eview	company review object

Sequence Diagram

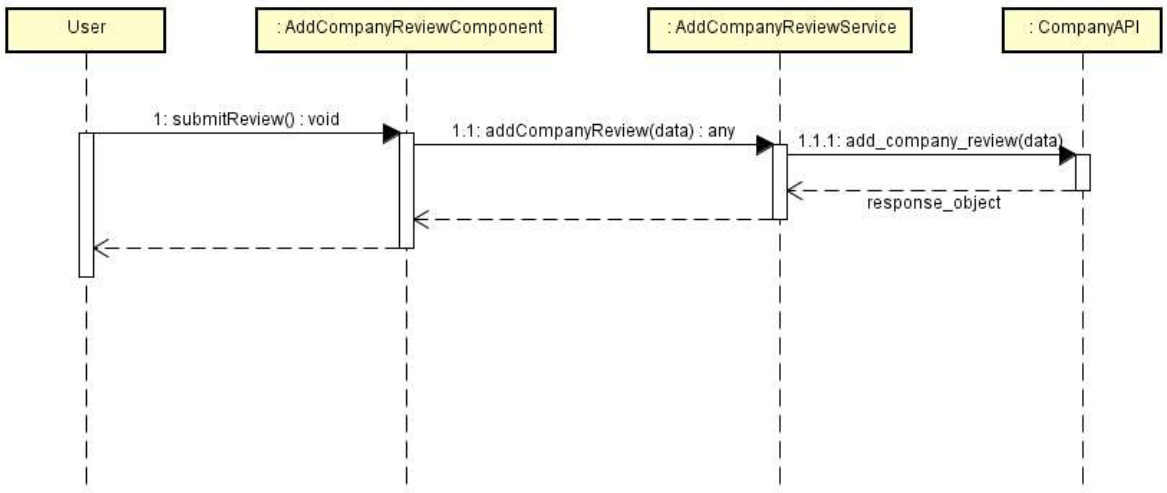


Figure 4-57: Job seeker adds a company review sequence diagram

4.3.4.10 Job seeker edits a company review

Screen Design

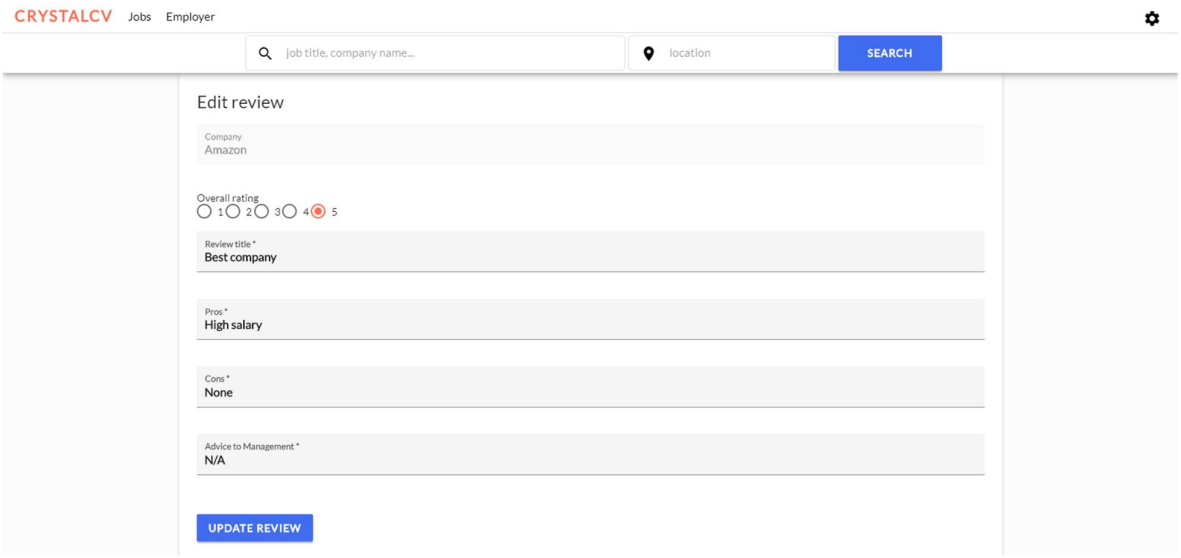


Figure 4-58: User edits new company review

Screen Explanation

Element	Type	Description
Company	Disabled text box	Shows the company name for the company being reviewed
Overall rating	Radio buttons	Overall rating over 5 for the reviewed company
Pros	Text area	Pros of working in company
Cons	Text area	Cons of working in company
Advice to management	Text area	Advice to company management
Terms agreement	Checkbox	Whether user agrees to website's Terms of Use. Required to submit

Class Diagram

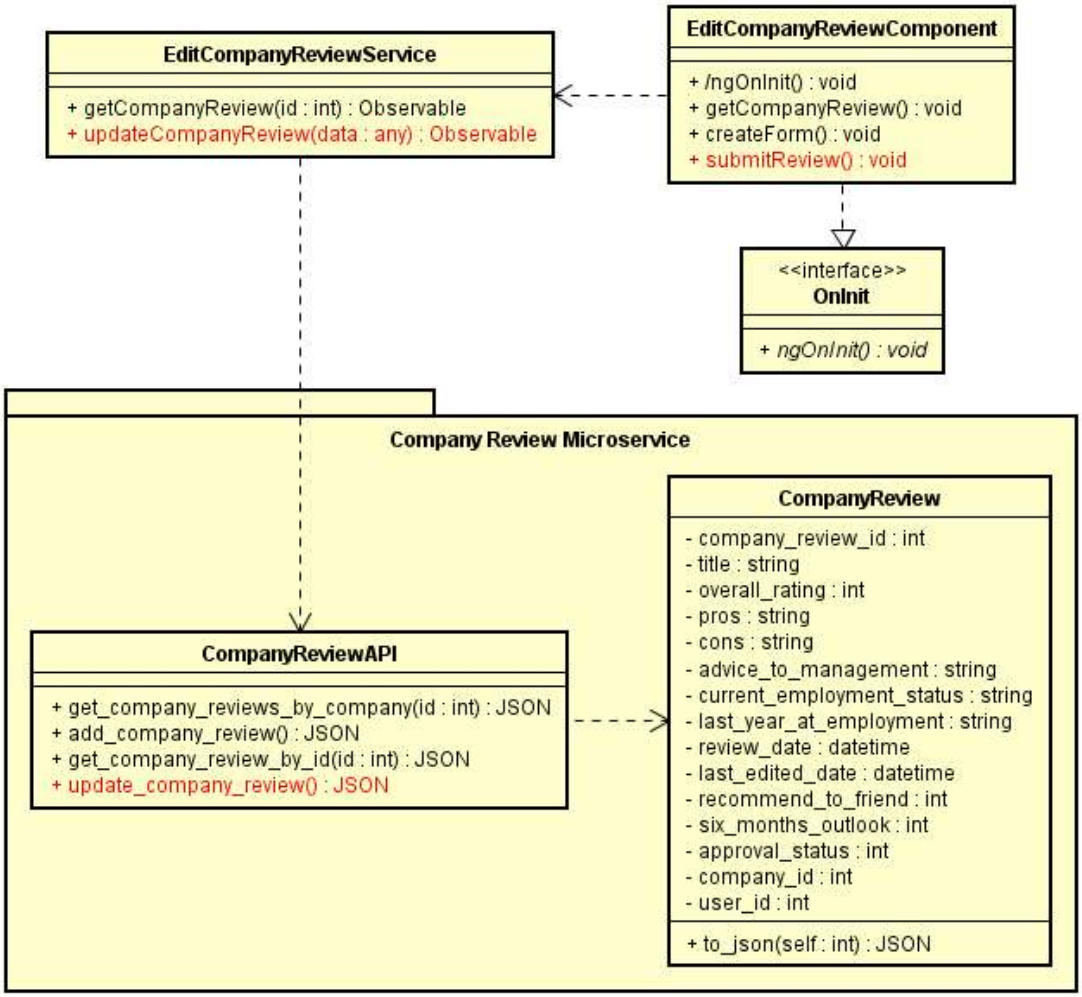


Figure 4-59: User edits a company review class diagram

Class Specification

EditCompanyReviewService

EditCompanyReviewService			
Physical address	/services/frontend/src/app/company/edit-company-review/edit-company-review.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
updateCompanyReview	Update company review		
Return Type	Observable		
Parameters	Name	Type	Description
	data	any	review data
getCompanyReview	Get company review by ID		
Return Type	Observable		
Parameters	Name	Type	Description

	id	int	ID of company selected review

EditCompanyReviewComponent

EditCompanyReviewComponent			
Physical address	/services/frontend/src/app/ add-company-review/add-company-review.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
ngOnInit	Page initialization, used to populate company data		
Return Type			
Parameters	Name	Parameters	Name
getCompanyReview	get company review from URL parameter		
Return Type			
Parameters	Name	Parameters	Name
submitReview	Submit form to update review		
Return Type			
Parameters	Name	Parameters	Name

CompanyReviewAPI

CompanyReviewAPI			
Physical address	/services/company/project/api/companies.py		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
get_company_review_by_id	Get company review by ID		
Return Type	JSON	Return Type	JSON
Parameters	Name	Parameters	Name
	id		id
update_company_review	Update company review from sent data		
Return Type	JSON	Return Type	JSON
Parameters	Name	Parameters	Name

CompanyReview

Company			
Physical address	/services/company/project/api/models.py		
Base class	Class		
Attributes			
No	Name	Type	Description

1	company_review_id	int	Company review ID
2	title	string	Review title
3	overall_rating	int	Overall rating
4	pros	string	Pros of company
5	cons	string	Cons of company
6	advice_to_management	string	Advice to management
7	current_employment_status	string	Current employment status
8	last_year_at_employment	string	Last year at employment (if unemployed)
9	review_date	datetime	When review was made
10	last_edited_date	datetime	Last edited date
11	recommend_to_friend	int	URL to company logo image
12	six_month_outlook	int	6-month outlook of company
13	approval_status	int	Approval status
14	company_id	int	Company ID
15	user_id	int	Reviewing user ID
Operation			
to_json	return dict of model in json key/value format		
Return Type	dict		
Parameters	Name	Type	Description
	self	CompanyR-eview	company review object

Sequence Diagram

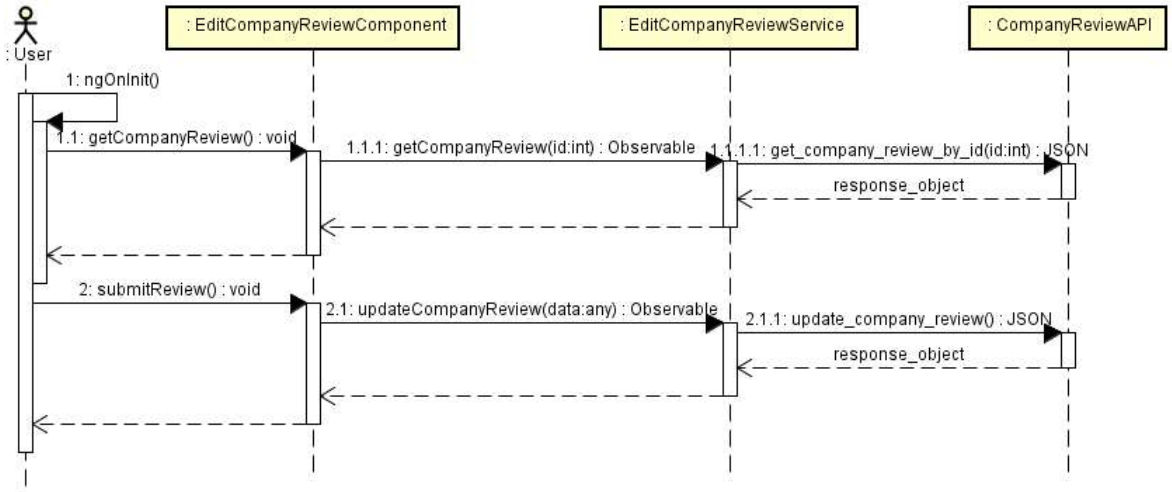


Figure 4-60: Job seeker edits a company review sequence diagram

4.3.4.11 Job seeker creates an interview review about a company

4.3.4.12 Job seeker edits review about the company's interview

4.3.4.13 Job seeker follows/unfollows a company

Screen Design

Figure 4-61: Follow/Unfollow screen design

Screen Explanation

Element	Type	Description
Company name	Text	Name of company
Company logo	Image	Logo of company
Follow	Button	Follow the company

Class Diagram

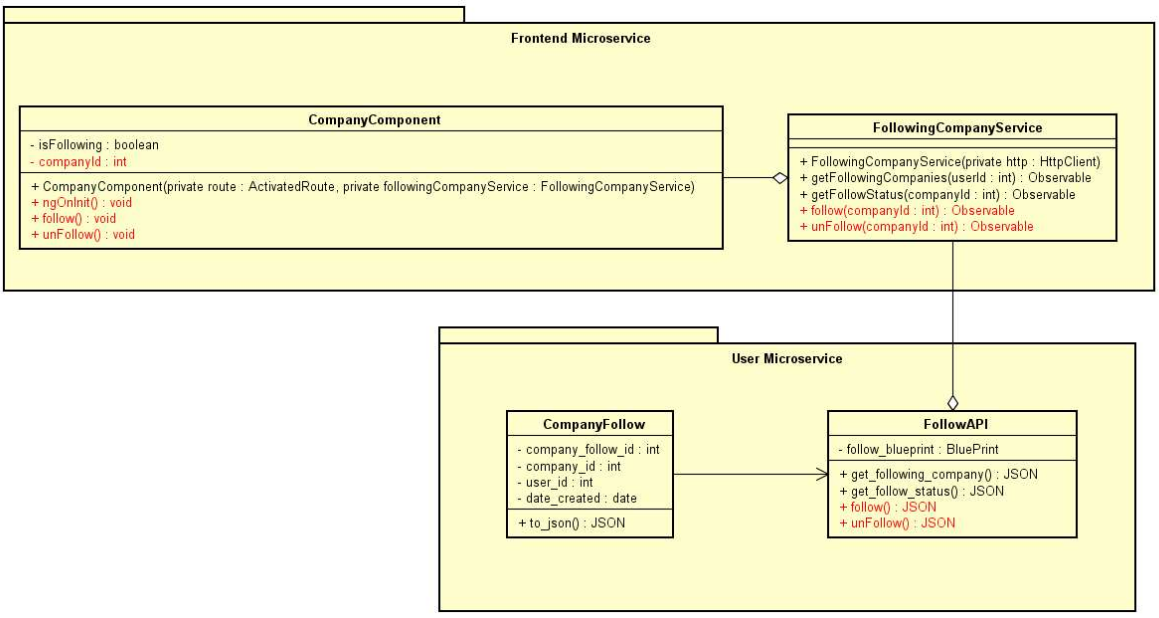


Figure 4-62: Follow/Unfollow class diagram

Class Specification

CompanyComponent

CompanyComponent			
Physical address	/services/frontend/src/app/company/company.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description

1	isFollowing	Boolean	Follow status of a user
2	companyId	Int	ID of current company
Operation			
follow	Call follow method from FollowService, then change user's follow status to true		
Return Type	void		
Parameters	Name	Type	Description
unFollow	Call unFollow method from FollowService, then change user's follow status to false		
Return Type	void		
Parameters	Name	Type	Description
	_		

FollowingCompanyService

FollowingCompanyService			
Physical address	/services/frontend/src/app/user-profile/following - company/following-company.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
follow	Call Follow API to add a follow with companyId and userId as parameters		
Return Type	Observable		
Parameters	Name	Type	Description
	companyId	Int	Company's ID
unFollow	Call Follow API to remove a follow with companyId and userId as parameters		
Return Type	Observable		
Parameters	Name	Type	Description
	companyId	Int	Company's ID

FollowAPI

FollowAPI			
Physical address	/services/user/project/api/following-company.py		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
follow	Add a new company follow		
Return Type	JSON		
Parameters	Name	Parameters	Name

unFollow	Remove a company follow		
Return Type	JSON		
Parameters	Name	Parameters	Name

CompanyFollow

CompanyFollow			
Physical address	/services/company/project/api/models.py		
Base class	Class		
Attributes			
No	Name	Type	Description
1	company_follow_id	int	CompanyFollow id
2	company_id	Int	ID of company
3	user_id	Int	ID of user
4	date_created	date	Date created
Operation			
to_json	return dict of model in json key/value format		
Return Type	dict		
Parameters	Name	Type	Description
	self	CompanyFollow	CompanyFollow object

Sequence Diagram

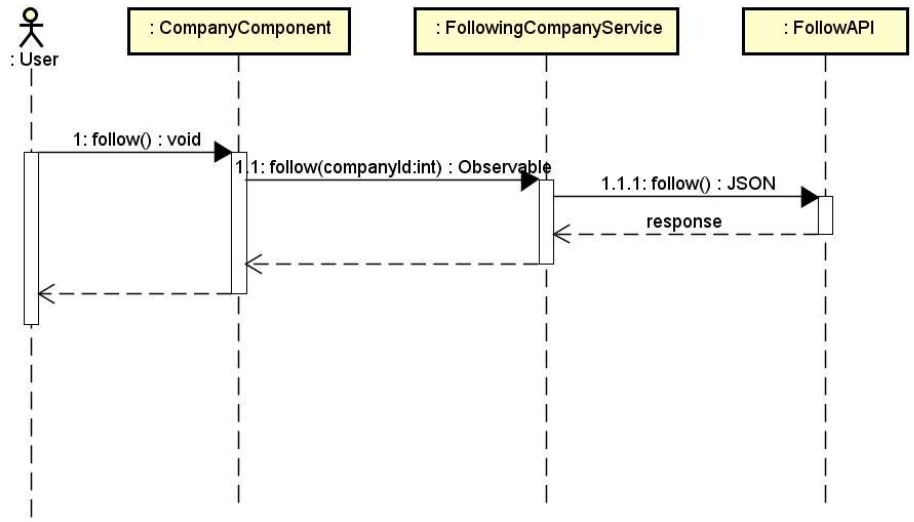


Figure 4-63: Follow/Unfollow sequence diagram

4.3.4.14 Job seeker views all following companies

Screen Design

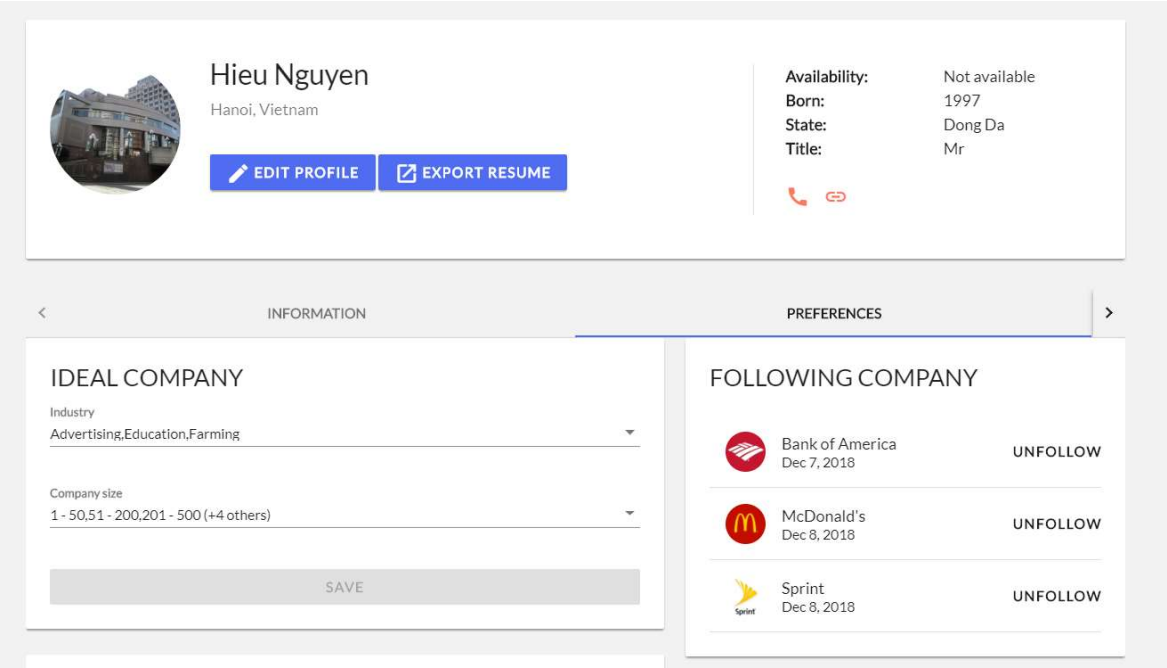


Figure 4-64: View all following companies screen design

Screen Explanation

Element	Type	Description
Company list	List	List of companies being followed
Company name	Text	Name of company
Company logo	Image	Logo of company
Date created	Text	Date created
Unfollow	Button	Unfollow the company

Class Diagram

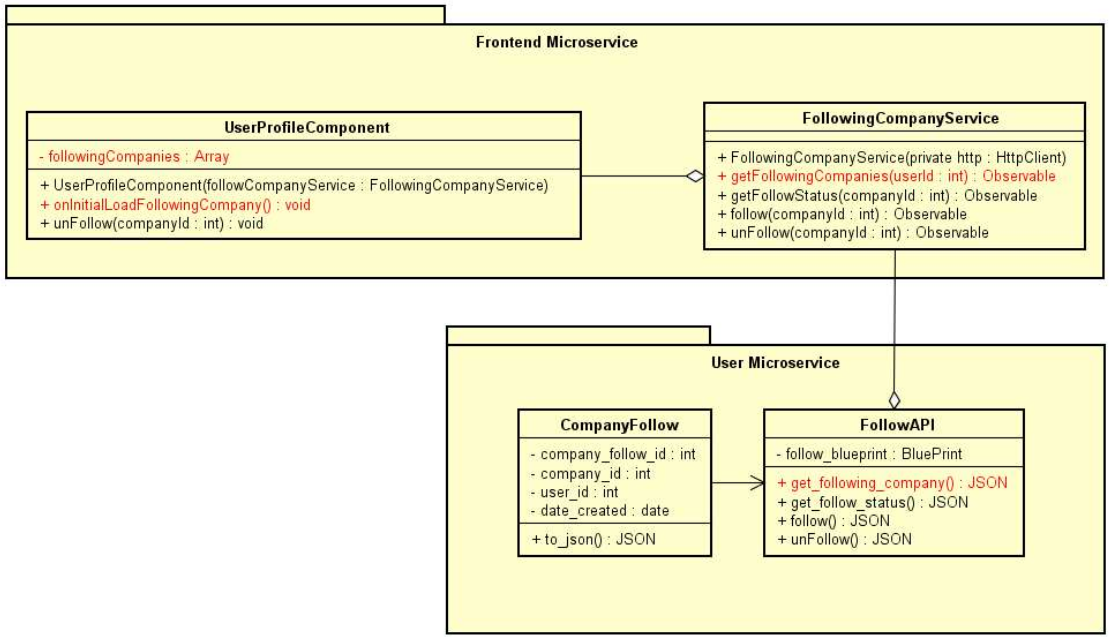


Figure 4-65: View all following companies class diagram

Class Specification

UserProfileComponent

UserProfileComponent			
Physical address	/services/frontend/src/app/user-profile/user-profile.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
1	followingCompanies	Array	List of companies followed by the user
Operation			
onInitialLoadFollowingCompany	List all companies followed by the user		
Return Type	void		
Parameters	Name	Type	Description
unFollow	Call unFollow method from FollowService, then remove the company from follow list		
Return Type	void		
Parameters	Name	Type	Description

FollowingCompanyService

FollowingCompanyService	
Physical address	/services/frontend/src/app/user-profile/following - company/following-company.service.ts

Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
getFollowingCompanies	Call Follow API get all following companies with userId as parameter		
Return Type	Observable		
Parameters	Name	Type	Description
	userId	Int	User's ID
unFollow	Call Follow API to remove a follow with companyId and userId as parameters		
Return Type	Observable		
Parameters	Name	Type	Description
	companyId	Int	Company's ID

FollowAPI

FollowAPI			
Physical address	/services/user/project/api/following-company.py		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
get_following_company	Get all companies followed by the user		
Return Type	JSON		
Parameters	Name	Parameters	Name
unFollow	Remove a company follow		
Return Type	JSON		
Parameters	Name	Parameters	Name

CompanyFollow

CompanyFollow			
Physical address	/services/company/project/api/models.py		
Base class	Class		
Attributes			
No	Name	Type	Description
1	company_follow_id	int	CompanyFollow id
2	company_id	Int	ID of company
3	user_id	Int	ID of user
4	date_created	date	Date created
Operation			
to_json	return dict of model in json key/value format		
Return Type	dict		
Parameters	Name	Type	Description
	self	CompanyFollow	CompanyFollow object

Sequence Diagram

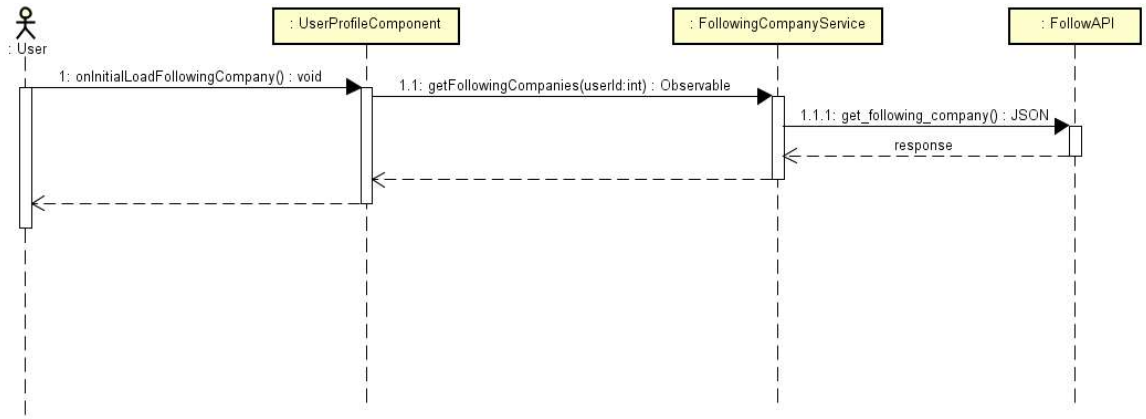


Figure 4-66: View all following companies sequence diagram

4.3.4.15 Job seeker edits preferred locations

Screen Design

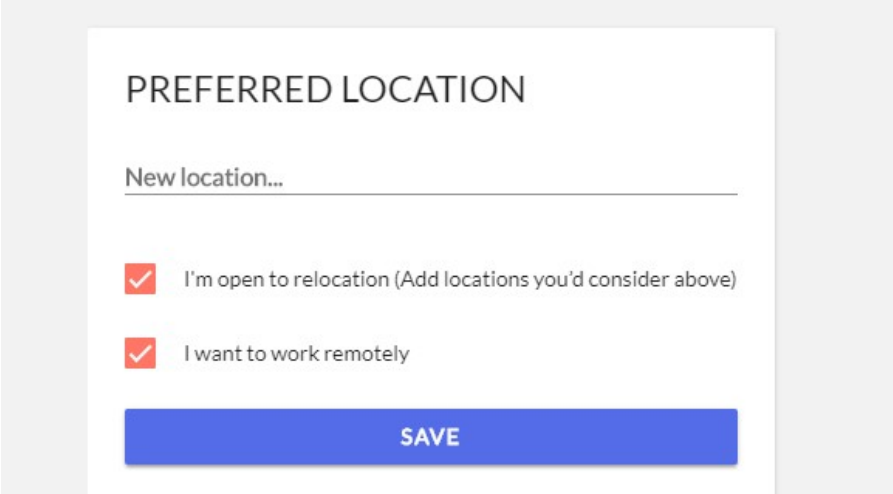


Figure 4-67: Job seeker edits preferred companies class diagram

Class Diagram

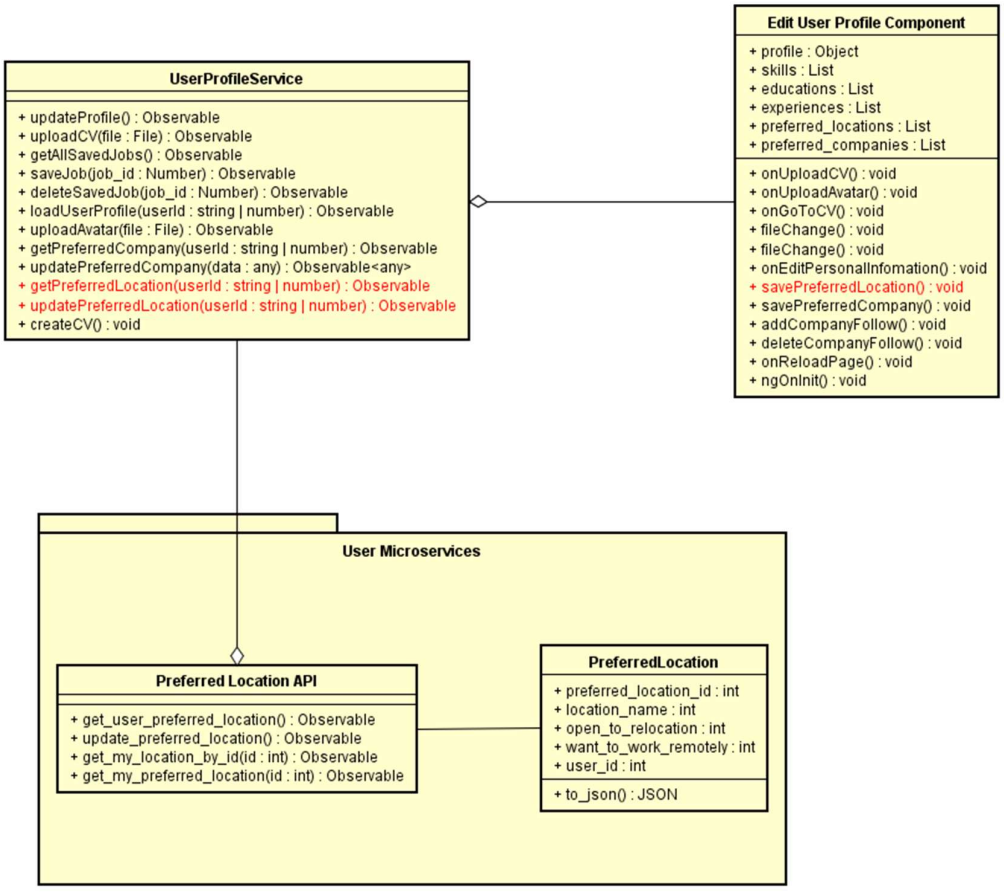


Figure 4-68: Job seeker edits preferred locations class diagram

Class Specification

UserProfileService

UserProfileService			
Physical address	services\frontend\src\app\user-profile\user-profile.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
UserProfileService	Constructor		
Return Type			
Parameters	Name	Type	Description
	httpClient	HttpClient	instance of HttpClient
	authService	Authenticat-onService	instance of AuthenticationSer-vice
getPreferredLocation	Get all user preferred locations		
Return Type	Observable		

Parameters	Name	Type	Description
	user_id	Int	User ID
updatePreferredLocation	Update user preferred company list		
Return Type	Observable		
Parameters	Name	Type	Description
	preferred_locations	List	List of all user preferred locations

User API

User API			
Physical address	services\user\project\api\users.py		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
update_preferred_location	Update user's preferred location list		
Return Type	JSON		
Parameters	Name	Type	Description
get_my_preferred_location	Get all user's preferred location list		
Return Type	JSON		
Parameters	Name	Type	Description

EditUserProfileComponent

EditUserProfileComponent			
Physical address	services\frontend\src\app\user-profile\user-profile.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
savePreferredLocation	Save updating user's list of preferred locations		
Return Type	void		
Parameters	Name	Type	Description
	preferred_locations	List	List of user's preferred locations

Sequence Diagram

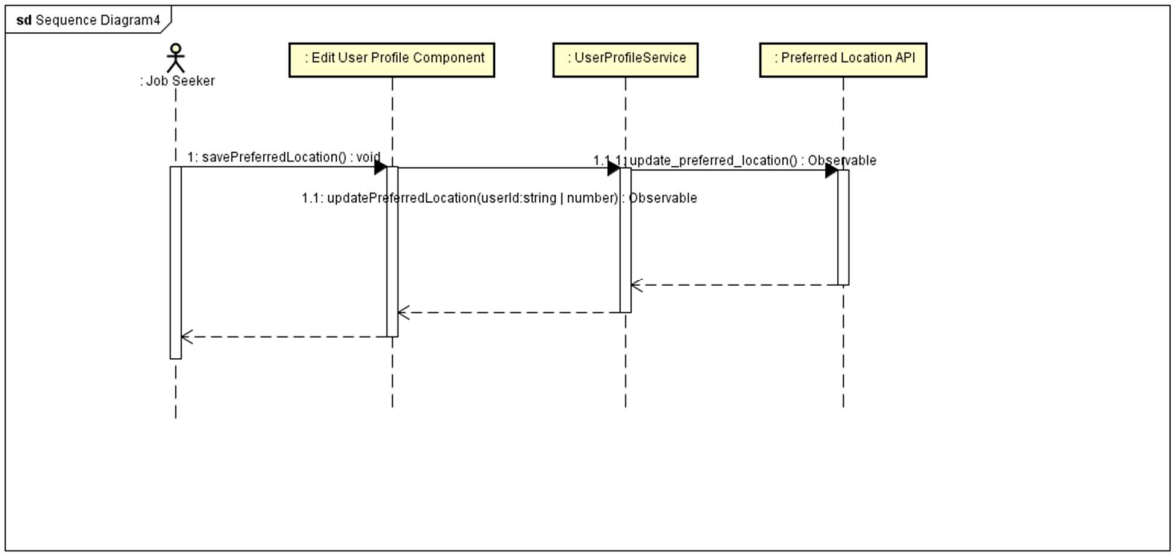


Figure 4-69: Job seeker edits preferred locations sequence diagram

4.3.4.16 Job seeker edits preferred companies

Screen Design

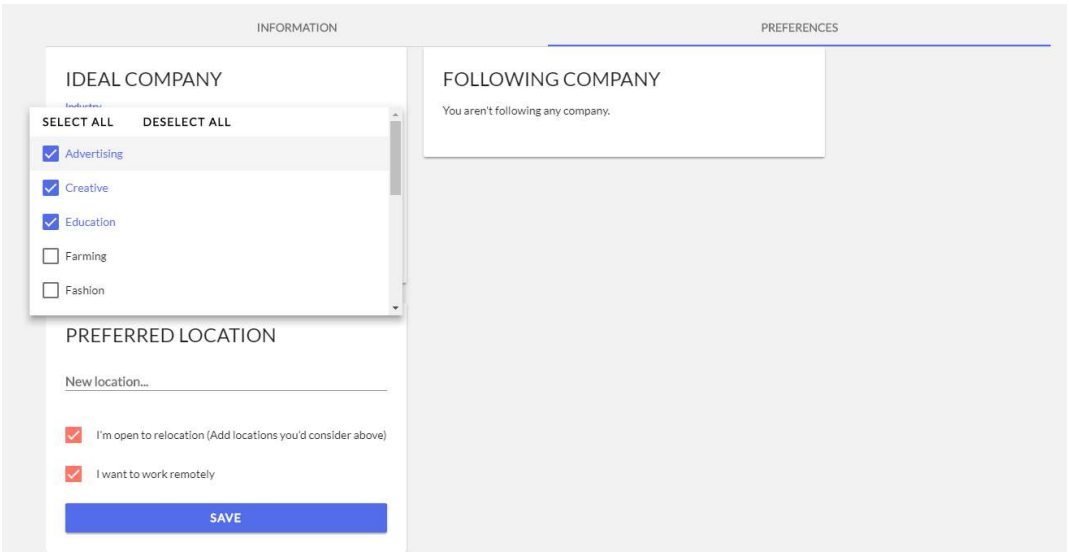


Figure 4-70: Job seeker edits preferred companies screen design

Class Diagram

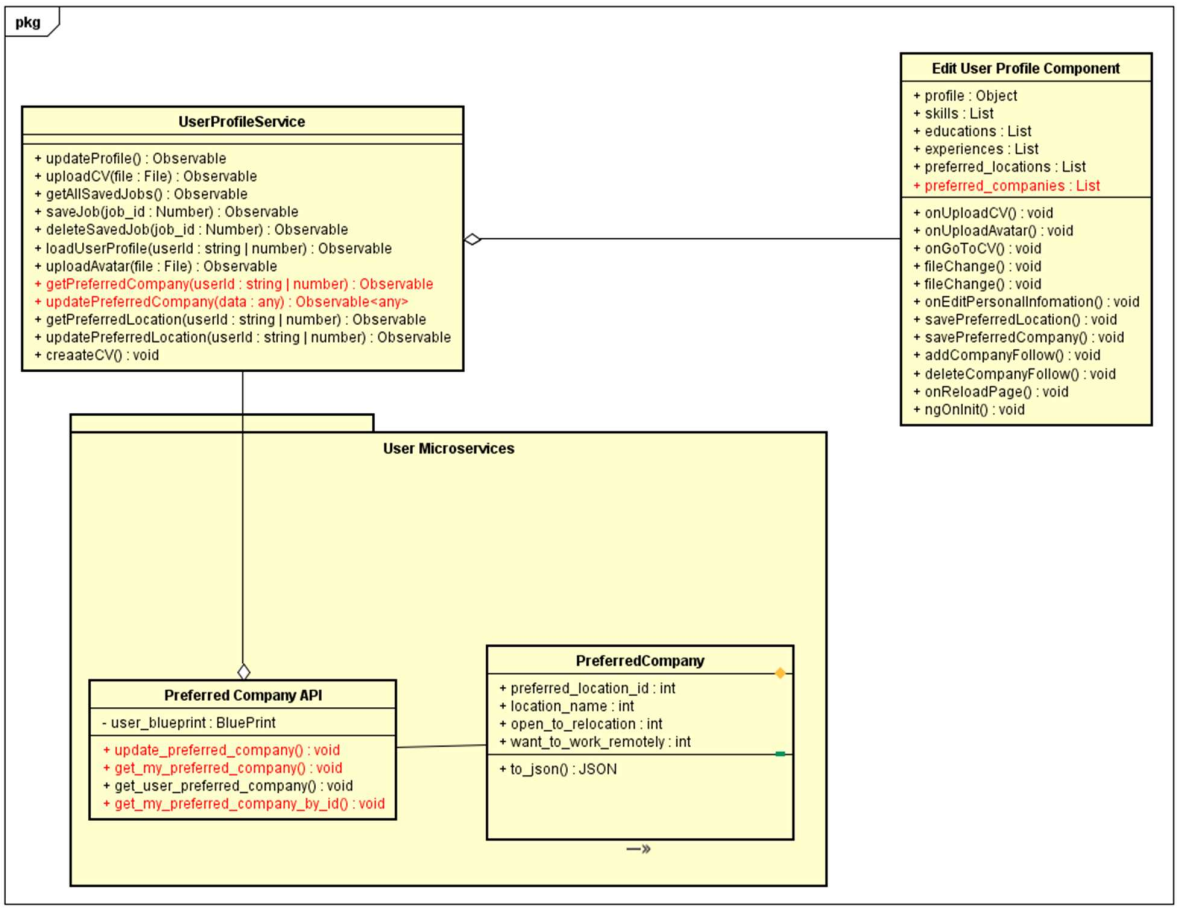


Figure 4-71: Job seeker edits preferred companies class diagram

Class Specification

UserProfileService

UserProfileService			
Physical address	services\frontend\src\app\user-profile\user-profile.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
UserProfileService	Constructor		
Return Type			
Parameters	Name	Type	Description
	httpClient	HttpClient	instance of HttpClient
	authService	Authenticat-onService	instance of AuthenticationSer-vice
getPreferredCompany	Get all user preferred company		
Return Type	Observable		
Parameters	Name	Type	Description

updatePreferredCompany	Update user preferred company list		
Return Type	Observable		
Parameters	Name	Type	Description
	preferred_company	List	List of all user preferred companies

User API

User API			
Physical address	services\user\project\api\users.py		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
update_preferred_company	Update user's preferred company list		
Return Type	JSON		
Parameters	Name	Type	Description
get_my_preferred_company	Get all user's preferred company list		
Return Type	JSON		
Parameters	Name	Type	Description

EditUserProfileComponent

EditUserProfileComponent			
Physical address	services\frontend\src\app\user-profile\user-profile.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
savePreferredCompany	Save updating user's list of preferred companies		
Return Type	void		
Parameters	Name	Type	Description
	preferred_companies	List	List of user's preferred companies

Sequence Diagram

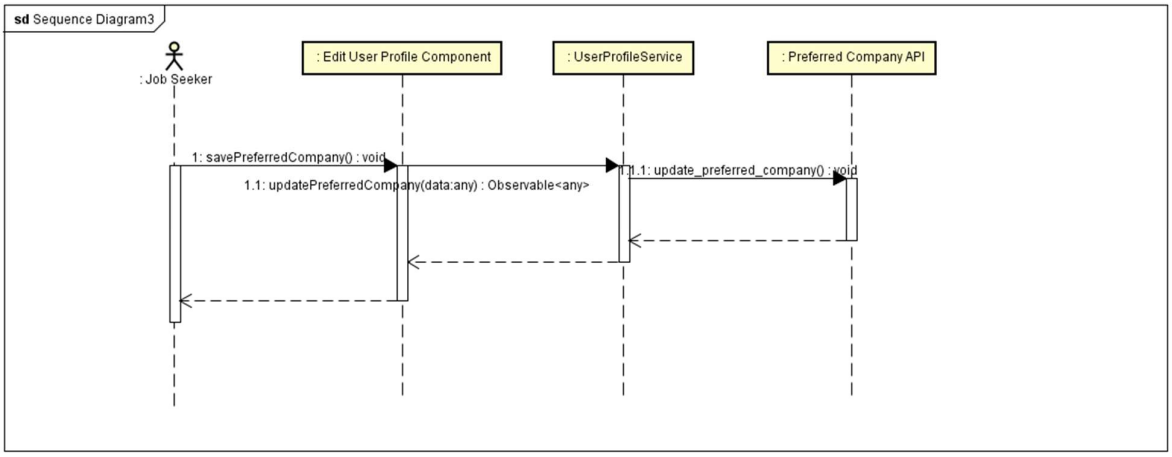


Figure 4-72: Job seeker edits preferred companies sequence diagram

4.3.4.17 Job seeker updates profile

Screen Design

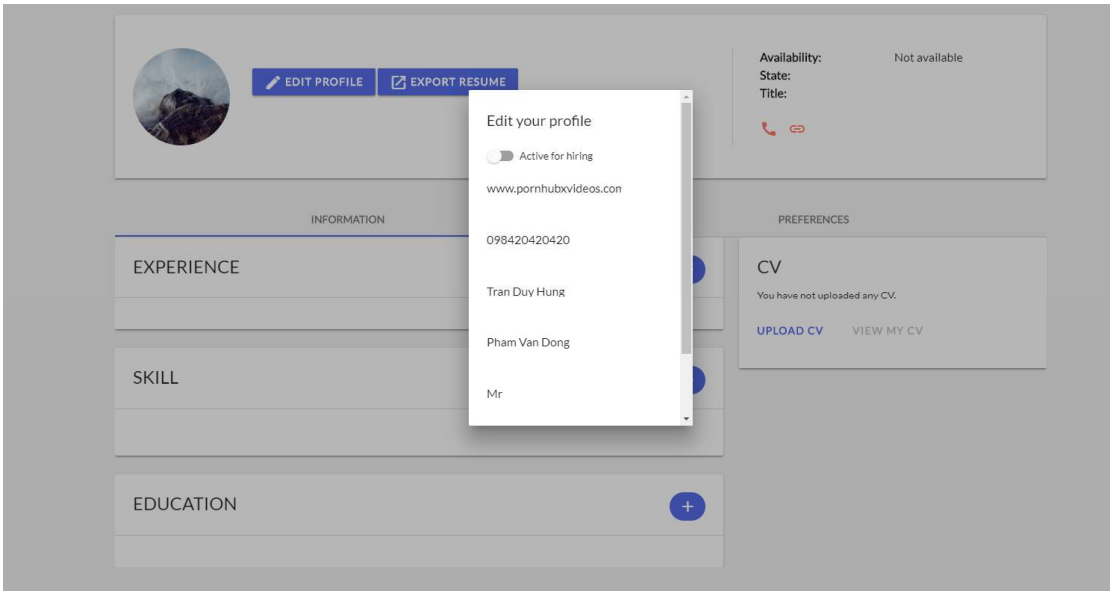


Figure 4-73: Job seeker updates profile class diagram

Class Diagram

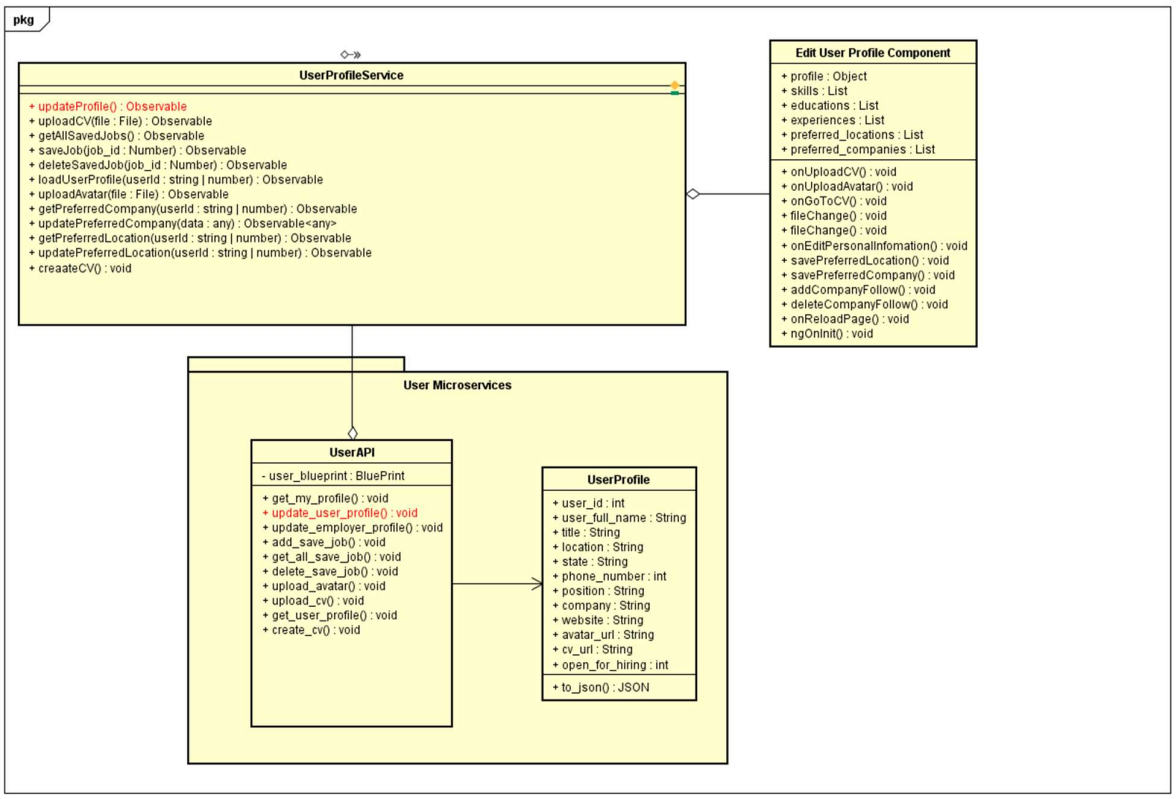


Figure 4-74: Job seeker updates profile class diagram

Class Specification

UserProfileService

UserProfileService			
Physical address	services\frontend\src\app\user-profile\user-profile.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
UserProfileService	Constructor		
Return Type			
Parameters	Name	Type	Description
	httpClient	HttpClient	instance of HttpClient
	authService	Authenticat-onService	instance of AuthenticationSer-vice
updateProfile	Update user profile with user input data		
Return Type	Observable		
Parameters	Name	Type	Description
	website	String	Website address
	phone_number	String	User's phone number

	state	String	State where user lives
	location	String	Location of the user
	open_for_hiring	int	Availability
	title	int	ID of job post

User API

User API			
Physical address	services\user\project\api\users.py		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
update_user_profile	Update user profile		
Return Type	JSON		
Parameters	Name	Type	Description
	profile	JSON	Profile details
get_my_profile	Get user profile details		
Return Type	JSON		
Parameters	Name	Type	Description
	profile	JSON	Profile details
upload_avatar	Allow user to update profile picture		
Return Type	JSON		
Parameters	Name	Type	Description
	avatar_url	JSON	URL linked to user's avatar

EditUserProfileComponent

EditUserProfileComponent			
Physical address	services\frontend\src\app\user-profile\user-profile.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
onEditPersonalInfomation()	Update user's personal details		
Return Type	void		
Parameters	Name	Type	Description
	profile	Observable	User personal details

UserProfile

UserProfile			
Physical address	services\user\project\api\models.py		
Base class	Class		
Attributes			
No	Name	Type	Description
1	user_id	int	User ID
2	location	string	User location
3	title	string	User title

4	state	string	User's state
5	phone_number	string	User's phone number
6	position	string	User's job position
7	company	string	User's company
8	website	string	User's website
9	avatar_url	string	User's avatar URL
10	cv_url	string	User's CV URL
11	open_for_hiring	int	User's availability for hiring
Operation			
to_json	return JSON of model		
Return Type	JSON		
Parameters	Name	Type	Description

Sequence Diagram

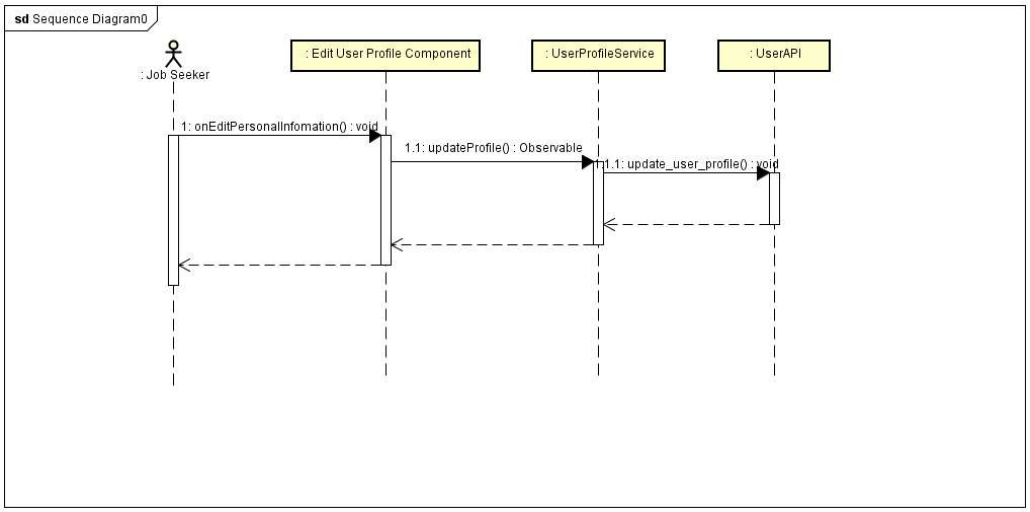


Figure 4-75: Job seeker updates profile sequence diagram

4.3.4.18 Job seeker uploads CV

Screen Design

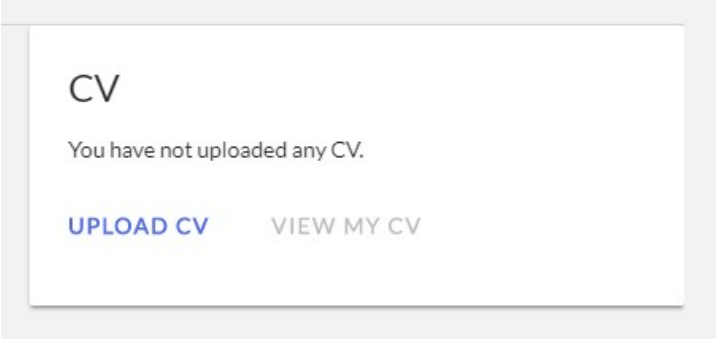


Figure 4-76: Job seeker uploads CV screen design

Class Diagram

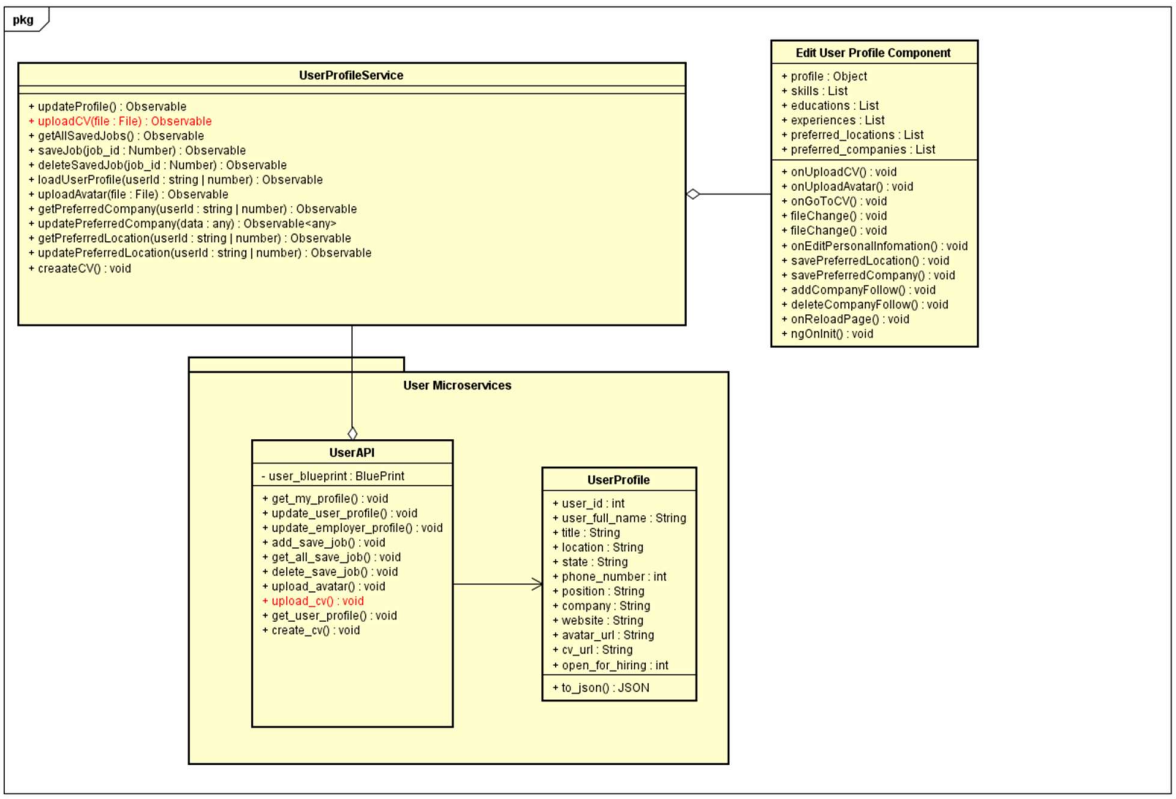


Figure 4-77: Job seeker uploads CV class diagram

Class Specification

UserProfileService

UserProfileService			
Physical address	services\frontend\src\app\user-profile\user-profile.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
UserProfileService	Constructor		
Return Type			
Parameters	Name	Type	Description
	httpClient	HttpClient	instance of HttpClient
	authService	Authenticat-onService	instance of AuthenticationSer-vice
uploadCV	Upload user's created CV		
Return Type	Observable		
Parameters	Name	Type	Description
	file (.pdf)	File	User .pdf format CV

User API

User API			
Physical address	services\user\project\api\users.py		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
upload_cv	Upload user's created CV to Google Storage		
Return Type	JSON		
Parameters	Name	Type	Description
	file	File	User serializable binary file

EditUserProfileComponent

EditUserProfileComponent			
Physical address	services\frontend\src\app\user-profile\user-profile.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
onUploadCV	Upload user's CV to server storage service		
Return Type	void		
Parameters	Name	Type	Description
	file (.pdf)	File	User .pdf format CV

UserProfile

UserProfile			
Physical address	services\user\project\api\models.py		
Base class	Class		
Attributes			
No	Name	Type	Description
1	user_id	int	User ID
2	location	string	User location
3	title	string	User title
4	state	string	User's state
5	phone_number	string	User's phone number
6	position	string	User's job position
7	company	string	User's company
8	website	string	User's website
9	avatar_url	string	User's avatar URL
10	cv_url	string	User's CV URL
11	open_for_hiring	int	User's availability for hiring
Operation			
to_json	return JSON of model		
Return Type	JSON		
Parameters	Name	Type	Description

Sequence Diagram

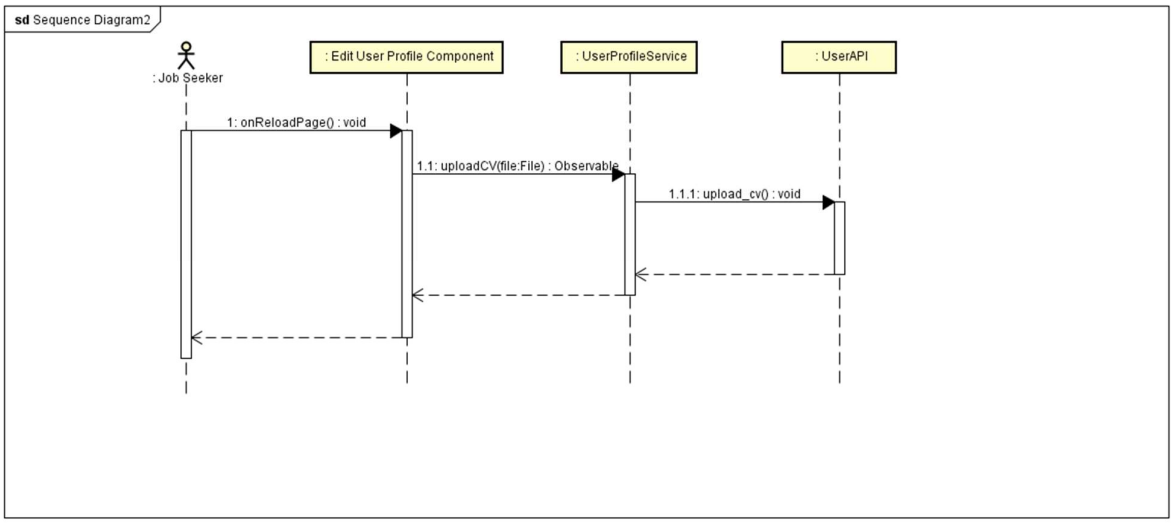


Figure 4-78: Job seeker uploads CV sequence diagram

4.3.4.19 Job seeker saves a job

Screen Design

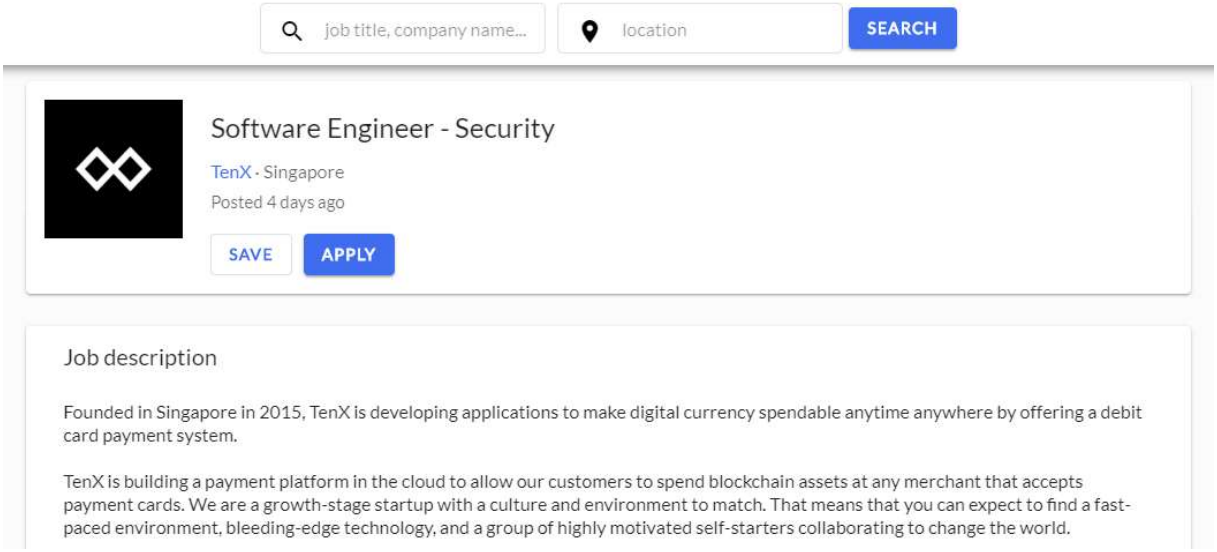


Figure 4-79: Job seeker saves a job screen design

Class Diagram

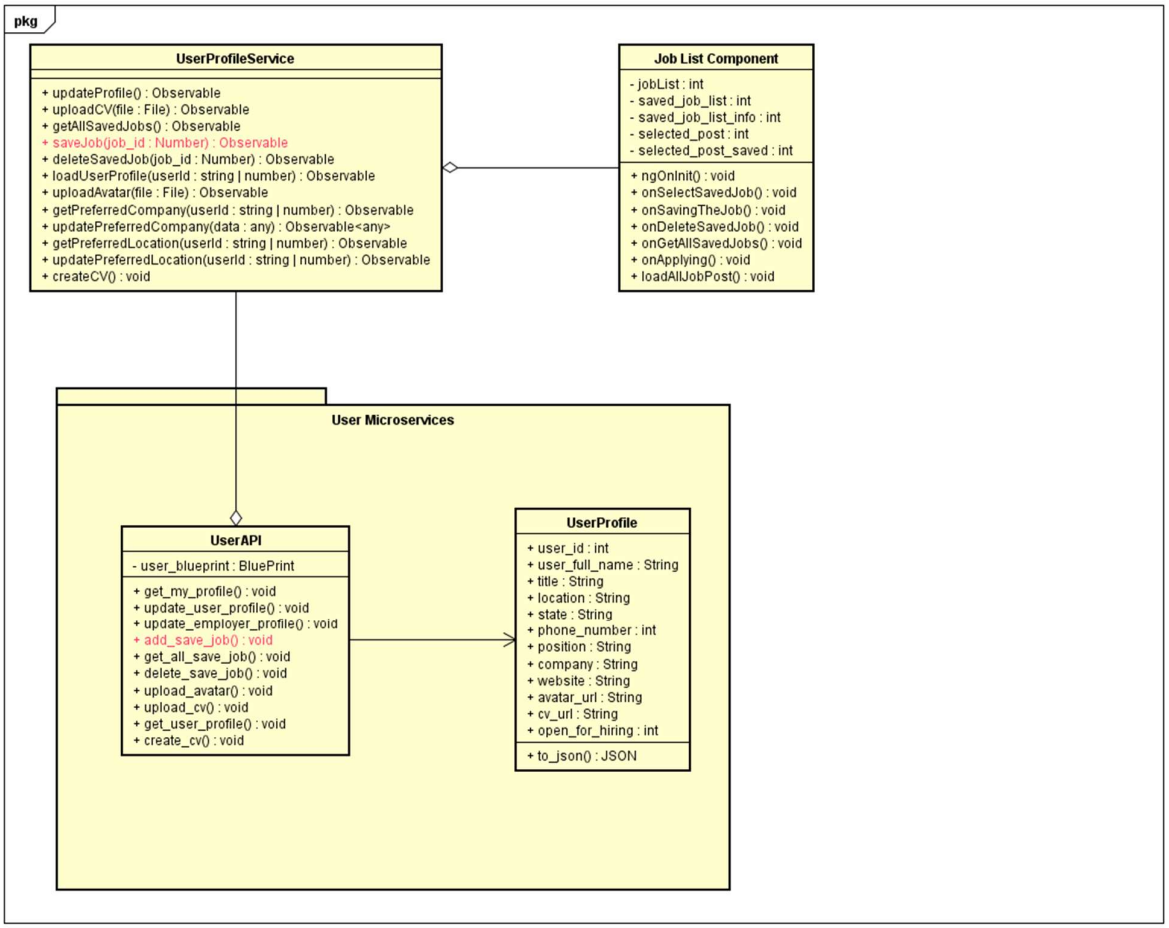


Figure 4-80: Job seeker saves a job class diagram

Class Specification

UserProfileService

UserProfileService			
Physical address	services\frontend\src\app\user-profile\user-profile.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
UserProfileService	Constructor		
Return Type			
Parameters	Name	Type	Description
	httpClient	HttpClient	instance of HttpClient
	authService	Authenticat-onService	instance of AuthenticationSer-vice
saveJob	Save a job post by the user		
Return Type	Observable		

Parameters	Name	Type	Description
	job_id	Int	Job ID of the job

User API

User API			
Physical address	services\user\project\api\users.py		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
add_save_job	Add user's saved job post to database		
Return Type	JSON		
Parameters	Name	Type	Description
	job_id	int	Job ID of the job saved

EditUserProfileComponent

EditUserProfileComponent			
Physical address	services\frontend\src\app\user-profile\user-profile.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
onSavingTheJob	Save the job		
Return Type	void		
Parameters	Name	Type	Description
	Job_post_id	Int	Job ID
onGetAllSavedJob	Get all the jobs saved by user		
Return Type	void		
Parameters	Name	Type	Description

Sequence Diagram

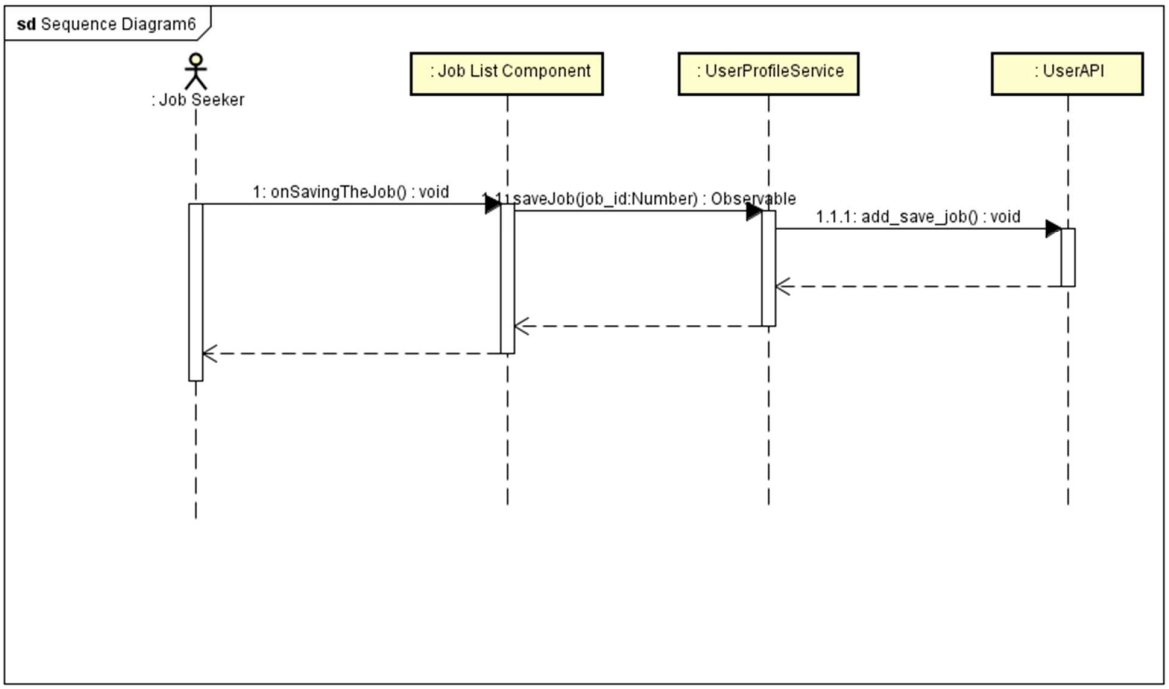


Figure 4-81: Job seeker saves a job sequence diagram

4.3.4.20 Job seeker creates mail notification

4.3.4.21 Job seeker updates/deletes saved jobs

Class Diagram

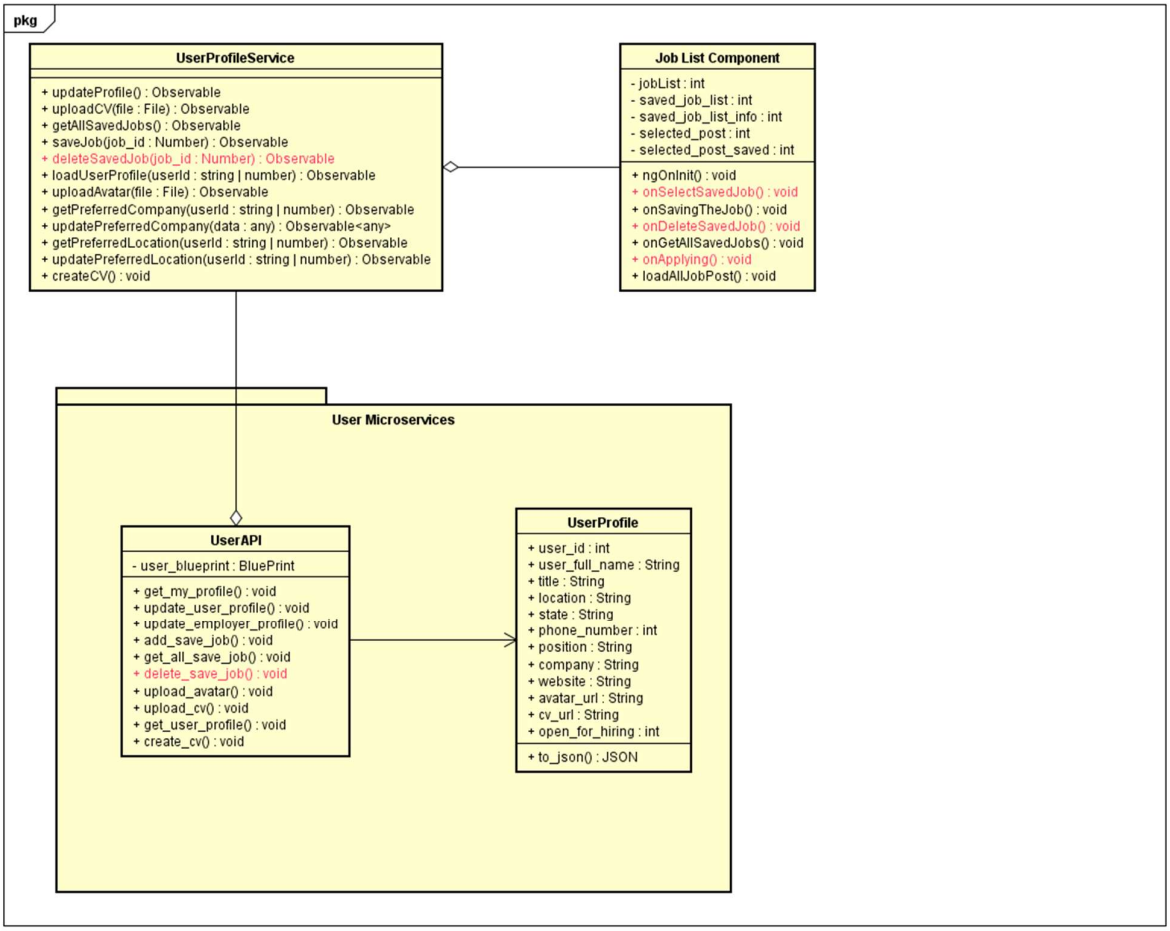


Figure 4-82: Job seeker deletes saved jobs class diagram

Class Specification

UserProfileService

UserProfileService			
Physical address	services\frontend\src\app\user-profile\user-profile.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
UserProfileService	Constructor		
Return Type			
Parameters	Name	Type	Description
	httpClient	HttpClient	instance of HttpClient
	authService	Authenticat-onService	instance of AuthenticationSer-vice
deleteSavedJob	Delete user's saved job		
Return Type	Observable		

Parameters	Name	Type	Description
	job_id	Int	Job ID of the job

User API

User API			
Physical address	services\user\project\api\users.py		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
delete_save_job	Delete user's saved job post from database		
Return Type	JSON		
Parameters	Name	Type	Description
	job_id	int	Job ID of the job saved

EditUserProfileComponent

EditUserProfileComponent			
Physical address	services\frontend\src\app\user-profile\user-profile.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
onDeleteSavedJob	Delete the job post saved by the user		
Return Type	void		
Parameters	Name	Type	Description
	job_post_id	Int	Job ID

Sequence Diagram

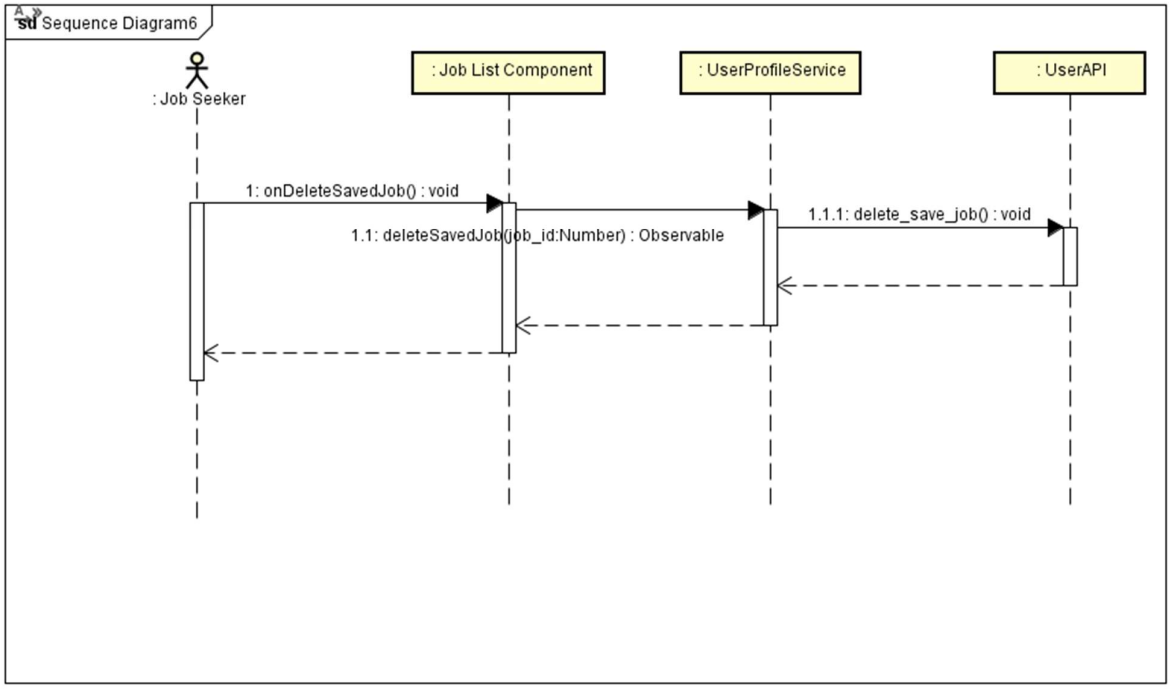


Figure 4-83: Job seeker deletes saved jobs sequence diagram

4.3.4.22 Job seeker applies for a job

Screen Design

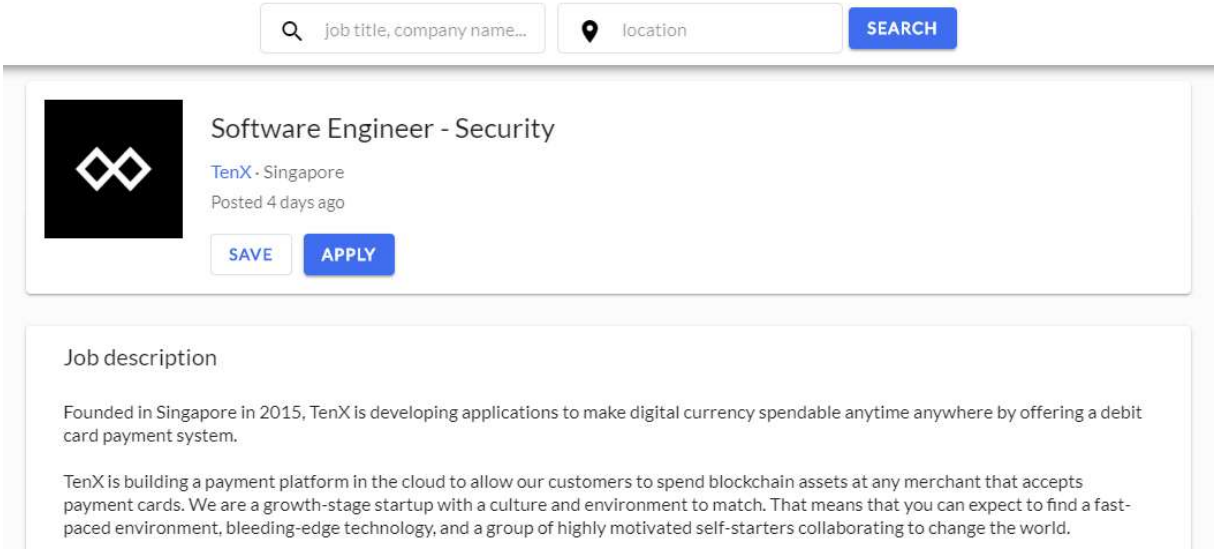


Figure 4-84: Job seeker applies for a job screen design

Class Diagram

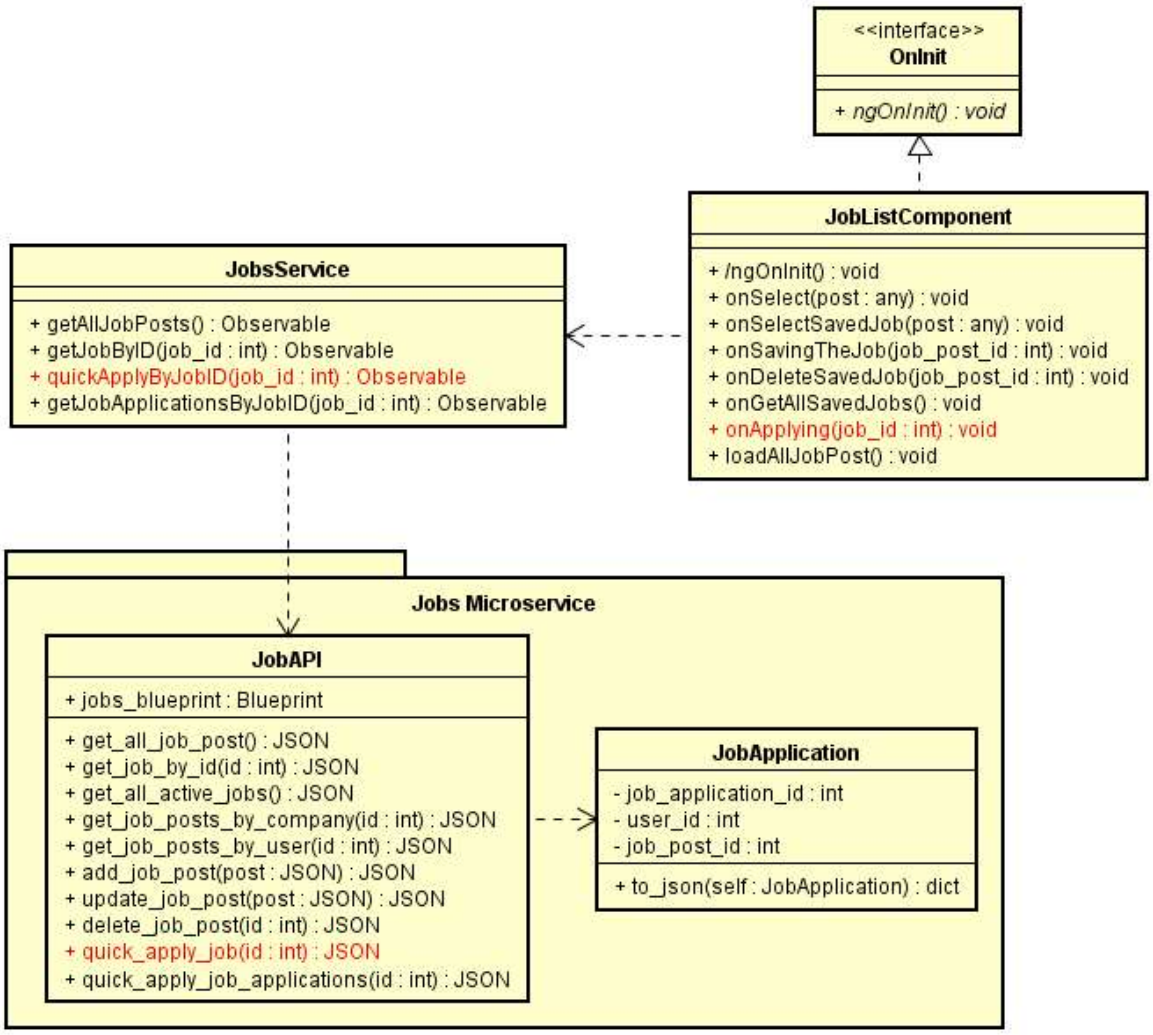


Figure 4-85: Job seeker applies for a job class diagram

Class Specification

JobsService

JobsService			
Physical address	/services/frontend/src/app/jobs/jobs.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
quickApplyByJobID	Quick apply for job post		
Return Type	Observable		
Parameters	Name	Type	Description
	job_id	int	Job post ID

JobListComponent

JobListComponent

Physical address	/services/frontend/src/app/employer/employer.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
ngOnInit	Page initialization, used to populate job posts		
Return Type			
Parameters	Name	Type	Description
onSelect	Action to do when a job post is selected		
Return Type	void		
Parameters	Name	Type	Description
	post	Observable	selected job post

JobAPI

JobAPI			
Physical address	/services/job/project/api/jobs.py		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
quick_apply_job_applications	Get applicants for selected job post		
Return Type	JSON		
Parameters	Name	Parameters	Name
	id		Job post ID

JobApplication

JobApplication			
Physical address	/services/job/project/api/models.py		
Base class	Class		
Attributes			
No	Name	Type	Description
1	job_application_id	int	Job Application ID
2	user_id	int	User ID
3	job_post_id	int	Job Post ID
Operation			
to_json	return dict of model in json key/value format		
Return Type	dict		
Parameters	Name	Type	Description
	self	JobApplication	Job application object

Sequence Diagram

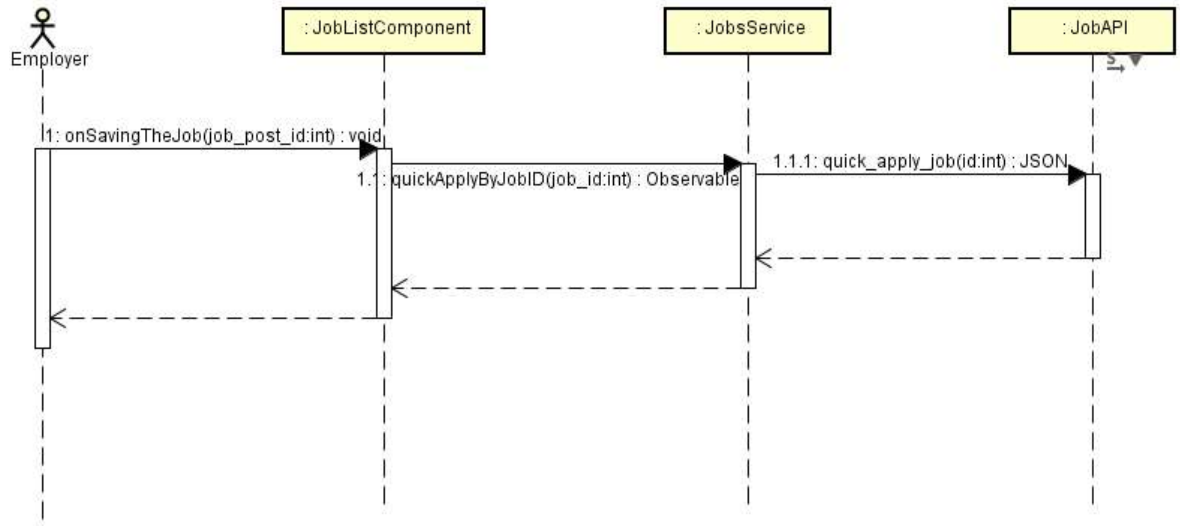


Figure 4-86: Job seeker applies for a job sequence diagram

4.3.4.23 Job seeker views all available jobs

4.3.4.24 Job seeker views recommended job posts

Screen Design

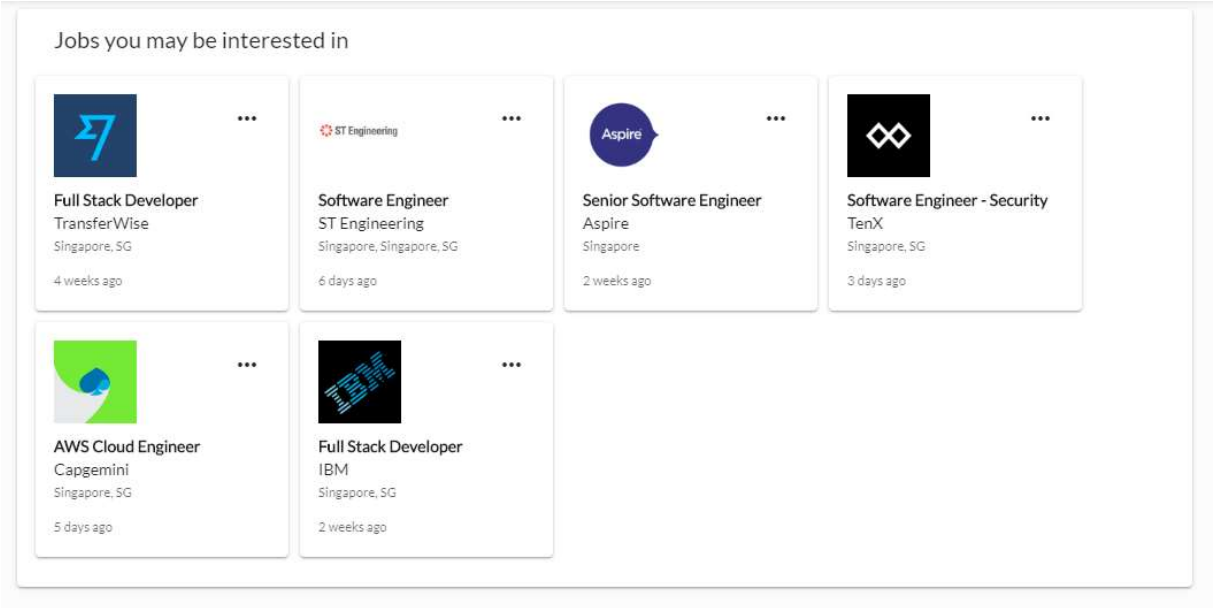


Figure 4-87: Job seeker views recommended job posts screen design

Class Diagram

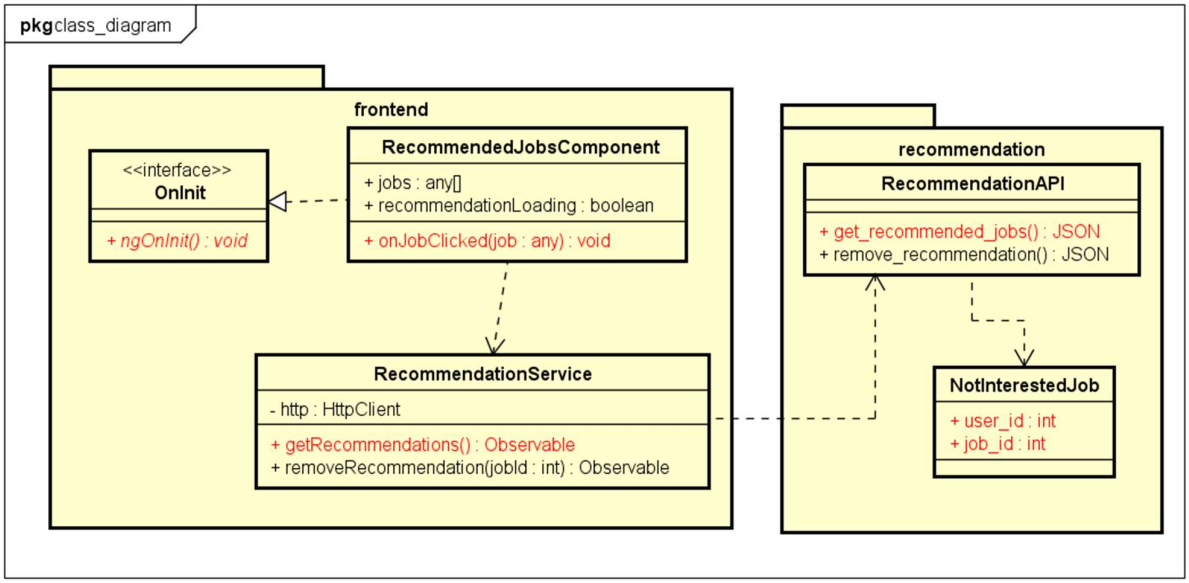


Figure 4-88: Job seeker views recommended job posts class diagram

Class Specification

RecommendationService

RecommendationService			
Physical address	/services/frontend/src/app/recommended-jobs/recommendation.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
getRecommendations	Get recommended job posts		
Return Type	Observable		
Parameters	Name	Type	Description

RecommendedJobsComponent

RecommendedJobsComponent			
Physical address	/services/frontend/src/app/employer/employer.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
ngOnInit	Page initialization, used to populate job posts		
Return Type			
Parameters	Name	Type	Description
onJobClicked	Action to do when a job post is clicked		

Return Type	void		
Parameters	Name	Type	Description
	job	any	Job ID

RecommendationAPI

RecommendationAPI			
Physical address	/services/recommendation/project/api/recommendations.py		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
get_recommended_jobs	Get recommended jobs for current user		
Return Type	JSON		
Parameters	Name	Parameters	Name

Implementation

Pseudo code:

```
Set user_id to current user ID.
Load saved user_embeddings, item_embeddings, item_biases from file on disk
Set past_items to the list of items that use has already interacted with.
Set not_interested to the list of items that user has marked as not being interested
in.
Set jobs_to_score = list of job IDs excluding past items
Initialize score_dictionary
For each job in jobs_to_score:
   score = dot product of user_embeddings[job] and (item_embeddings[job] +
item_biases[job])
   add score to score_dictionary
Rank scores in score_dictionary
Return top 10 jobs that have the highest score
```

Sequence Diagram

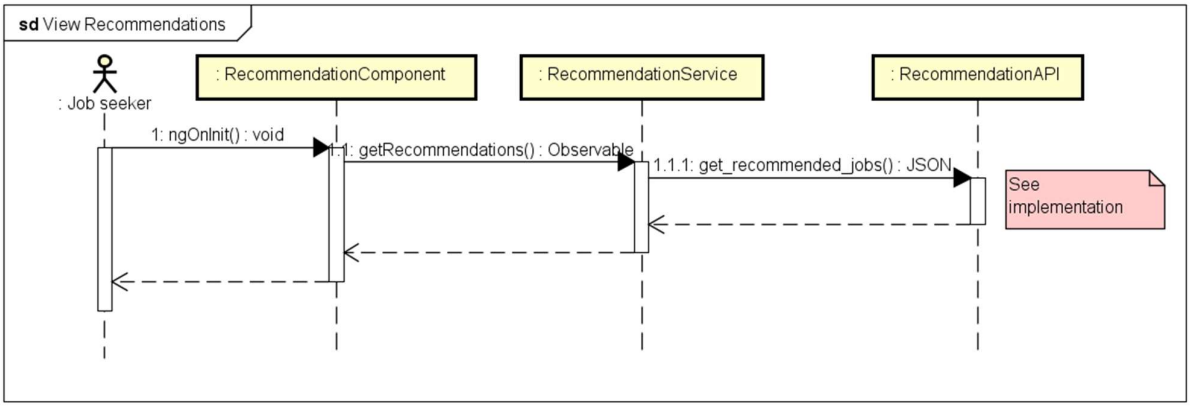


Figure 4-89: Job seeker views recommended job posts sequence diagram

4.3.4.25 Job seeker removes a recommended job post

Screen Design

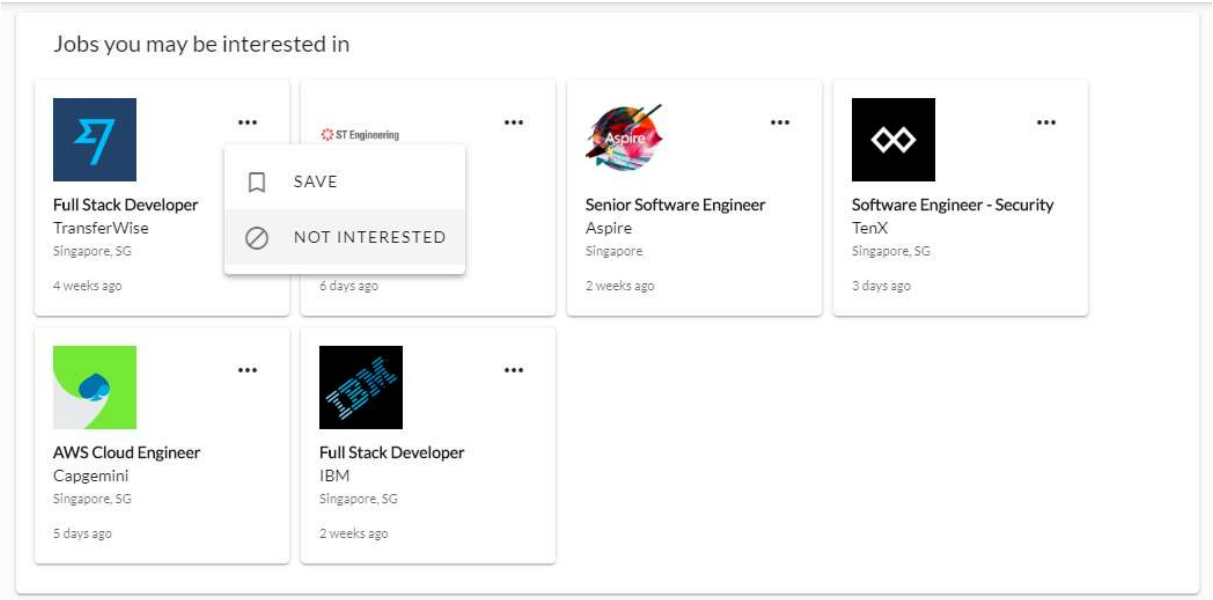


Figure 4-90: Job seeker removes a recommended job post screen design

Class Diagram

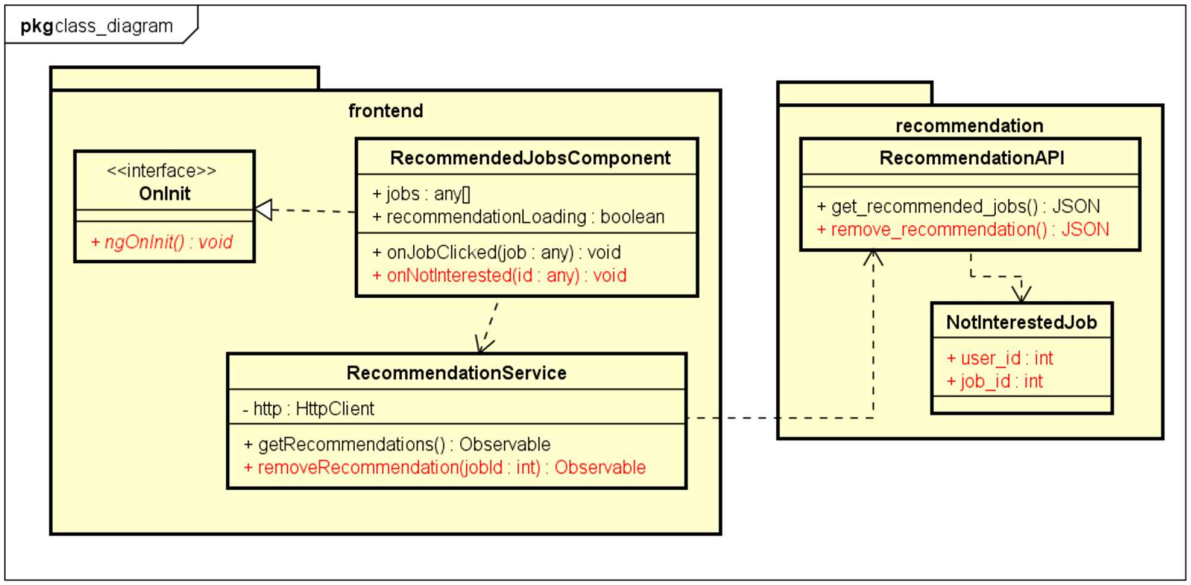


Figure 4-91: Job seeker removes recommended job post class diagram

Class Specification

RecommendationService	
Physical address	/services/frontend/src/app/recommended-jobs/recommendation.service.ts
Base class	Class
Attributes	

No	Name	Type	Description
Operation			
removeRecommendation	Remove a recommended job post		
Return Type	Observable		
Parameters	Name	Type	Description
	jobId	number	Job ID

RecommendedJobsComponent

RecommendedJobsComponent			
Physical address	/services/frontend/src/app/employer/employer.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
ngOnInit	Page initialization, used to populate job posts		
Return Type			
Parameters	Name	Type	Description
onNotInterested	Action to do when a recommended job post is removed		
Return Type	void		
Parameters	Name	Type	Description
	id	any	Job ID

RecommendationAPI

RecommendationAPI			
Physical address	/services/recommendation/project/api/recommendations.py		
Base class	Blueprint		
Attributes			
No	Name	Type	Description
Operation			
remove_recommendation	Remove recommended jobs		
Return Type	JSON		
Parameters	Name	Parameters	Name

NotInterestedJob

NotInterestedJob			
Physical address	/services/recommendation/project/api/models.py		
Base class	Model		
Attributes			
No	Name	Type	Description
1	user_id	int	User ID
2	job_id	int	Job ID

Implementation

Pseudo code:

```
Set user_id to current user ID.
```

```
Load saved model from disk.
Fit model with new data.
Save new user_embeddings, item_embeddings, item_biases to disk.
```

Sequence Diagram

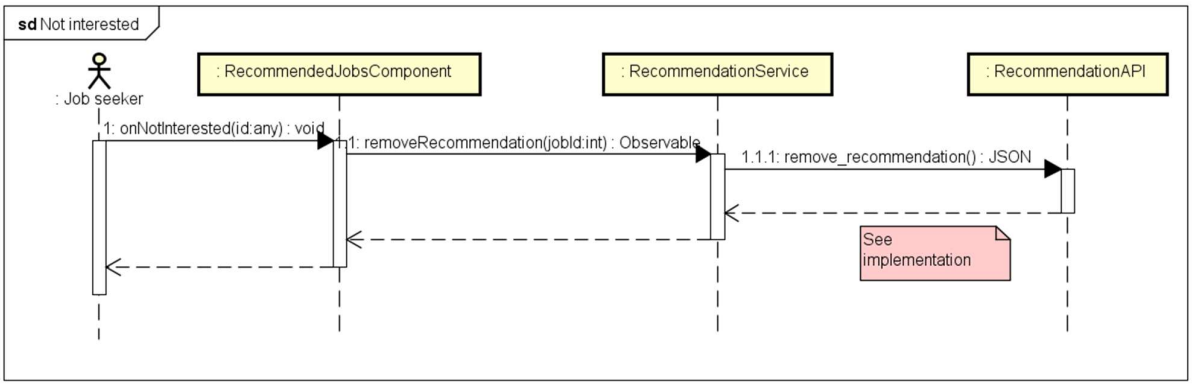


Figure 4-92: Job seeker removes recommended job post sequence diagram

4.3.4.26 Employer creates a job post

Screen Design

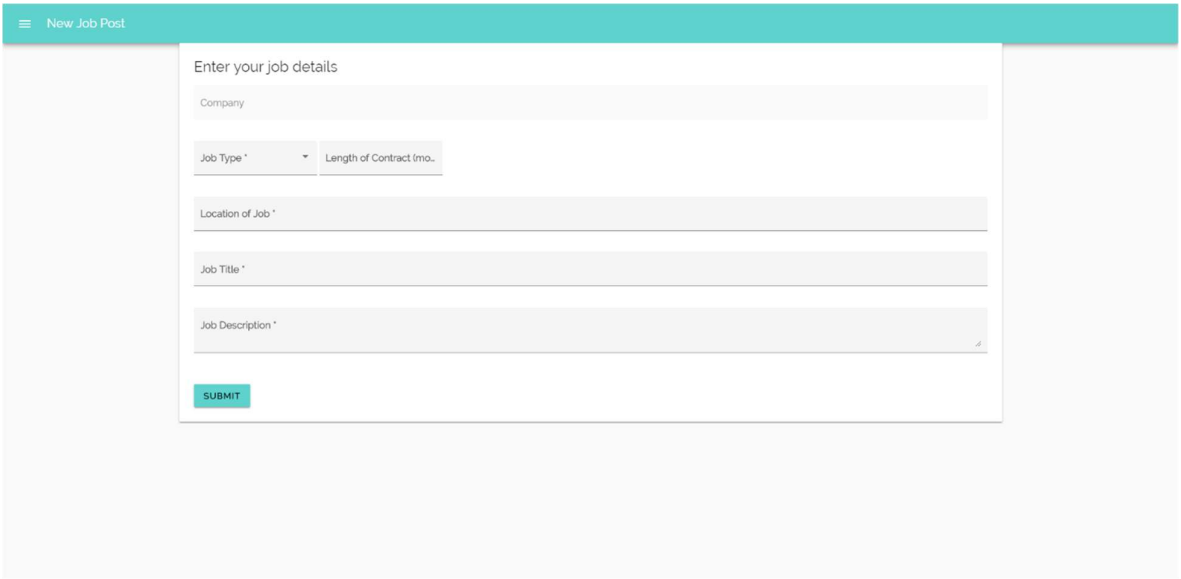


Figure 4-93: Employer user add job post

Screen Explanation

Element	Type	Description
Company	Disabled text input	Show company name for the job post
Job Type	Drop-down Input	Choose job type between On-site, Off-site

Length of Contract	Input number	Length of job contract in months
Location of Job	Input text	Location where job is situated
Job Title	Input text	Title of job position held
Job Description	Text area	Detailed description of job

Class Diagram

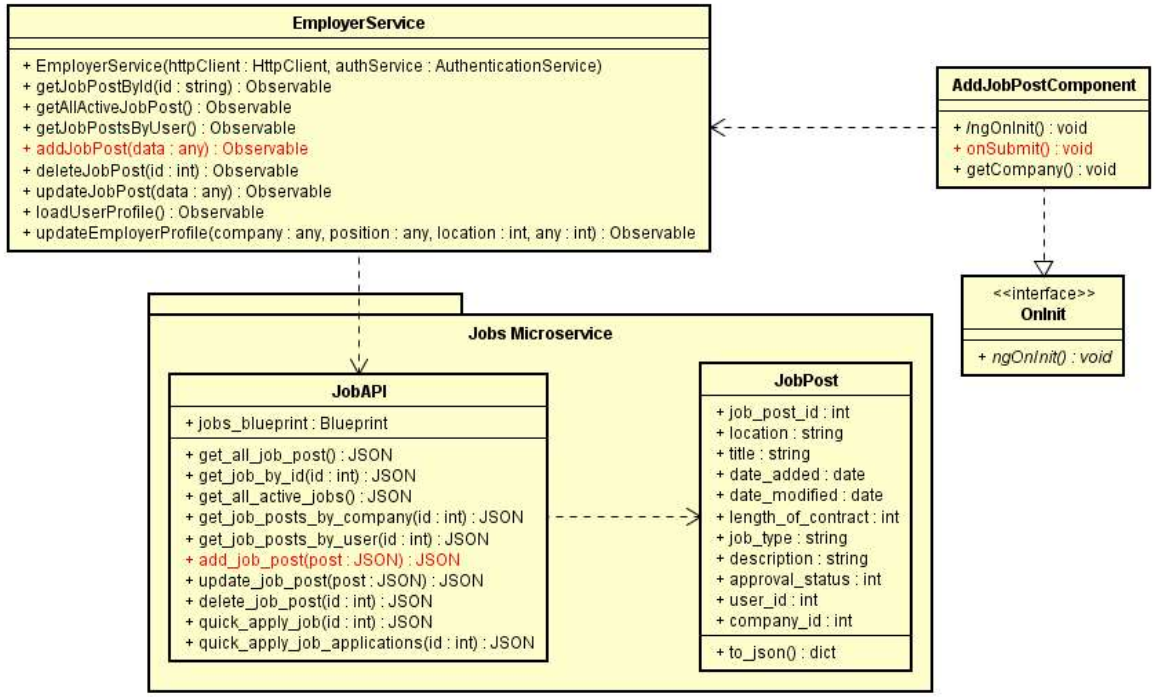


Figure 4-94: Employer user add new job class diagram

Class Specification

EmployerService

EmployerService			
Physical address	/services/frontend/src/app/employer/employer.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
EmployerService	Constructor		
Return Type			
Parameters	Name	Type	Description
	httpClient	HttpClient	instance of HttpClient

	authService	Authenticat-onService	instance of AuthenticationSer-vice
getJobPostById	Get Job post by post ID		
Return Type	Observable		
Parameters	Name	Type	Description
	id	int	ID of job post
getAllActiveJobPosts	Get all active job posts that are posted by current user		
Return Type	Observable		
Parameters	Name	Type	Description
getJobPostsByUser	Get all job posts posted by current user		
Return Type	Observable		
Parameters	Name	Type	Description
addJobPost	Add new job post		
Return Type	Observable		
Parameters	Name	Type	Description
	data	any	job post data
deleteJobPost	Delete selected job post		
Return Type	Observable		
Parameters	Name	Type	Description
	id	int	ID of job post
updateJobPost	Update selected job post		
Return Type	Observable		
Parameters	Name	Type	Description
	data	any	job post data
loadUserProfile	Load user profile of logged in user		
Return Type	Observable		
Parameters	Name	Type	Description
updateEmployerProfile	Update employer profile of logged in user		
Return Type	Observable		
Parameters	Name	Type	Description
	company	any	company name
	position	any	job position
	location	any	work location

AddJobPostComponent

AddJobPostComponent			
Physical address	/services/frontend/src/app/employer/edit-job-post/add-job-post.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
getCompany	Get company by company ID		
Return Type			
Parameters	Name	Type	Description

	id	int	company ID
onSubmit	Submit button action		
Return Type			
Parameters	Name	Type	Description

JobAPI

EmployerService			
Physical address	/services/job/project/api/jobs.py		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
add_job_post	Add new job post to database		
Return Type	JSON		
Parameters	Name	Type	Description

JobPost

JobPost			
Physical address	/services/job/project/api/models.py		
Base class	Class		
Attributes			
No	Name	Type	Description
1	job_post_id	int	Job post id
2	location	string	job location
3	title	string	job title
4	date_added	datetime	when post was added
5	date_modified	datetime	when post was last modified
6	length_of_contract	int	length of contract (in months)
7	job_type	string	type of job
8	description	string	job description
9	approval_status	int	approval status
10	user_id	int	user ID of poster
11	company_id	int	ID of company of job post
Operation			
to_json	return dict of model in json key/value format		
Return Type	dict		
Parameters	Name	Type	Description
	self	JobPost	Job post object

Sequence Diagram

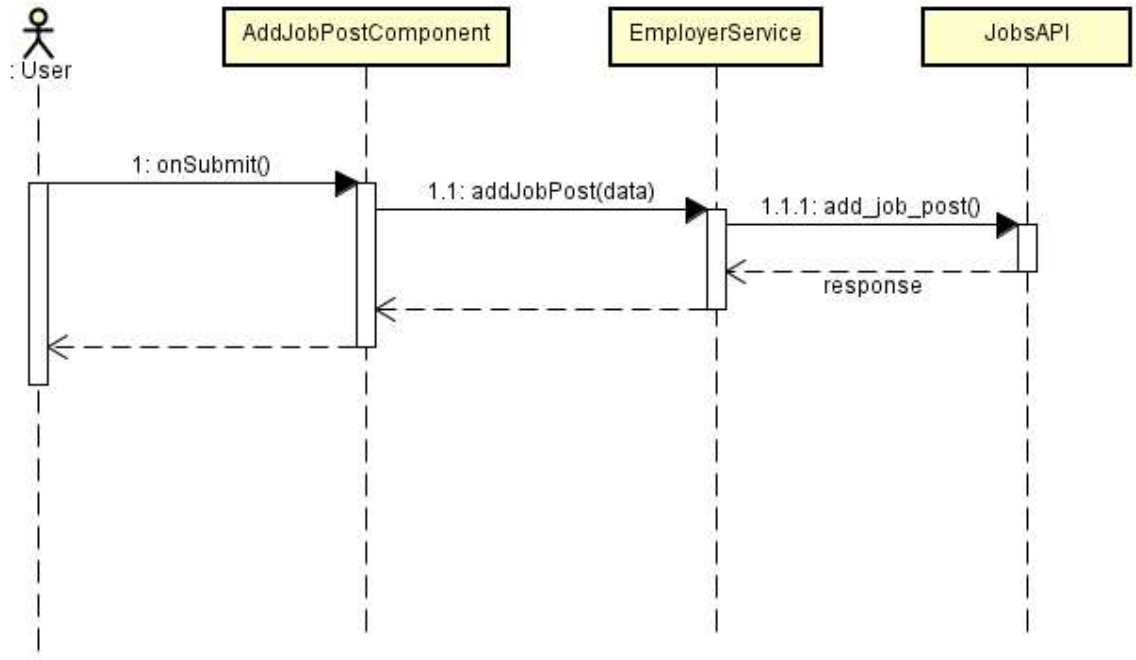


Figure 4-95: Employer user add job post

4.3.4.27 Employer edits/deletes created job posts

Screen Design

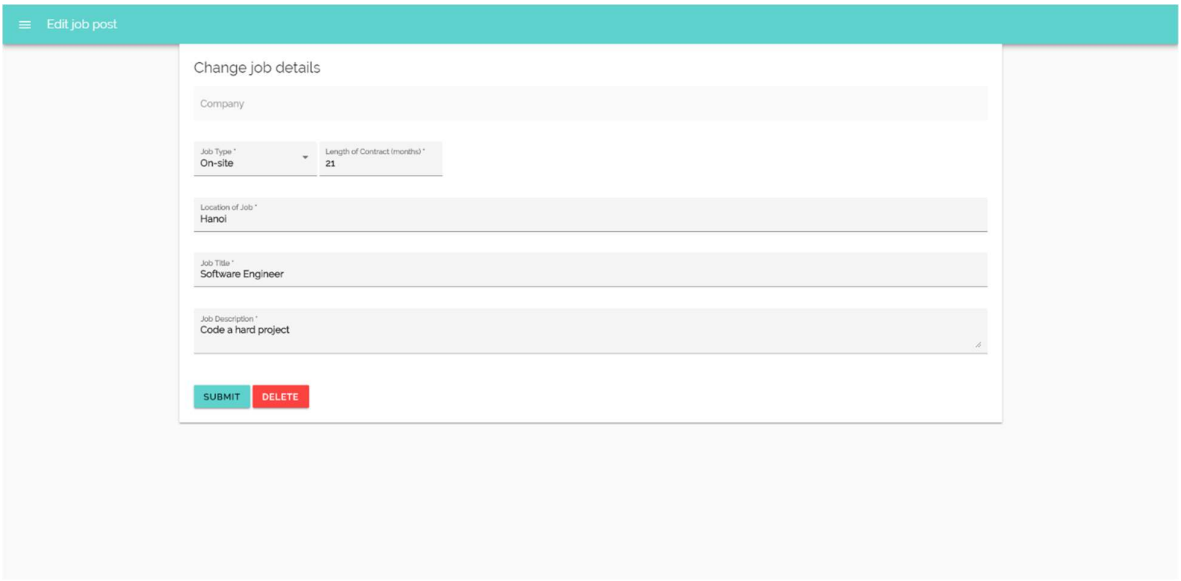


Figure 4-96: Employer edits/deletes a job post screen design

Screen Explanation

Element	Type	Description

Company	Disabled text input	Show company name for the job post
Job Type	Drop-down Input	Choose job type between On-site, Off-site
Length of Contract	Input number	Length of job contract in months
Location of Job	Input text	Location where job is situated
Job Title	Input text	Title of job position held
Job Description	Text area	Detailed description of job

Class Diagram

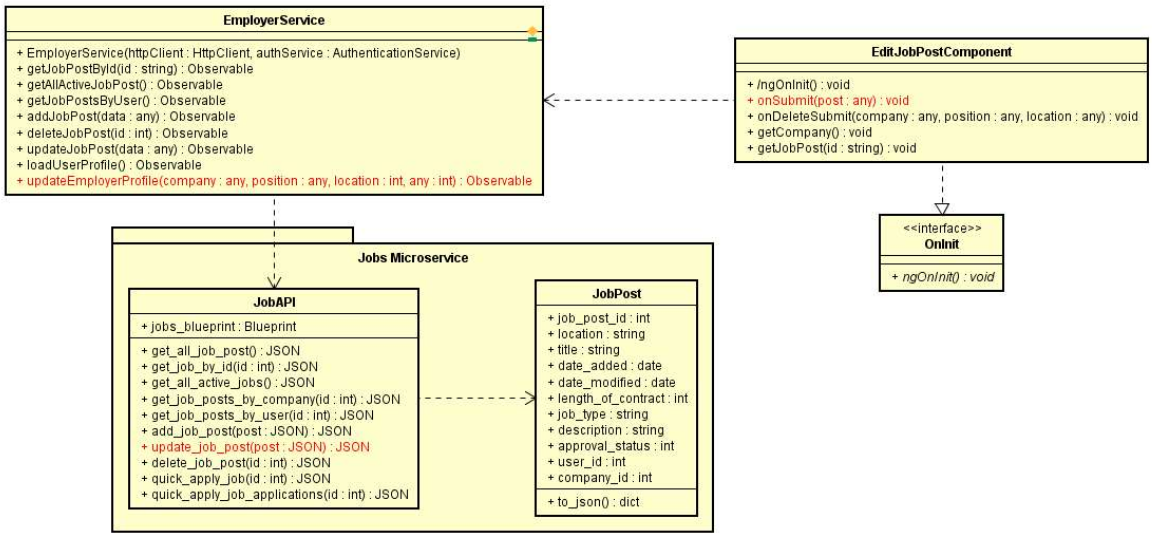


Figure 4-97: Employer edits a job post class diagram

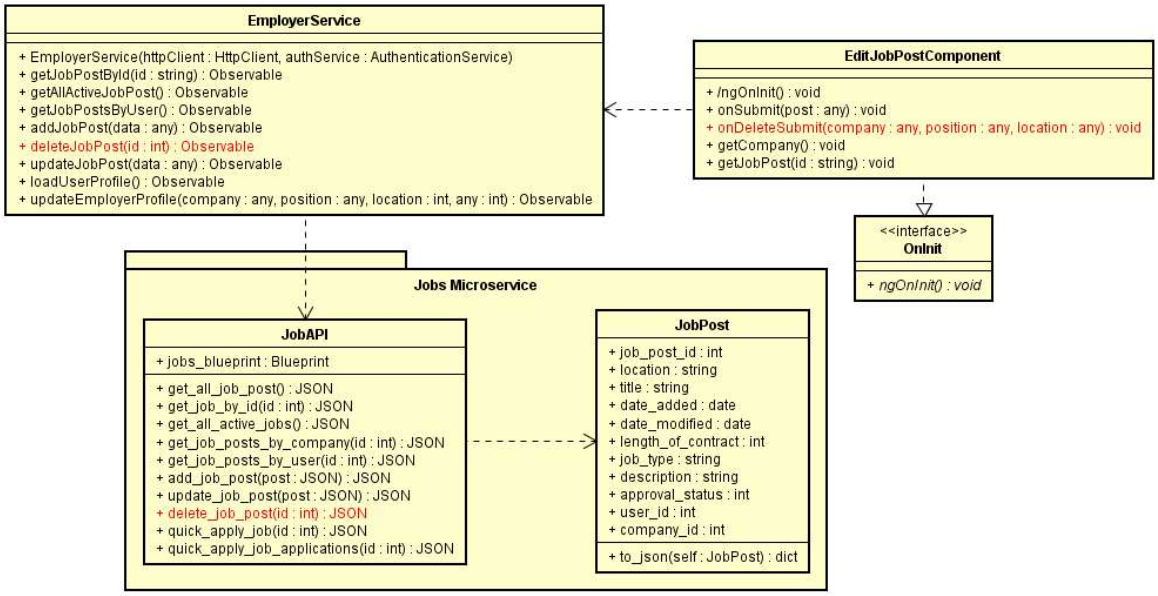


Figure 4-98: Employer deletes a job post class diagram

Class Specification

EmployerService

EmployerService			
Physical address	/services/frontend/src/app/employer/employer.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
EmployerService	Constructor		
Return Type			
Parameters	Name	Type	Description
	httpClient	HttpClient	instance of HttpClient
	authService	Authenticat-onService	instance of AuthenticationSer-vice
getJobPostById	Get Job post by post ID		
Return Type	Observable		
Parameters	Name	Type	Description
	id	int	ID of job post
getAllActiveJobPosts	Get all active job posts that are posted by current user		
Return Type	Observable		
Parameters	Name	Type	Description
getJobPostsByUser	Get all job posts posted by current user		
Return Type	Observable		
Parameters	Name	Type	Description

addJobPost	Add new job post		
Return Type	Observable		
Parameters	Name	Type	Description
	data	any	job post data
deleteJobPost	Delete selected job post		
Return Type	Observable		
Parameters	Name	Type	Description
	id	int	ID of job post
updateJobPost	Update selected job post		
Return Type	Observable		
Parameters	Name	Type	Description
	data	any	job post data
loadUserProfile	Load user profile of logged in user		
Return Type	Observable		
Parameters	Name	Type	Description
updateEmployerProfile	Update employer profile of logged in user		
Return Type	Observable		
Parameters	Name	Type	Description
	company	any	company name
	position	any	job position
	location	any	work location

EditJobPostComponent

EditJobPostComponent			
Physical address	/services/frontend/src/app/employer/edit-job-post/edit-job-post.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
getJobPost	Get Job post by post ID		
Return Type			
Parameters	Name	Type	Description
	id	int	ID of job post
getCompany	Get company by company ID		
Return Type			
Parameters	Name	Type	Description
	id	int	ID of company
onSubmit	Submit button action		
Return Type			
Parameters	Name	Type	Description
onDeleteSubmit	Delete button action		
Return Type			
Parameters	Name	Type	Description

JobAPI

EmployerService			
Physical address	/services/job/project/api/jobs.py		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
update_job_post	Update selected job post in database		
Return Type	JSON		
Parameters	Name	Type	Description

JobPost

JobPost			
Physical address	/services/job/project/api/models.py		
Base class	Class		
Attributes			
No	Name	Type	Description
1	job_post_id	int	Job post id
2	location	string	job location
3	title	string	job title
4	date_added	datetime	when post was added
5	date_modified	datetime	when post was last modified
6	length_of_contract	int	length of contract (in months)
7	job_type	string	type of job
8	description	string	job description
9	approval_status	int	approval status
10	user_id	int	user ID of poster
11	company_id	int	ID of company of job post
Operation			
to_json	return dict of model in json key/value format		
Return Type	dict		
Parameters	Name	Type	Description
	self	JobPost	Job post object

Sequence Diagram

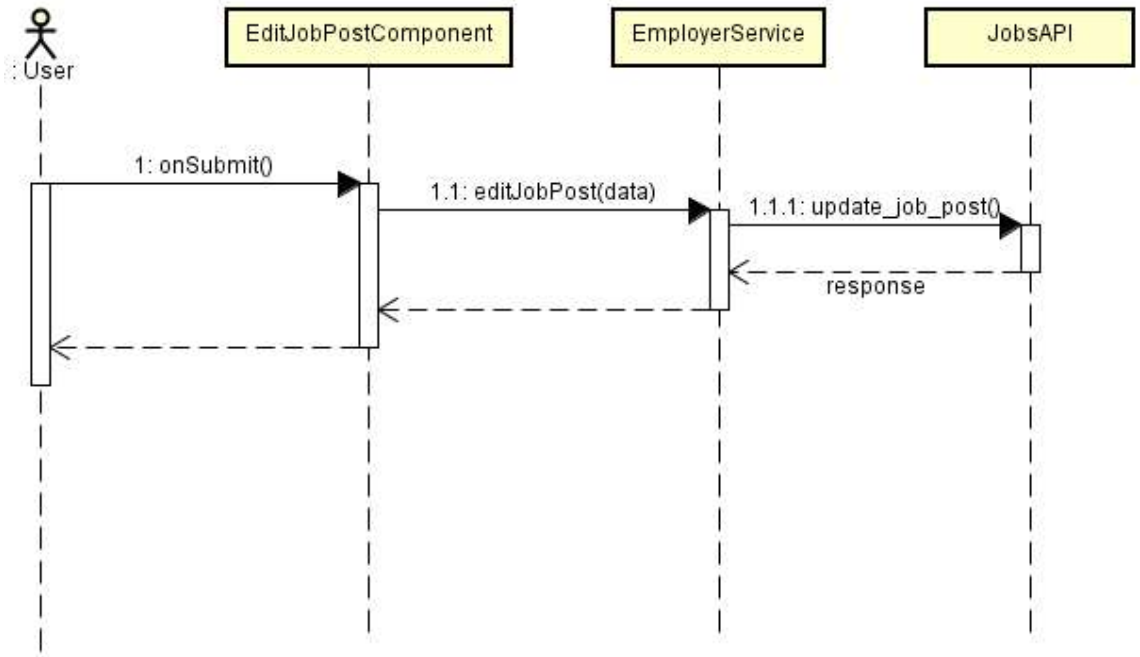


Figure 4-99: Employer edits a job post sequence diagram

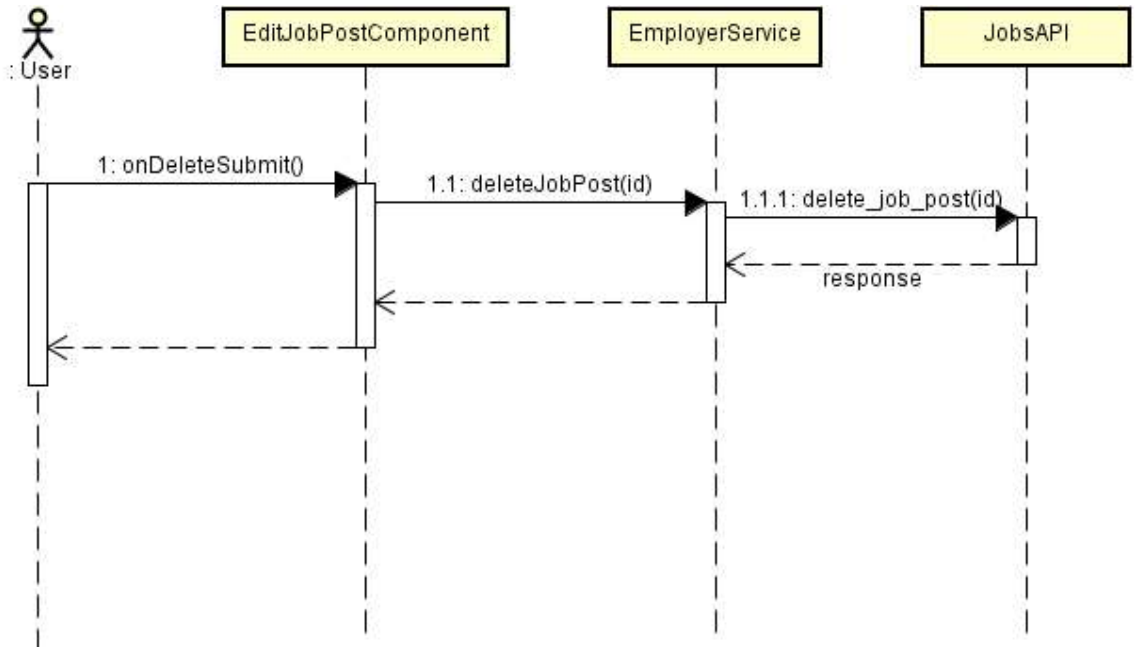


Figure 4-100: Employer deletes a job post sequence diagram

4.3.4.28 Employer creates employer profile

Class Diagram

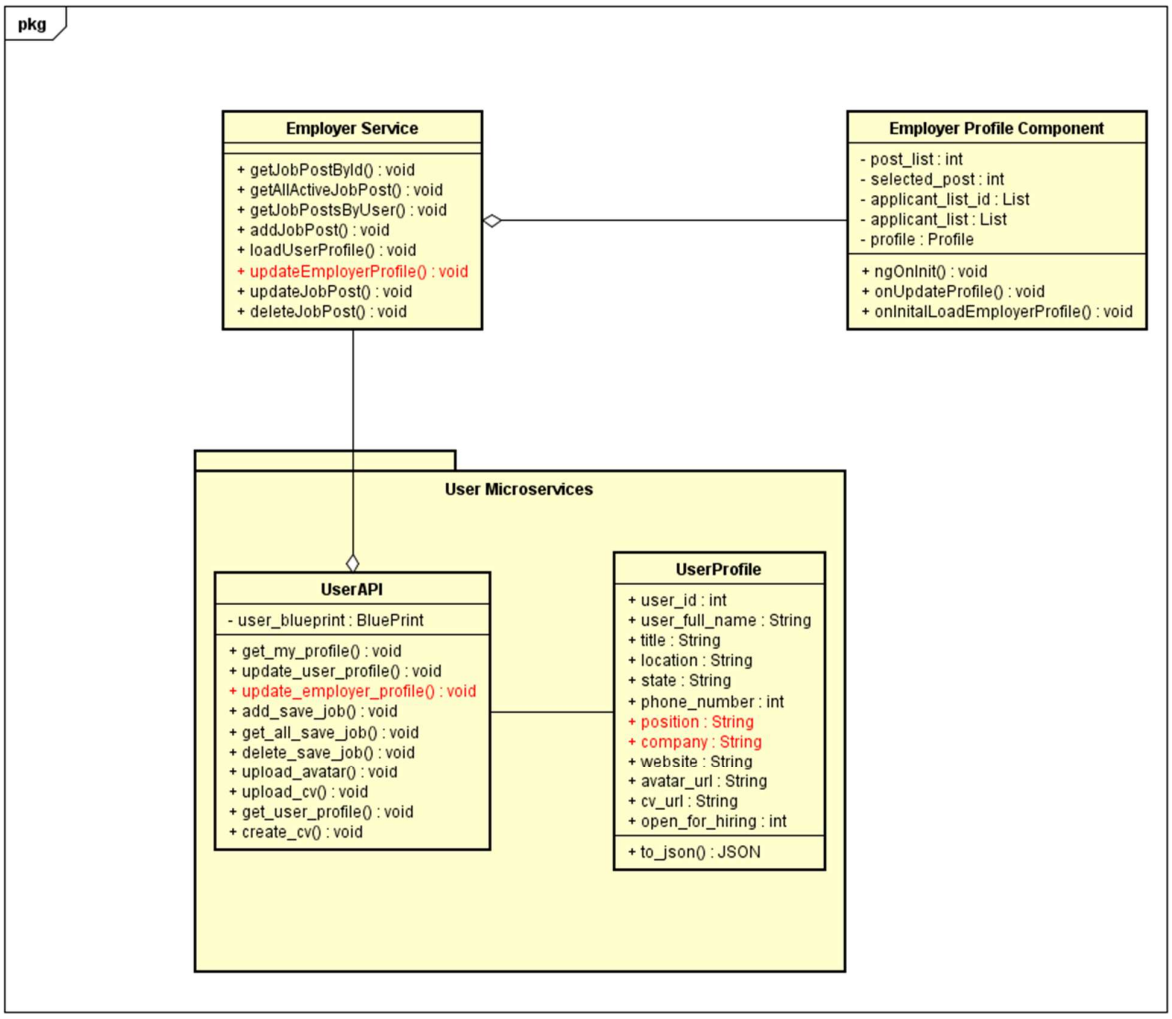


Figure 4-101: Employer creates employer profile class diagram

Class Specification

UserProfileService

UserProfileService			
Physical address	services\frontend\src\app\user-profile\user-profile.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
UserProfileService	Constructor		
Return Type			
Parameters	Name	Type	Description
	httpClient	HttpClient	instance of HttpClient
	authService	Authenticat-onService	instance of AuthenticationSer-vice

updateEmployerProfile	Update user's employer profile		
Return Type	Observable		
Parameters	Name	Type	Description
	employerProfile	Object	Detail of user's employer profile

User API

User API			
Physical address	services\user\project\api\users.py		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
update_employer_profile	Update user's employer profile to database		
Return Type	JSON		
Parameters	Name	Type	Description

EditUserProfileComponent

EditUserProfileComponent			
Physical address	services\frontend\src\app\user-profile\user-profile.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
onInitialLoadEmployerProfile	Load user's employer profile		
Return Type	void		
Parameters	Name	Type	Description
onUpdateProfile	Save updated user's employer profile		
Return Type	void		
Parameters	Name	Type	Description
	employer_profile	Object	User's employer profile

Sequence Diagram

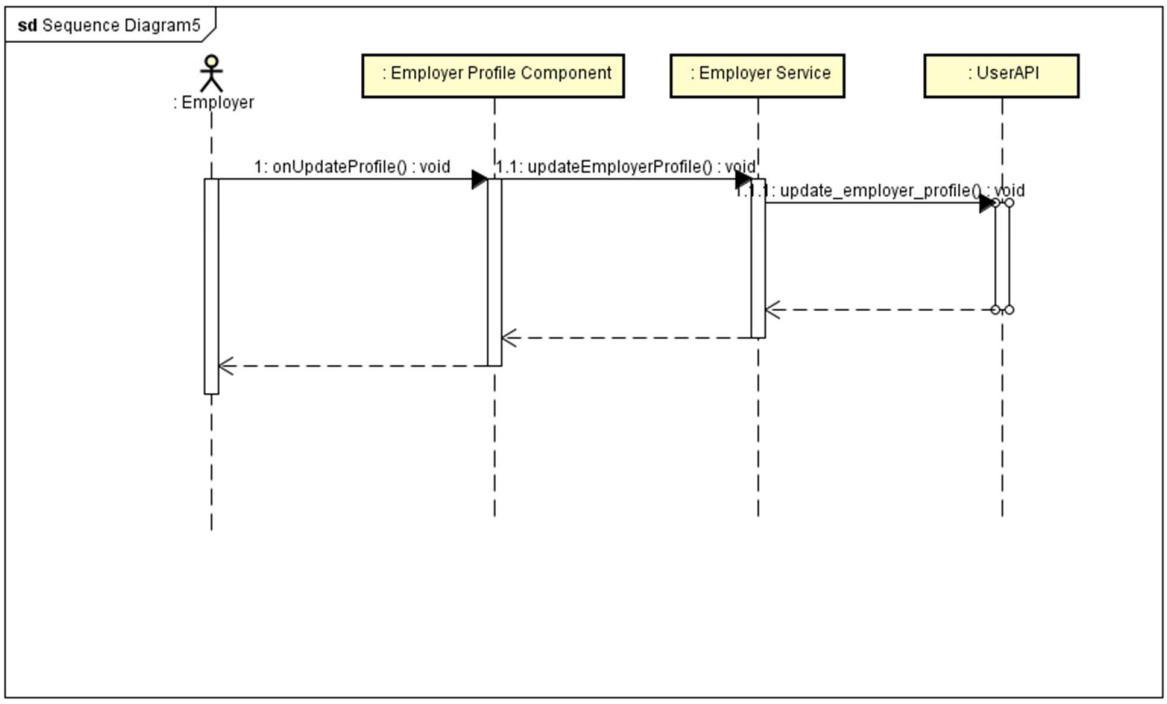


Figure 4-102: Employer creates employer profile sequence diagram

4.3.4.29 Employer views a job seeker resume

Class Diagram

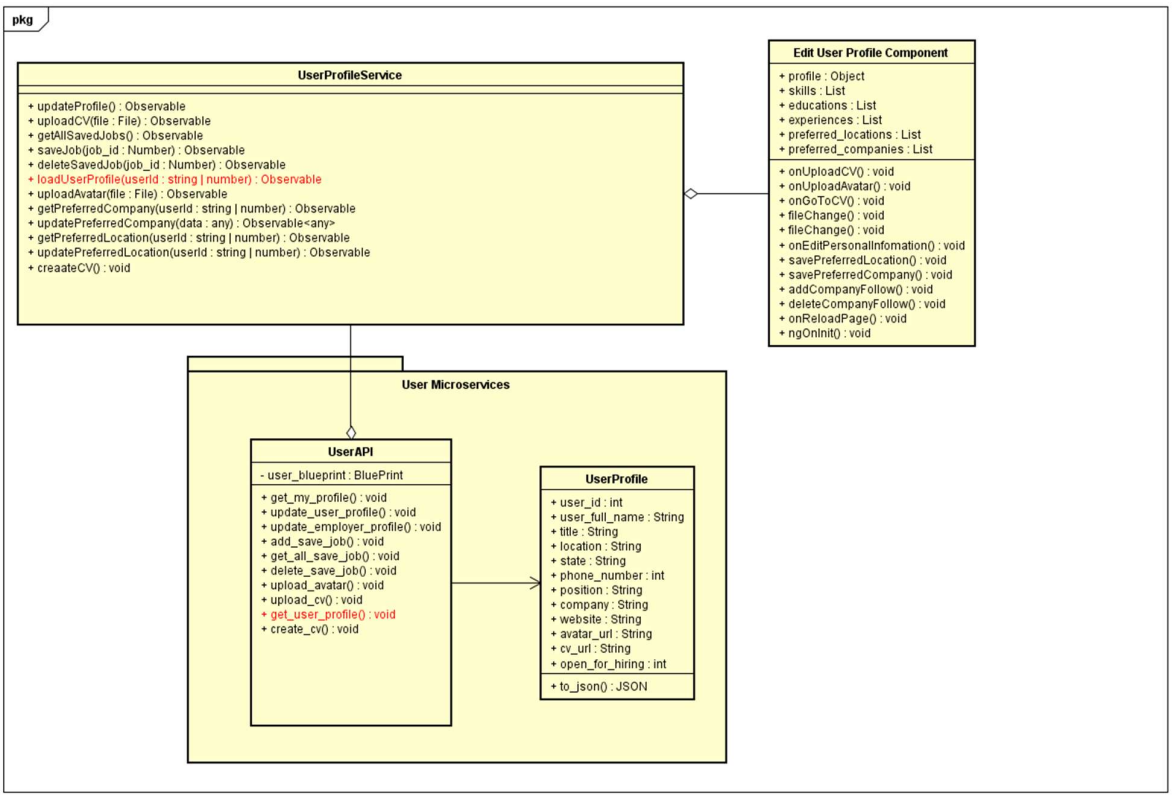


Figure 4-103: Employer views a job seeker resume class diagram

Class Specification

UserProfileService

UserProfileService			
Physical address	services\frontend\src\app\user-profile\user-profile.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
UserProfileService	Constructor		
Return Type			
Parameters	Name	Type	Description
	httpClient	HttpClient	instance of HttpClient
	authService	Authenticat-onService	instance of AuthenticationSer-vice
loadUserProfile	Update user's personal details		
Return Type	Observable		
Parameters	Name	Type	Description
	userId	String	User ID

User API

User API			
Physical address	services\user\project\api\users.py		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
get_user_profile	Get user profile details		
Return Type	JSON		
Parameters	Name	Type	Description
	profile	JSON	Profile details

EditUserProfileComponent

EditUserProfileComponent			
Physical address	services\frontend\src\app\user-profile\user-profile.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
onInitalLoadUserProfile	Load user personal details		
Return Type	void		
Parameters	Name	Type	Description

UserProfile

UserProfile			
Physical address	services\user\project\api\models.py		
Base class	Class		
Attributes			
No	Name	Type	Description
1	user_id	int	User ID
2	location	string	User location
3	title	string	User title
4	state	string	User's state
5	phone_number	string	User's phone number
6	position	string	User's job position
7	company	string	User's company
8	website	string	User's website
9	avatar_url	string	User's avatar URL
10	cv_url	string	User's CV URL
11	open_for_hiring	int	User's availability for hiring
Operation			
to_json	return JSON of model		
Return Type	JSON		
Parameters	Name	Type	Description

Sequence Diagram

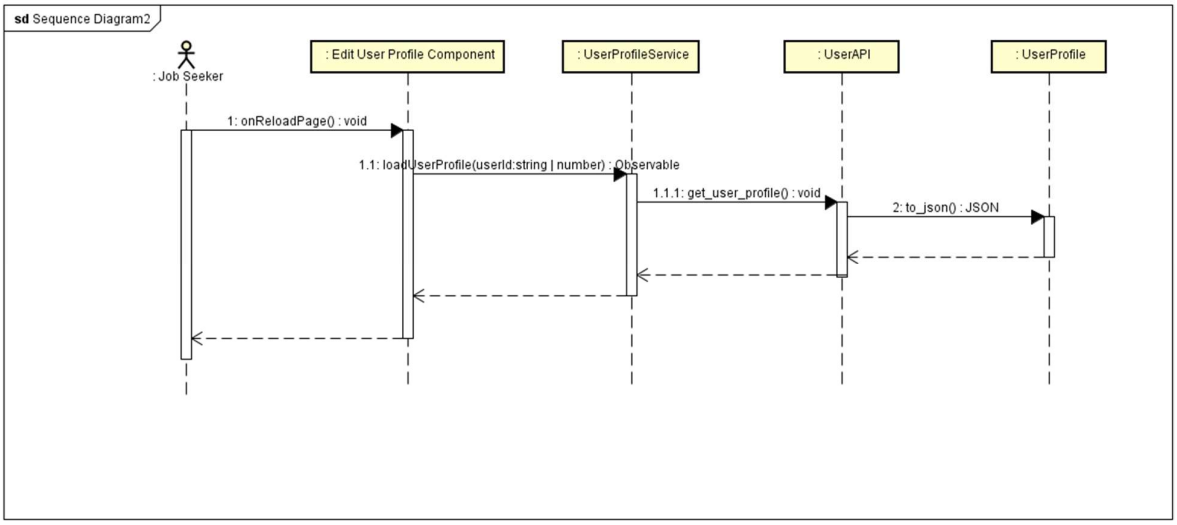


Figure 4-104: Employer views a job seeker resume sequence diagram

4.3.4.30 Employer views all applicants

Screen Design

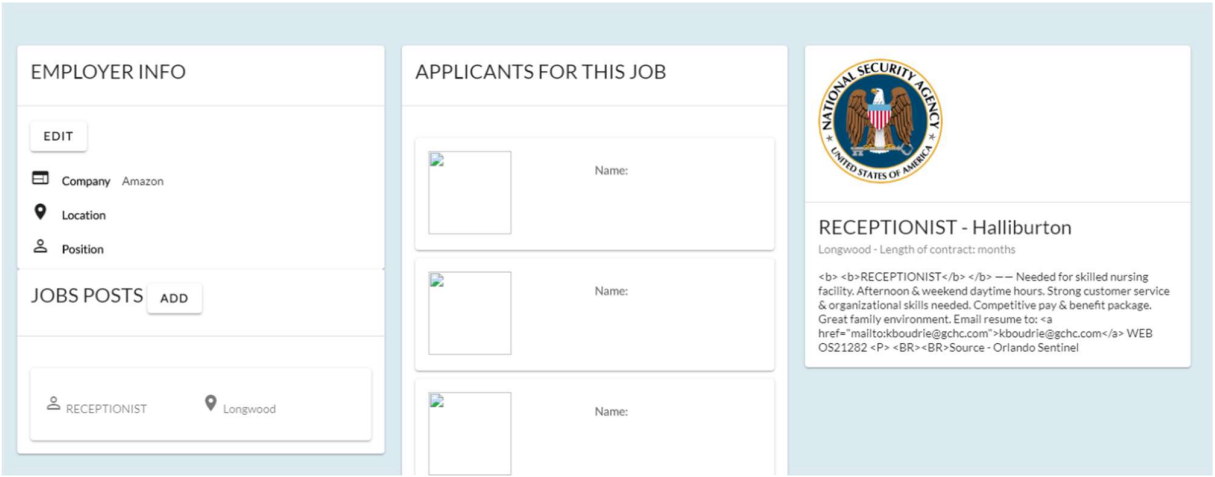


Figure 4-105: Employer view job details & applicants

Class Diagram

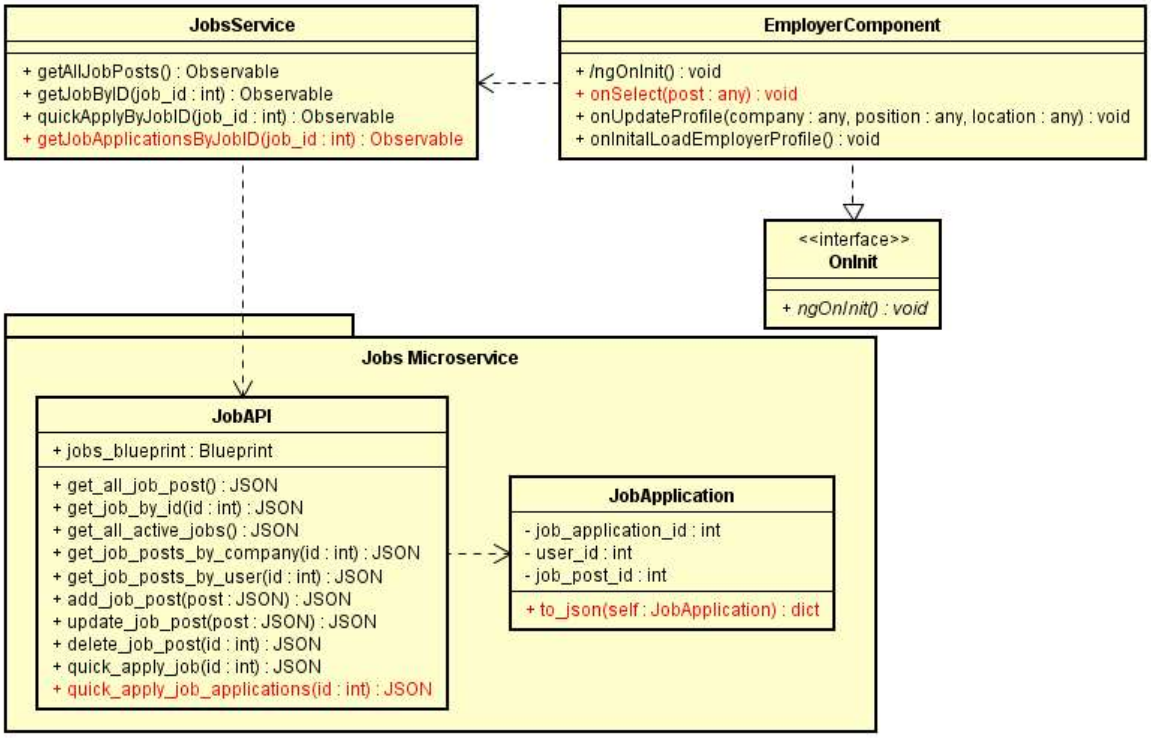


Figure 4-106: Employer user view all quick-applied applicants

Class Specification

JobsService

JobsService			
Physical address	/services/frontend/src/app/jobs/jobs.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
getJobApplicationsByJ-obID	Get all job applicants for a job		
Return Type	Observable		
Parameters	Name	Type	Description
	job_id	int	Job post ID

EmployerComponent

EmployerComponent			
Physical address	/services/frontend/src/app/employer/employer.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
ngOnInit	Page initialization, used to populate job posts		

Return Type			
Parameters	Name	Type	Description
onSelect	Action to do when a job post is selected		
Return Type	void		
Parameters	Name	Type	Description
	post	Observable	selected job post

JobAPI

JobAPI			
Physical address	/services/job/project/api/jobs.py		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
quick_apply_job_applications	Get applicants for selected job post		
Return Type	JSON	Return Type	JSON
Parameters	Name	Parameters	Name
	id		Job post ID

JobApplication

JobApplication			
Physical address	/services/job/project/api/models.py		
Base class	Class		
Attributes			
No	Name	Type	Description
1	job_application_id	int	Job Application ID
2	user_id	int	User ID
3	job_post_id	int	Job Post ID
Operation			
to_json	return dict of model in json key/value format		
Return Type	dict		
Parameters	Name	Type	Description
	self	JobApplication	Job application object

Sequence Diagram

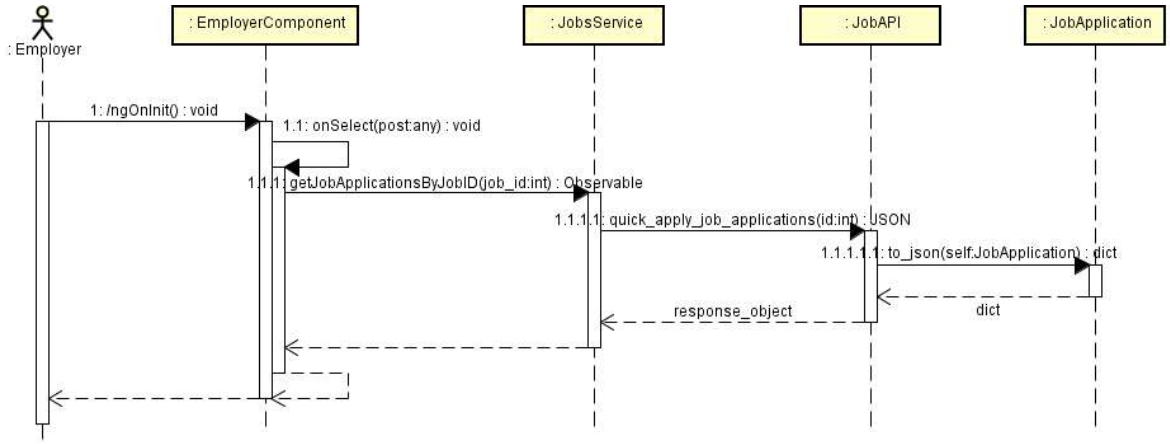


Figure 4-107: Employer user view all applicants sequence diagram

4.3.4.31 Staff views all active users

Screen Design

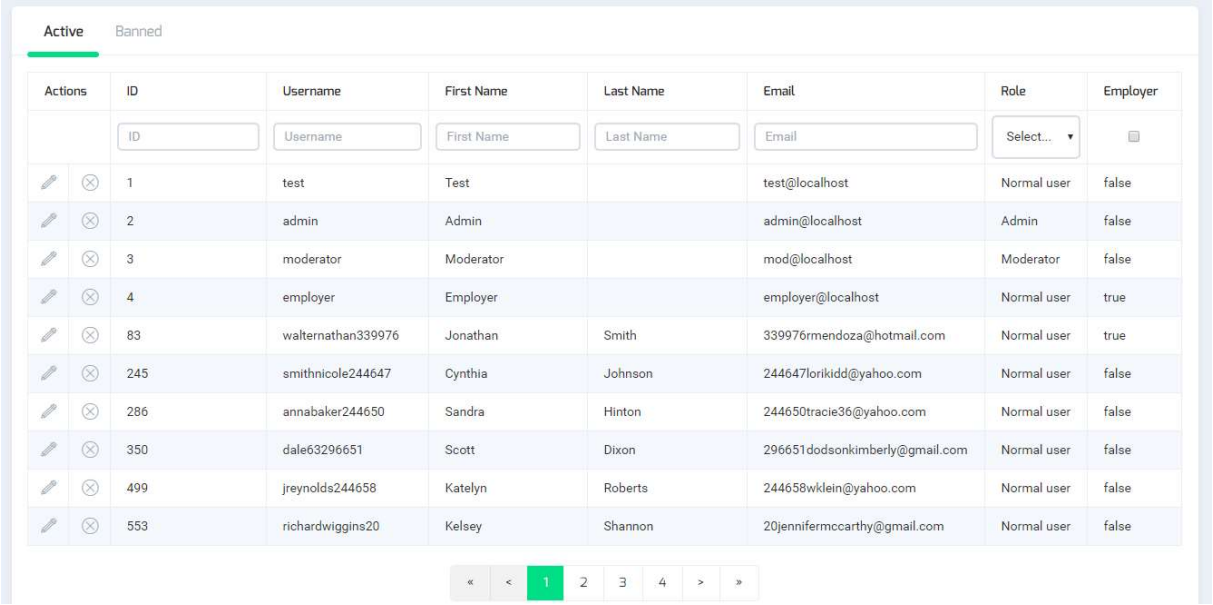


Figure 4-108: Staff views all active users

Class Diagram

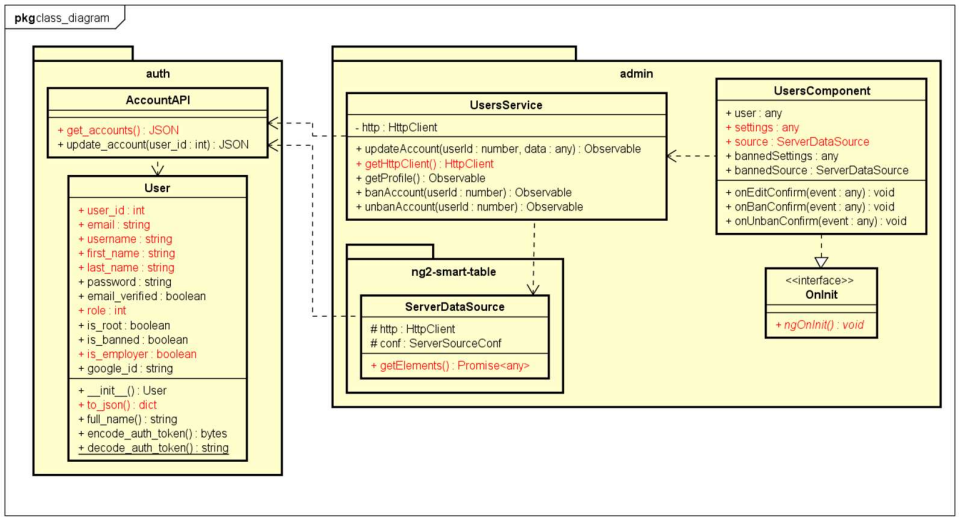


Figure 4-109: Staff views all active users class diagram

Class Specification

UsersComponent

UsersComponent			
Physical address	/services/frontend/src/app/pages/users/users.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
1	user	any	Current logged in user
2	settings	any	Settings for user table
3	source	ServerDataSource	Data source for the table
Operation			
ngOnInit	Initialize component		
Return Type	void		
Parameters	Name	Type	Description

UsersService

UsersService			
Physical address	/services/admin/src/app/@core/data/users.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
1	http	HttpClient	Angular HttpClient to communicate with API endpoints
Operation			
getHttpClient	Get HttpClient instance of the service		
Return Type	HttpClient		

Parameters	Name	Type	Description

AccountAPI

AccountAPI			
Physical address	/services/auth/project/api/accounts.py		
Base class	Blueprint		
Attributes			
No	Name	Type	Description
Operation			
get_accounts	Get available accounts		
Return Type	JSON		
Parameters	Name	Type	Description

User

User			
Physical address	/services/auth/project/api/models.py		
Base class	Model		
Attributes			
No	Name	Type	Description
1	user_id	int	ID of the user
2	email	string	Email of the user
3	username	string	Username of the user
4	first_name	string	First name of the user
5	last_name	string	Last name of the user
6	role	string	Role of the user: 0, 1, 2, corresponding to Normal User, Moderator, Administrator
7	is_employer	boolean	Indicates whether the user is an employer or not
Operation			
to_json	Create dictionary object from User		
Return Type	dict		
Parameters	Name	Type	Description

ServerDataSource

ServerDataSource			
Physical address	ng2-smart-table		
Base class	LocalDataSource		
Attributes			
No	Name	Type	Description
1	http	HttpClient	Angular HttpClient for communication with API
2	conf	ServerSourceConf	Configuration for source
Operation			
getElements	Fetch data from API		
Return Type	Promise<any>		

Parameters	Name	Type	Description

Sequence Diagram

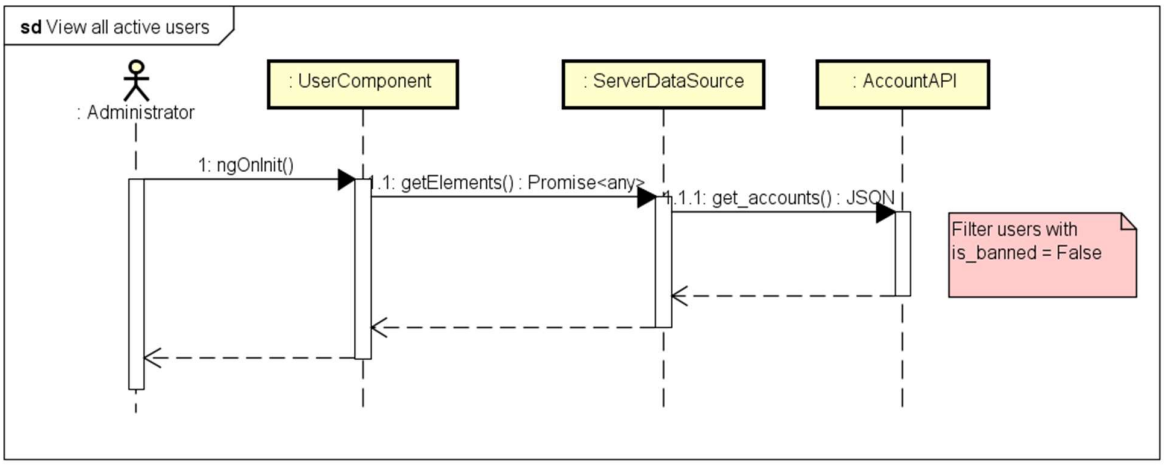


Figure 4-110: Staff views all active users sequence diagram

4.3.4.32 Administrator views banned users

Screen Design

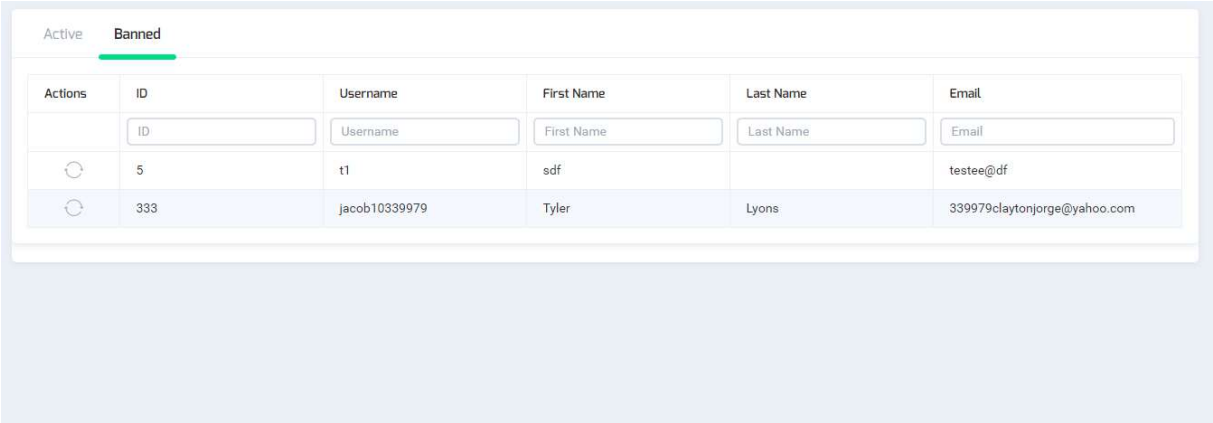


Figure 4-111: Administrator views banned users

Class Diagram

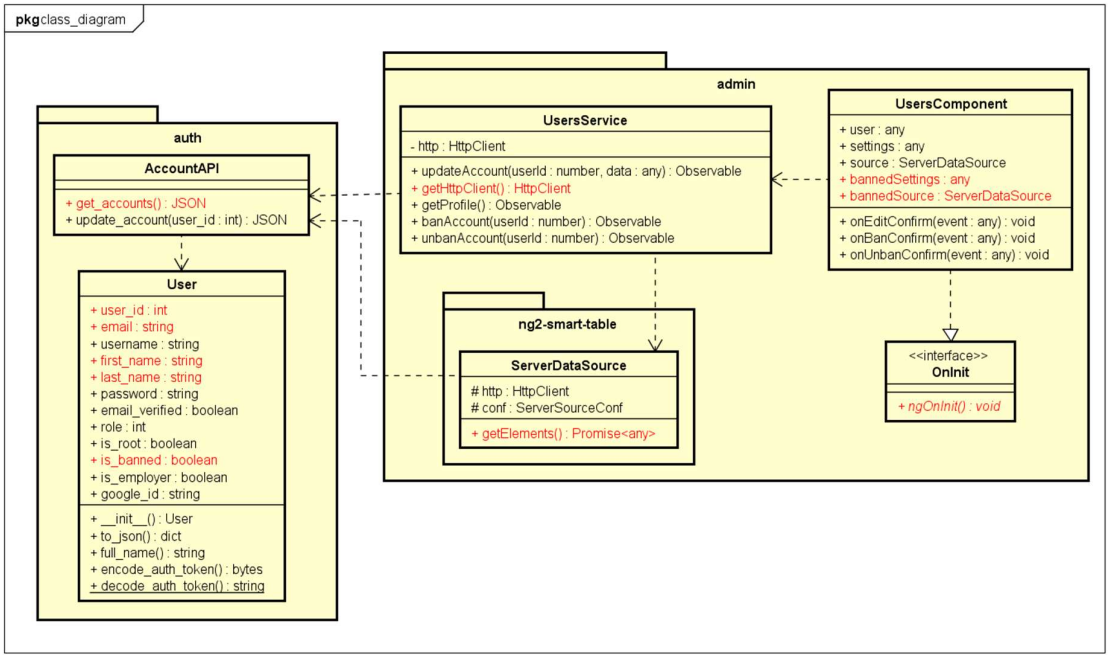


Figure 4-112: Administrator views banned users class diagram

Class Specification

UsersComponent

UsersComponent			
Physical address	/services/frontend/src/app/pages/users/users.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
1	user	any	Current logged in user
2	bannedSettings	any	Settings for banned users table
3	bannedSource	ServerDataSource	Data source for the banned users table

UsersService

UsersService			
Physical address	/services/admin/src/app/@core/data/users.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
1	http	HttpClient	Angular HttpClient to communicate with API endpoints
Operation			
getHttpClient	Get HttpClient instance of the service		
Return Type	HttpClient		
Parameters	Name	Type	Description

ngOnInit	Initialize component		
Return Type	void		
Parameters	Name	Type	Description

AccountAPI

AccountAPI			
Physical address	/services/auth/project/api/accounts.py		
Base class	Blueprint		
Attributes			
No	Name	Type	Description
Operation			
get_accounts	Get available accounts		
Return Type	JSON		
Parameters	Name	Type	Description

User

User			
Physical address	/services/auth/project/api/models.py		
Base class	Model		
Attributes			
No	Name	Type	Description
1	user_id	int	ID of the user
3	username	string	Username of the user
4	first_name	string	First name of the user
5	last_name	string	Last name of the user
6	is_banned	boolean	Indicates whether the user is banned or not

ServerDataSource

ServerDataSource			
Physical address	ng2-smart-table		
Base class	LocalDataSource		
Attributes			
No	Name	Type	Description
1	http	HttpClient	Angular HttpClient for communication with API
2	conf	ServerSourceConf	Configuration for source
Operation			
getElements	Fetch data from API		
Return Type	Promise<any>		
Parameters	Name	Type	Description

Sequence Diagram

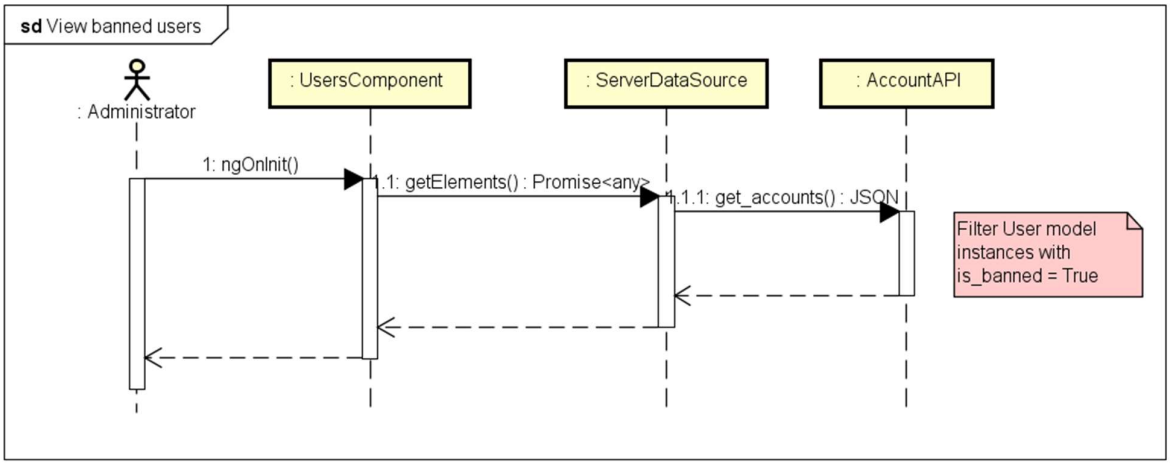


Figure 4-113: Administrator views banned user sequence diagram

4.3.4.33 Staff updates a user's role or employer status

Screen Design

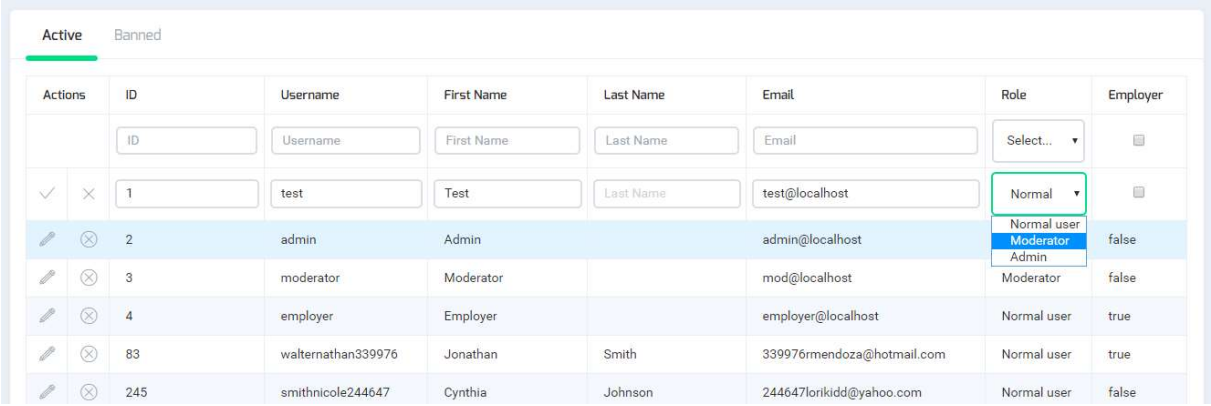


Figure 4-114: Staff updates a user's role

Class Diagram

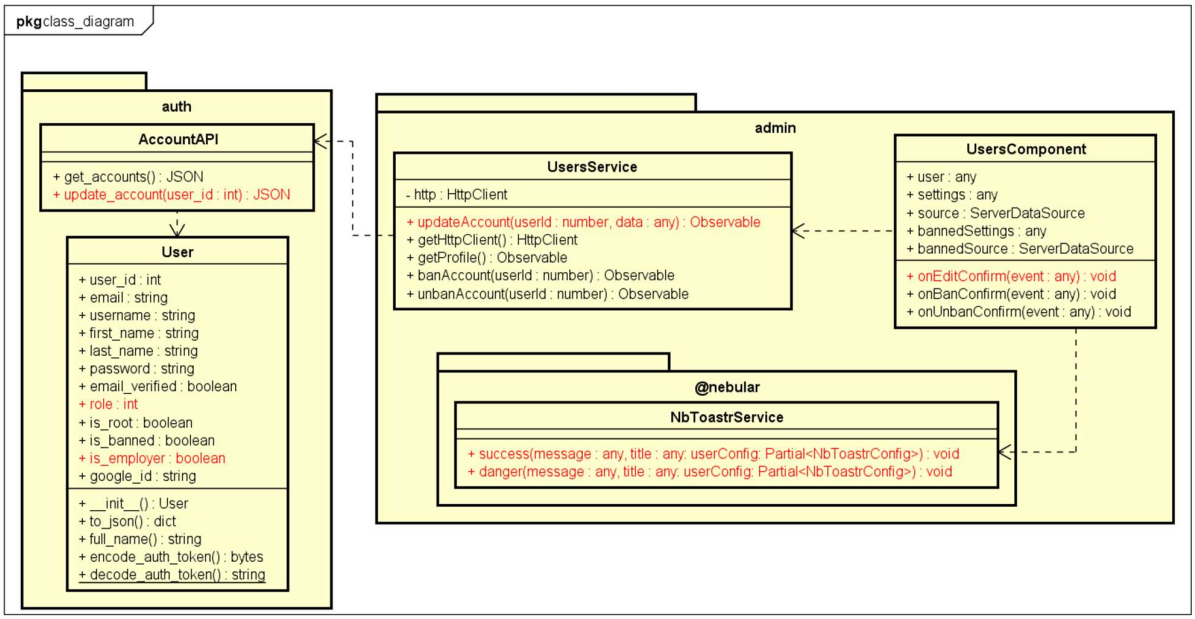


Figure 4-115: Staff updates a user's role or employer status class diagram

Class Specification

UsersComponent

UsersComponent			
Physical address	/services/frontend/src/app/pages/users/users.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
onEditConfirm	Update user		
Return Type	void		
Parameters	Name	Type	Description
	event	any	Event object containing the data

UsersService

UsersService			
Physical address	/services/admin/src/app/@core/data/users.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
updateAccount	Send user update request to the API		
Return Type	Observable		
Parameters	Name	Type	Description

	userId	number	User ID
	data	any	New data to update for the user

AccountAPI

AccountAPI			
Physical address	/services/auth/project/api/accounts.py		
Base class	Blueprint		
Attributes			
No	Name	Type	Description
Operation			
update_account	Update user account with new information		
Return Type	JSON		
Parameters	Name	Type	Description
	user_id	int	ID of the user

User

User			
Physical address	/services/auth/project/api/models.py		
Base class	Model		
Attributes			
No	Name	Type	Description
1	role	int	Role of the user
2	is_employer	boolean	Indicates whether the user is an employer or not

NbToastrService

NbToastrService			
Physical address	@nebular		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
success			
Return Type	void		
Parameters	Name	Type	Description
	message	string	Message to be displayed
	title	string	Title of the message
	userConfig	Partial<NbToastrConfig>	Config for the toast notification
danger			
Return Type	void		
Parameters	Name	Type	Description
	message	string	Message to be displayed
	title	string	Title of the message
	userConfig	Partial<NbToastrConfig>	Config for the toast notification

Sequence Diagram

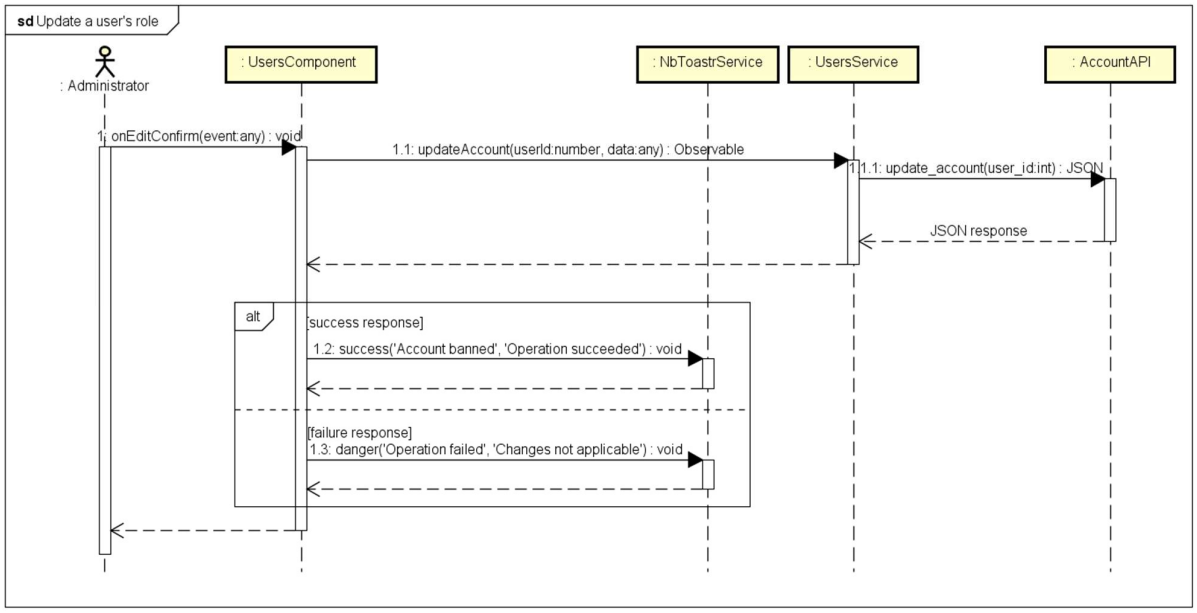


Figure 4-116: Update a user's role or employer status sequence diagram

4.3.4.34 Administrator bans/unbans a user

Screen Design

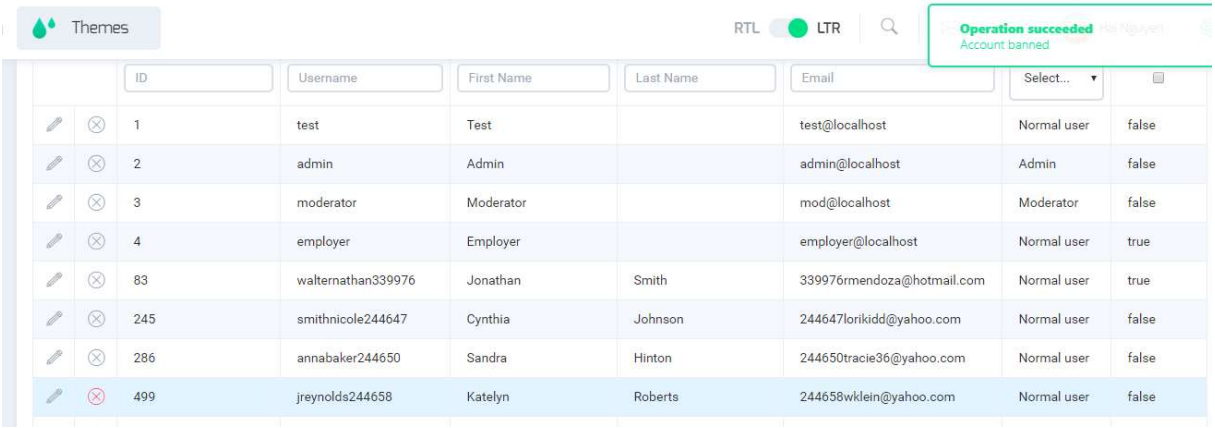


Figure 4-117: Administrator bans a user

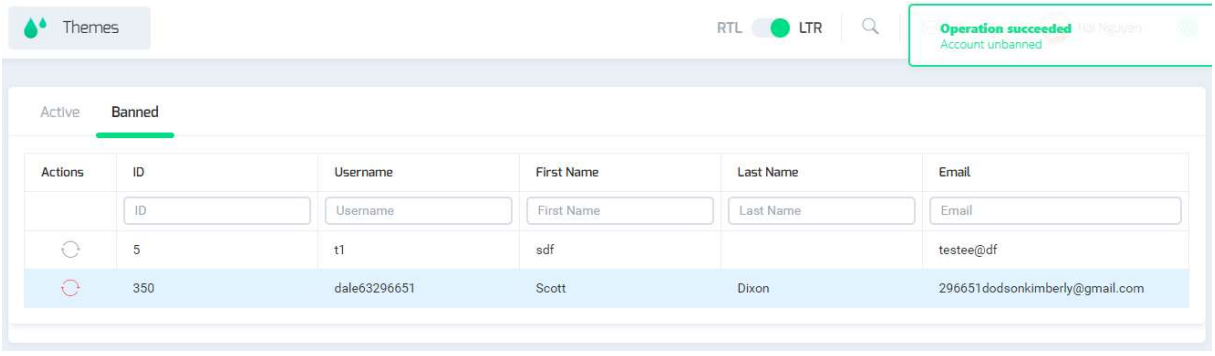


Figure 4-118: Administrator unbans a user

Class Diagram

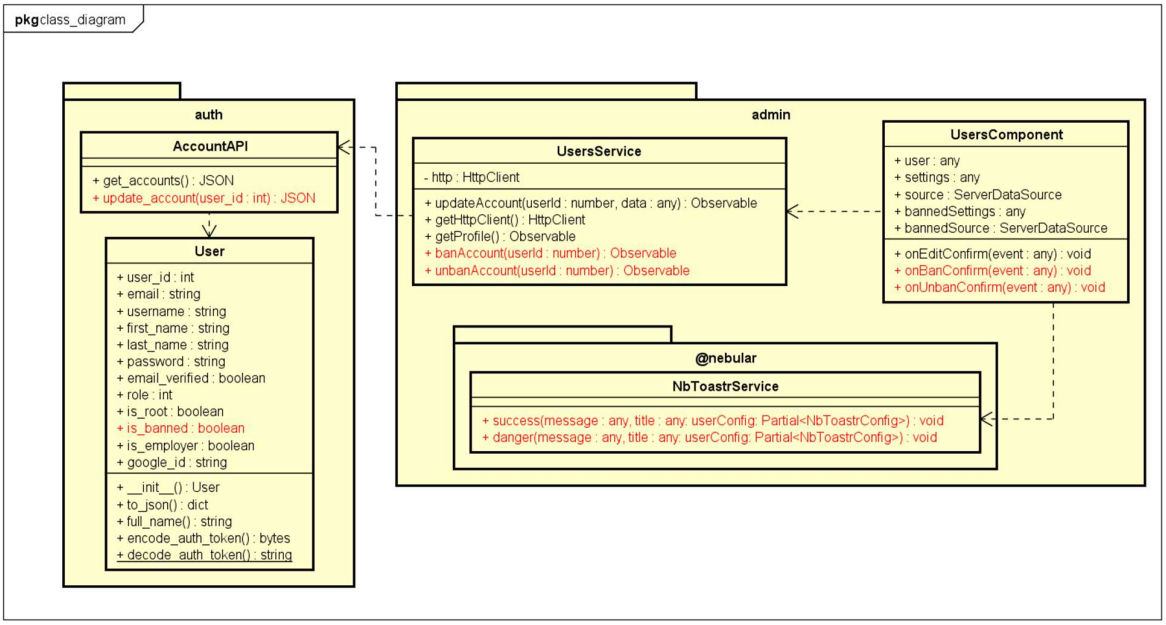


Figure 4-119: Administrator bans/unbans a user class diagram

Class Specification

UsersComponent

UsersComponent			
Physical address	/services/frontend/src/app/pages/users/users.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
onBanConfirm	Ban user event		
Return Type	void		
Parameters	Name	Type	Description
	event	any	Event object containing user data
onUnbanConfirm	Unban user event		
Return Type	void		
Parameters	Name	Type	Description
	event	any	Event object containing user data

UsersService

UsersService			
Physical address	/services/admin/src/app/@core/data/users.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			

banAccount	Send user ban request to the API		
Return Type	Observable		
Parameters	Name	Type	Description
	userId	number	User ID
unbanAccount	Send user ban request to the API		
Return Type	Observable		
Parameters	Name	Type	Description
	userId	number	User ID

AccountAPI

AccountAPI			
Physical address	/services/auth/project/api/accounts.py		
Base class	Blueprint		
Attributes			
No	Name	Type	Description
Operation			
update_account	Update user account with new information		
Return Type	JSON		
Parameters	Name	Type	Description
	user_id	int	ID of the user

User

User			
Physical address	/services/auth/project/api/models.py		
Base class	Model		
Attributes			
No	Name	Type	Description
1	is_banned	boolean	Indicates whether the user is banned or not

NbToastrService

NbToastrService			
Physical address	@nebular		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
success			
Return Type	void		
Parameters	Name	Type	Description
	message	string	Message to be displayed
	title	string	Title of the message
	userConfig	Partial<NbToastrConfig>	Config for the toast notification
danger			
Return Type	void		
Parameters	Name	Type	Description
	message	string	Message to be displayed
	title	string	Title of the message

	userConfig	Partial<NbToastrConfig>	Config for the toast notification

Sequence Diagram

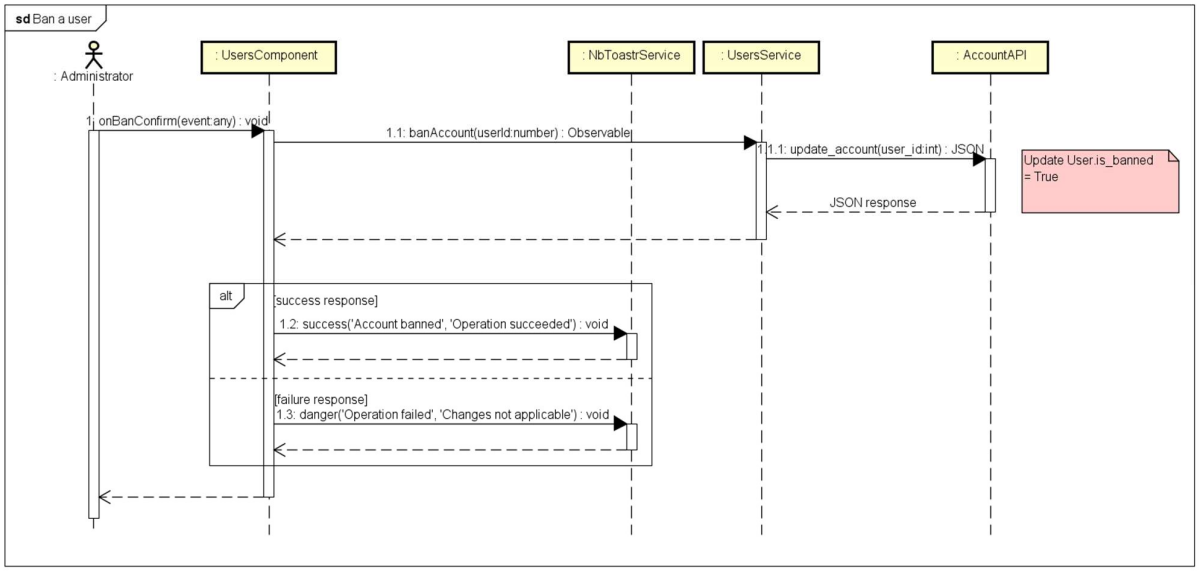


Figure 4-120: Administrator bans a user sequence diagram

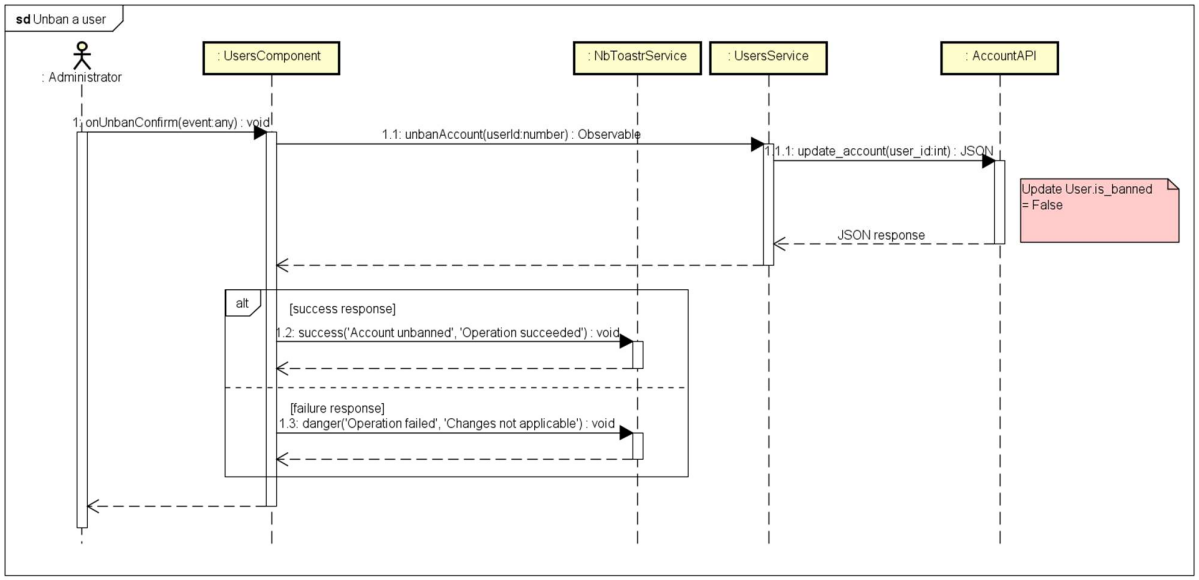


Figure 4-121: Administrator unbans a user sequence diagram

4.3.4.35 Administrator views all companies

Screen Design

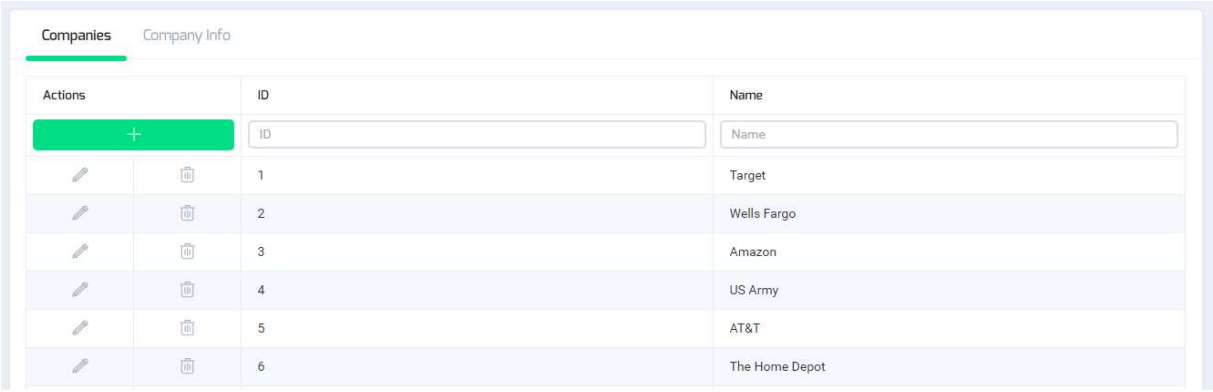


Figure 4-122: Administrator views all companies screen design

Class Diagram

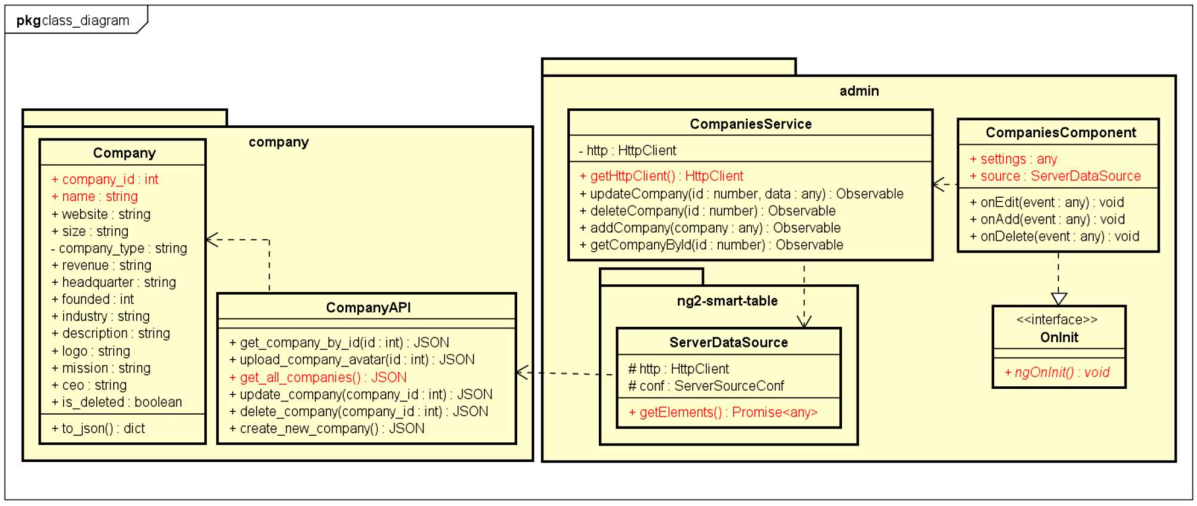


Figure 4-123: Administrator views all companies class diagram

Class Specification

CompaniesComponent

CompaniesComponent			
Physical address	/services/frontend/src/app/pages/companies/companies.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
1	settings	any	Settings for user table
2	source	ServerDataSource	Data source for the table
Operation			
ngOnInit	Initialize component		
Return Type	void		
Parameters	Name	Type	Description

CompaniesService

CompaniesService			
Physical address	/services/admin/src/app/@core/data/companies.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
1	http	HttpClient	Angular HttpClient to communicate with API endpoints
Operation			
getHttpClient	Get HttpClient instance of the service		
Return Type	HttpClient		
Parameters	Name	Type	Description

CompanyAPI

CompanyAPI			
Physical address	/services/company/project/api/companies.py		
Base class	Blueprint		
Attributes			
No	Name	Type	Description
Operation			
get_all_companies	Get available companies		
Return Type	JSON		
Parameters	Name	Type	Description

Company

Company			
Physical address	/services/company/project/api/models.py		
Base class	Model		
Attributes			
No	Name	Type	Description
1	company_id	int	ID of the company
2	name	string	Name of the company
Operation			
to_json	Create dictionary object from the model		
Return Type	dict		
Parameters	Name	Type	Description

ServerDataSource

ServerDataSource			
Physical address	ng2-smart-table		
Base class	LocalDataSource		
Attributes			
No	Name	Type	Description
1	http	HttpClient	Angular HttpClient for communication with API

2	conf	ServerSourceConf	Configuration for source
Operation			
getElements	Fetch data from API		
Return Type	Promise<any>		
Parameters	Name	Type	Description

Sequence Diagram

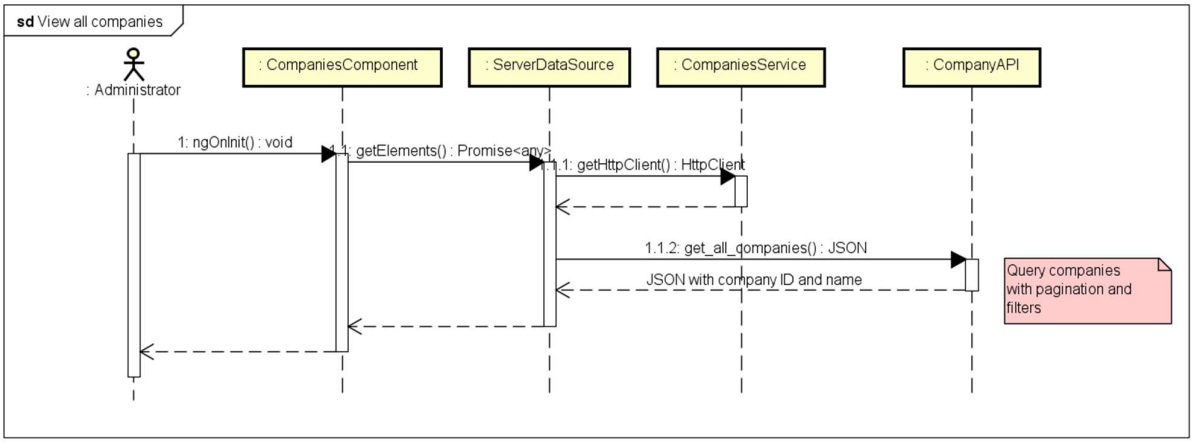


Figure 4-124: Administrator views all companies sequence diagram

4.3.4.36 Administrator creates a new company

Screen Design

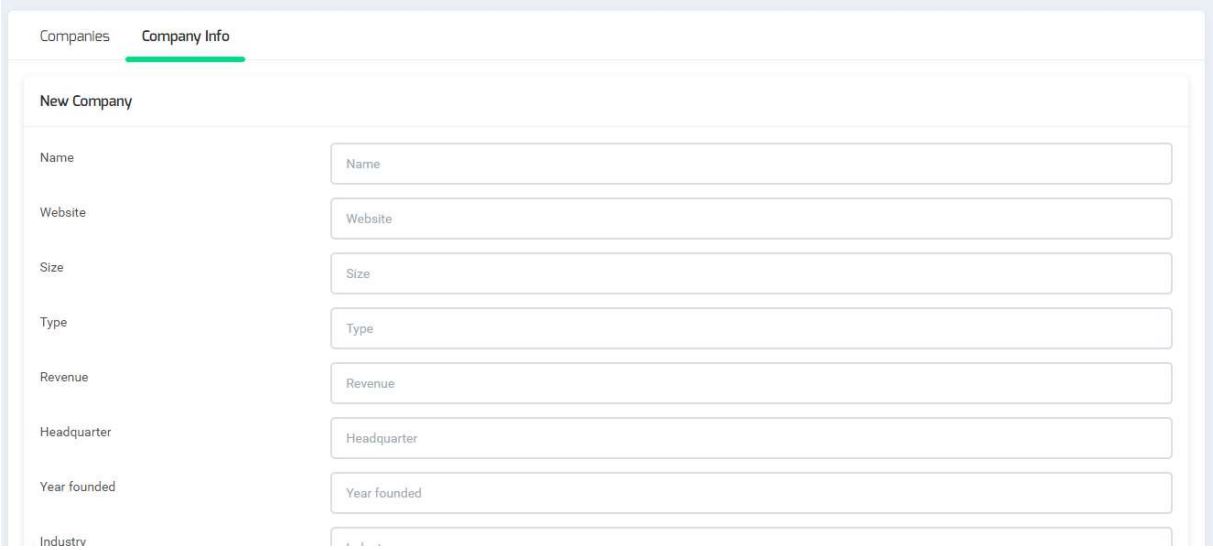


Figure 4-125: Administrator creates a new company screen design

Class Diagram

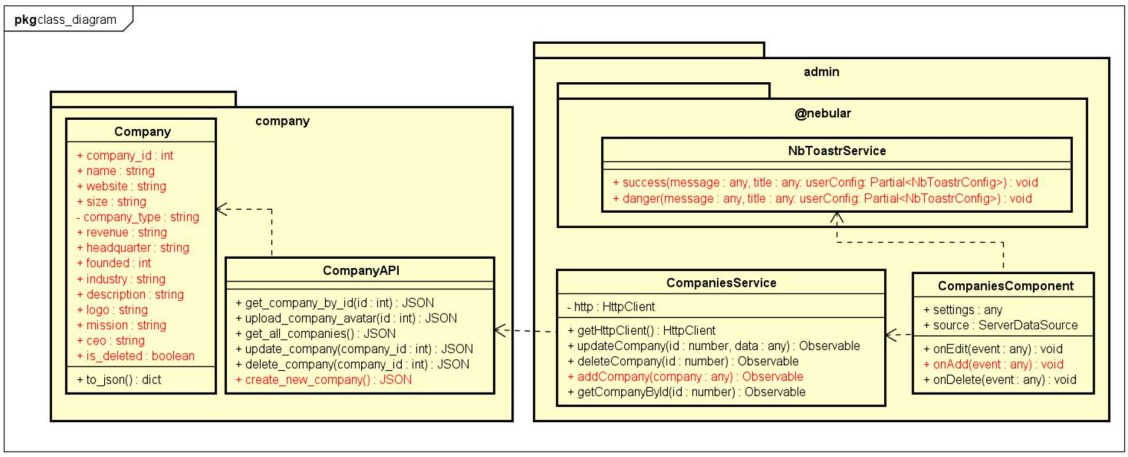


Figure 4-126: Administrator creates a new company class diagram

Class Specification

CompaniesComponent

CompaniesComponent			
Physical address	/services/frontend/src/app/pages/companies/companies.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
onAdd	Add company event		
Return Type	void		
Parameters	Name	Type	Description
	event	any	Event object

CompaniesService

CompaniesService			
Physical address	/services/admin/src/app/@core/data/companies.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
addCompany	Send company creation request to the API		
Return Type	Observable		
Parameters	Name	Type	Description
	company	any	Data of the new company

CompanyAPI

CompanyAPI	
Physical address	/services/company/project/api/companies.py
Base class	Blueprint

Attributes			
No	Name	Type	Description
Operation			
create_new_company	Create a new company		
Return Type	JSON		
Parameters	Name	Type	Description

Company

Company			
Physical address	/services/company/project/api/models.py		
Base class	Model		
Attributes			
No	Name	Type	Description
1	company_id	int	ID of the company
2	name	string	Company name
3	website	string	Website URL of the company
4	size	string	Size of the company
5	company_type	string	Type of the company
6	revenue	string	Revenue of the company
7	headquarter	string	Headquarter location of the company
8	founded	int	Year the company was founded
9	industry	string	Industries the company work in
10	description	string	Description of the company
11	logo	string	Logo URL of the company
12	mission	string	Mission of the company
13	ceo	string	CEO name of the company
14	is_deleted	boolean	Indicates whether the company is deleted

NbToastrService

NbToastrService			
Physical address	@nebular		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
success			
Return Type	void		
Parameters	Name	Type	Description
	message	string	Message to be displayed
	title	string	Title of the message
	userConfig	Partial<NbToastrConfig>	Config for the toast notification
danger			
Return Type	void		
Parameters	Name	Type	Description
	message	string	Message to be displayed
	title	string	Title of the message

	userConfig	Partial<NbToastrConfig>	Config for the toast notification

Sequence Diagram

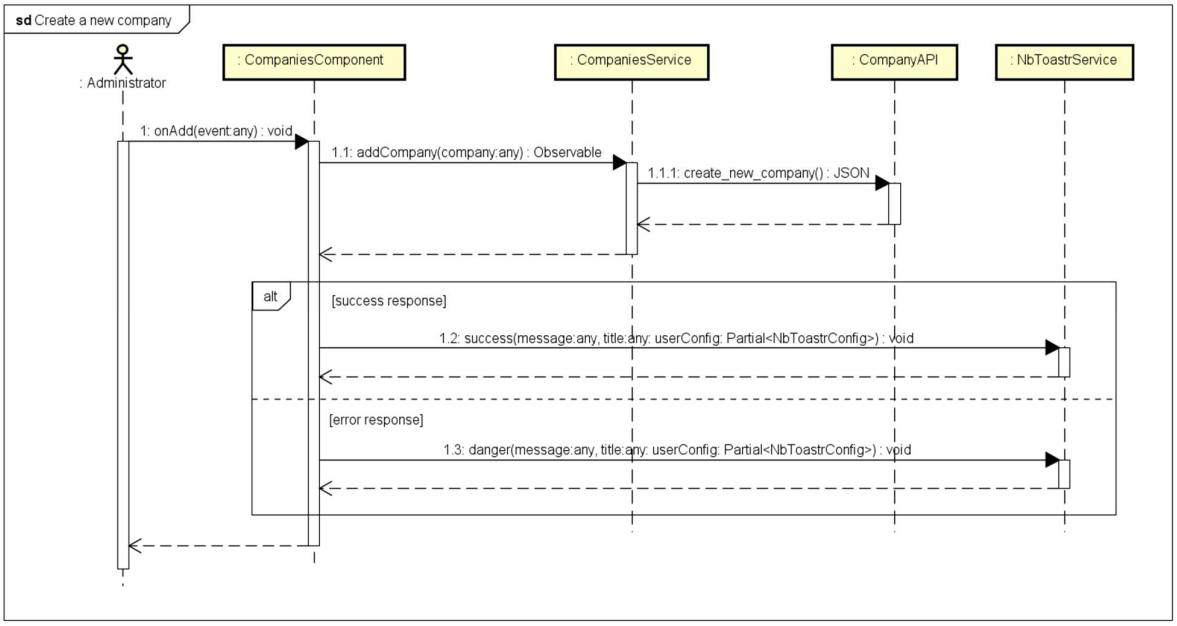


Figure 4-127: Administrator creates a new company

4.3.4.37 Administrator updates/deletes a company

Screen Design

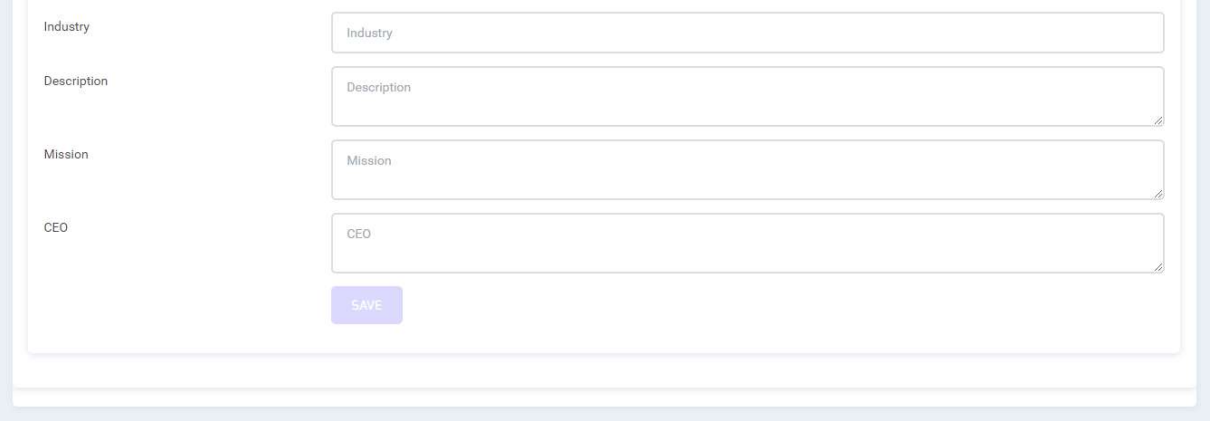


Figure 4-128: Administrator updates a company screen design

Figure 4-129: Administrator deletes a company screen design

Class Diagram

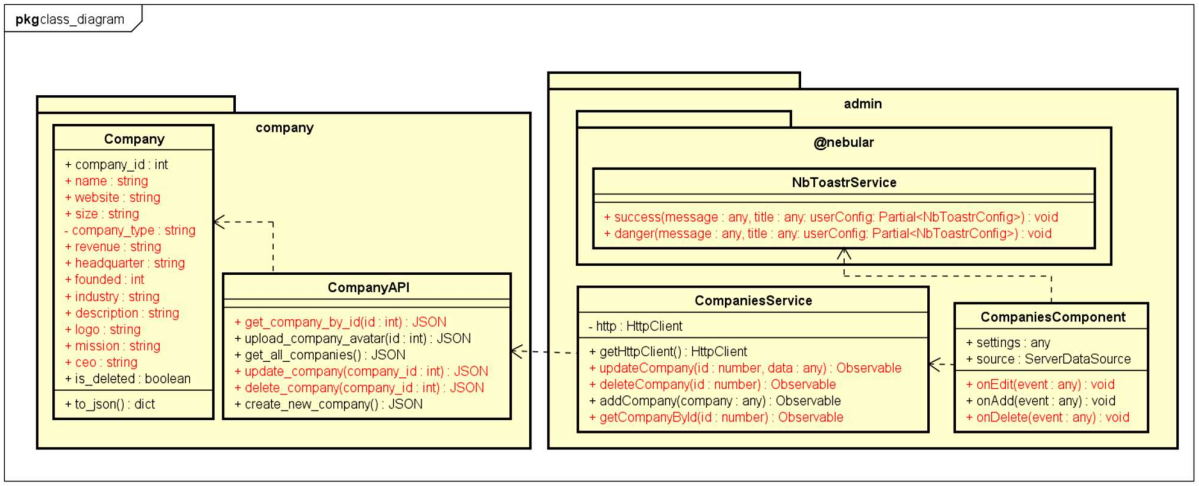


Figure 4-130: Administrator updates/deletes a company class diagram

Class Specification

CompaniesComponent

CompaniesComponent			
Physical address	/services/frontend/src/app/pages/companies/companies.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
onEdit	Edit company event		

Return Type	void		
Parameters	Name	Type	Description
	event	any	Event object containing company data
onDelete	Delete company event		
Return Type	void		
Parameters	Name	Type	Description
	event	any	Event object containing company data

CompaniesService

CompaniesService			
Physical address	/services/admin/src/app/@core/data/companies.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
updateCompany	Send company creation request to the API		
Return Type	Observable		
Parameters	Name	Type	Description
	id	number	ID of the company
	data	any	New company data to update
deleteCompany	Send company deletion request to the API		
Return Type	Observable		
Parameters	Name	Type	Description
	id	number	ID of the company
getCompanyById	Get company information from the API		
Return Type	Observable		
Parameters	Name	Type	Description
	id	number	ID of the company

CompanyAPI

CompanyAPI			
Physical address	/services/company/project/api/companies.py		
Base class	Blueprint		
Attributes			
No	Name	Type	Description
Operation			
delete_company	Delete a company by ID		
Return Type	JSON		
Parameters	Name	Type	Description
	company_id	int	ID of the company
get_company_by_id	Get company by ID		
Return Type	JSON		
Parameters	Name	Type	Description
	id	int	ID of the company
update_company	Update a company by ID		
Return Type	JSON		
Parameters	Name	Type	Description

	company_id	int	ID of the company

Company

Company			
Physical address	/services/company/project/api/models.py		
Base class	Model		
Attributes			
No	Name	Type	Description
1	name	string	Company name
2	website	string	Website URL of the company
3	size	string	Size of the company
4	company_type	string	Type of the company
5	revenue	string	Revenue of the company
6	headquarter	string	Headquarter location of the company
7	founded	int	Year the company was founded
8	industry	string	Industries the company work in
9	description	string	Description of the company
10	logo	string	Logo URL of the company
11	mission	string	Mission of the company
12	ceo	string	CEO name of the company

NbToastrService

NbToastrService			
Physical address	@nebular		
Base class	Class		
Attributes			
No	Name	Type	Description
Operation			
success			
Return Type	void		
Parameters	Name	Type	Description
	message	string	Message to be displayed
	title	string	Title of the message
	userConfig	Partial<NbToastrConfig>	Config for the toast notification
danger			
Return Type	void		
Parameters	Name	Type	Description
	message	string	Message to be displayed
	title	string	Title of the message
	userConfig	Partial<NbToastrConfig>	Config for the toast notification

Sequence Diagram

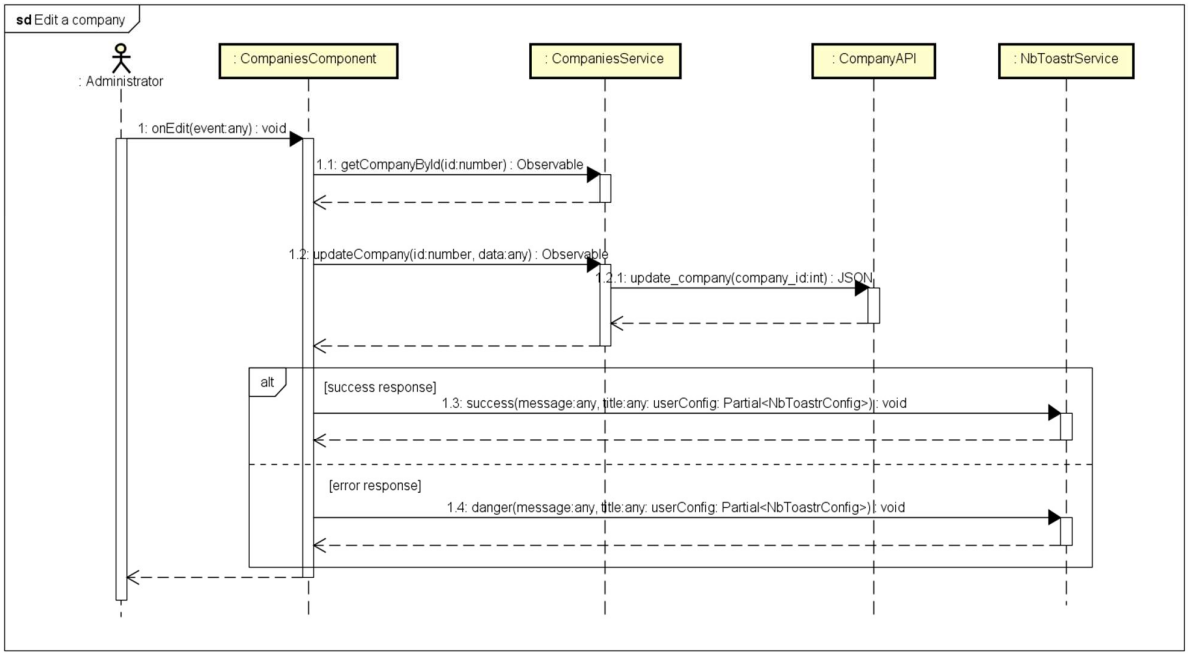


Figure 4-131: Administrator updates a company sequence diagram

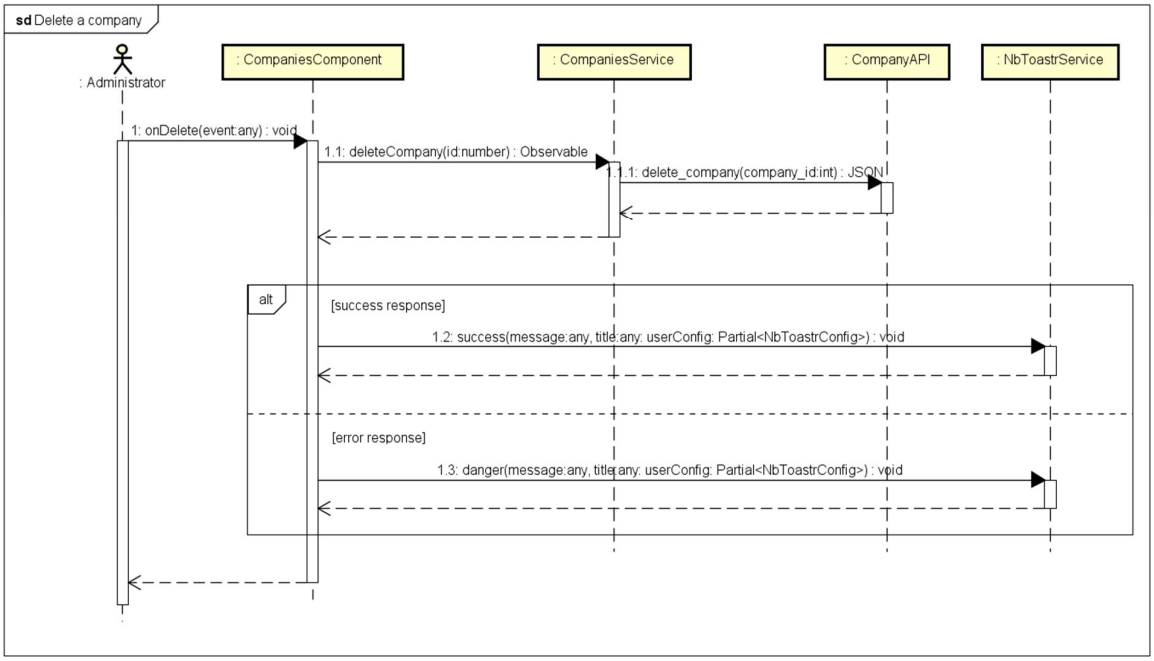


Figure 4-132: Administrator deletes a company sequence diagram

4.3.4.38 Staff views employer requests

Screen Design

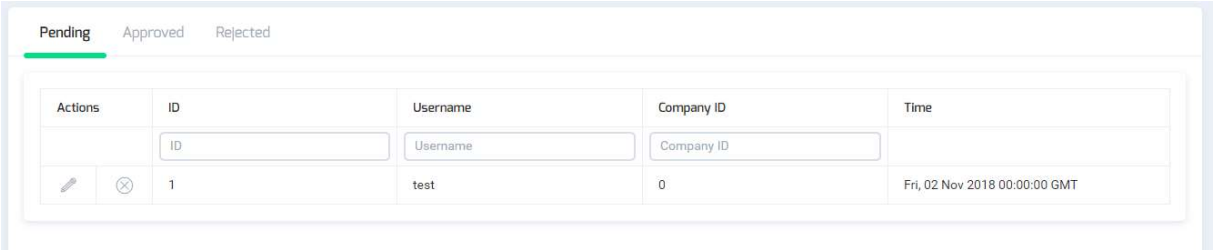


Figure 4-133: Staff views employer requests screen design

Class Diagram

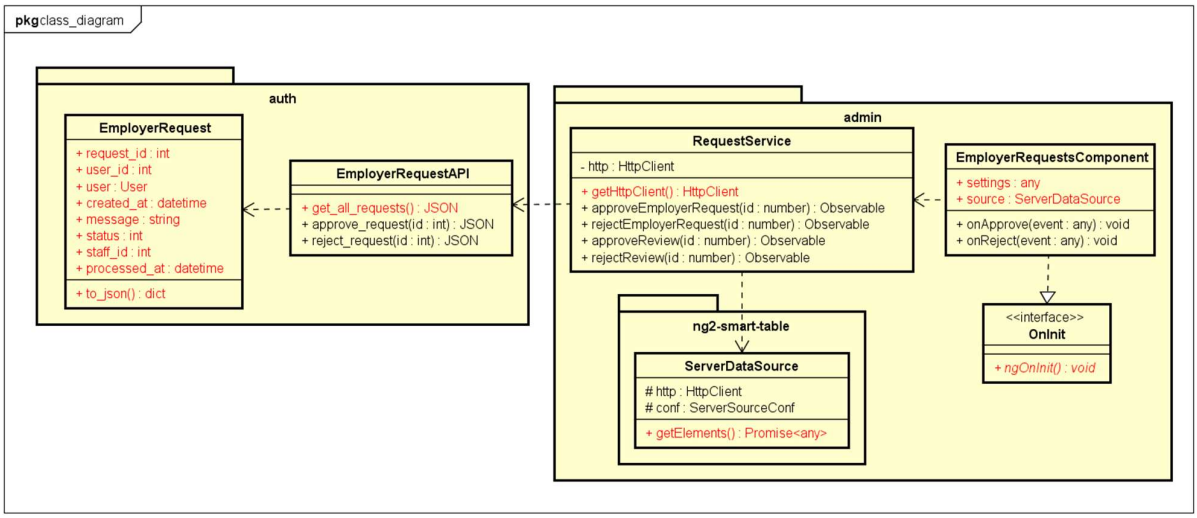


Figure 4-134: Staff views employer requests class diagram

Class Specification

EmployerRequestsComponent

EmployerRequestsComponent			
Physical address	/services/frontend/src/app/pages/requests/employer-requests.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
1	settings	any	Settings for user table
2	source	ServerDataSource	Data source for the table
Operation			
ngOnInit	Initialize component		
Return Type	void		
Parameters	Name	Type	Description

RequestService

RequestService	
Physical address	/services/admin/src/app/@core/data/request.service.ts

Base class	Class		
Attributes			
No	Name	Type	Description
1	http	HttpClient	Angular HttpClient to communicate with API endpoints
Operation			
getHttpClient	Get HttpClient instance of the service		
Return Type	HttpClient		
Parameters	Name	Type	Description

EmployerRequestAPI

EmployerRequestAPI			
Physical address	/services/auth/project/api/employer_requests.py		
Base class	Blueprint		
Attributes			
No	Name	Type	Description
Operation			
get_all_requests	Get available requests		
Return Type	JSON		
Parameters	Name	Type	Description

EmployerRequest

EmployerRequest			
Physical address	/services/auth/project/api/models.py		
Base class	Model		
Attributes			
No	Name	Type	Description
1	request_id	int	ID of the request
2	user_id	int	ID of the user who makes the request
3	user	User	User who makes the request
4	created_at	datetime	Time the request was made
5	message	string	Message of the user
6	status	int	Status of the request
7	staff_id	int	ID of the staff who process the request
8	processed_at	datetime	Time the request was processed
Operation			
to_json	Create dictionary object from the model		
Return Type	dict		
Parameters	Name	Type	Description

ServerDataSource

ServerDataSource			
Physical address	ng2-smart-table		
Base class	LocalDataSource		
Attributes			
No	Name	Type	Description

1	http	HttpClient	Angular HttpClient for communication with API
2	conf	ServerSourceConf	Configuration for source
Operation			
getElements	Fetch data from API		
Return Type	Promise<any>		
Parameters	Name	Type	Description

Sequence Diagram

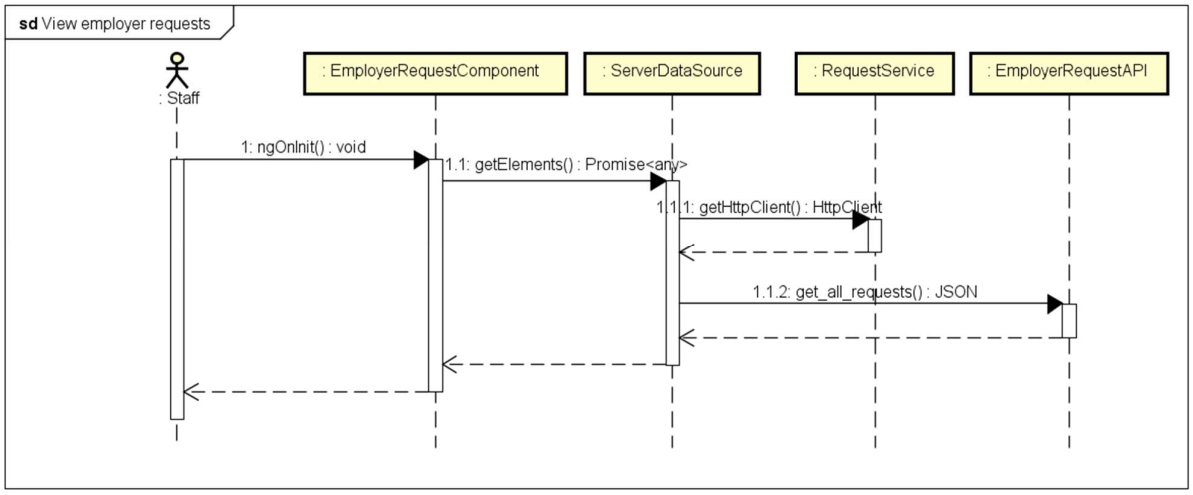


Figure 4-135: Staff views employer requests

4.3.4.39 Staff approves/rejects employer requests

Screen Design

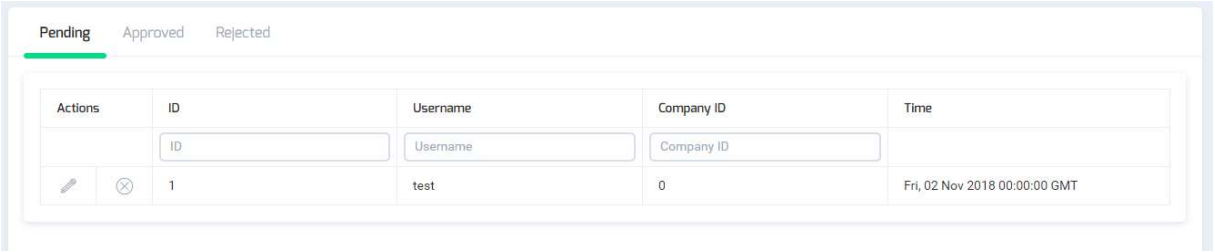


Figure 4-136: Staff approves/rejects employer requests screen design

Class Diagram

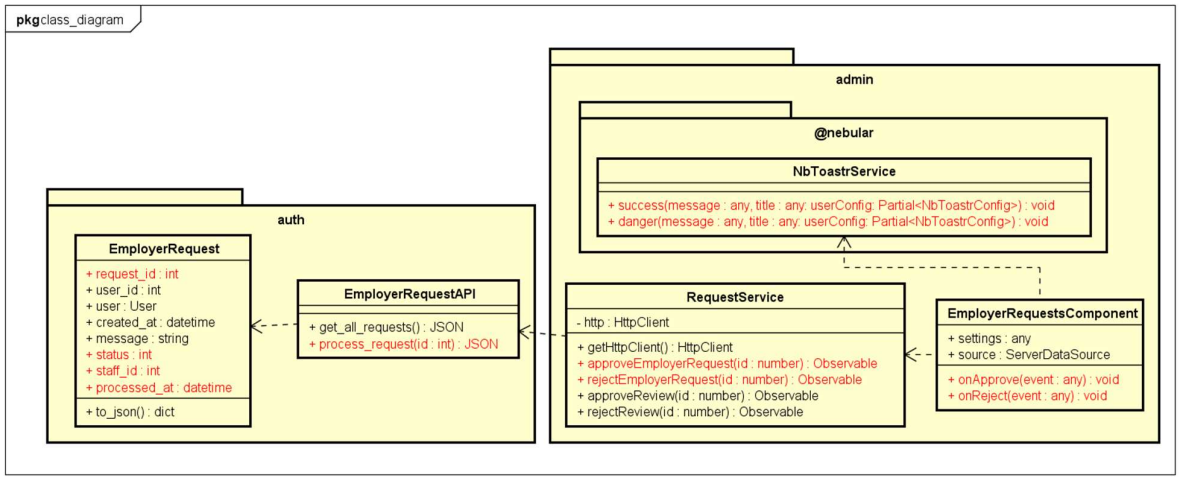


Figure 4-137: Staff approves/rejects employer requests class diagram

Class Specification

EmployerRequestsComponent

EmployerRequestsComponent			
Physical address	/services/frontend/src/app/pages/requests/employer-requests.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
1	settings	any	Settings for user table
2	source	ServerDataSource	Data source for the table
Operation			
onApprove	Request approval event		
Return Type	void		
Parameters	Name	Type	Description
	event	any	Event object containing row data
onReject	Request rejection event		
Return Type	void		
Parameters	Name	Type	Description
	event	any	Event object containing row data

RequestService

RequestService			
Physical address	/services/admin/src/app/@core/data/request.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
1	http	HttpClient	Angular HttpClient to communicate with API endpoints
Operation			
approveEmpoyer Request	Send approval to API		
Return Type	Observable		

Parameters	Name	Type	Description
	id	number	Request ID
rejectEmployerRequest	Send rejection to API		
Return Type	Observable		
Parameters	Name	Type	Description
	id	number	Request ID

EmployerRequestAPI

EmployerRequestAPI			
Physical address	/services/auth/project/api/employer_requests.py		
Base class	Blueprint		
Attributes			
No	Name	Type	Description
Operation			
process_request	Approve or reject the specified request		
Return Type	JSON		
Parameters	Name	Type	Description
	id	int	Request ID

EmployerRequest

EmployerRequest			
Physical address	/services/auth/project/api/models.py		
Base class	Model		
Attributes			
No	Name	Type	Description
1	request_id	int	ID of the request
2	user_id	int	ID of the user who makes the request
3	user	User	User who makes the request
4	created_at	datetime	Time the request was made
5	message	string	Message of the user
6	status	int	Status of the request
7	staff_id	int	ID of the staff who process the request
8	processed_at	datetime	Time the request was processed
Operation			
to_json	Create dictionary object from the model		
Return Type	dict		
Parameters	Name	Type	Description

Sequence Diagram

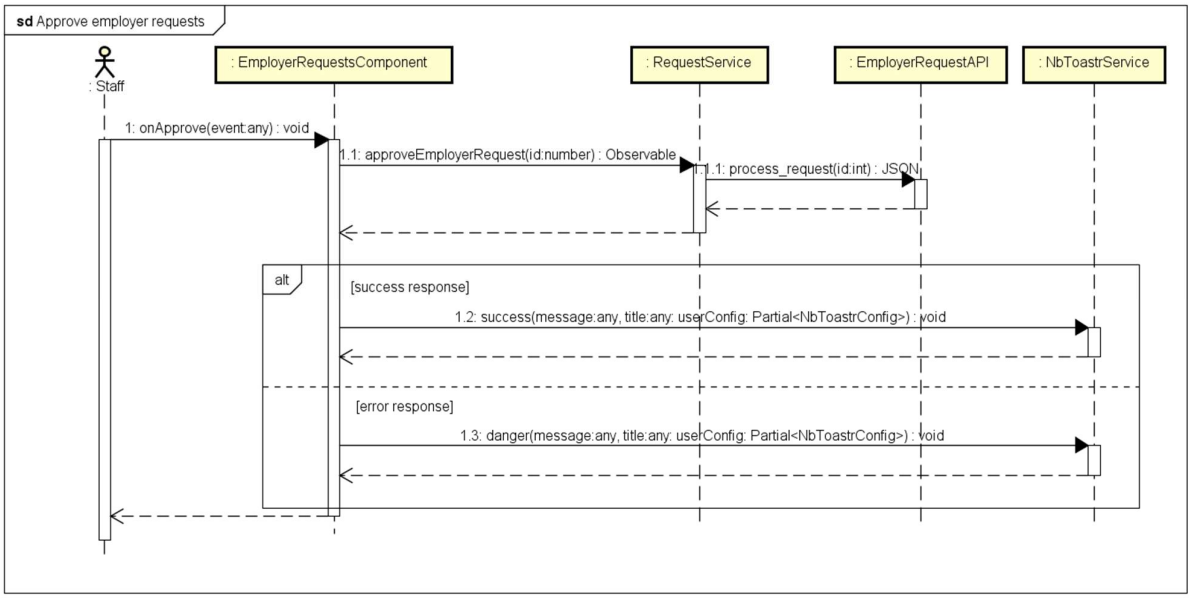


Figure 4-138: Staff approves employer requests sequence diagram

4.3.4.40 Staff views user reviews

Screen Design

Figure 4-139: Staff views user reviews screen design

Class Diagram

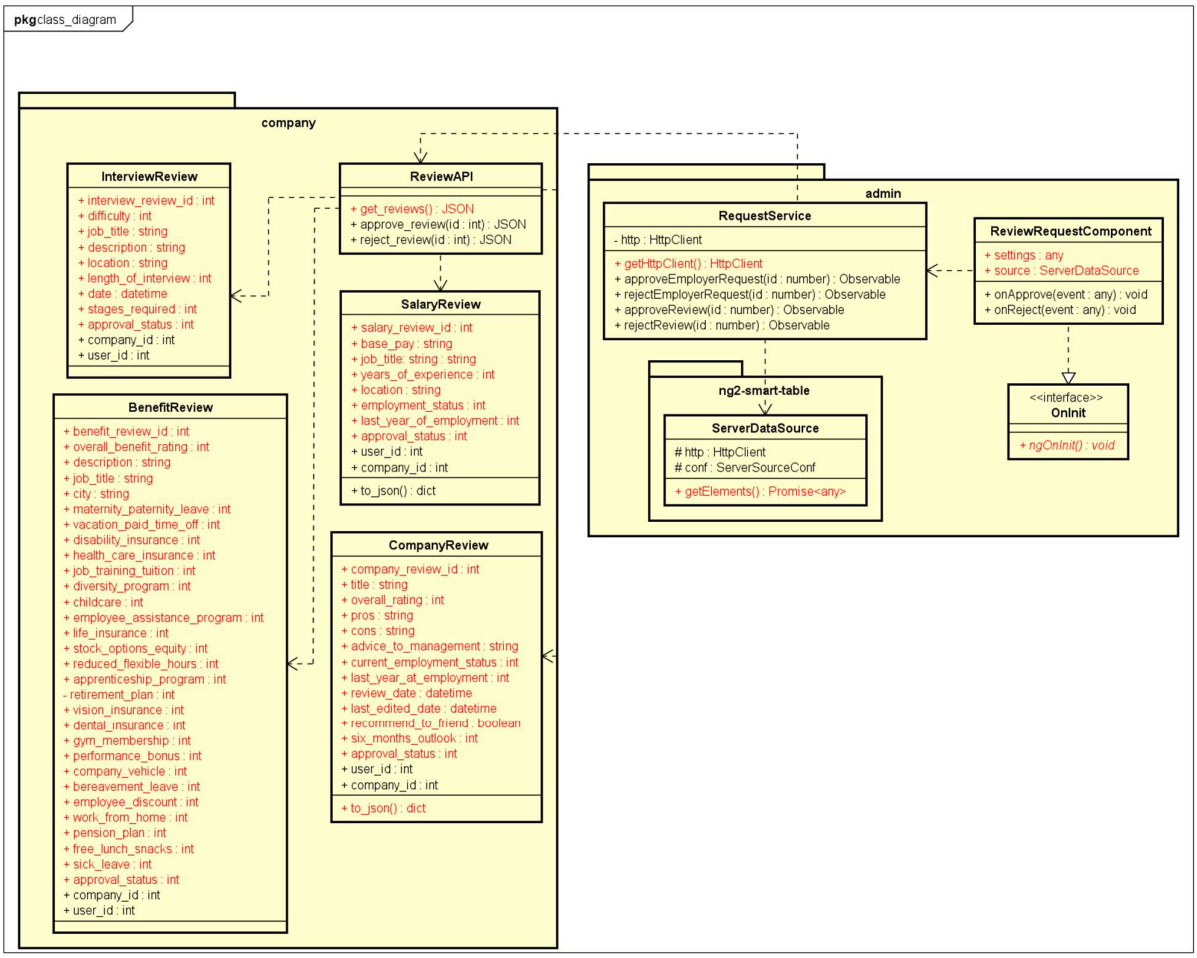


Figure 4-140: Staff views user reviews class diagram

Class Specification

ReviewRequestsComponent

ReviewRequestsComponent			
Physical address	/services/frontend/src/app/pages/requests/review-requests.component.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
1	settings	any	Settings for user table
2	source	ServerDataSource	Data source for the table
Operation			
ngOnInit	Initialize component		
Return Type	void		
Parameters	Name	Type	Description

RequestService

RequestService	
Physical address	/services/admin/src/app/@core/data/request.service.ts

Base class	Class		
Attributes			
No	Name	Type	Description
1	http	HttpClient	Angular HttpClient to communicate with API endpoints
Operation			
getHttpClient	Get HttpClient instance of the service		
Return Type	HttpClient		
Parameters	Name	Type	Description

ReviewAPI

ReviewAPI			
Physical address	/services/company/project/api/reviews.py		
Base class	Blueprint		
Attributes			
No	Name	Type	Description
Operation			
get_reviews	Get available reviews		
Return Type	JSON		
Parameters	Name	Type	Description

ServerDataSource

ServerDataSource			
Physical address	ng2-smart-table		
Base class	LocalDataSource		
Attributes			
No	Name	Type	Description
1	http	HttpClient	Angular HttpClient for communication with API
2	conf	ServerSourceConf	Configuration for source
Operation			
getElements	Fetch data from API		
Return Type	Promise<any>		
Parameters	Name	Type	Description

Sequence Diagram

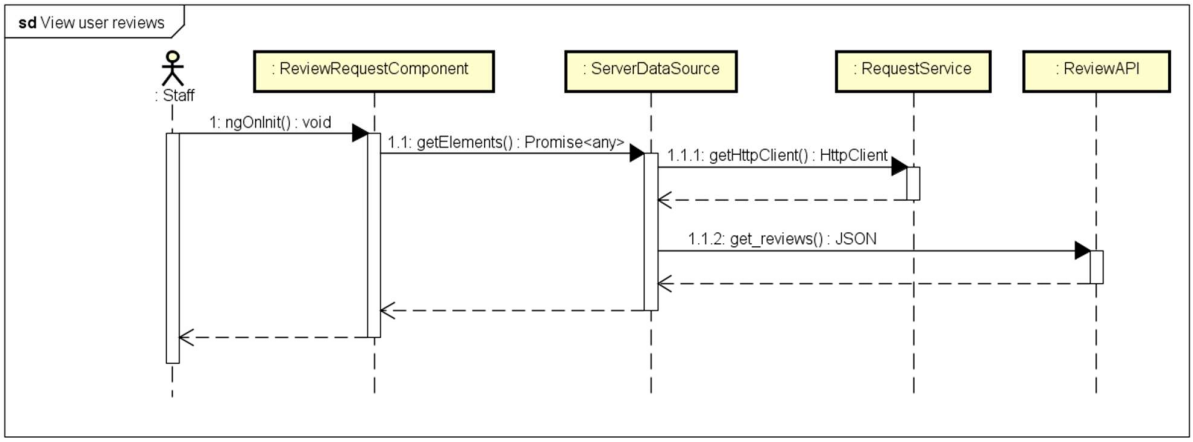


Figure 4-141: Staff views user reviews sequence diagram

4.3.4.41 Staff approves/rejects user reviews

Screen Design

Figure 4-142: Staff approves/rejects user reviews screen design

Class Diagram

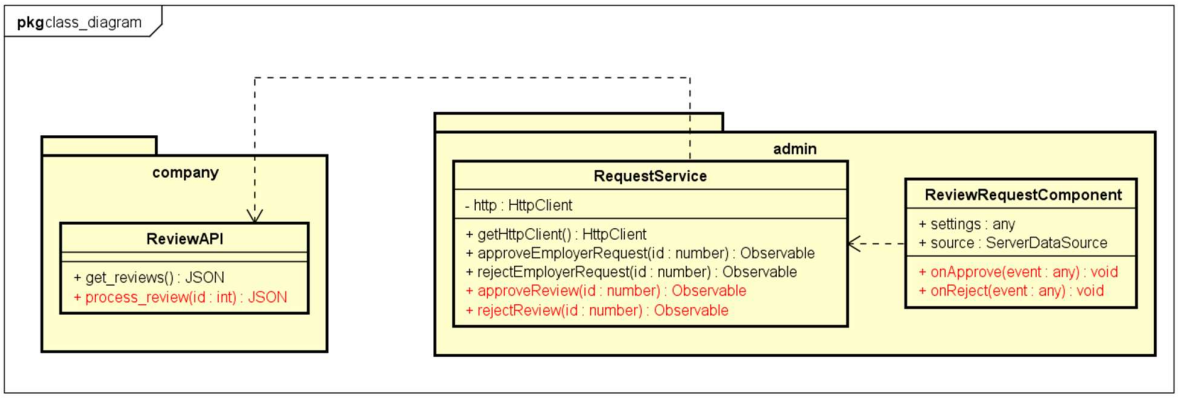


Figure 4-143: Staff approves/rejects user reviews class diagram

Class Specification

ReviewRequestsComponent

ReviewRequestsComponent	
Physical address	/services/frontend/src/app/pages/requests/review-requests.component.ts

Base class	Class		
Attributes			
No	Name	Type	Description
1	settings	any	Settings for user table
2	source	ServerDataSource	Data source for the table
Operation			
onApprove	Review approval event		
Return Type	void		
Parameters	Name	Type	Description
	event	any	Event object containing row data
onReject	Review rejection event		
Return Type	void		
Parameters	Name	Type	Description
	event	any	Event object containing row data

RequestService

RequestService			
Physical address	/services/admin/src/app/@core/data/request.service.ts		
Base class	Class		
Attributes			
No	Name	Type	Description
1	http	HttpClient	Angular HttpClient to communicate with API endpoints
Operation			
approveReview	Send review approval request to API		
Return Type	JSON		
Parameters	Name	Type	Description
	id	number	Review ID
rejectReview	Send review rejection request to API		
Return Type	JSON		
Parameters	Name	Type	Description
	id	number	Review ID

ReviewAPI

ReviewAPI			
Physical address	/services/company/project/api/reviews.py		
Base class	Blueprint		
Attributes			
No	Name	Type	Description
Operation			
process_review	Approve or reject a review		
Return Type	JSON		
Parameters	Name	Type	Description
	id	int	Review ID

Sequence Diagram

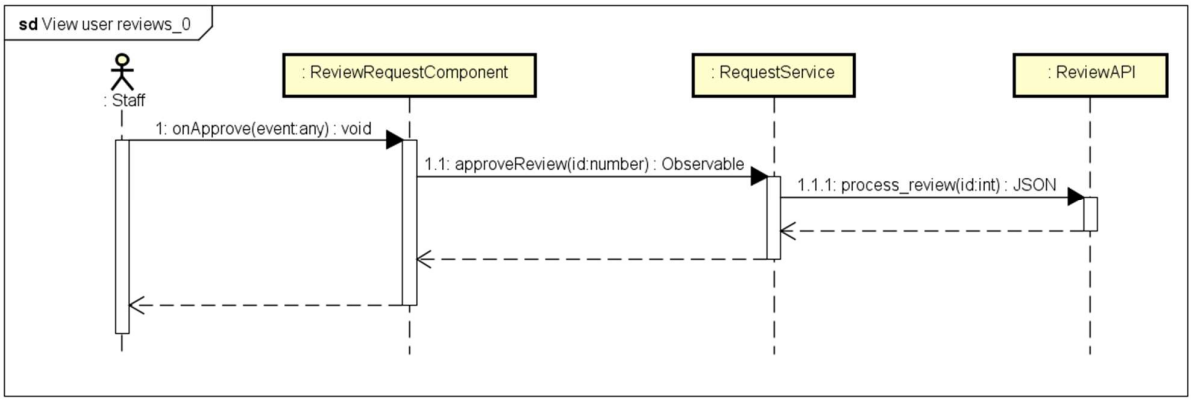


Figure 4-144: Staff approves/rejects user reviews sequence diagram

Chapter 5 : Software Testing Documentation

5.1 Introduction

5.1.1 Purpose

This chapter's primary goal is to create testing plans and execute the defects detection and prevention procedures, which may cause software malfunctioning. Another objective of this chapter is to provide details about the software quality and to ensure that the end result meets the business and user requirements. This chapter consists of these following parts:

- Scope of Testing.
- Testing Tool and Environment.
- Resources & responsibilities.
- Test strategy: Test approach, test stages.
- Test schedule.
- Feature to be tested.
- Feature not to be tested.
- Defect Log.
- Test report.

5.1.2 Scope of testing

- **Stages of testing:**

There are 4 phases in the Testing Process: Unit testing, Integration testing, System testing.

ID	Test Stages	Description
1	**Unit testing**	Unit testing is a software testing method by which small units of source code are tested to determine whether they meet the requirements.
2	**Integration testing**	Integration testing is a software testing method in which individual software modules are combined and tested as a group. Integration test's input modules that have been unit tested are aggregated and undergoes integration test plan and delivers an output that is ready for system testing.
3	**System testing**	System Testing is the testing of a complete and fully integrated software product. System Testing is actually a series of different tests whose purpose is to exercise the full computer-based system.[15]

- **Type of testing**

The test team has to test the following type on Google Chrome

- GUI test
- Performance test
- Regression test
- Unit test

[15] https://www.guru99.com/system-testing.html

- **Range of testing**
- Team performs all functions defined in the SRS based on the approved version.

5.2 Test plan

5.2.1 Testing tools and environment

5.2.1.1 Testing tools

5.2.1.1.1 GlassCV Front-end and Project testing

- **Chrome Developer Tools: To view logs, inspect elements.**

Figure 5-1: Chrome Dev Tools

- **Backlog: A bug control service to log, manage and resolve bugs.**
- **Microsoft Excel: To manage test cases**

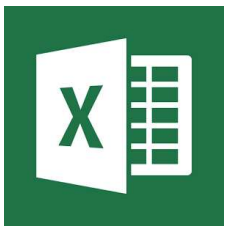


Figure 5-2: Microsoft Excel

- **Postman: A tool to test API endpoints and returned results.**

Figure 5-3: Postman

5.2.1.1.2 GlassCV API testing

5.2.1.2 Testing environment

Type of testing	Software	Hardware

System test	- Chrome version 71.0.3578.98 (Official Build 64-bit)	Personal computer for developing with the minimum configuration: - Windows 10 Education 64-bit. - Intel® Core™ i7 7700K. - Installed memory (RAM): 16.00GB
Unit test and API testing	- Postman	Personal computer for run testing with minimum configuration - Windows 10 Education 64-bit. - Intel® Core™ i7 7700K. - Installed memory (RAM): 16.00GB

5.2.2 Resources and responsibilities

ID	Resources	Responsibilities
1	Project Manager	• Responsible for Project Schedules and overall success of the project. • Review Test-case and report.
2	Tester	• Preforming the actual system testing. • Manage test resource and assign test tasks. • Create Test Plan. • Create Test Cases. • Create Test Report. • Execute Test. • Test Log report.
3	Developer	• Create unit test and integration test scripts. • Fix bugs.

5.2.3 Test strategy

5.2.3.1 Test model

Overall, GlassCV's deploys the Iterative and Incremental Software Process Model, and the entire system is comprised of 2 main systems: backend API services & front-end services.

Since APIs lack a GUI and need to change source code rapidly as GlassCV Frontend requires, so that GlassCV API applies Test-driven development (TDD) and Behavior Driven Development (BDD) process, which covers source code by Unit testing and API testing at the message layer. In the development time, whenever we add a new feature or change the old features, we will add/modify the tests first, then write code to make the test pass then refactor the code and refactor the test at the last.

GlassCV API has 2 levels of test:

- Unit testing: Automation tests that cover logic of Models and Libraries
- API testing: Automation tests that involve testing APIs directly (in isolation) to determine whether APIs return the correct response (in the expected format) for a broad range of feasible requests, react properly to edge cases such as failures and unexpected/extreme inputs.

5.2.3.2 Test types

- **Unit testing:**
  - Testing individual methods, functions, model class and library class.
  - Unit test also includes database testing to verify constraint, transaction, default value, data types, data format, and check null and junk characters which are mentioned in database design and software requirement.
  - Test case will have to cover all logic branch that function or method could execute with difference data input. Another alternative logic branch should be covered if not, that logic branch should be detected at API testing level.
- **API testing:**
  - Involves testing APIs directly to determine if they meet expectations for functionality, reliability, performance, and security. API testing will test all of individual implemented API of GlassCV API.
  - Test case will verify constraint of data which be mention in Business rule
  - Basically, almost all API test cases are executed as automation test. After that all API with standard sample datasets will be saved and confirmation tests will be executed by using Postman with developer's local database.
- **UI testing**
  - User Interface testing verifies a user's interaction with the software. The goal of GUI testing is to ensure that the GUI provides the user with an appropriate access and navigation through the functions of the target-of-test. In addition, GUI testing ensures that the objects within the GUI function as expected and conform to requirement
  - GUI test will be performed fully on all screens.
  - This test targets to cover the verification of the overall look and feel of the GlassCV system including initial position, font, text size, color, focus, initial button, tab order, label, screen sizes and sentences width.
  - All GUI elements for size, position, width, length and acceptance of characters or numbers. Text font is readable.
    - Images have high resolution and properly adjusted for different screen sizes.
    - GUI elements are properly aligned for different screen resolutions.
- **Regression testing**
  - Regression testing is to confirm that the bug is removed with regards to their impacts when a developer fixes a bug. We conduct regression testing include bug fixes, configurations changed, software enhancements.

5.2.3.3 Test schedule

Table below is the Test Schedule for GlassCV Project:

Test Schedule	Start Date	End Date
Phase 1: Job seeker features	20/09/2018	10/10/2018
Unit Testing and API Testing	20/09/2018	26/09/2018
User Interface Testing	22/09/2018	27/09/2018
System Testing	28/09/2018	30/09/2018
Regression Tests	29/09/2018	10/10/2018

Phase 2: Employer & Staff	15/10/2018	26/11/2018
Unit Testing and API Testing	15/10/2018	20/10/2018
User Interface Testing	22/10/2018	26/11/2018
System Testing	26/10/2018	15/11/2018
Regression Tests	16/11/2018	26/11/2018

Table 5-1: Test schedule

5.2.4 Features to be tested

All features are listed in the use case list.

5.2.5 Features not to be tested

All features are listed in 1.5.1.3 above will not to be tested.

5.3 Test Case

5.3.1 Unit testing and API testing

Unit testing and API testing will be done by the developers and approved by the development team leader.

The GlassCV development team embrace these features to gain the following advantages:

- Reduce the number of bugs in production code.
- Save development & testing time.
- Automation tests can be run frequently.
- Make it easier to change and refactor code by improving the design of code especially with Test-Driven Development.
- Easier to maintain than GUI tests which are difficult to maintain with the short release cycles and frequent changes and with a complex system
- Reduce cost of resource to corresponding GUI testing.

5.3.1.1 Unit testing framework

- For API testing and Unit testing, we use Python's default unittest testing framework and the *coverage* package to generate coverage reports.
- For front-end unit testing, we use Jasmine testing framework with Karma as the test runner. Coverage reporting is done via istabul library.
- All testing frameworks and libraries will be installed to GlassCV front-end automatically by using npm package manager.

```
$ npm install
```

- Unit tests use a separate environment which is similar to production environment and includes: Database, Google Application ID, Google Cloud Storage, Google PubSub. For testing complex code that requires running external libraries, unittest.mock library is used to simulate operations.

```
FLASK_ENV=development
```

```
APP_SETTINGS=project.config.DevelopmentConfig
SECRET_KEY=SECRET_KEY
MYSQL_USER=root
MYSQL_PASSWORD=1234321
MYSQL_HOST=glasscv-mysql
MYSQL_PORT=3306
PYTHONUNBUFFERED=1
GOOGLE_APPLICATION_CREDENTIALS=/usr/src/credentials/GlassCV-aa2097eeba55.json
GOOGLE_PROJECT_ID=glasscv-217004
TOPIC_USER=user
TOPIC_USER_TEST=user-test
TOPIC_JOB=job
TOPIC_JOB_TEST=job-test
TOPIC_MAIL=mail
TOPIC_MAIL_TEST=mail-test
MAIL_SUB_USER=mail-sub-user
MAIL_SUB_USER_TEST=mail-sub-user-test
PROFILE_SUB_USER=profile-sub-user
PROFILE_SUB_USER_TEST=profile-sub-user-test
TRAINING_SUB_USER=training-sub-user
TRAINING_SUB_USER_TEST=training-sub-user-test
TRAINING_SUB_JOB=training-sub-job
TRAINING_SUB_JOB_TEST=training-sub-job-test
PROFILE_SUB_JOB=profile-sub-job
PROFILE_SUB_JOB_TEST=profile-sub-job-test
JOB_SUB_MAIL=job-sub-mail
JOB_SUB_MAIL_TEST=job-sub-mail-test
ELK_VERSION=6.5.1
GOOGLE_OAUTH_CLIENT_ID=884516740145-
nsgjulvp1aicqq2pftdlvfmb3r33aukh.apps.googleusercontent.com
FRONTEND_URL=http://localhost:4200
FRONTEND_URL_PROD=http://crystalcv.gustof.win
```

Figure 5-4: Test environment configuration file (.env)

- Unit testing scripts are created manually and saved to **api/tests** directory of GlassCV API services and along with components in the frontend application.

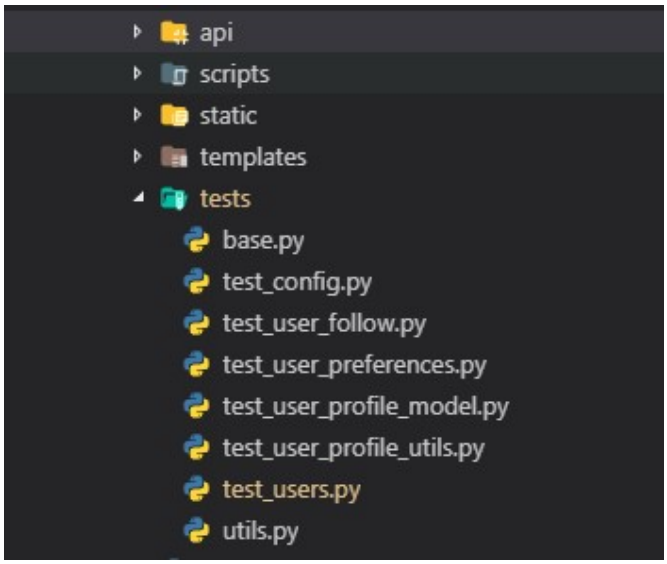


Figure 5-5: Test directory structure

- Unit tests focus on individual functions in a class and are created as in the picture.

```python
class TestUserPreferences(BaseTestCase):

    def test_create_preferred_company(self):
        """Ensure company preferences can be created in the database."""
        create_or_update_profile(1, 'Test')
        preferred_company = create_or_update_preferred_company(1, '["51 - 200", "201 - 500"]', '["Advertising", "Creative"]')
        self.assertTrue(isinstance(preferred_company, PreferredCompany))

    def test_update_preferred_company(self):
        """Ensure company preferences can be updated in the database."""
        create_or_update_profile(1, 'Test')
        create_or_update_preferred_company(1, '["51 - 200", "201 - 500"]', '["Advertising", "Creative"]')
        preferred_company = create_or_update_preferred_company(1, '["0 - 50", "201 - 500"]', '["Creative"]')
        self.assertTrue(isinstance(preferred_company, PreferredCompany))

    def test_preferred_company(self):
        """Test get Preferred company API endpoint"""
        create_or_update_profile(999, 'Test')
        preferred_company = create_or_update_preferred_company(999, '["51 - 200", "201 - 500"]', '["Advertising", "Creative"]')
        with self.app.test_client() as client:
            resp = client.get('/me/preferred_company',
                              headers={'GLASSCV-USER-ID': 999})
            data = json.loads(resp.data.decode())
            self.assertEqual(200, resp.status_code)
            self.assertEqual('["Advertising", "Creative"]', data['data']['industry_name'])
            self.assertEqual('["51 - 200", "201 - 500"]', data['data']['company_size'])
            self.assertEqual(999, data['data']['user_id'])
```

Figure 5-6: Unit test case sample
This factory function used to create and update User location and comapy preferences

Coverage report: 93%

Module ↓	statements	missing	excluded	branches	partial	coverage
project/api/accounts.py	92	0	0	40	0	100%
project/api/auth.py	112	0	0	16	1	99%
project/api/employer_requests.py	89	0	0	32	0	100%
project/api/filters/auth.py	21	5	0	0	0	76%
project/api/filters/common.py	3	2	0	0	0	33%
project/api/filters/company.py	23	4	0	0	0	83%
project/api/filters/decorators.py	40	16	0	18	2	52%
project/api/filters/job.py	12	2	0	0	0	83%
project/api/filters/recommend.py	7	1	0	0	0	86%
project/api/filters/user.py	37	2	0	0	0	95%
project/api/models.py	71	0	0	2	0	100%
project/api/utils.py	12	0	0	2	0	100%
Total	**519**	**32**	**0**	**110**	**3**	**93%**

coverage.py v4.5.1, created at 2018-12-24 15:29

Figure 5-7: Test coverage report – auth API

5.3.2 System testing

Detailed Test cases will be described in ***TestCase_Final.xlsx*** file.

As a standard definition, GlassCV Project defines that a test case is:

- A set of test data and test programs (test scripts) and their expected results. A test case validates one or more system requirements and generates a pass or fail
- A good test case should follow two basic aspects, the Contents and the Style. Test cases for functional testing are derived from the target of test's use cases Test cases should be developed for each use case scenario. The use case scenarios are identified by describing the paths through the use case that traverse the basic flow and alternate flows start to finish through the use case.
- By using good automation test and using, GlassCV Project System testing will not focus on common logic of system like length of text but focus on behavior of website and aims to validate that all software module dependencies are functionally correct, and that data integrity is maintained between separate modules for the entire solution.

5.3.3 Defect Log

GlassCV project used http://www.nulab.com to manager tasks and defects.

Every member of GlassCV project creates an account in NuLab to take part in activities: control bugs, fix bugs, re-test bugs and close bug. Bug will be log by tester or developer in develop progress.

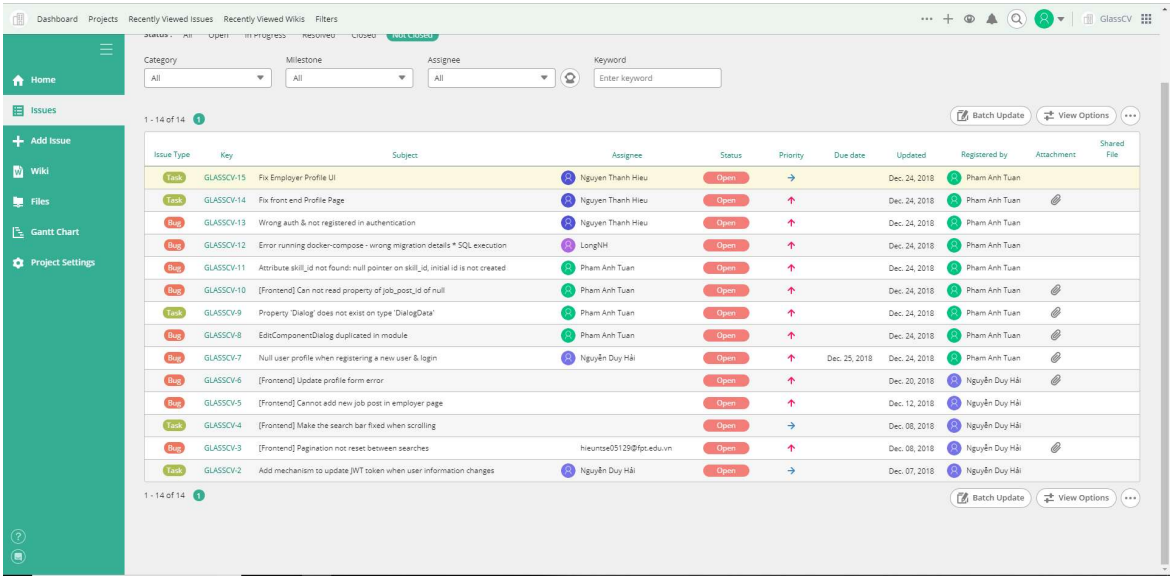


Figure 5-8: Control task and bug with NuLab

5.4 Test Report

5.4.1 Unit test case report

The contents of the Unit Test Case Report are shown in the table below:

Type of test case	Test Case	Pass	Fail	Not available	Number of Test Case
API Test - Job seeker	View user profile	2	0	0	2
	Request to become an employer to create jobs recruitment posts	3	0	0	3
	Create a review about a company	3	0	0	3
	Edit a company review	3	0	0	3
	Create interview review about a company	3	0	0	3
	Edit review about the company's interview	3	0	0	3
	Follow/Unfollow a company	2	0	0	2
	View all companies being followed by the user	1	0	0	1
	Edit preferred locations	3	0	0	3
	Edit preferred company	3	0	0	3
	Update User Profile	3	0	0	3
	Upload CV	3	0	0	3
	Total	**32**	**0**	**0**	**32**
API Test - Staff	View all users	13	0	0	13
	Update user	7	0	0	7
	Ban/Unban a user	5	0	0	5
	View all companies	8	0	0	3
	Create a new company	3	0	0	21
	Update/delete a company	3	0	0	11
	Total	**39**	**0**	**0**	**39**
	Create a job post	3	0	0	26

API Test - Employer	Edit/Delete created job posts	3	0	0	33
	Create employer profile	3	0	0	32
	Total	**9**	**0**	**0**	**9**
API Test - Authentication	Login	10	0	0	10
	Register	3	0	0	3
	Log out	1	0	0	1
	Verify user	3	0	0	3
	Total	**17**	**0**	**0**	**17**
Unit test - Library	Elastic search	8	0	0	8
	Total	**8**	**0**	**0**	**8**
Unit test - Miscellaneous	Utility functions	6	0	0	6
	Worker scripts	5	0	0	5
	Configurations	24	0	0	24
	Total	**35**	**0**	**0**	**35**
Unit test - Model	Job	2	0	0	2
	User	8	0	0	8
	Company	6	0	0	6
	Authentication	11	0	0	11
	Total	**27**	**0**	**0**	**27**
Total of Test Cases		**167**	**0**	**0**	**167**

Table 5-2: Unit test case report

5.4.2 Unit test report

Unit testing is an integral part of development process. So that, there are 2 phases as 2 phases of Iterative and Incremental Software Process Model.

Test case	Phase 1		Phase 2		Final
	Pass	Fail	Pass	Fail	
API Test - Job seeker	32	0	32	0	32
API Test - Staff	0	0	39	0	39
API Test – Employer	0	0	9	0	9
API Test – Authentication	17	0	17	0	17
Unit test – Library	8	0	8	0	8
Unit test – Miscellaneous	12	0	35	0	35
Unit test – Model	13	0	27	0	27
Total of test cases	**82**	**0**	**167**	**0**	**167**

Table 5-3: Unit test report

Phase	Coverage
Phase 1	85%
Phase 2	93%

Table 5-4: Unit test coverage report

5.4.3 System test case report

The contents of the System Test Case Report are shown in the table below:

Client	Test Case	Pass	Fail	Not available	Number of Test Case
	View user profile	**2**	0	0	2

Job Seeker	Request to become an employer to create jobs recruitment posts	4	0	0	4
	Create a review about a company	3	0	0	3
	Edit a company review	3	0	0	3
	Create interview review about a company	3	0	0	3
	Edit review about the company's interview	3	0	0	3
	Follow/Unfollow company	3	0	0	3
	View all companies being followed by the user	1	0	0	1
	Edit preferred locations	2	0	0	2
	Edit preferred company	2	0	0	2
	Update User Profile	4	0	0	4
	Upload CV	5	0	0	5
	Save a job	2	0	0	2
	Create mail notification	2	0	0	2
	Update/Delete saved jobs	3	0	0	3
	Quick apply for a job	3	0	0	3
	View all available jobs	5	0	0	5
	View recommended job posts	5	0	0	5
	Remove a recommended job post	2	0	0	2
Employer	Create a job post	4	0	0	4
	Update/Delete a job post	5	0	0	5
Guest	Normal sign in	5	0	0	5
	Sign in with Google account	2	0	0	2
	Sign up	8	0	0	8
Staff	View all users	4	0	0	4
	Update users	8	0	0	8
	Ban/Unban a user	2	0	0	2
	View all companies	4	0	0	4
	Create a new company	4	0	0	4
	Update/delete a company	5	0	0	5
Total of Test Cases		**108**	**0**	**0**	**108**

Table 5-5: System test case report

5.4.4 System test report

We execute test with 2 stages with 2 phases of process model, to finish project.

The contents of the Test Report are shown in the table below

Test Case		**Stage 1**		**Stage 2**		**Final**
		Pass	Fail	Pass	Fail	
Client	Job Seeker	57	0	0	0	57

	Employer	0	0	9	0	9
	Guest	15	0	0	0	15
	Staff	0	0	27	0	27
Total all Test Case		**72**	**0**	**36**	**0**	108

Table 5-6: System test report

Chapter 6 : User manual

6.1 Deployment guidelines

6.1.1 Environment for development

- Operating System: Windows 10 Pro
- Code Editor: Visual Studio Code
- Plugins:
  - EditorConfig for VS Code
  - TSLint
  - GitLens
  - Python
  - Docker
- Python: 3.6.6
- Docker: 18.09.0
- Docker images:
  - python
  - MySQL
  - Elasticsearch

6.1.1.1 Setup development environment

6.1.1.1.1 Install Visual Studio Code

- Go to URL https://code.visualstudio.com, click "Download".

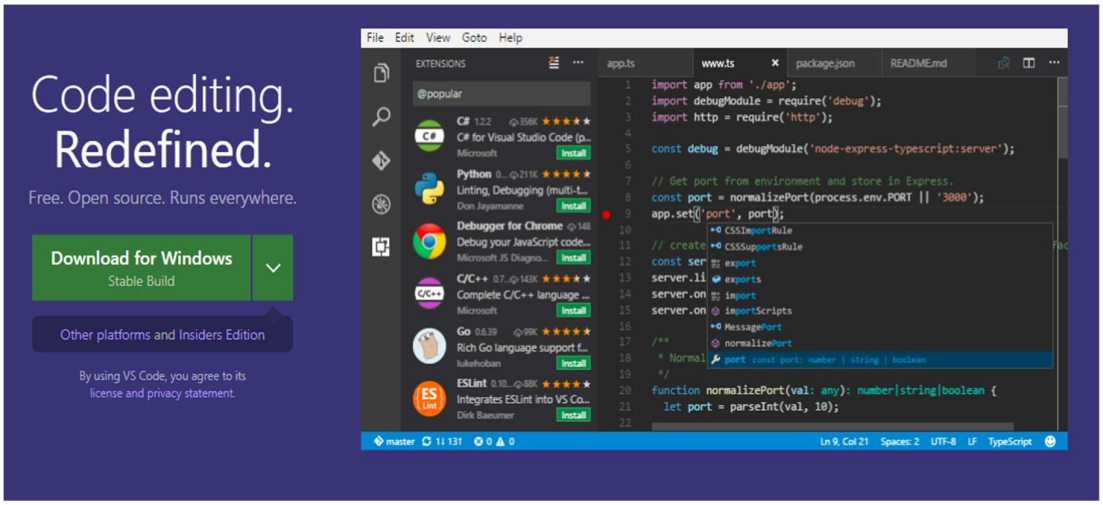


Figure 6-1: Download Visual Studio Code

- Run the downloaded executable file, follow the installation instructions.
- Start the application by opening the shortcut on desktop or from the start menu.

6.1.1.1.2 Install Docker

- Go to URL https://store.docker.com/editions/community/docker-ce-desktop-windows, Log in or create an account.
- After that, click the "Get Docker" button.

Figure 6-2: Download Docker for Windows

- Open the downloaded executable file and follow the installation instructions. Be sure to keep using Linux container when prompted to switch to Windows container.
- Start "Docker for Windows" from the shortcut on the desktop or from the start menu.
- Open a new PowerShell at the project source code folder and run the following command:

```
> docker-compose up --build -d
```

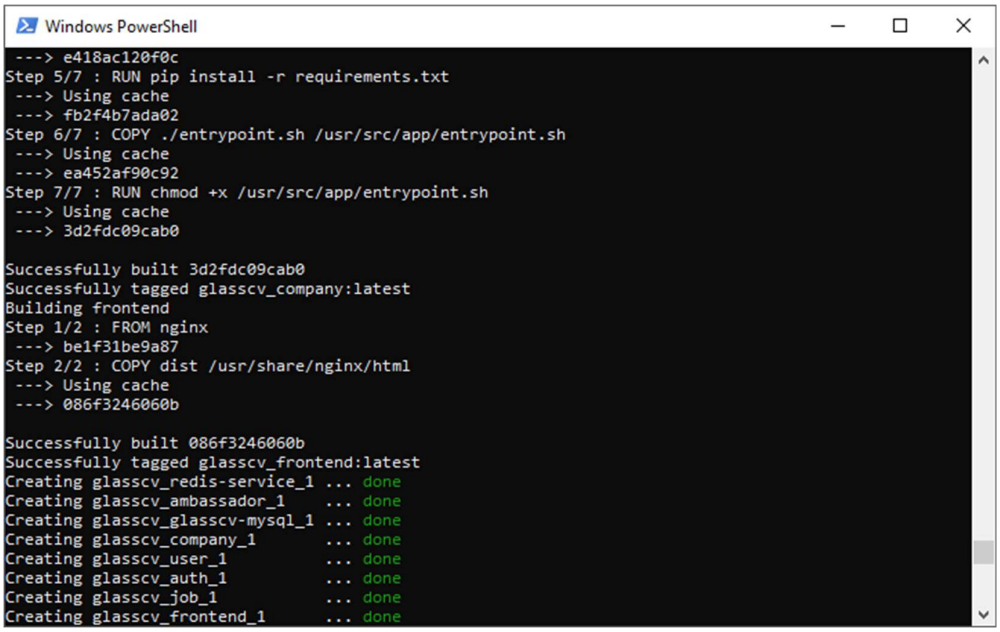


Figure 6-3: Start docker containers

- To view output generated inside containers, run:

```
> docker-compose logs
```

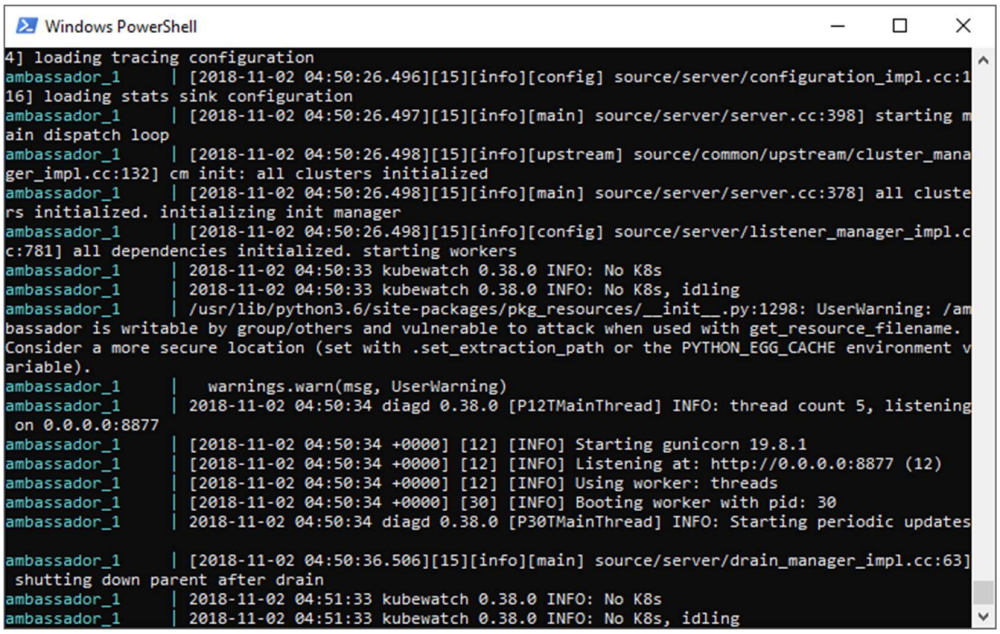


Figure 6-4: View container logs

- After development, to stop the docker containers and clean up resources, run:

```
> docker-compose down
```

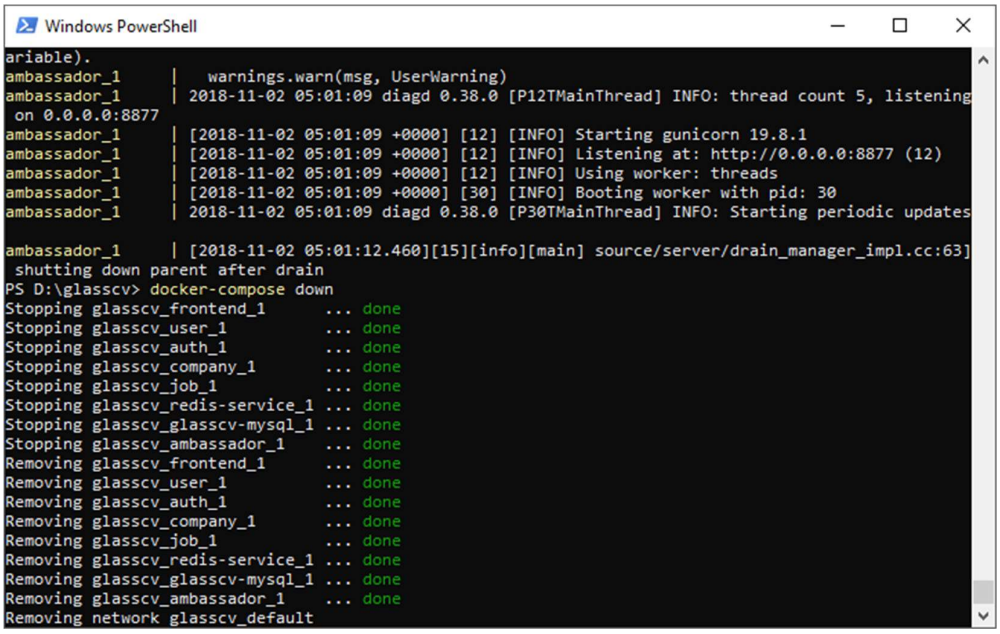


Figure 6-5: Shutdown docker containers

6.1.1.2 Run tests and linting

Go to the appropriate folders of each microservice and run the corresponding commands below:

- Angular:
  - Linting:

```
> npm run lint
```

- Testing:

```
> npm test
```

- Flask:
  - Linting:

```
> flake8 project
```

  - Testing:

```
> python manage.py test
```

  - Testing + coverage:

```
> python manage.py cov
```

6.1.1.3 Local network setup

- Run “Notepad” as Administrator.
- From File menu, click “Open”.
- Navigate to <Windows Drive>\Windows\system32\drivers\etc\.
- Choose “All Files (*.*)” from the dropdown to the right of the “File name” box.
- Choose “hosts” file and click “Open”.
- Add two following new lines at the end of the file:

```
127.0.0.1    crystalcv.test
127.0.0.1    crystalapi.test
127.0.0.1    crystaladmin.test
```

- The front-end, admin front-end and API can be accessed from http://crystalcv.test, http://crystaladmin.test, and http://crystalapi.test respectively.

6.1.2 Environment for deployment

Platform: Kubernetes Engine on Google Cloud Platform.

6.1.2.1 Install Google Cloud SDK

- Download the Google Cloud SDK installer:

  https://dl.google.com/dl/cloudsdk/channels/rapid/GoogleCloudSDKInstaller.exe
- Launch the installer and follow the prompts.
- After installation has completed, the installer presents several options:

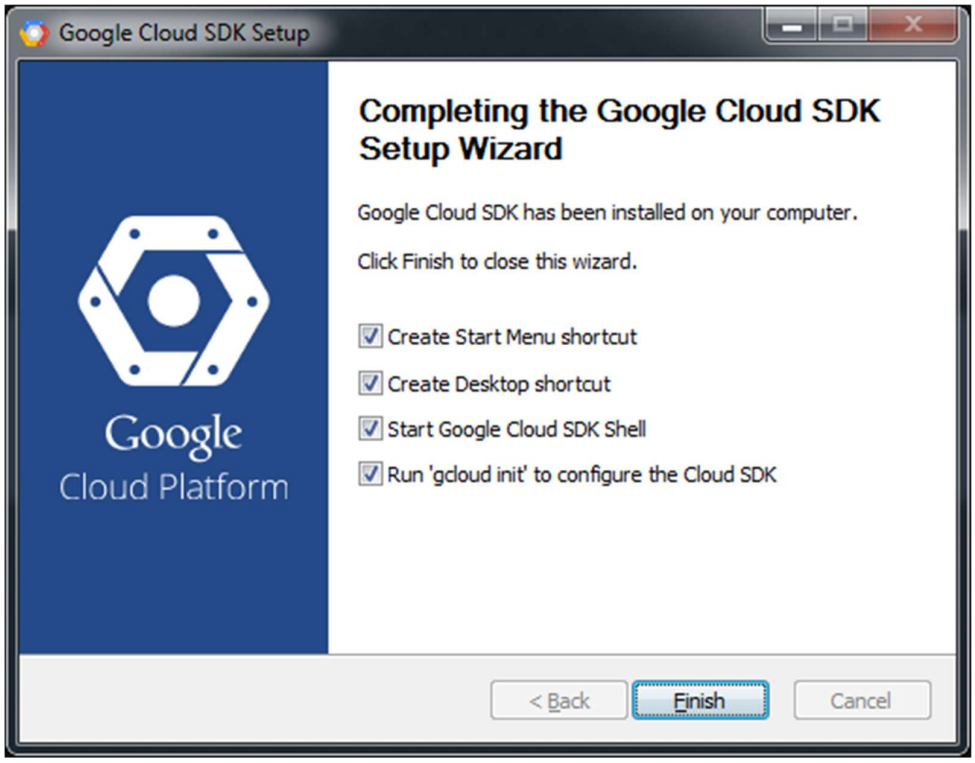


Figure 6-6: Google Cloud SDK installation

Make sure that the following are selected:

- Start Google Cloud SDK Shell
- Run 'gcloud init'

The installer then starts a terminal window and runs the gcloud init command.

Reference: https://cloud.google.com/sdk/docs/quickstart-windows

6.1.2.2 Configure Google Cloud SDK

- Run the following at a command prompt:

```
> gcloud init
```

- Accept the option to log in using your Google user account:

```
To continue, you must log in. Would you like to log in (Y/n)? Y
```

- In your browser, log in to your Google user account when prompted and click Allow to grant permission to access Google Cloud Platform resources.
- At the command prompt, select a Cloud Platform project from the list of those where you have Owner, Editor or Viewer permissions:

```
Pick cloud project to use:
  [1] [my-project-1]
 [2] [my-project-2]
 …
Please enter your numeric choice:
```

If you only have one project, gcloud init selects it for you.

- If you have the Google Compute Engine API enabled, gcloud init allows you to choose a default Compute Engine zone:

```
Which compute zone would you like to use as project default?
 [1] [asia-east1-a]
 [2] [asia-east1-b]
 …
 [14] Do not use default zone
Please enter your numeric choice:
```

gcloud init confirms that you have complete the setup steps successfully:

```
gcloud has now been configured!
You can use [gcloud config] to change more gcloud settings.

Your active configuration is: [default]
```

Reference: https://cloud.google.com/sdk/docs/quickstart-windows

6.1.2.3 Setup Kubernetes Engine on Google Cloud

- Create a kubernetes cluster named "glasscv-cluster":

```
> gcloud container clusters create glasscv-cluster --zone asia-southeast1-a
```

- Get credentials for kubectl:

```
> gcloud container clusters get-credentials glasscv-cluster --zone asia-southeast1-a
```

- Log into Dockerhub if not already:

```
> docker login --username example
```

- Create secrets for MySQL:

```
> kubectl create secret generic mysql-pass --from-literal=password=YOUR_PASSWORD
```

6.1.2.4 Build and push docker images

- For frontend and admin frontend service, remember to build the production app first.

```
> npm run build
```

- Then build the docker images (run these commands inside corresponding microservice project folder e.g. frontend, user, etc.)

```
>  docker build -f Dockerfile -t haivp3010/<service_name>
```

- Push the built image to Dockerhub.

```
>  docker push haivp3010/<service_name>
```

6.1.2.5 Create Kubernetes deployments

Navigate the terminal to the deployment directory, then for each yaml file in the deployment folder, run:

```
> kubectl apply -f <deployment_name>.yaml --record
```

6.1.2.6 Setup SendGrid

6.1.2.6.1 Register an account at SendGrid

- Go to https://signup.sendgrid.com/.
- Follow the registration screen to register a new account.

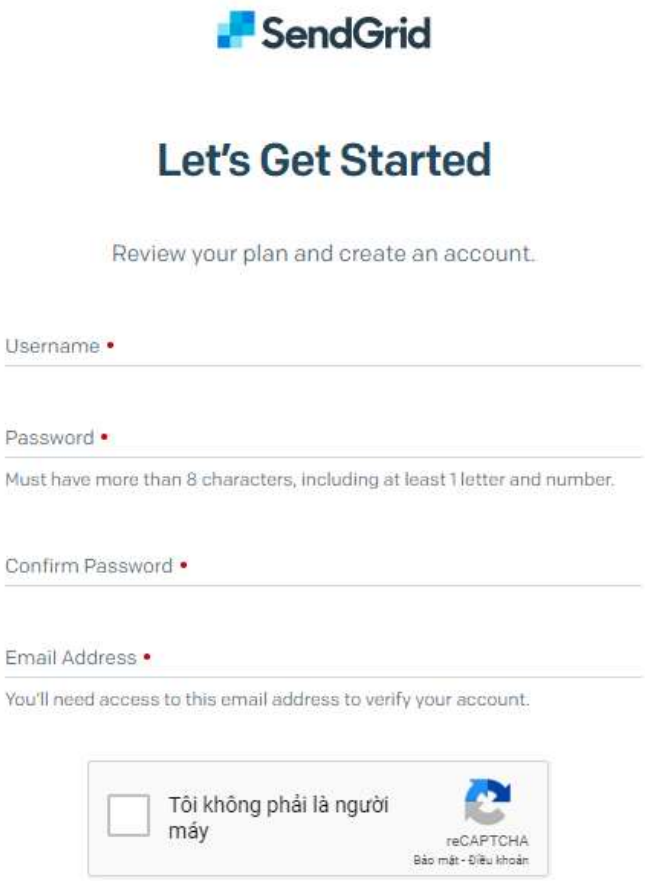


Figure 6-7: Register account at SendGrid

6.1.2.6.2 Create a API Key

- Go to https://app.sendgrid.com and sign in
- From the menu bar on the left, click **Settings** > **API Keys** to navigate to API Keys management page.
- Click **Create API Key** at the top right corner and follow instructions to create a new API Key.

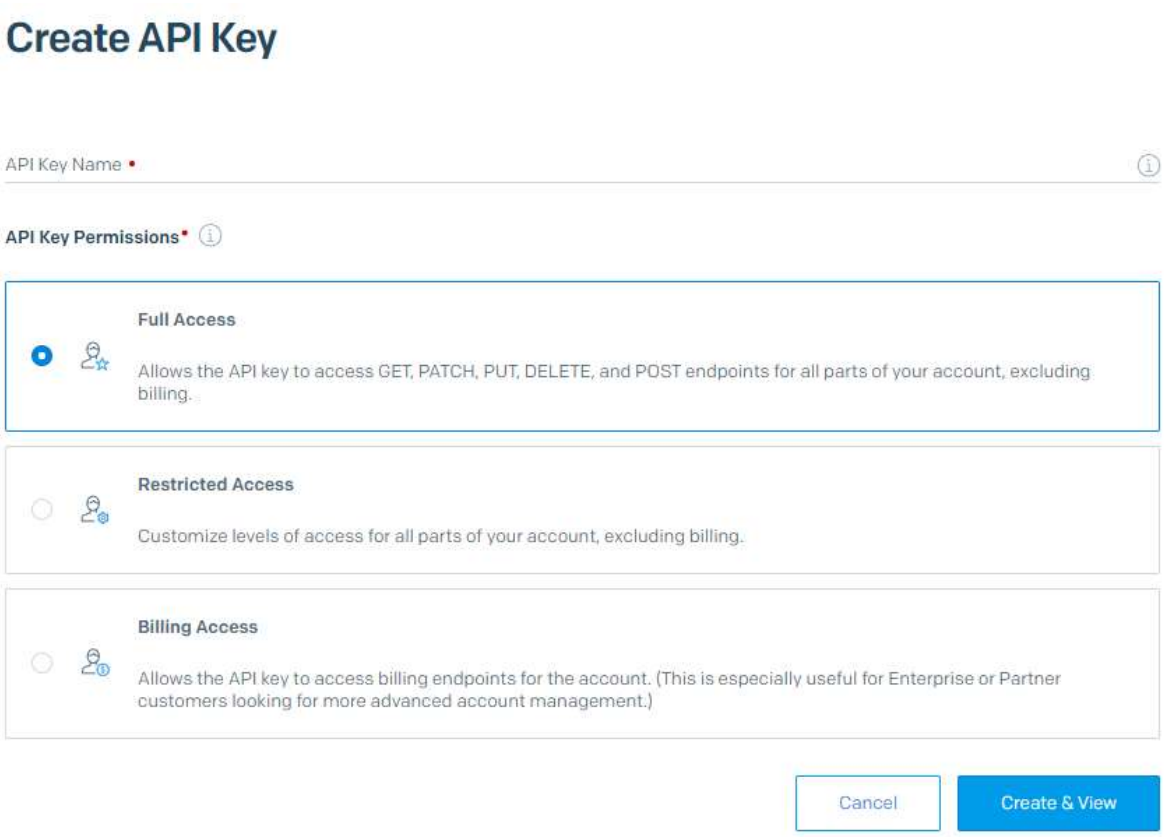


Figure 6-8: SendGrid API Key creation

6.2 User guidelines

6.2.1 Google sign-in

1. On any page, user clicks on gear icon at the topmost right-hand corner on the navigation bar

Figure 6-9: Navigation bar

2. System displays Google sign in dialog.

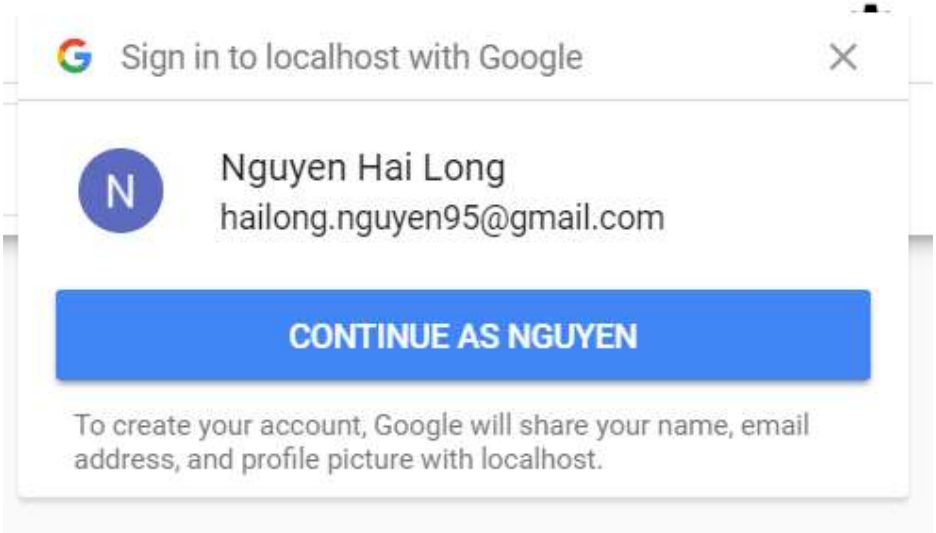


Figure 6-10: Google sign-in form

3. User sign in with his/her Gmail email account.

6.2.2 User sign-out

1. On any page, user clicks on gear icon at the topmost right-hand corner on the navigation bar

Figure 6-11: Navigation bar

2. User clicks Logout in dropdown menu.

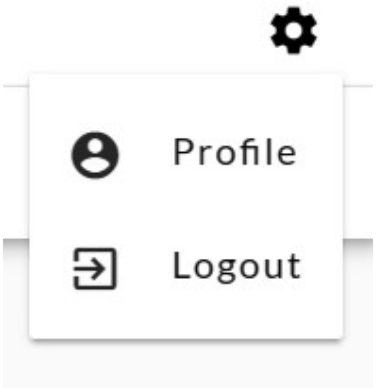


Figure 6-12: Navigation bar dropdown menu

3. User is now logged out of the system and redirected to the homepage.

6.2.3 User searches for jobs and companies

1. On any page, user enters the search keyword(s) in the search bar below the navigation bar and press Enter or click on the “Search” button.

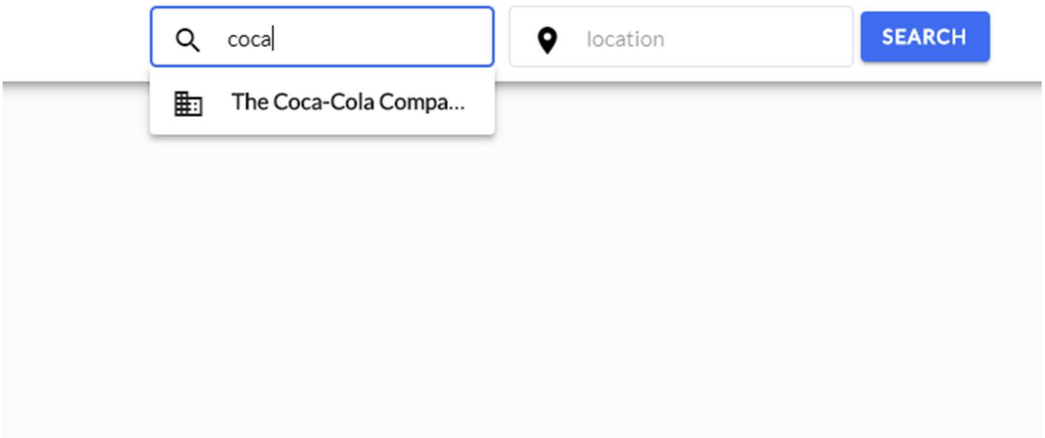


Figure 6-13: Search bar

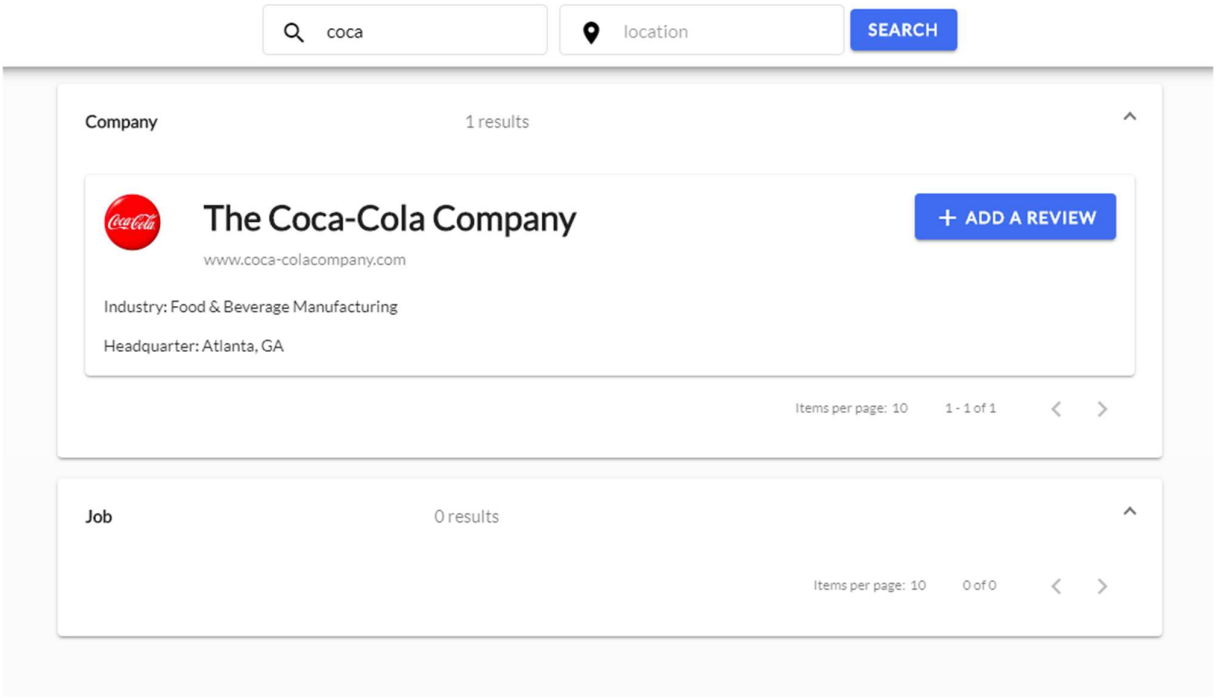


Figure 6-14: Search results

6.2.4 User views his/her profile

1. On any page, user clicks on gear icon at the topmost right-hand corner on the navigation bar.

Figure 6-15: Navigation bar

2. User clicks Profile in the dropdown menu.

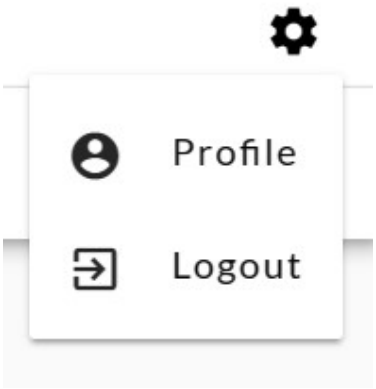


Figure 6-16: Navigation bar dropdown menu

3. User is redirected to User Profile page.

6.2.5 User edits his/her profile

1. User views his/her profile.

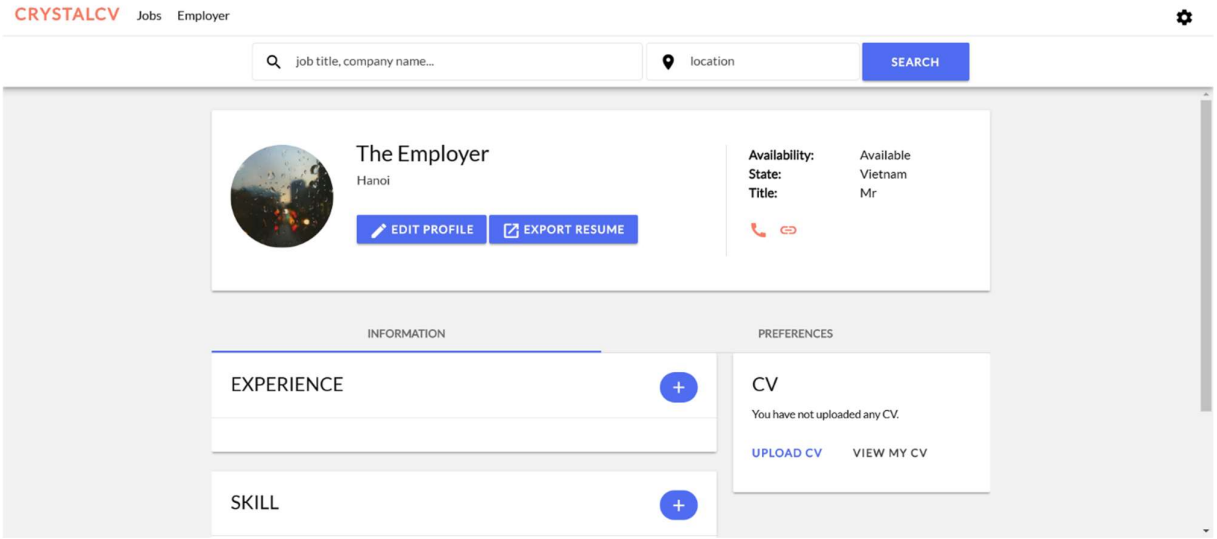


Figure 6-17: User profile

2. User clicks on “Edit profile”
3. Edit related fields
4. Clicks on “Update”

6.2.6 User adds work experience to User profile

1. User views his/her profile.

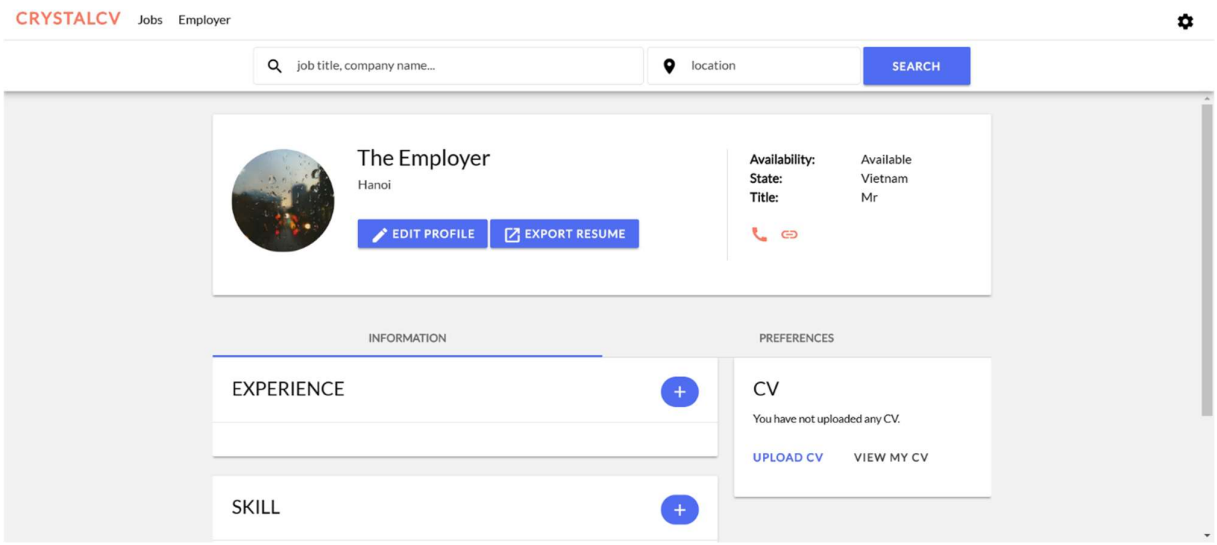


Figure 6-18: User profile

2. User clicks on "+" icon next to "Experience".
3. User edits the respective text boxes.

Figure 6-19: Add work experience

4. User clicks on "Add"

6.2.7 User adds skills to User profile

1. User views his/her profile.

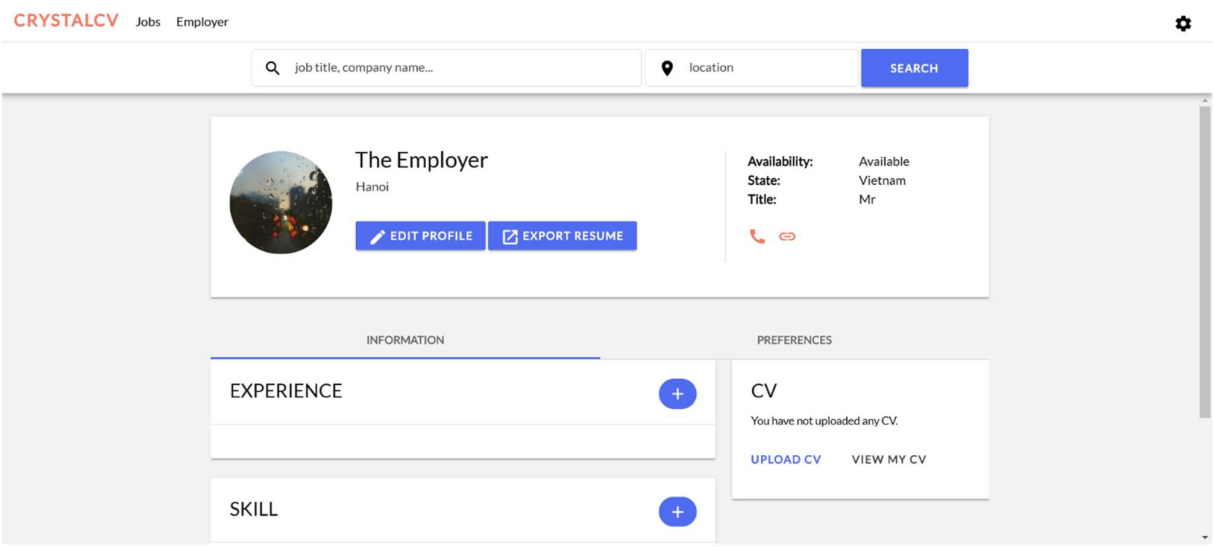


Figure 6-20: User profile

2. User clicks on "+" icon next to "Skill"
3. User edits the respective text fields

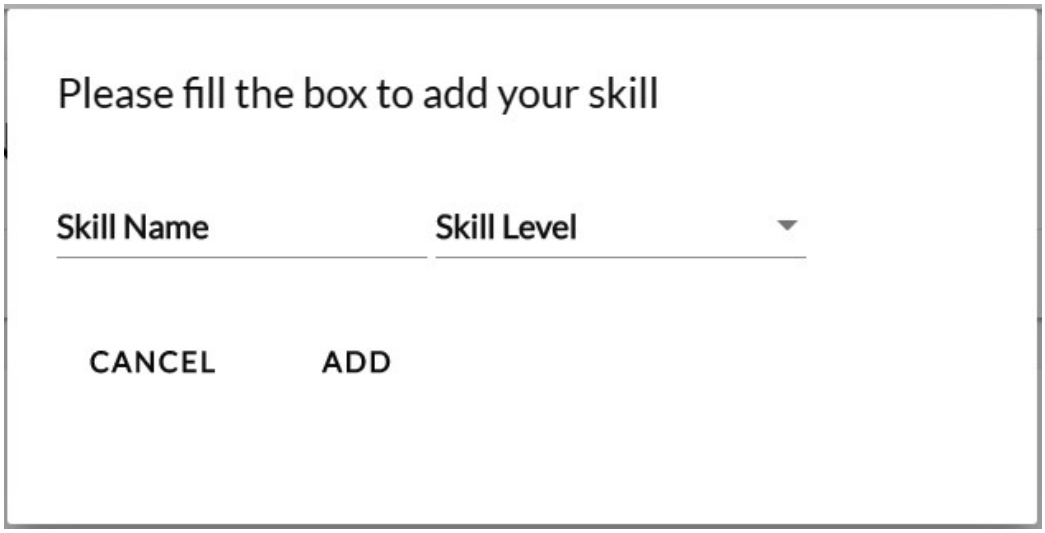


Figure 6-21: Add skill form

4. User clicks on "Add"

6.2.8 User adds education to User profile

1. User views his/her User profile.

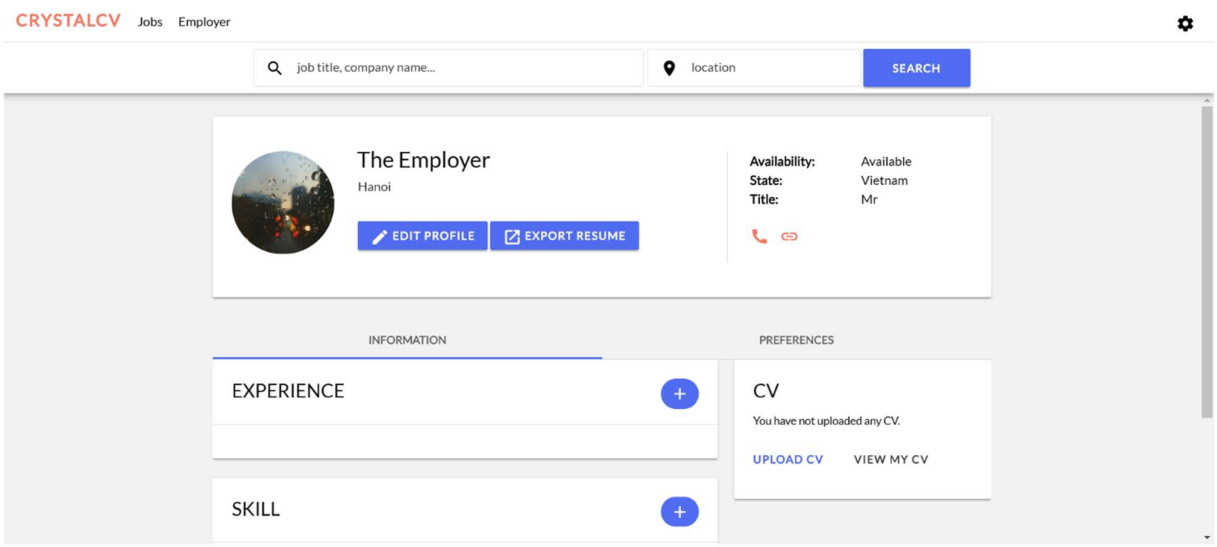


Figure 6-22: User profile

2. User clicks on "+" button next to "Education".
3. User fills the respective text fields.

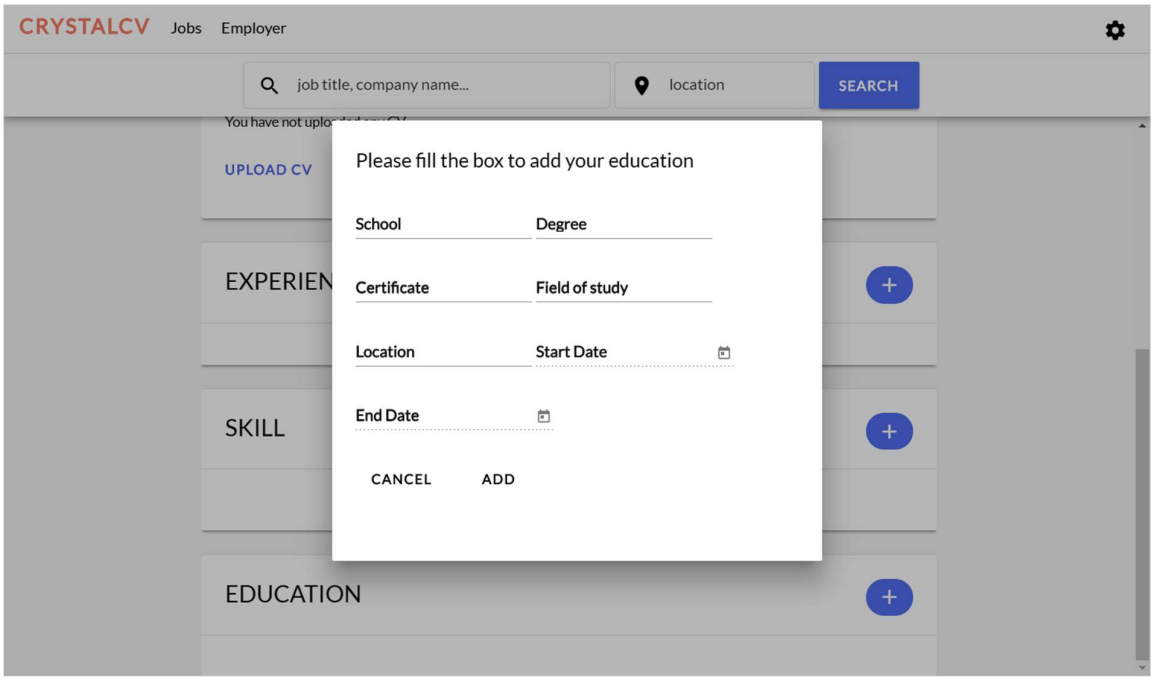


Figure 6-23: Add education form

4. User clicks on "Add"

6.2.9 User edits preferred company

1. User views his/her User profile.
2. User clicks on "Preferences" tab

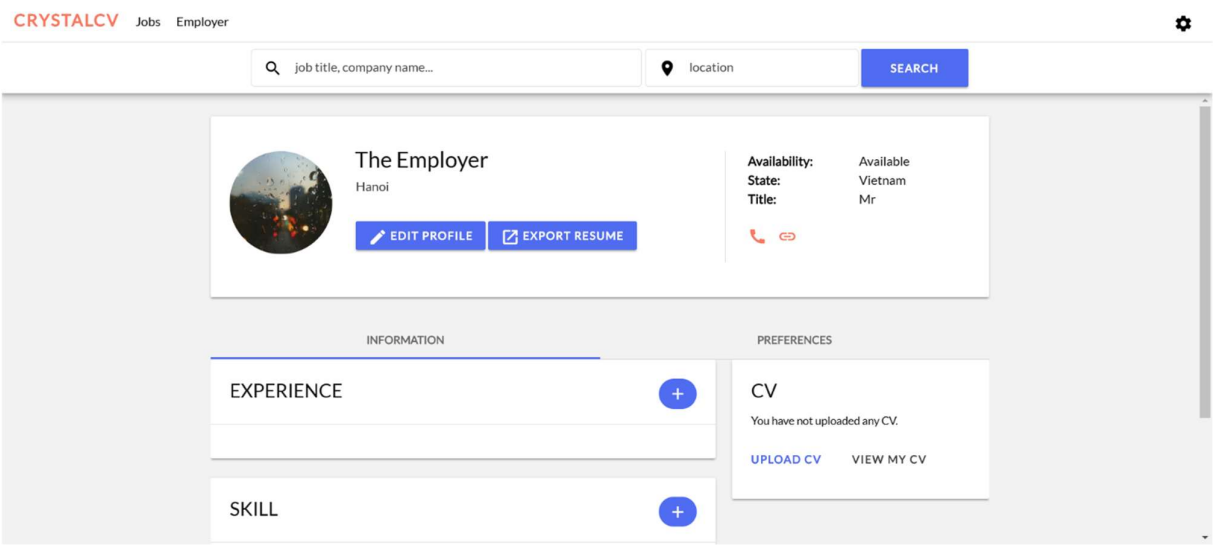


Figure 6-24: User profile

3. User edits the necessary fields then click “Save”

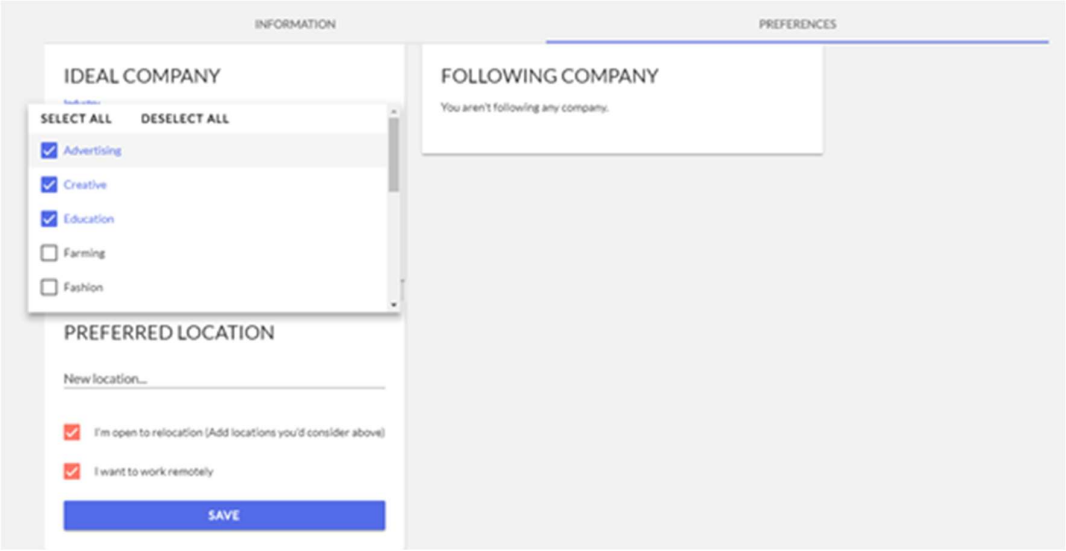


Figure 6-25: Edit preferred company

6.2.10 User edits preferred location

1. User views his/her User profile.
2. User clicks on “Preferences” tab

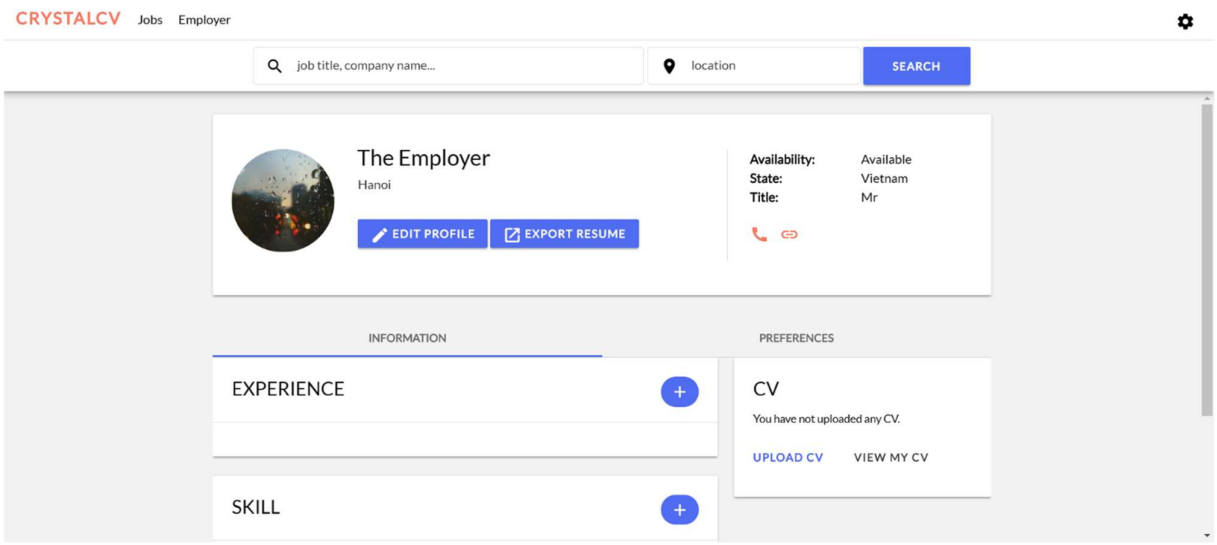


Figure 6-26: User profile

3. User edits the necessary fields then click “Save”

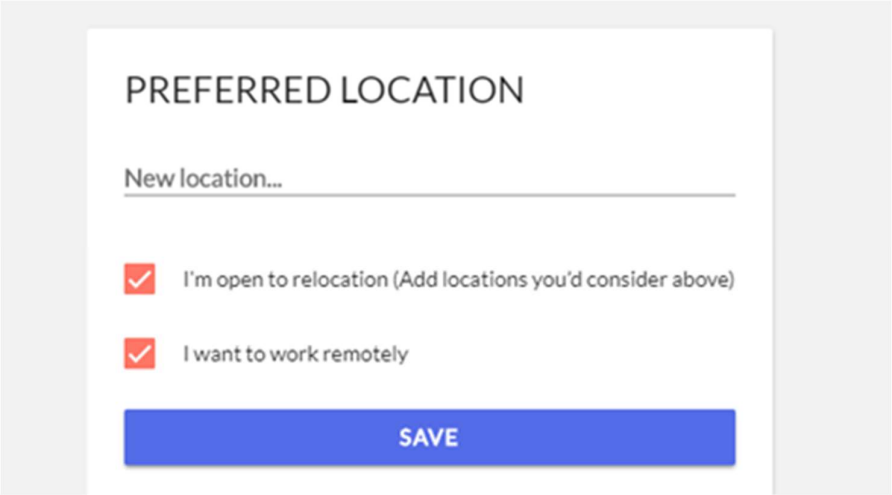


Figure 6-27: Edit preferred location

6.2.11 User uploads CV

1. User views his/her User profile.
2. User clicks on “Upload CV” and choose his/her CV in pdf format.

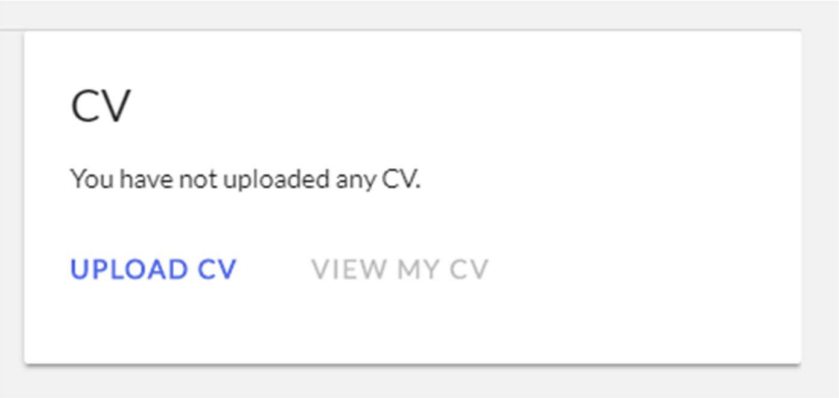


Figure 6-28: Upload CV

6.2.12 Employer user adds new job post

1. Employer user views his/her employer profile by clicking on "Employer" on the navigation bar.

Figure 6-29: Navigation bar

2. User is redirected to Employer profile page

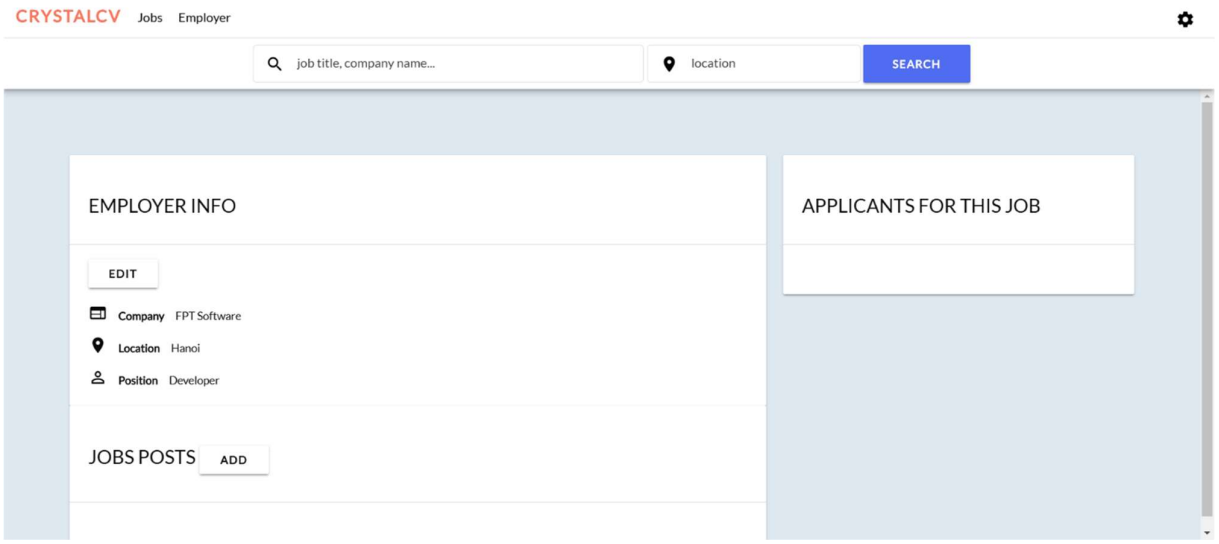


Figure 6-30: Employer profile

3. User clicks on "Add".

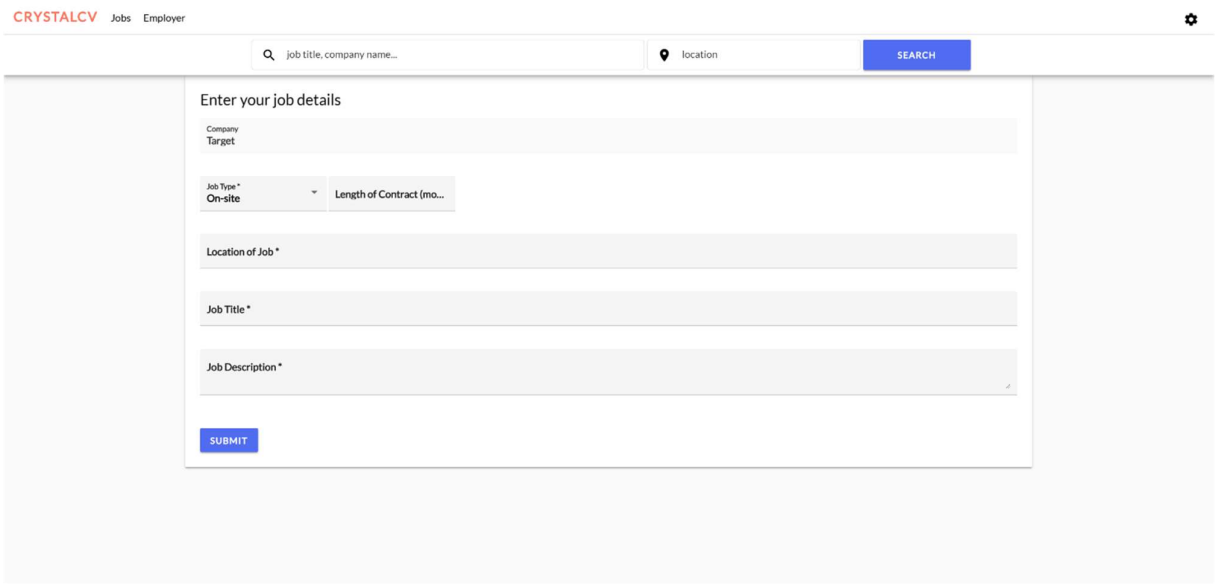


Figure 6-31: Add job post form

4. User fills out the form and click “Submit”

APPENDIX 1:

38% sinh viên Việt Nam tìm kiếm công việc qua Facebook

Tại Việt Nam, sinh viên thường làm gì để trang trải nhu cầu cuộc sống trong khi một số họ đến các thành phố lớn để học tập. Chuyên trang nghiên cứu thị trường Q&Me đã tiến hành một khảo sát về công việc làm thêm của sinh viên.

Figure 7-1: Students finding jobs via Facebook[16]

[16] https://qandme.net/vi/baibaocao/38-phan-tram-sinh-vien-Viet-Nam-tim-cong-viec-qua-Facebook.html