



TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

Final Report
On
JOB PORTAL SYSTEM (HIRE-NEPAL)

Submitted By:

Suraj Pathak (076BCT090)
Tikaharu Sharma (076BCT094)
Sayoush Subedi (076BCT075)

Submitted To:

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
PULCHOWK CAMPUS

MARCH 23, 2023

ACKNOWLEDGEMENTS

We are immensely grateful to the Department of Electronics and Computer Engineering, Institute of Engineering, Pulchowk Campus, for their generous support in providing us with the necessary resources and facilities to carry out this project. We extend our sincere thanks to Asst. Prof. Lok Nath Regmi and Asst. Prof. Nischal Acharya for their encouragement and guidance. We would also like to acknowledge the invaluable support of our supervisor, Assistant Prof. Dr. Aman Shakya, and our mentors, Mr. Manish Modi and Mr. Amit from Khalti.

We extend our sincere gratitude to our esteemed advisors, Associate Prof. Dr. Jyoti Tandukar, Asst. Prof. Santosh Giri, and Asst. Prof. Dr. Basanta Joshi, for their invaluable guidance and support throughout the project. Their expertise and wisdom have been instrumental in the success of this endeavor. We would also like to express our heartfelt appreciation to all the teachers, staff, and seniors who have offered their unwavering assistance and support throughout the project.

We would also like to express our deep appreciation to our friends and family members, who have offered us their unwavering support and encouragement throughout the project. We are also grateful to the seniors who have offered us their invaluable guidance and support.

Once again, we extend our heartfelt gratitude to everyone who has contributed to the success of this project.

Authors:

Suraj Pathak

Tikaharu Sharma

Sayoush Subedi

ABSTRACT

The "Hire-Nepal" job platform is a comprehensive solution designed to connect job seekers with potential employers in Nepal. The platform utilizes technologies such as Django and React to deliver a seamless user experience. Additionally, the platform incorporates a recommendation system built using Tfidfvectorizer and cosine similarity, which provides personalized job recommendations to job seekers based on their interests and skills. With features such as resume and cover letter generation, as well as the ability to explore job openings from other websites, the platform offers a convenient and efficient job search experience. The ultimate goal of the project is to contribute to the reduction of unemployment in Nepal by connecting job seekers with job openings that align with their interests and skills.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	1
ABSTRACT	2
LIST OF FIGURES	6
LIST OF ABBREVIATIONS	7
1. INTRODUCTION	1
1.1 Background	1
1.2 Problem Statement	1
1.3 Objective	1
1.4 Scope	2
2. LITERATURE REVIEW	3
3. THEORY	4
3.1 TF-IDF Vectorization	4
A. Term Frequency	4
B. Inverse Document Frequency	4
C. Vectorization	5
3.2 Cosine Similarity	6
3.3 REST API	6
3.4 ORM (Object Relational Mapper)	7
3.5 MVT (Model-View Template)	8
3.6 Component-based architecture	9
3.7 CSRF (Cross-Site Request Forgery)	9
3.8 JWT (JSON Web Token)	10
4. SOFTWARE AND LIBRARY REQUIREMENTS	11
4.1 Django REST framework	11
4.2 React	12
4.3 Scikit-learn (TFID-Vectorizer & Cosine Similarity)	12
4.4 PostgreSQL	12
4.5 Git and Github	12
5. METHODOLOGIES	13
5.1 Requirement Gathering and Analysis	13
5.1.1 Functional Requirements	13
R.1 View and Apply for Jobs	13
R.2 Create and Post Resumes	14
R.3 Generate Cover Letters	14
R.4 Create Job Postings	14
R.5 Receive Job Recommendations	15
R.6 Scrape Other Job Websites	15
5.1.2 Non-Functional Requirements	16
R.1 Performance	16
R.2 Security	16

R.3 Scalability	16
R.4 Usability	17
R.5 Compatibility	17
R.6 Responsive Design	17
5.2 System Design	18
5.2.1 Use Case Diagram	18
A. Use Case: Authentication (Login and Register)	19
B. Use Case: View Profile	20
C. Use Case: View Jobs	21
D. Use Case: Post Jobs	22
E. Use Case: Apply for Jobs	23
F. Use Case: Resume Generation	24
G. Use Case: Recommendation	25
5.2.2 Activity Diagram	26
A. Apply Jobs	26
B. Upload Jobs	27
C. Search Jobs	28
D. Resume Generation	29
5.2.3 Sequence Diagram	30
A. Authentication	30
B. View Profile	31
C. View Job	32
D. Post Job	33
E. Generate Resume	34
F. Apply for Job	35
G. Recommendation	36
5.2.4 Class Diagram	37
A. Resume	38
B. User-System	39
C. Hiring	40
5.3 Database Design and Implementation	41
A. Models Definition	41
2) Fields Definition	42
3) Relationship Definition	45
4) Migrations Creations	46
5) Data Migrations	46
5.4 Backend Development	46
A. Design	47
B. Implementation	47
C. Testing	47
D. Integration	48
5.5 Frontend Development	48
A. Design	48
B. Implementation	48

C. Testing	49
5.6 Recommendation System Development	49
A. Data collection and Cleaning	49
B. Vectorizing the data	49
C. Computing the cosine similarity	49
D. Combining similarity scores	50
E. Finding top N recommendations	50
F. Adding similarity scores	50
G. Returning results	50
5.7 Integration and Testing	50
6. RESULT AND ANALYSIS	51
7. EPILOGUE	52
7.1 Project Schedule	52
7.2 Conclusion	52
7.3 Limitation	53
7.2 Future Enhancement	53
REFERENCES	54

LIST OF FIGURES

Figure 3.1: REST API	7
Figure 5.1: System Use Case Diagram	18
Figure 5.2: Use Case Diagram for ‘Authentication’	19
Figure 5.3: Use Case Diagram for ‘View Profile’	20
Figure 5.4: Use Case Diagram for ‘View Jobs’	21
Figure 5.5: Use Case Diagram for ‘Post Jobs’	22
Figure 5.6: Use Case Diagram for ‘Apply for Jobs’	23
Figure 5.7: Use Case Diagram for ‘Resume Generation’	24
Figure 5.8: Use Case Diagram for ‘Recommendation’	25
Figure 5.9: Activity Diagram for ‘Apply Jobs’	26
Figure 5.10: Activity Diagram for ‘Upload Jobs’	27
Figure 5.11: Activity Diagram for ‘Search Jobs’	28
Figure 5.12: Activity Diagram for ‘Resume Generation’	29
Figure 5.13: Sequence Diagram for ‘Authentication’	30
Figure 5.14: Sequence Diagram for ‘View Profile’	31
Figure 5.13: Sequence Diagram for ‘View Job’	32
Figure 5.14: Sequence Diagram for ‘Post Jobs’	33
Figure 5.15: Sequence Diagram for ‘Generate Resume’	34
Figure 5.16: Sequence Diagram for ‘Apply for Job’	35
Figure 5.17: Sequence Diagram for ‘Recommendation’	36
Figure 5.18: System Class Diagram	37
Figure 5.19: Class Diagram for ‘Resume’	38
Figure 5.20: Class Diagram for ‘User-System’	39
Figure 5.21: Class Diagram for ‘Hiring’	40

LIST OF ABBREVIATIONS

AI	: Artificial Intelligence
API	: Application Programming Interface
IT	: Information Technology
JSON	: Java Script Object Notation
XML	: Extensible Markup Language
REST	: Representational State Transfer
HTTP	: Hypertext Transfer Protocol
SQL	: Standard Query Language
ACID	: Atomicity Consistency Isolation Durability
UML	: Unified Modeling Language
ORM	: Object-Relational Mapping
MVT	: Model View Template
JWT	: JSON Web Tokens
URL	: Uniform Resource Locator
TF-IDF	: Term Frequency–Inverse Document Frequency
CSS	: Cascading Style Sheets

1. INTRODUCTION

1.1 Background

There is a high demand for convenient and effective job search platforms that can connect job seekers with relevant job opportunities. However, many existing job portals do not offer the level of customization and personalization that is needed to help users find jobs that truly match their skills and experience. The proposed job portal aims to fill this gap by providing a range of features that will enable users to find and apply for job openings that are tailored to their needs.

1.2 Problem Statement

Nepal has a high unemployment rate, especially among the youths :

- Employers face challenges finding suitable candidates for their job openings.
- Job seekers also face challenges finding jobs that match their skills and interests.
- This results in a mismatch between the demand and supply of jobs.
- This problem creates economic and social challenges for Nepali society.
- Our project aims to address this problem by creating a platform.
- The platform will connect job seekers with suitable job opportunities.
- It will also help employers find qualified candidates for their job openings

1.3 Objective

The main objectives of the proposed job portal are as follows:

- To provide a platform for job seekers to browse and apply for job openings that match their skills and experience.
- To allow users to create and upload their resumes, as well as generate customized cover letters.
- To provide personalized job recommendations to users based on their resume and job preferences.
- To enable employers to post job openings and attract qualified candidates.
- To analyze profile and suggest job opportunities based on the content.

1.4 Scope

The scope of the job portal development project includes the following:

- Design and development of the job portal website and its associated features and tools.
- Implementation of a secure login and registration system for both job seekers and employers.
- Integration of the AI tools for resume analysis.
- Integration of a system for scraping other job websites for relevant openings.

The following items are outside the scope of the project:

- Marketing and promotion of the job portal after its launch.
- Ongoing maintenance and updates to the website.

2. LITERATURE REVIEW

A literature review of job portals for IT jobs in Nepal was conducted to gather information on the current state of the market. Several job portals such as Merojob, JobsNepal, and Hamrodevjobs were found to be popular among job seekers in Nepal. In Nepal, online job portals have become increasingly popular in recent years as a way for job seekers to find employment and for employers to find qualified candidates. Some of the most popular job portals in Nepal include:

- **Jobs Dynamics:** A job portal that offers career counseling, resume writing, and interview preparation services for job seekers. It also provides recruitment solutions and employer branding for employers.
- **JobAxe:** A career building and recruitment platform that connects right jobs and right people together. It uses artificial intelligence and machine learning to match candidates with suitable jobs.
- **RamroJob:** A job portal that is well maintained and updated daily. It has a simple and user-friendly interface that allows users to browse jobs by category, location, or keyword.
- **KantipurJob:** A job portal that is affiliated with Kantipur Media Group, one of the largest media houses in Nepal. It has a wide range of job opportunities from various sectors and levels.
- **MeroJob:** A job portal that claims to be the most innovative human resource consulting firm in Nepal. It offers personalized services such as career counseling, training programs, psychometric tests, and video resumes for job seekers.
- **JobsNepal.com:** An online job search and career development platform that provides resume creating and matching for online job posts.
- **Jobee.com:** A fast-growing job portal that allows users to search among thousands of jobs from top companies, industries, and locations of their choice.
- **Merorogari.com:** A free service that posts promising job vacancies for various sectors and levels.

3. THEORY

3.1 TF-IDF Vectorization

TF-IDF (Term Frequency-Inverse Document Frequency) is a statistical measure used to evaluate the importance of words in a document. TF-IDF vectorization is a technique used in Natural Language Processing (NLP) to convert text data into numerical vectors that can be used in machine learning algorithms.

A. Term Frequency

Term Frequency (TF) is a statistical measure used in Natural Language Processing (NLP) to evaluate the importance of a word in a document. It represents the number of times a particular word appears in a document, divided by the total number of words in the document.

TF is often used as a basic building block for more advanced NLP techniques, such as TF-IDF vectorization and sentiment analysis. It helps to identify which words are most relevant to the meaning of a document, and can be used to generate numerical vectors that can be processed by machine learning algorithms.

The term frequency can be calculated using the following formula:

$$TF = (\text{Number of times a word appears in a document}) / (\text{Total number of words in the document})$$

B. Inverse Document Frequency

Inverse Document Frequency (IDF) is another statistical measure used in Natural Language Processing (NLP) to evaluate the importance of a word in a collection of documents. Unlike Term Frequency (TF), IDF considers not only the frequency of a word in a single document, but also its frequency across all documents in the collection.

The idea behind IDF is that words that appear frequently across many documents are less important for discriminating between them, and therefore less informative than words that appear in only a few documents. IDF is calculated as the logarithm of the total number of documents in the collection divided by the

number of documents containing the word, and then taking the inverse of this value.

The inverse document frequency can be calculated using the following formula:

$$\text{IDF} = \log (\text{Total number of documents in the collection}) / (\text{Number of documents containing the word})$$

The IDF value is high for words that appear in very few documents and low for words that appear in many documents. By combining the TF and IDF values, we can calculate the TF-IDF score for each word, which gives a measure of its relevance to a particular document in the collection. TF-IDF is commonly used for information retrieval, text classification, and sentiment analysis in NLP.

C. Vectorization

Vectorization in TF-IDF refers to the process of converting a text document into a numerical vector representation that can be used for machine learning algorithms. This process involves calculating the TF-IDF score for each word in the document, and representing each word as a vector in a high-dimensional space.

The TF-IDF score for each word is calculated by multiplying its Term Frequency (TF) by its Inverse Document Frequency (IDF). The resulting TF-IDF scores are then normalized to account for differences in the length of documents and the overall frequency of words in the corpus.

Once the TF-IDF scores have been calculated for all the words in a document, they can be combined into a single vector representation using a mathematical formula such as the L2 norm. The resulting vector represents the document in a high-dimensional space, with each dimension corresponding to a different word in the vocabulary.

Vectorization in TF-IDF is a powerful technique for text analysis and information retrieval, as it allows textual data to be processed using machine learning

algorithms that are designed to work with numerical data. It can be used for a variety of NLP tasks, such as text classification, clustering, and recommendation systems.

3.2 Cosine Similarity

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. In the context of Natural Language Processing (NLP), cosine similarity is often used to determine the similarity between two text documents based on their vector representations.

To calculate cosine similarity between two vectors, we first compute the dot product of the two vectors, which is the sum of the element-wise multiplication of the corresponding elements in the two vectors. Then, we divide the dot product by the product of the L2 norms of the two vectors. The resulting value is the cosine similarity between the two vectors, which ranges from -1 to 1.

A cosine similarity of 1 indicates that the two vectors are identical, while a cosine similarity of -1 indicates that they are completely dissimilar. A cosine similarity of 0 indicates that the two vectors are orthogonal, or have no similarity.

In NLP, cosine similarity is often used in conjunction with TF-IDF vectorization to compare the similarity of documents, sentences, or words based on their vector representations. It is widely used in applications such as document similarity analysis, text clustering, and recommendation systems.

3.3 REST API

REST API refers to the representational state transfer application programming interface. It communicates with a standard database for creating, updating, reading and deleting the data using the standard HTTP protocol. It used the standard HTTP requests like (GET, POST, PUT, DELETE) for the communication with the database server. The state of a resource at any particular instant, or timestamp, is known as the resource representation. This information can be delivered to a client in virtually any format

including JavaScript Object Notation (JSON), HTML, XLT, Python, PHP, or plain text. JSON is popular because it's readable by both humans and machines.

A request is sent from client to server in the form of a web request such as HTTP GET, POST, PUT, DELETE) Then the server responds with the resource in the form of JSON, HTML, XML or just plain text.

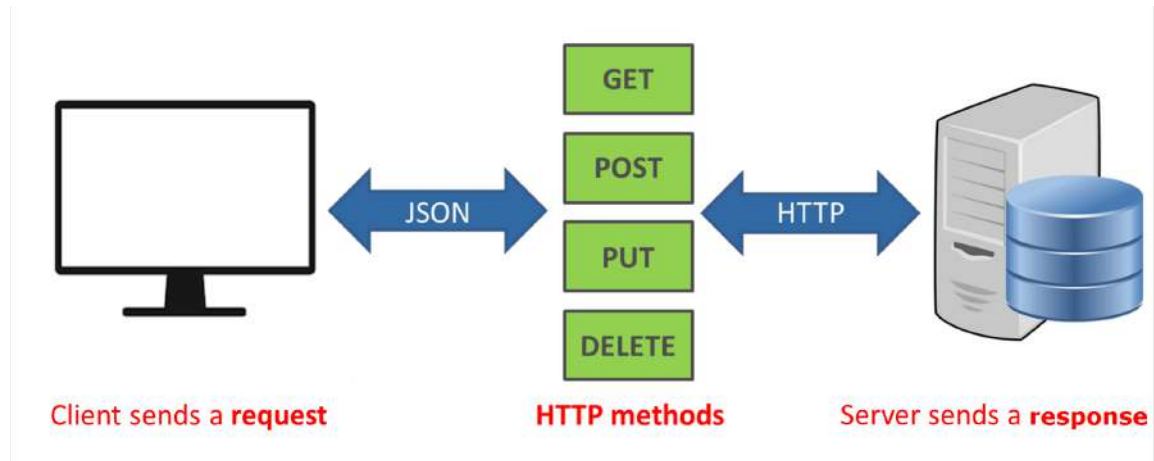


Figure 3.1: REST API

3.4 ORM (Object Relational Mapper)

ORM, or Object-Relational Mapping, is a programming technique that allows developers to map object-oriented models to relational databases. In simpler terms, ORM is a way to convert data between two incompatible type systems: an object-oriented programming language and a relational database management system (RDBMS).

The basic idea behind ORM is to represent database tables as objects, and to create mappings between them. In other words, each table in a database is represented by a corresponding class in the application, and each row in the table is represented by an instance of that class. The properties of the class correspond to the columns in the table, and the methods of the class correspond to the operations that can be performed on the data.

ORM frameworks typically provide an API (Application Programming Interface) that allows developers to create and manipulate database objects using the familiar syntax of their programming language. For example, instead of writing SQL statements to interact with the database, developers can write code that uses ORM methods to create, read, update, and delete objects.

3.5 MVT (Model-View Template)

MVT stands for Model-View-Template, which is a software architecture pattern that is commonly used in web development frameworks such as Django. MVT separates the application logic into three components:

- **Model:** This component defines the data structure and business logic of the application. It interacts with the database and provides an abstract representation of the data.
- **View:** This component handles the user interface and presentation logic of the application. It receives requests from the user, communicates with the model, and returns responses to the user.
- **Template:** This component defines the layout and style of the web pages. It is a file that contains HTML code and placeholders for dynamic content that is filled by the view.

The main advantage of MVT architecture is that it promotes modularity, reusability, and maintainability of code. It also allows developers to work on different components independently and use different technologies or tools for each component. For example, one can use different databases or languages for the model, different frameworks or libraries for the view, and different templates or themes for the template.

MVT architecture is similar to MVC (Model-View-Controller) architecture, which is another popular pattern for web development. The main difference is that in MVC, the view is responsible for generating the HTML code, while in MVT, the view passes the data to a template engine that generates the HTML code. This makes MVT more flexible and easier to use for web development, as it separates the presentation logic from the business logic more clearly.

3.6 Component-based architecture

The component architecture in React is a fundamental concept that allows developers to create reusable UI elements, or components, which can be combined to form larger, more complex user interfaces. This architecture is based on the idea of "composition" and "separation of concerns", which means that each component is responsible for a single piece of functionality or behavior, and can be reused and combined in different ways to create more complex user interfaces.

In React, a component is a JavaScript function or class that encapsulates a specific piece of functionality or behavior. Components can be either "functional" or "class" components, depending on how they are defined. Functional components are simpler and more lightweight, and are typically used for simpler UI elements, while class components are more complex and provide additional functionality such as state management and lifecycle methods.

React components are typically organized in a hierarchical tree structure, with each component having a parent and one or more child components. This hierarchy is known as the "component tree" or "DOM tree". Each component in the tree can have its own state and properties, which are passed down from the parent component to the child components.

The key advantage of the component architecture in React is that it allows for a high degree of reusability and modularity. Developers can create small, self-contained components that can be easily reused in different parts of the application, and can be combined in different ways to create more complex UI elements. This makes it easier to maintain and update the application, and also makes it easier to test and debug individual components.

3.7 CSRF (Cross-Site Request Forgery)

CSRF stands for Cross-Site Request Forgery, which is a type of web security vulnerability that allows an attacker to trick a user into performing unwanted actions on a website where they are already authenticated. For example, an attacker can send a

malicious link or image to a user that, when clicked, will send a request to transfer money from the user's bank account to the attacker's account.

A CSRF token is a random and unique value that is generated by the server and sent to the client as a hidden form field or a cookie. The client must include the CSRF token in every request that modifies the state of the server, such as POST, PUT, DELETE, etc. The server will then verify that the CSRF token matches the one that was previously sent to the client, and only process the request if they match. This way, the server can prevent CSRF attacks, because an attacker cannot guess or forge the CSRF token.

A CSRF token works as follows:

- When a user visits a website that uses CSRF tokens, the server generates a random and unique value for the CSRF token and sends it to the client along with the web page.
- The client stores the CSRF token in a hidden form field or a cookie, depending on the implementation.
- When the user submits a form or performs any action that modifies the state of the server, the client includes the CSRF token in the request header or body.
- The server checks that the CSRF token in the request matches the one that was previously sent to the client, and only processes the request if they match. If they do not match, the server rejects the request and returns an error message.
- The server generates a new CSRF token for each request and sends it back to the client along with the response.

By using CSRF tokens, a website can protect itself from CSRF attacks, because an attacker cannot forge or guess the CSRF token. Even if an attacker manages to trick a user into clicking on a malicious link or image, the request will not have a valid CSRF token and will be rejected by the server. Therefore, CSRF tokens are an effective way to prevent unauthorized requests from being executed on behalf of a user.

3.8 JWT (JSON Web Token)

JWT stands for JSON Web Token, which is a standard for securely transmitting information between parties as a JSON object. JWTs are commonly used for

authentication and authorization in web applications, as they allow the server to verify the identity and permissions of the client without storing session data or cookies.

A JWT consists of three parts: a header, a payload, and a signature. The header contains metadata about the token, such as the algorithm used to sign it and the type of token. The payload contains the claims or data that the token carries, such as the user ID, the expiration time, and the roles or scopes. The signature is a cryptographic hash of the header and the payload, using a secret key or a public/private key pair. The signature ensures that the token has not been tampered with or forged.

A JWT works as follows:

- When a user logs in to a web application, the server generates a JWT with the user's information and permissions and sends it back to the client.
- The client stores the JWT in a secure place, such as local storage or a cookie, and includes it in every request to the server that requires authentication or authorization.
- The server verifies the JWT by checking its signature and its expiration time, and extracts the user's information and permissions from the payload.
- The server processes the request based on the user's identity and permissions, and returns a response to the client.

By using JWTs, a web application can implement stateless authentication and authorization, which means that the server does not need to store any session data or cookies for each user. This makes the server faster and more scalable.

4. SOFTWARE AND LIBRARY REQUIREMENTS

4.1 Django REST framework

- Django-REST: widely used, mature, battle-tested
- Serialization: easily convert database data to JSON or XML format
- Authentication and permissions: built-in support for user authentication and authorization
- Support for building API-driven applications: easily build RESTful APIs with Django-REST's built-in views and serializers

4.2 React

- React: popular, large community
- Component-based architecture: managing complex UI
- Fast, efficient: real-time updates
- Declarative syntax, Virtual DOM: simplifying debugging and testing

4.3 Scikit-learn (TFIDF-Vectorizer & Cosine Similarity)

- TfidfVectorizer and Cosine Similarity: common techniques for recommendation systems
- TfidfVectorizer: converts text to numerical features for analysis
- Cosine Similarity: measures similarity between two sets of features
- Both techniques are widely used, easy to implement, and can provide effective recommendations.

4.4 PostgreSQL

- Open-source: PostgreSQL is a free, open-source relational database management system that can be used on a variety of platforms.
- ACID compliance: PostgreSQL is known for its adherence to the ACID (Atomicity, Consistency, Isolation, Durability) properties, ensuring the reliability and consistency of data.
- Extensible: PostgreSQL is highly extensible and can be customized using extensions and user-defined functions, making it suitable for a wide range of applications and use cases.
- Community-driven: PostgreSQL has a large and active community of developers and users who contribute to its development and provide support through online forums and user groups.

4.5 Git and Github

We used Git for version control. GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.

5. METHODOLOGIES

5.1 Requirement Gathering and Analysis

Requirement gathering and analysis is the process of identifying, documenting, and understanding the needs and constraints of stakeholders for a software system or project. We identified the following requirements after requirement gathering and analysis.

5.1.1 Functional Requirements

R.1 View and Apply for Jobs

Description: Allows users to view and apply for job listings on the website

R1.1 View job listings

Input: None

Output: A list of all job listings on the website

R1.2 View job listing details

Input: Job listing ID

Output: Detailed information about the selected job listing including job title, location, description, requirements, and application instructions

R1.3 Apply for a job

Input: Job listing ID, resume, cover letter (optional)

Output: User's application is submitted and sent to the employer for review

R1.4 View applications history

Input: User ID

Output: User can view the history of their job applications

R.2 Create and Post Resumes

Description: Allows users to create resumes on the website

R2.1 Create a new resume

Input: Personal information (name, contact details), education, work experience, skills

Output: A new resume is created and can be downloaded

R2.2 Apply with a resume

Input: Job listing ID, Resume ID

Output: User can apply for a job listing using a selected resume

R.3 Generate Cover Letters

Description: Allows users to generate custom cover letters for job applications

R3.1 Create a new cover letter

Input: Job listing title, user's personal information, job listing's requirements

Output: A new cover letter is generated, tailored to the specific job listing and the user's qualification.

R.4 Create Job Postings

Description: Allows employers to create and post job listings on the website

R4.1 Create a new job listing

Input: Job title, job type, location, description, requirements, application instructions, company details

Output: A new job listing is created and published on the website

R4.2 Edit/Delete a job listing

Input: Job listing ID

Output: The selected job listing is edited or deleted

R4.3 View job listings

Input: Employer ID

Output: A list of job listings associated with the employer's account

R.5 Receive Job Recommendations

Description: Allows users to receive personalized job recommendations based on their profile

R5.1 View recommended jobs

Input: User ID

Output: A list of job recommendations for the user, based on their profile.

R.6 Scrape Other Job Websites

Description: Allows the job portal website to scrape job listings from other websites and automatically add them to its own job listing database.

R6.1 Scrape job listings

Input: URLs of other job websites

Output: Job listings from the specified websites are scraped and added to the job portal website's database

5.1.2 Non-Functional Requirements

R.1 Performance

R1.1: Response time for job search and application should be fast.

R1.2: The website should be optimized for speed and performance.

R1.3: The system should provide fast load times for all pages.

R.2 Security

R2.1: User's data should be secured while storage and transmission.

R2.2: The system should provide secure methods for user's login and authentication.

R.3 Scalability

R3.1: The system should be able to grow and still work well as more job listings and users are added.

R3.2: The system should provide an easy and efficient way to add new features and functionalities.

R.4 Usability

R4.1: The website should be easy to navigate and understand for users of all levels.

R4.2: The website should be mobile-friendly and responsive.

R4.3: The system should provide clear and concise instructions for users throughout the process.

R4.4: The system should provide an easy way to update and manage user's profile.

R.5 Compatibility

R5.1: The website should be compatible with all modern web browsers.

R5.2: The website should be compatible with different devices and operating systems.

R.6 Responsive Design

R6.1: The website automatically adjusts its layout and functionality to optimize user experience for different screen sizes and devices.

R6.2: The system should provide a seamless user experience across different devices.

R6.3: The website adjusts visibility of elements according to the available screen space.

R6.4: The website is accessible and usable on all modern web browsers and devices.

A. Use Case: Authentication (Login and Register)

Actors: User, Server

Purpose: To authenticate a user's identity and grant them access to the system.

Precondition: User must have an email address and internet connection.

Postcondition: User will be able to access the protected endpoints of the system.

Description:

- The user will first attempt to log in to the system using their email and password.
- If the user is not registered, they will be prompted to register by providing their email, password, and other required information.
- The server will send a confirmation email after registration is complete.
- The server will verify the user's email and password before allowing access to the system if they are correct.

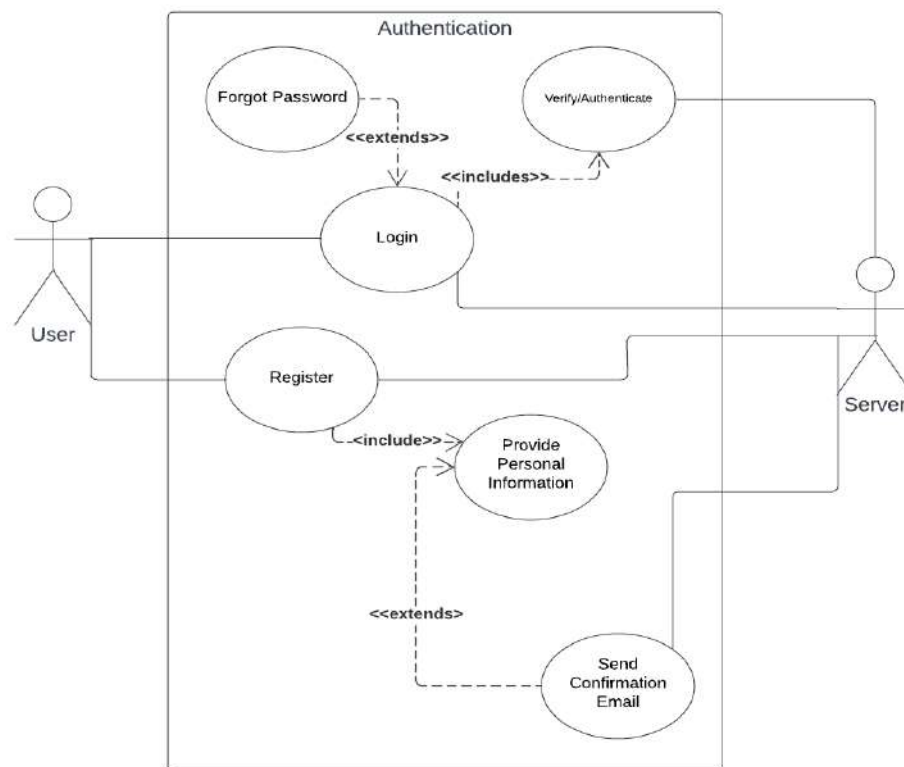


Figure 5.2: Use Case Diagram for 'Authentication'

B. Use Case: View Profile

Actors: Seeker, Server

Purpose: To allow a user to view and edit their profile information.

Precondition: Seeker must be authenticated and have an existing profile

Postcondition: Seeker will have access to their profile information

Description:

- The Seeker will navigate to the profile section of the system.
- The Seeker will have the capability to create a profile on the website.
- The Seeker will then be able to edit their profile information as needed and save the changes.

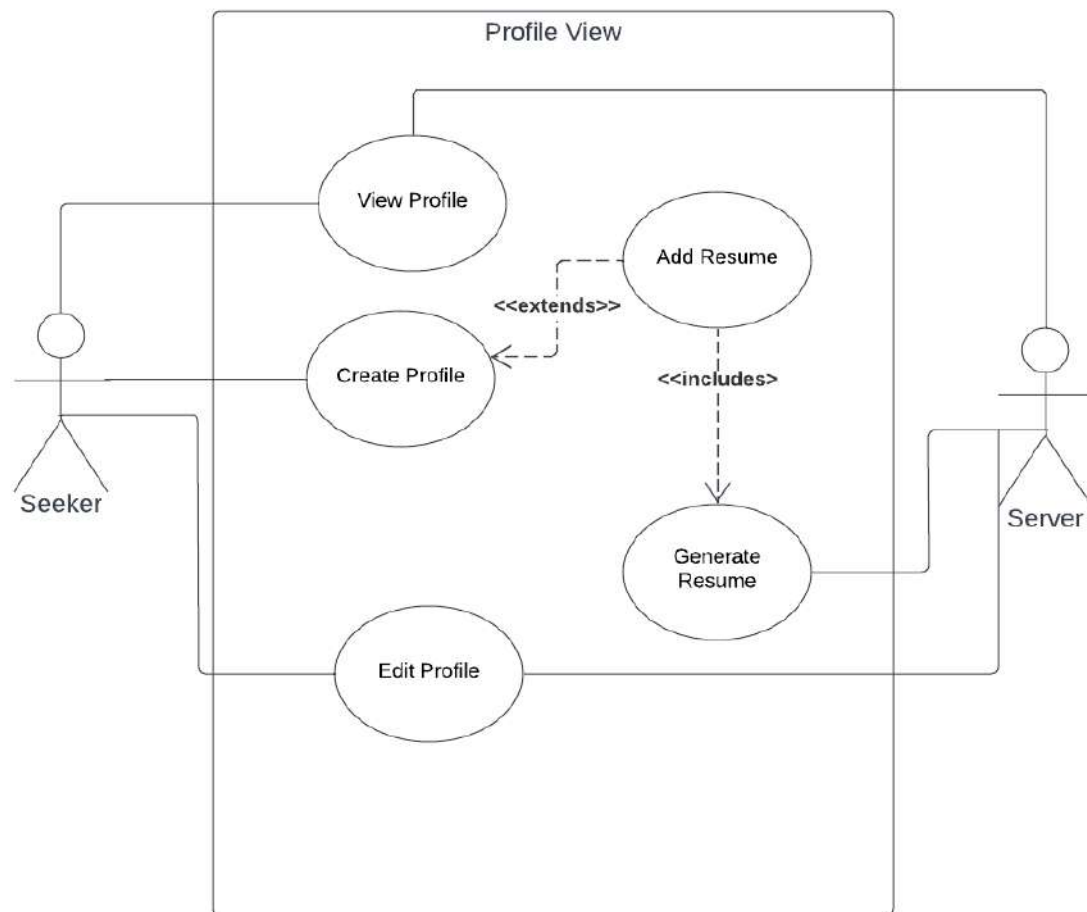


Figure 5.3: Use Case Diagram for 'View Profile'

C. Use Case: View Jobs

Actors: User, Server

Purpose: To allow a user to view available job postings.

Precondition: User must be authenticated

Postcondition: User will have access to a list of available job postings

Description:

- The user will navigate to the jobs section of the system.
- The server will retrieve a list of available job postings from the database and display them to the user.
- The user can search jobs by entering keywords, and filtering results
- The user will be able to view jobs they have previously applied to.
- The system will provide recommendations for jobs to the user, which they will be able to view.

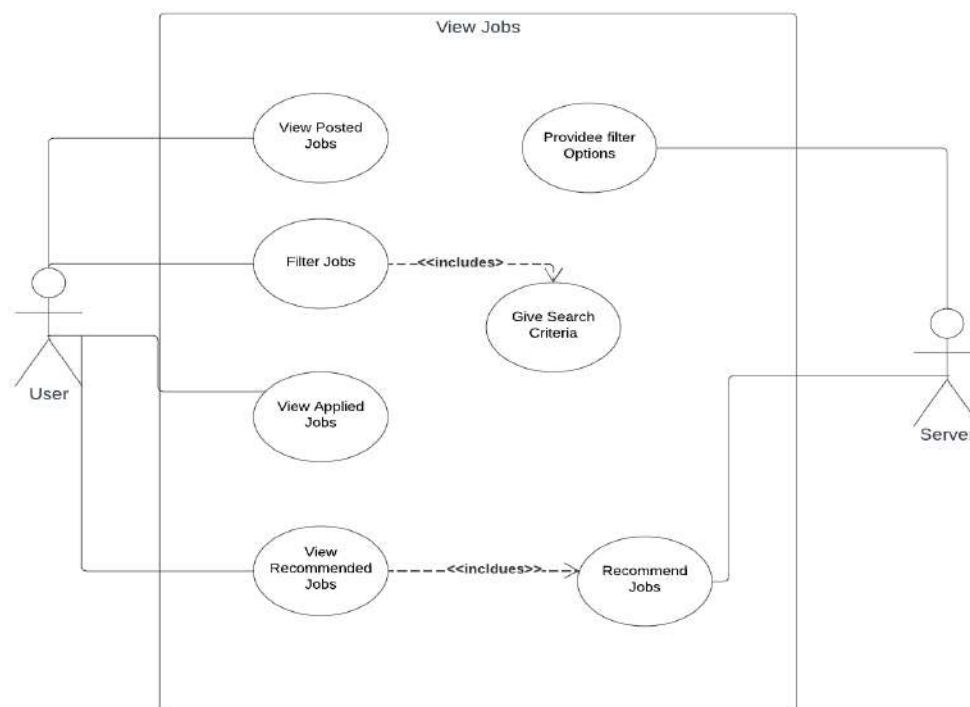


Figure 5.4: Use Case Diagram for 'View Jobs'

D. Use Case: Post Jobs

Actors: Employer, Server, Admin

Purpose: To allow an Employer to post new job listings on the system.

Precondition: Employer must be authenticated and have the necessary permissions to post jobs

Postcondition: New job listing will be added to the system

Description:

- The Employer will navigate to the post a job section of the system.
- The Employer will enter the job details like title,description, salary, location and required skills etc.
- The server will then save the job listing to the database.

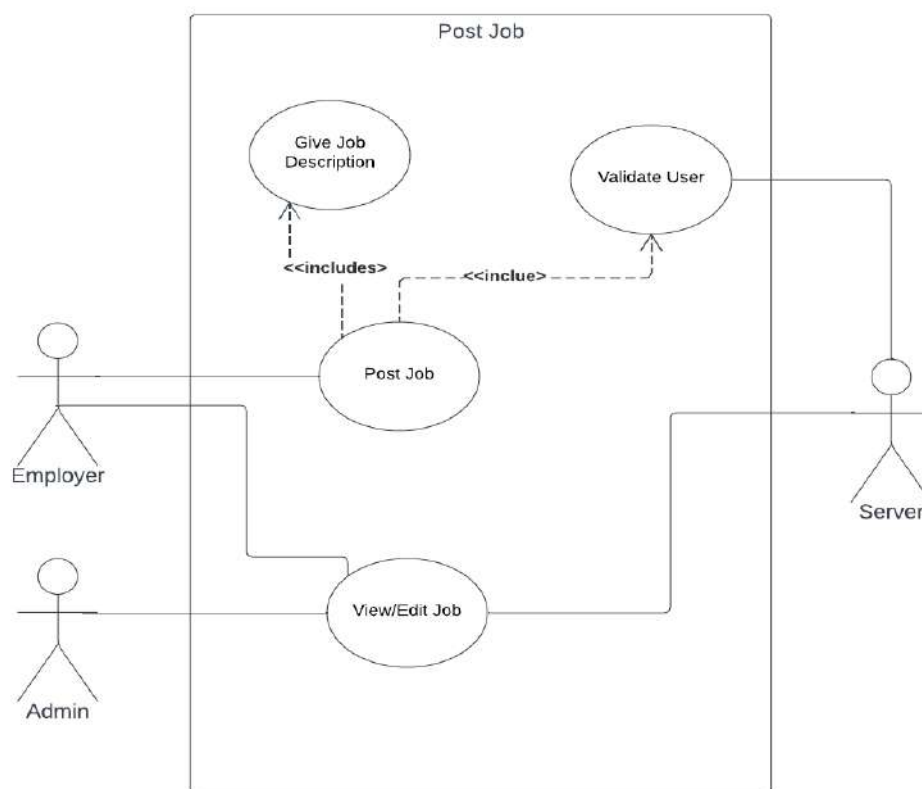


Figure 5.5: Use Case Diagram for 'Post Jobs'

E. Use Case: Apply for Jobs

Actors: Seeker, Server, Admin

Purpose: To allow a Seeker to apply for a job posting on the system.

Precondition: Seeker must be authenticated and job postings should not exceed their deadline.

Postcondition: Seeker's application will be associated with the job posting and saved to the database.

Description:

- The Seeker will navigate to a job listing that they are interested in.
- The server verifies whether or not the job postings have reached their designated expiration date.
- The seeker will be able to upload their resume as part of the application process.
- The seeker will also have the option to upload a cover letter as part of the application.
- The server will then associate the Seeker's application with the job listing and save it to the database.

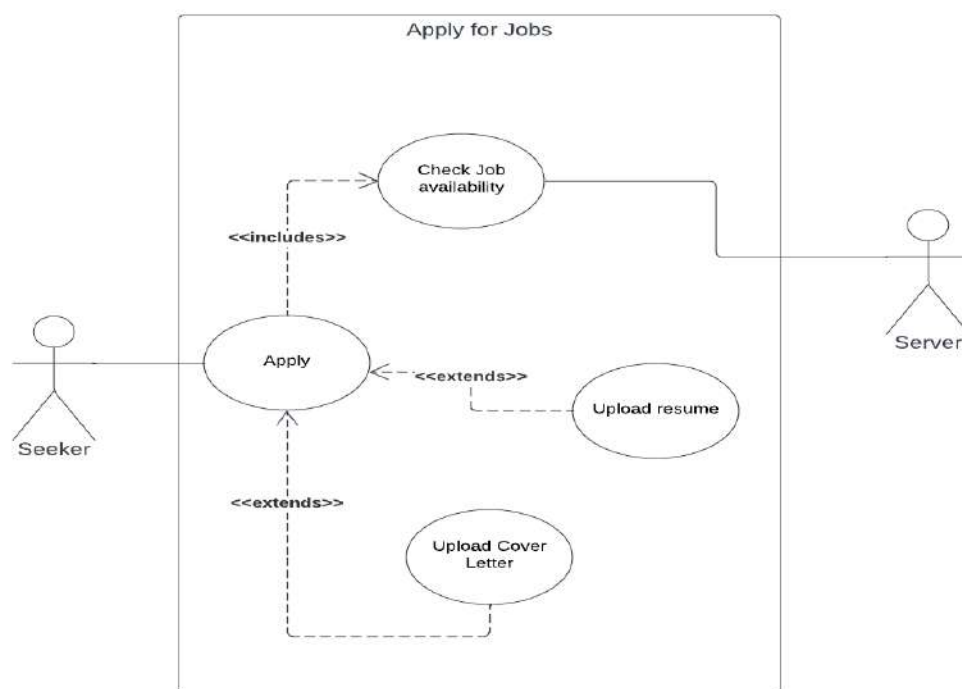


Figure 5.6: Use Case Diagram for 'Apply for Jobs'

F. Use Case: Resume Generation

Actors: Seeker, Server

Purpose: To allow a Seeker to generate a professional resume.

Precondition: Seeker must be authenticated and have filled in their resume information

Postcondition: Seeker will have a generated resume

Description:

- The Seeker will navigate to the resume generation section of the system.
- The server will use this information to generate a professional resume that the Seeker can download or print.

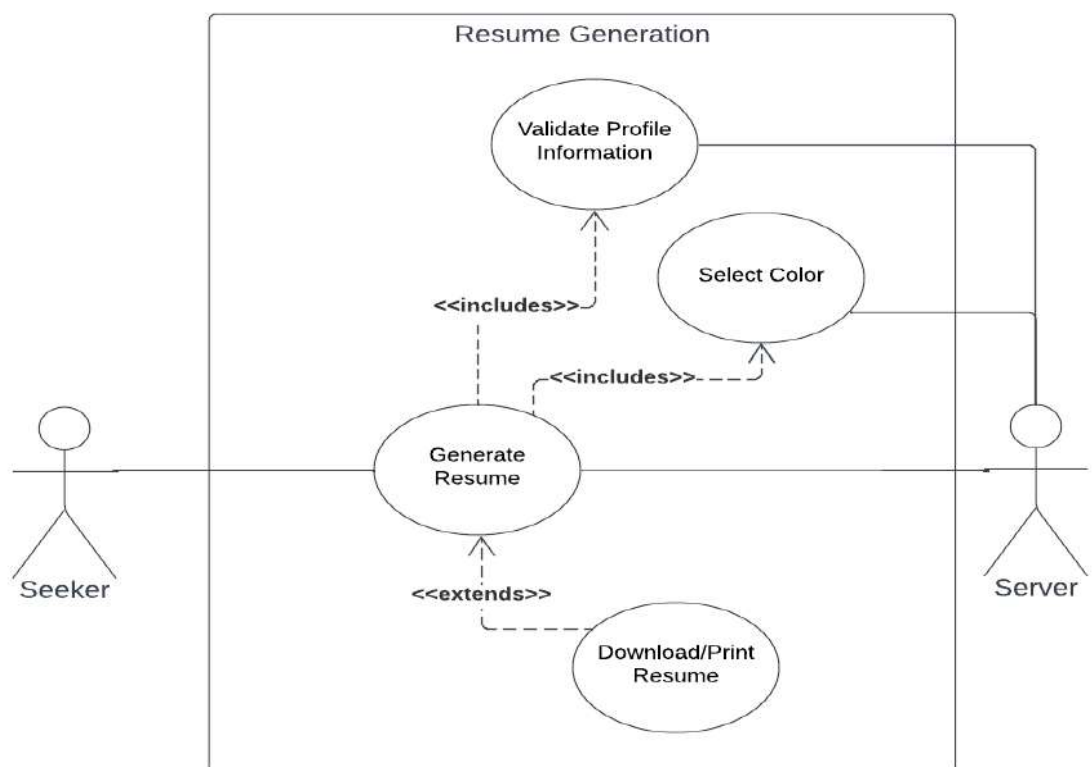


Figure 5.7: Use Case Diagram for 'Resume Generation'

G. Use Case: Recommendation

Actors: Seeker, Employer, Server

Purpose: To allow a Seeker and an Employer to receive recommendations for jobs or applicants.

Precondition: User must be authenticated

Postcondition: Seeker and Employer will receive relevant recommendations

Description:

- The Seeker or Employer will navigate to the recommendation section of the system.
- The server will use the AI tools for resume reading and job search criteria to recommend relevant job postings for the Seeker.
- The server will also use Applicant ranking to filter out the best candidates for the relevant job.

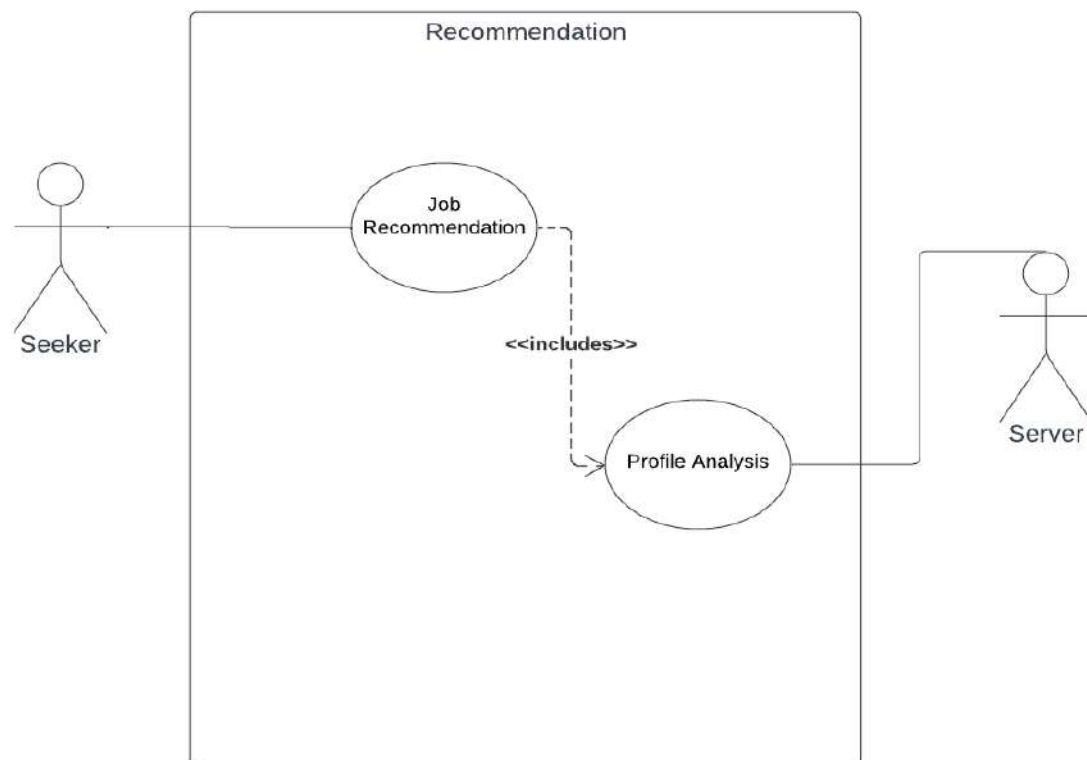


Figure 5.8: Use Case Diagram for 'Recommendation'

5.2.2 Activity Diagram

A. Apply Jobs

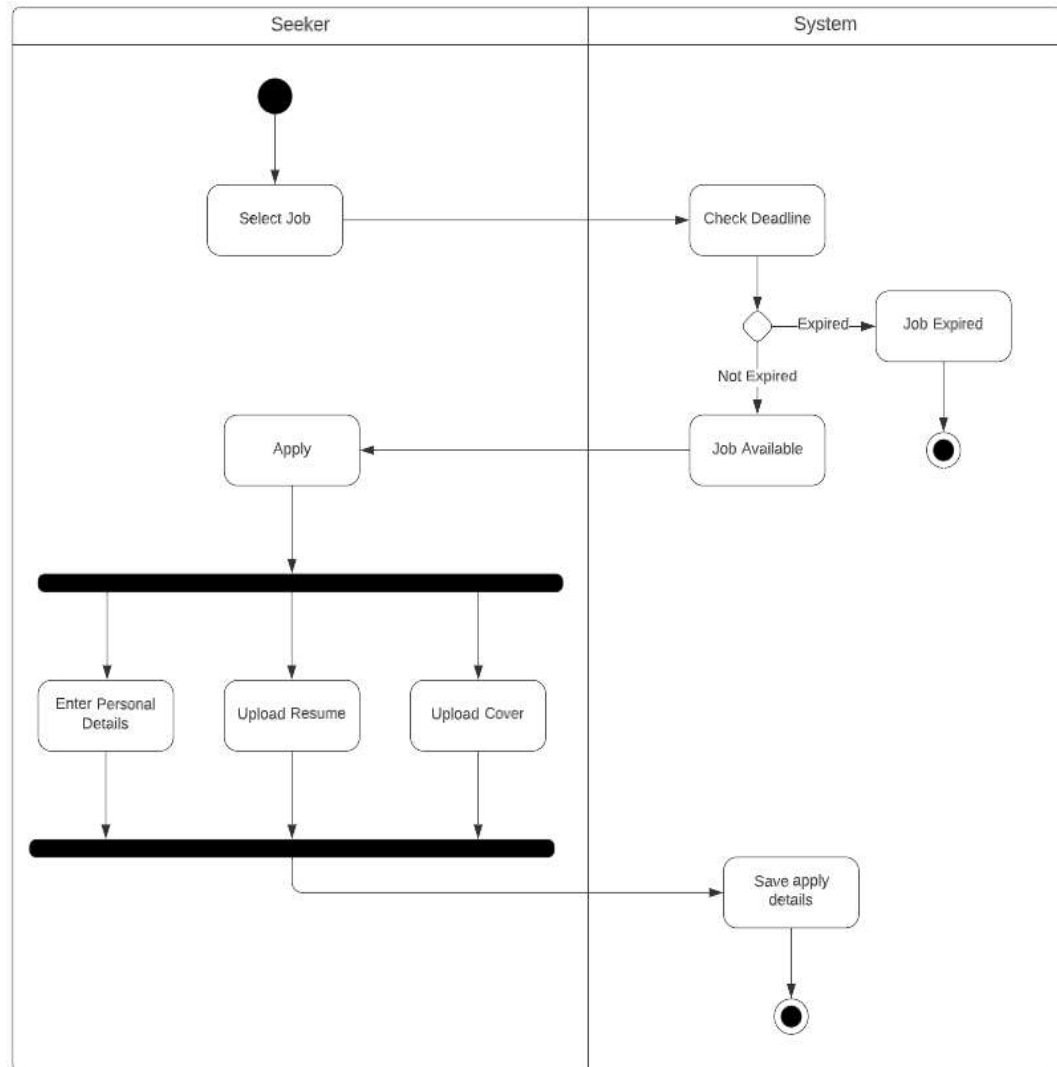


Figure 5.9: Activity Diagram for 'Apply Jobs'

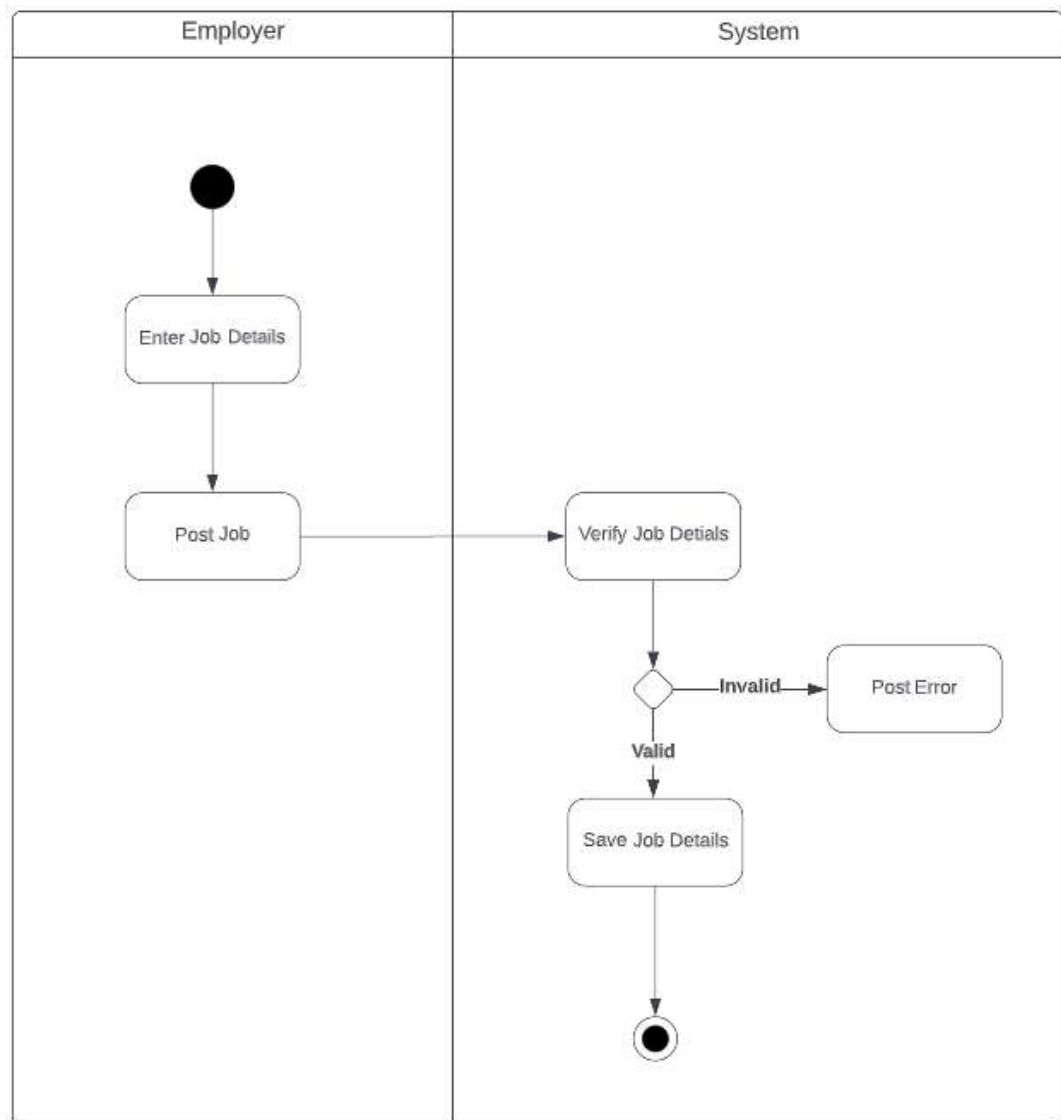
B. Upload Jobs

Figure 5.10: Activity Diagram for 'Upload Jobs'

C. Search Jobs

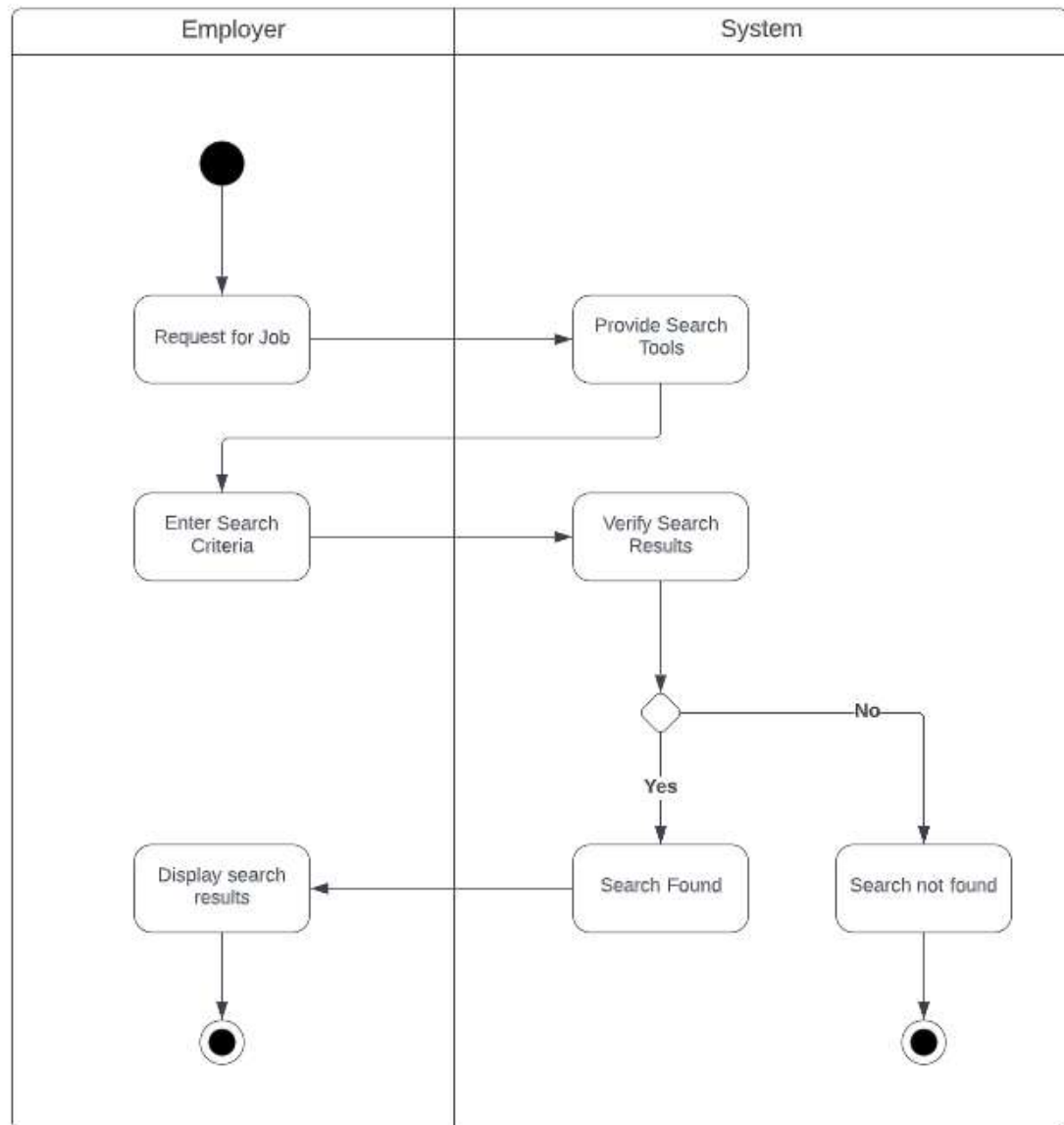


Figure 5.11: Activity Diagram for 'Search Jobs'

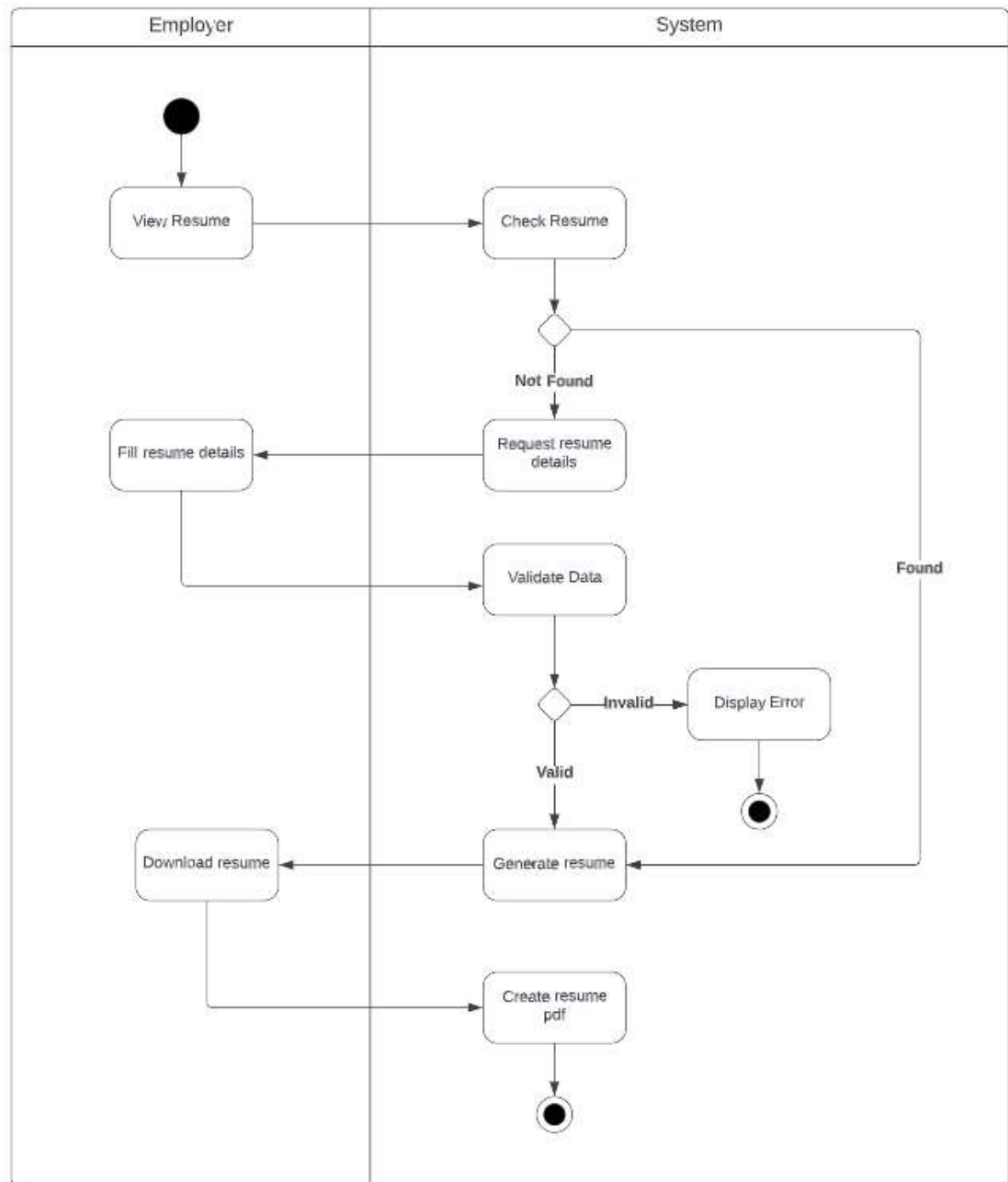
D. Resume Generation

Figure 5.12: Activity Diagram for 'Resume Generation'

5.2.3 Sequence Diagram

A. Authentication

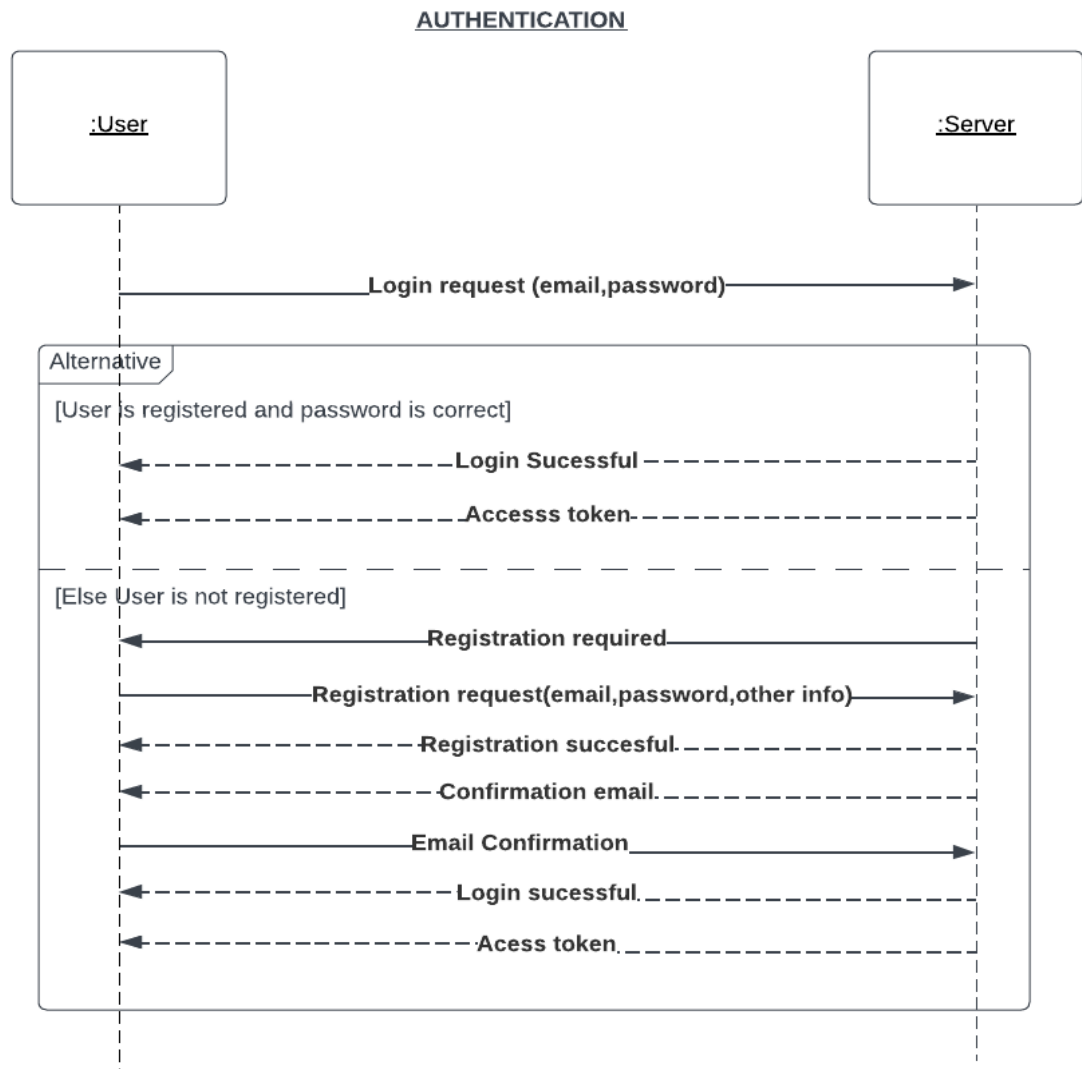


Figure 5.13: Sequence Diagram for ‘Authentication’

B. View Profile

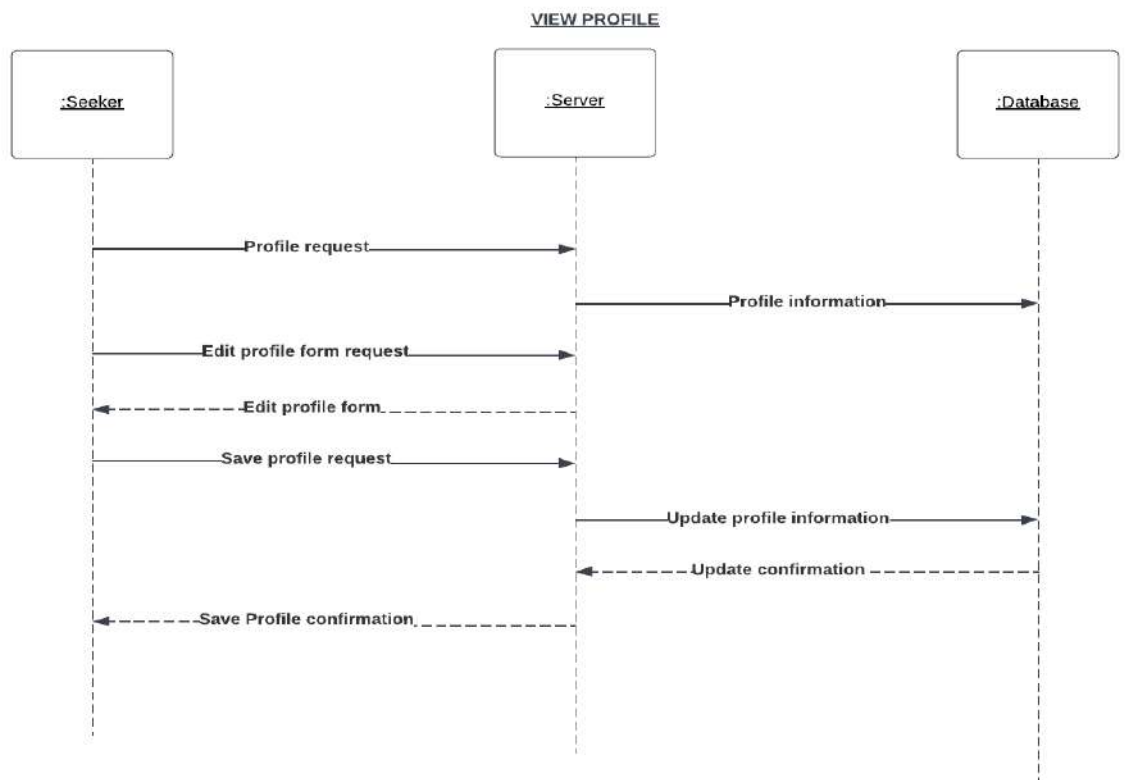


Figure 5.14: Sequence Diagram for 'View Profile'

C. View Job

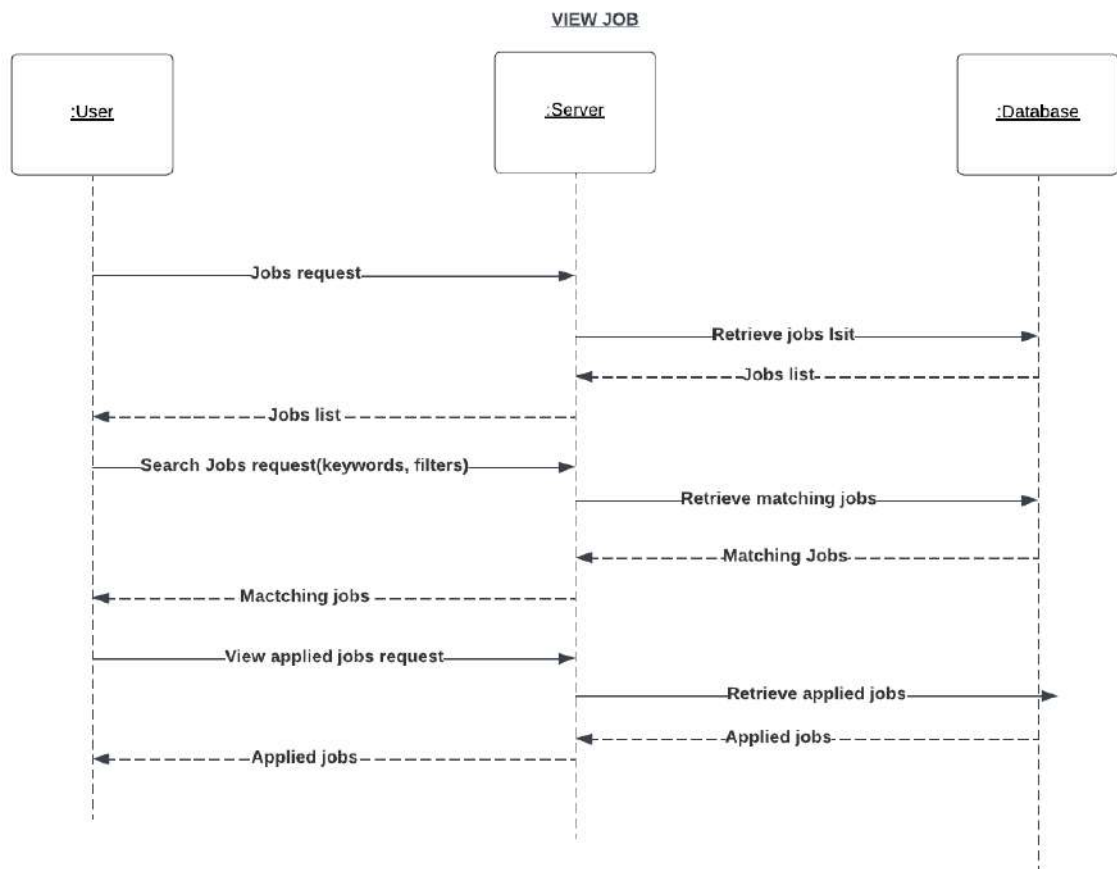


Figure 5.13: Sequence Diagram for 'View Job'

D. Post Job

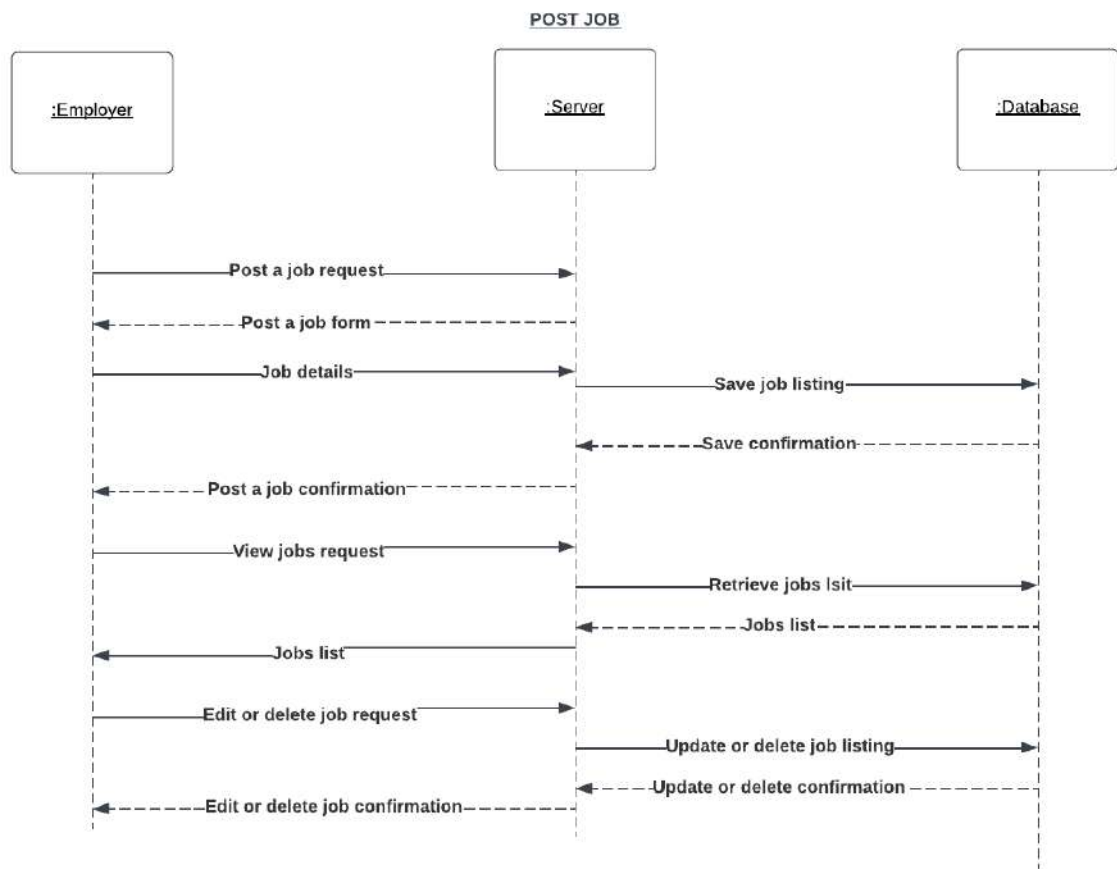


Figure 5.14: Sequence Diagram for 'Post Jobs'

E. Generate Resume

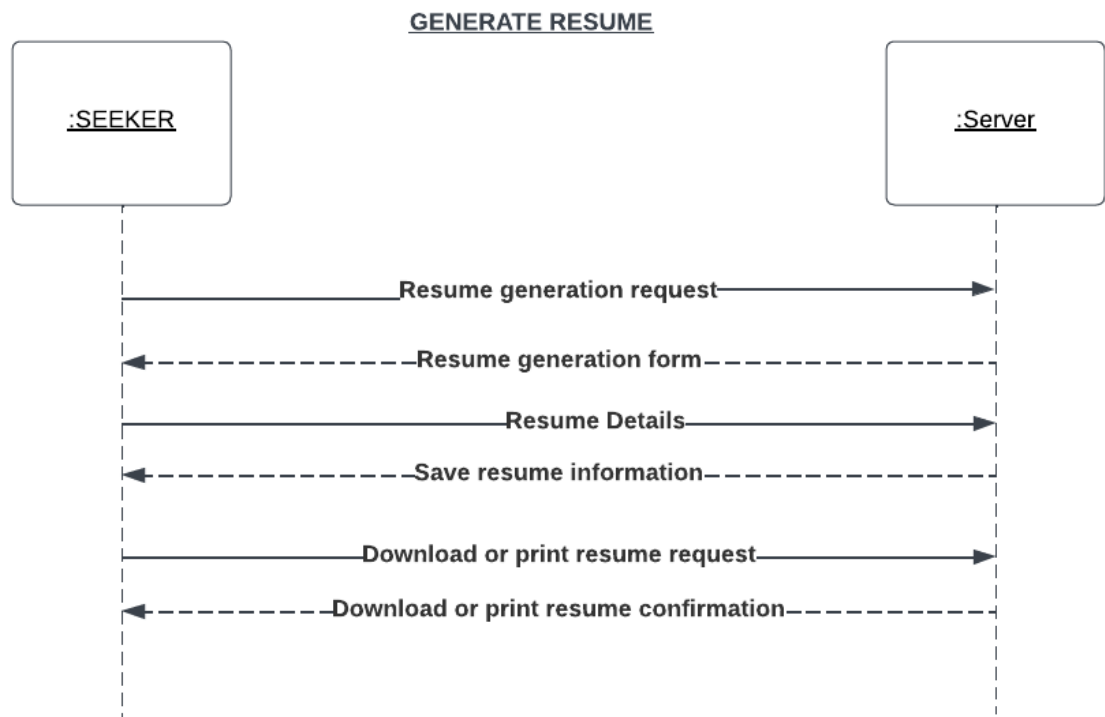


Figure 5.15: Sequence Diagram for 'Generate Resume'

F. Apply for Job

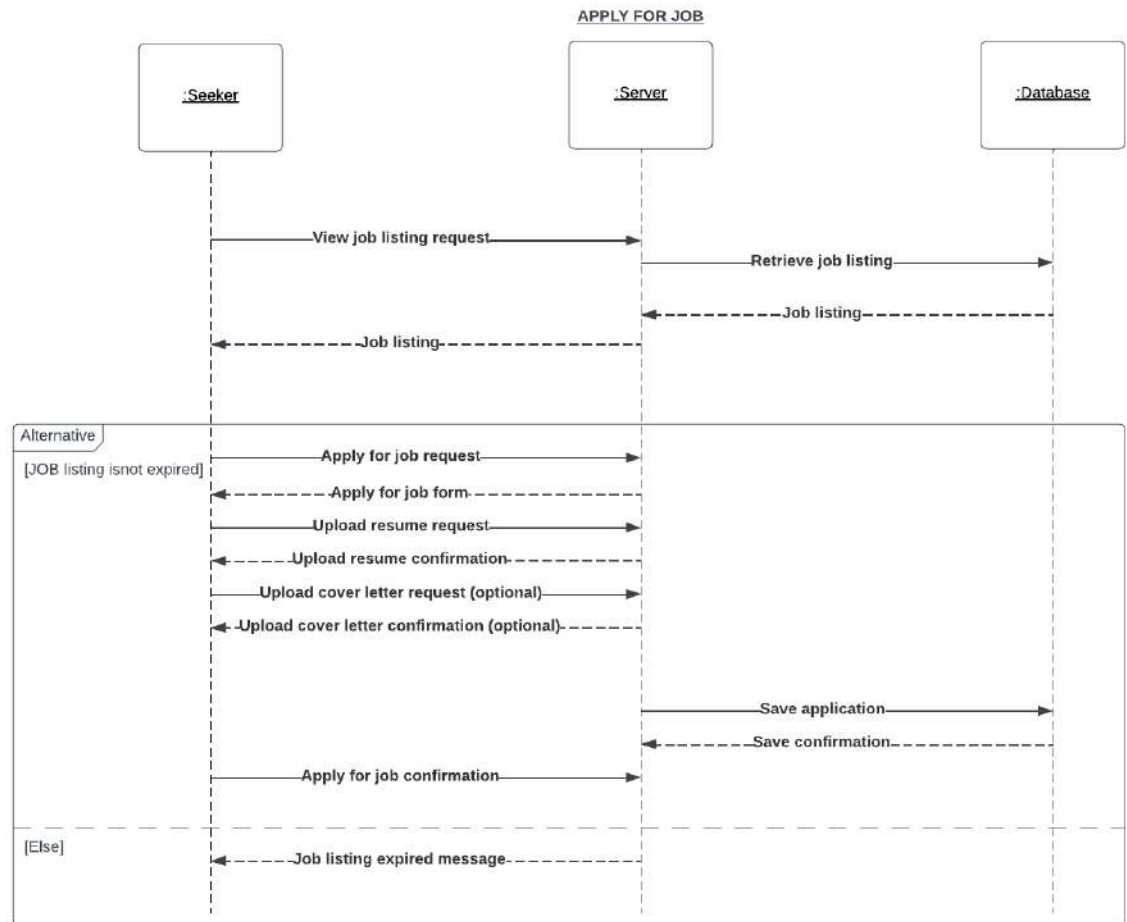


Figure 5.16: Sequence Diagram for 'Apply for Job'

G. Recommendation

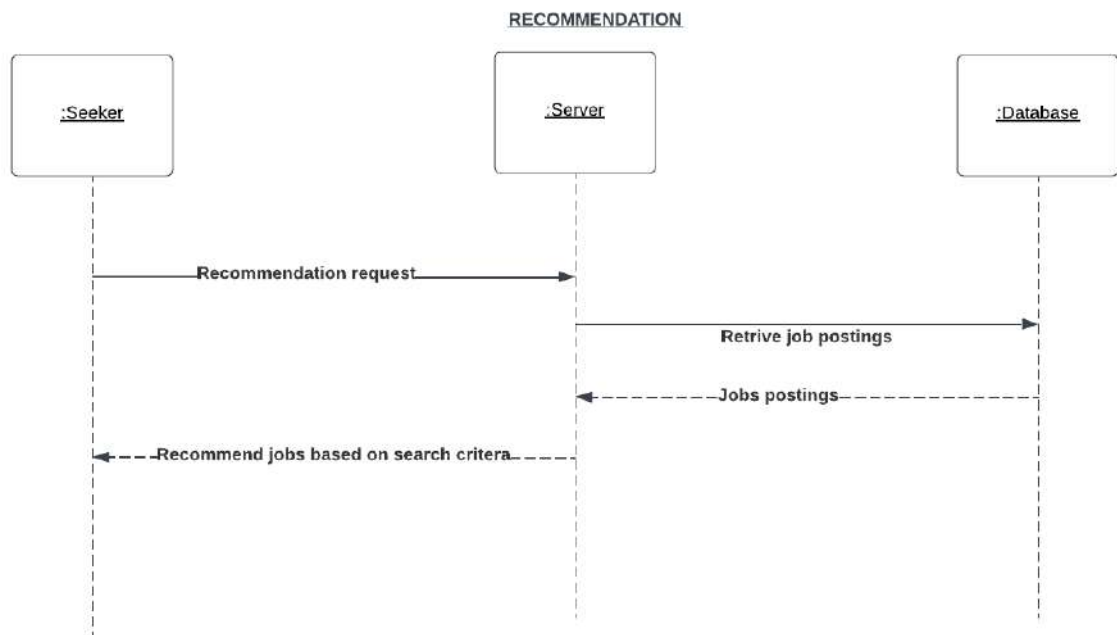


Figure 5.17: Sequence Diagram for 'Recommendation'

5.2.4 Class Diagram

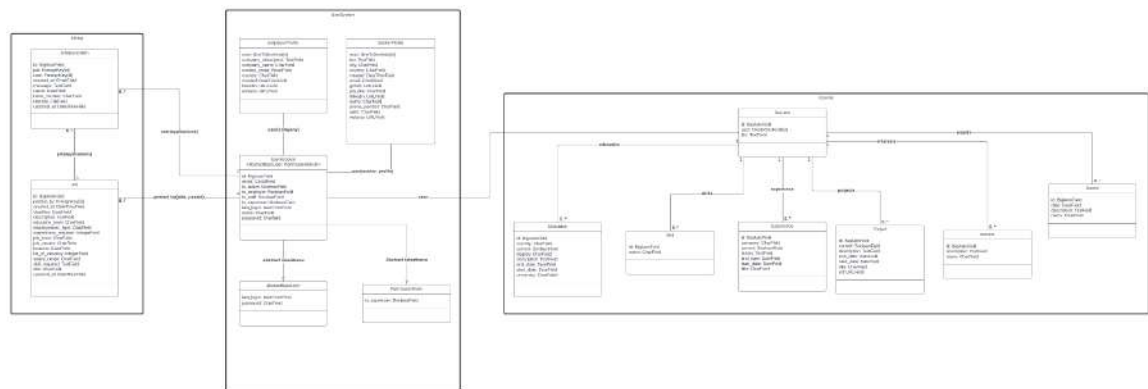


Figure 5.18: System Class Diagram

A. Resume

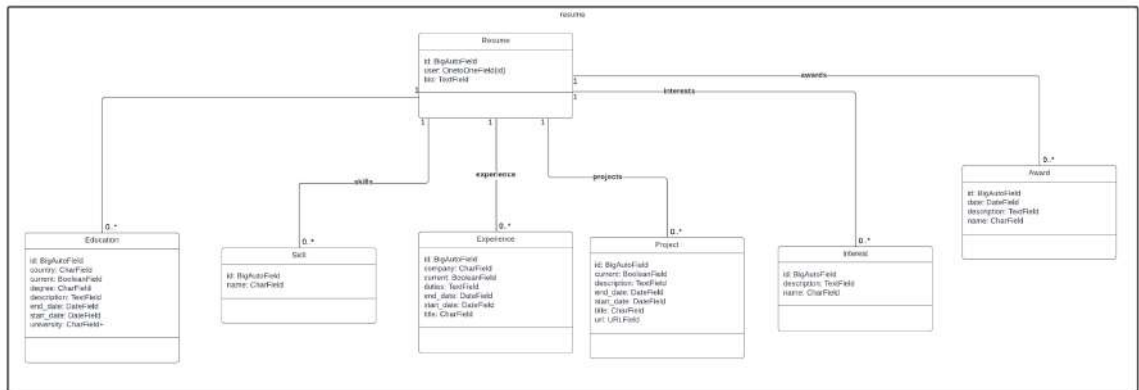


Figure 5.19: Class Diagram for Resume

B. User-System

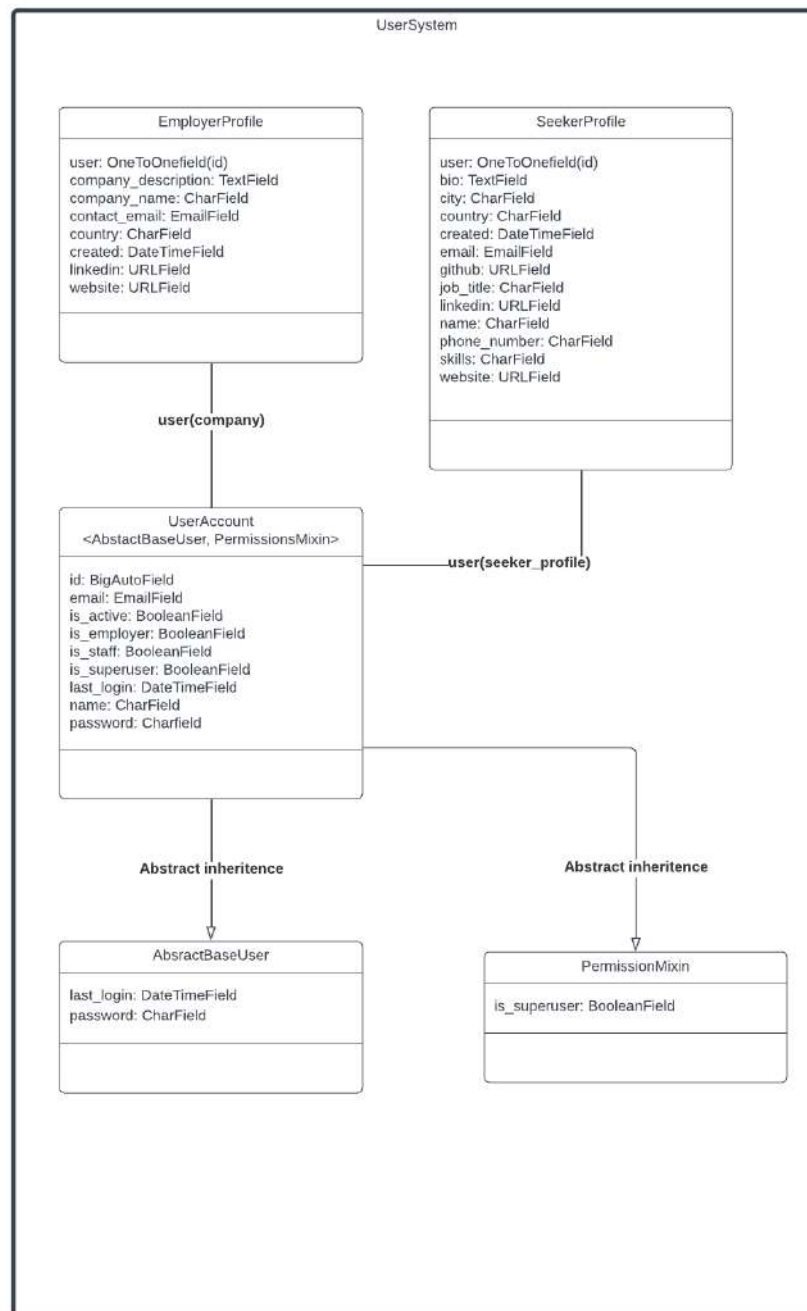


Figure 5.20: Class Diagram for 'User-System'

C. Hiring

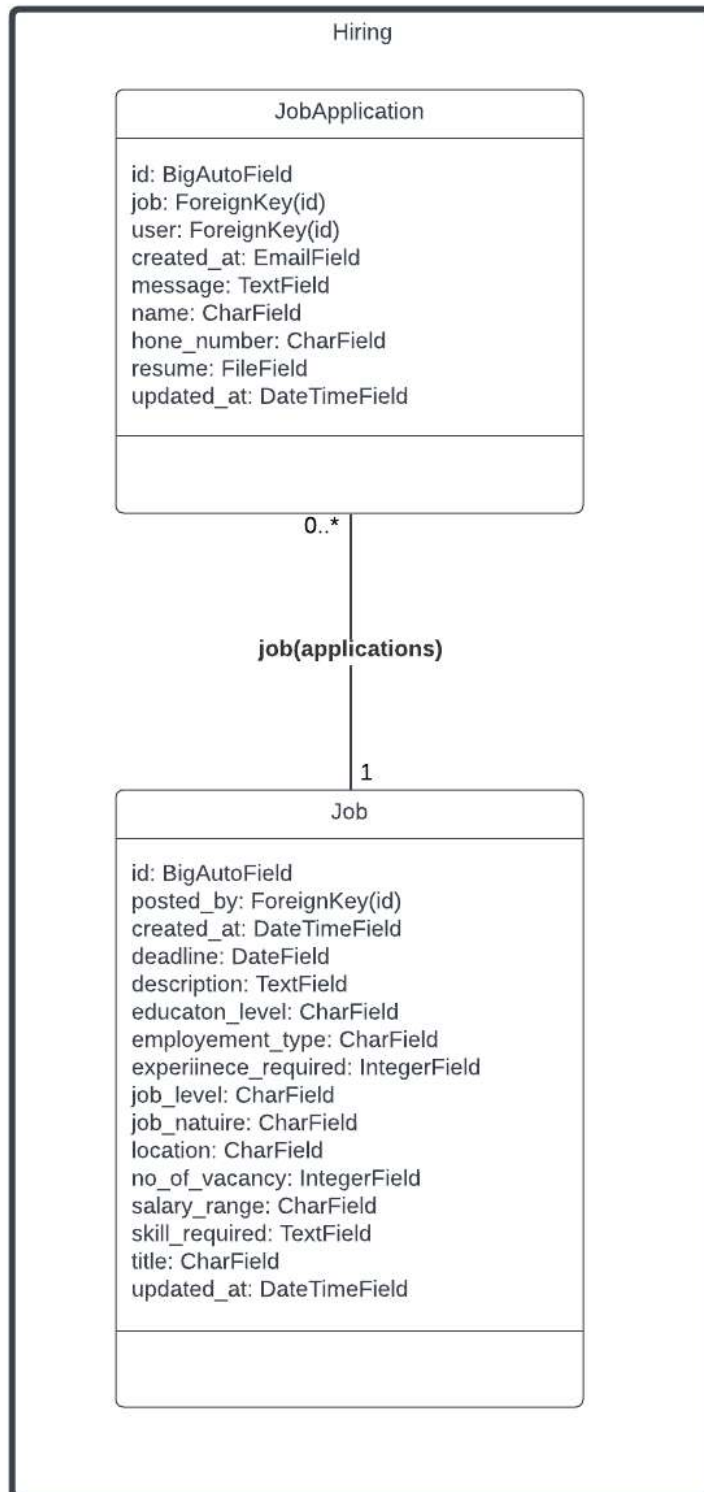


Figure 5.21: Class Diagram for 'Hiring'

5.3 Database Design and Implementation

The job platform was designed and implemented using Django ORM, a powerful and flexible object-relational mapping framework. The following steps were followed to design and implement the database for the job platform:

A. Models Definition

- a) **UserAccount:** This model represents a user account and extends Django's built-in `AbstractBaseUser` and `PermissionsMixin` models. It has fields like `email`, `name`, `is_active`, `is_staff`, and `is_employer` to handle multiple user types.
- b) **SeekerProfile:** This model represents a user's profile information who is a job seeker. It has fields like `name`, `email`, `city`, `country`, `phone_number`, `github`, `linkedin`, `website`, `job_title`, `skills`, `bio`, and `created`.
- c) **EmployerProfile:** This model represents a user's profile information who is an employer. It has fields like `company_name`, `company_location`, `country`, `company_description`, `linkedin`, `website`, `contact_email`, and `created`.
- d) **Education:** This model represents a user's education information. It has fields like `degree`, `university`, `country`, `start_date`, `end_date`, `current`, and `description`.
- e) **Skill:** This model represents a user's skills information. It has a field called `name`.
- f) **Experience:** This model represents a user's experience information. It has fields like `title`, `company`, `start_date`, `end_date`, `current`, and `duties`.
- g) **Project:** This model represents a user's project information. It has fields like `title`, `description`, `start_date`, `end_date`, `current`, and `url`.
- h) **Interest:** This model represents a user's interests information. It has fields like `name` and `description`.
- i) **Award:** This model represents a user's awards information. It has fields like `name`, `date`, and `description`.

- j) Resume: This model represents a user's resume information. It has fields like bio and ManyToManyFields to Education, Skill, Experience, Project, Interest, and Award models.

2) Fields Definition

1) UserAccount model:

- a) email: EmailField
- b) name: CharField
- c) is_active: BooleanField
- d) is_staff: BooleanField
- e) is_employer: BooleanField

2) SeekerProfile model:

- a) user: OneToOneField with UserAccount model
- b) name: CharField
- c) email: EmailField
- d) city: CharField
- e) country: CharField
- f) phone_number: CharField
- g) github: URLField
- h) linkedin: URLField
- i) website: URLField
- j) job_title: CharField
- k) skills: CharField
- l) bio: TextField
- m) created: DateTimeField

3) EmployerProfile model:

- a) user: OneToOneField with UserAccount model
- b) company_name: CharField
- c) company_location: CharField
- d) country: CharField
- e) company_description: TextField
- f) linkedin: URLField
- g) website: URLField
- h) contact_email: EmailField
- i) created: DateTimeField

4) Education model:

- a) degree: CharField
- b) university: CharField
- c) country: CharField
- d) start_date: DateField
- e) end_date: DateField
- f) current: BooleanField
- g) description: TextField

5) Skill model:

- a) name: CharField

6) Experience model:

- a) title: CharField
- b) company: CharField
- c) start_date: DateField
- d) end_date: DateField
- e) current: BooleanField

f) duties: TextField

7) Project model:

- a) title: CharField
- b) description: TextField
- c) start_date: DateField
- d) end_date: DateField
- e) current: BooleanField
- f) url: URLField

8) Interest model:

- a) name: CharField
- b) description: TextField

9) Award model:

- a) name: CharField
- b) date: DateField
- c) description: TextField

10) Resume model:

- a) user: OneToOneField with UserAccount model
- b) bio: TextField
- c) education: ManyToManyField with Education model
- d) skills: ManyToManyField with Skill model
- e) experience: ManyToManyField with Experience model
- f) projects: ManyToManyField with Project model

- g) interests: ManyToManyField with Interest model
- h) awards: ManyToManyField with Award model

3) Relationship Definition

- 1) UserAccount model
 - a) OneToOne relationship with SeekerProfile model through "user" field
 - b) OneToOne relationship with EmployerProfile model through "user" field
- 2) SeekerProfile model
 - a) OneToOne relationship with UserAccount model through "user" field
- 3) EmployerProfile model
 - a) OneToOne relationship with UserAccount model through "user" field
- 4) Resume Model
 - a) OneToOne relationship with UserAccount model through "user" field
 - b) ManyToMany relationship with Education model through "education" field
 - c) ManyToMany relationship with Skill model through "skills" field
 - d) ManyToMany relationship with Experience model through "experience" field
 - e) ManyToMany relationship with Project model through "projects" field
 - f) ManyToMany relationship with Interest model through "interests" field
 - g) ManyToMany relationship with Award model through "awards" field

4) Migrations Creations

In our project, we can use Django's ORM to create database migrations. Migrations are a way of propagating changes made to Django models to the database schema. Creating migrations is a feature provided by Django's ORM that allows us to define changes to our database schema in Python code, and then generate the SQL commands needed to implement those changes in your database. Migrations are used to manage changes to the structure of our database tables over time, including creating new tables, modifying existing tables, and deleting tables.

When we modify our models or add new ones, we need to create a migration to apply these changes to the database. The migration file contains instructions for modifying the database schema to match the new models.

5) Data Migrations

Migrating data is the process of transferring data from one database schema to another, usually when the schema has changed due to new database tables, fields or relationships being added or altered. In Django, migrating data involves running the '**python manage.py migrate**' command, which applies any pending database schema changes and transfers data to the new schema.

When running '**python manage.py migrate**', Django compares the database schema defined in the migration files to the current state of the database and applies any changes required to bring the schema up-to-date. This process can involve creating new tables or columns, dropping existing tables or columns, and modifying existing columns to match the new schema.

5.4 Backend Development

Our job portal website was developed using Django Rest Framework in the backend. Django Rest Framework is a powerful and flexible toolkit for building Web APIs, and provided the foundation for our RESTful API.

A. Design

Our job portal website was developed using Django Rest Framework in the backend. Django Rest Framework is a powerful and flexible toolkit for building Web APIs, and provided the foundation for our RESTful API.

B. Implementation

To implement various functions, we used packages such as Djoser for authentication and JSON Web Tokens (JWT) for authorization. We also leveraged Django Rest Framework's ModelSerializer to serialize and deserialize data between Python objects and JSON representations. Additionally, we used Django permissions to restrict access to certain endpoints based on the user's role.

Our implementation included the following separate points:

- a) Models : We created models for job, job application, user account, seeker profile and employer profile.
- b) Serializers: We created serializers to convert the model instances into Python data types and vice versa. These serializers handled the validation and conversion of request data, as well as the serialization of response data into JSON.
- c) Views: We created serializers to convert the model instances into Python data types and vice versa. These serializers handled the validation and conversion of request data, as well as the serialization of response data into JSON.

We created serializers to convert the model instances into Python data types and vice versa. These serializers handled the validation and conversion of request data, as well as the serialization of response data into JSON.

- d) URLs : We mapped the views to the appropriate API endpoint URLs.

C. Testing

We tested the backend API endpoints using Thunder-Client, a VSCode extension for RESTful API testing. We tested various responses and views to ensure that they were functioning as expected, and also checked for any errors or inconsistencies in the data.

D. Integration

We integrated the backend with our React frontend using the RESTful API. We ensured that the API endpoints were correctly mapped to the frontend components and tested the integration thoroughly to ensure that the frontend and backend were functioning correctly together.

5.5 Frontend Development

Our Job Portal Website was built using React.js as the frontend framework. We utilized Bootstrap5 for the CSS styling of our website, and used Axios for fetching data from the Django backend API. The following sections outline our approach and implementation for frontend development.

A. Design

We followed a modular approach while designing the frontend, breaking down the entire website into small, reusable components. This allowed us to build a highly scalable and maintainable codebase, with each component being responsible for a specific aspect of the website. We also implemented React Router to create a seamless navigation experience for the users, ensuring that each page of the website had its own unique URL.

B. Implementation

We utilized a range of libraries and packages to implement the various functions of the website. We created a series of reusable components, such as Navbar, JobItem, JobApplication, Profile, etc., and implemented them across the website as required. We also integrated Bootstrap5 for the CSS styling of our website, which provided a consistent and responsive layout across all devices.

For fetching data from the backend API, we used Axios, which allowed us to send HTTP requests to the API endpoints and retrieve the required data. We also used Axios interceptors to handle authentication, ensuring that only authorized users were able to access the relevant features of the website.

In our project, we used JWT for user authentication, where the user was required to provide their credentials in order to receive a JWT token. This token was then

used to authenticate the user for future requests to our API. JWT provides a secure way to transmit user information between the client and server, as it is digitally signed and can be verified to ensure that it has not been tampered with.

C. Testing

We tested the website's features by running the website locally and ensuring that each component was rendering correctly and that data was being fetched from the backend API without any issues. We also conducted thorough testing of the website's responsiveness, ensuring that the website was functioning correctly across a range of devices and screen sizes.

Overall, our use of React.js, Bootstrap5, and Axios allowed us to create a highly scalable and efficient frontend for our Job Portal Website, with a fast and responsive user experience that seamlessly integrated with our Django backend.

5.6 Recommendation System Development

Our recommendation system leverages the TfidfVectorizer and cosine similarity functionalities from the Scikit-learn library to develop a solution that analyzes job titles, job descriptions, and required skills. The system then suggests job openings that are most relevant to the job seeker's profile by calculating the similarity between the job seeker's profile and each job posting in the system.

A. Data collection and Cleaning

Jobs data is collected from the database as a queryset and then converted to a pandas dataframe. The job titles, descriptions, and skills are cleaned using regular expressions to remove punctuation, convert to lowercase, and remove unwanted characters.

B. Vectorizing the data

The cleaned data is vectorized using TfidfVectorizer and CountVectorizer from scikit-learn. The vectorized data is then used to create a tf-idf matrix and count matrix for job titles, descriptions, and skills.

C. Computing the cosine similarity

The cosine similarity scores between the input job and all other jobs are computed using `cosine_similarity()` from scikit-learn. Cosine similarity is a

measure of the similarity between two non-zero vectors of an inner product space.

D. Combining similarity scores

The cosine similarity scores for job titles, job descriptions, and skills are combined to get a weighted average of similarity scores. The weights for title, description, and skills are set as 0.4, 0.2, and 0.4, respectively.

E. Finding top N recommendations

The indices of the top N jobs with the highest cosine similarity scores are found using `np.argsort()`. The top N recommendations are then selected from the jobs dataframe.

F. Adding similarity scores

The similarity scores of the recommended jobs are added to the results dataframe. The scores are also multiplied by 100 and rounded to two decimal places. The company name is also added to the results dataframe by fetching it from the `EmployerProfile` model.

G. Returning results

The results dataframe is returned as the output of the recommendation system.

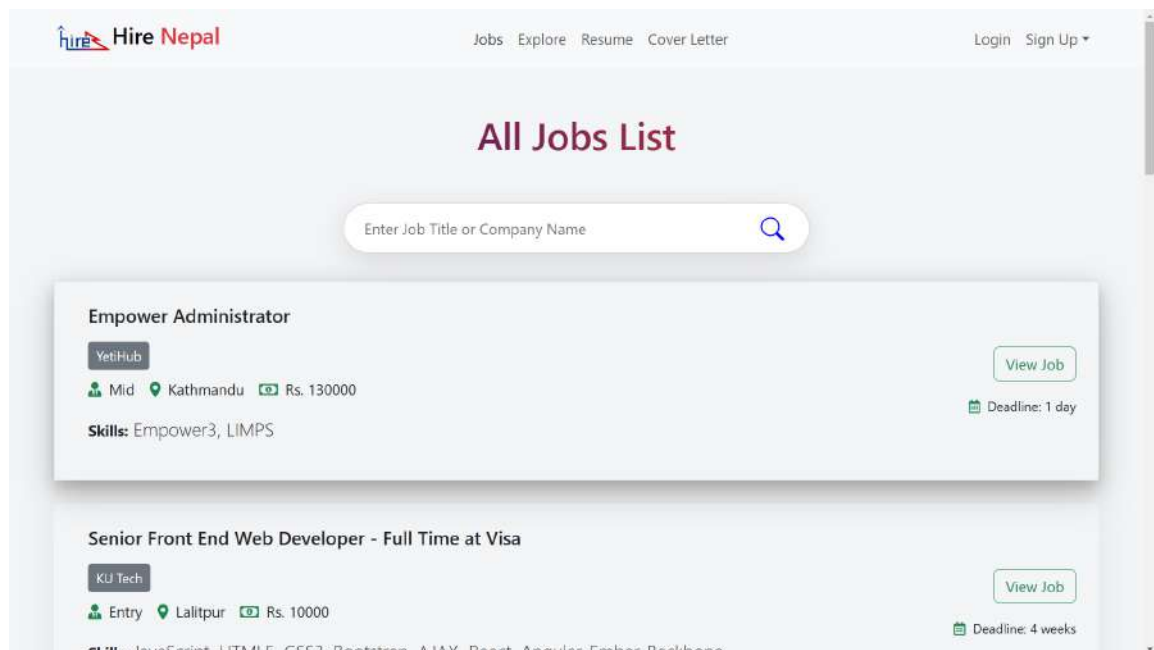
5.7 Integration and Testing

The job portal website underwent comprehensive integration and testing to ensure that the frontend and backend were functioning correctly together. The backend API endpoints were tested using Thunder-Client, and the integration with the React frontend was thoroughly tested to ensure correct mapping of API endpoints to frontend components. Additionally, the website's features were tested locally to ensure correct rendering and data fetching, and the website's responsiveness was tested across a range of devices and screen sizes. These tests helped to identify and resolve any errors or inconsistencies in the website's functionality, resulting in a well-integrated and thoroughly tested job portal website.

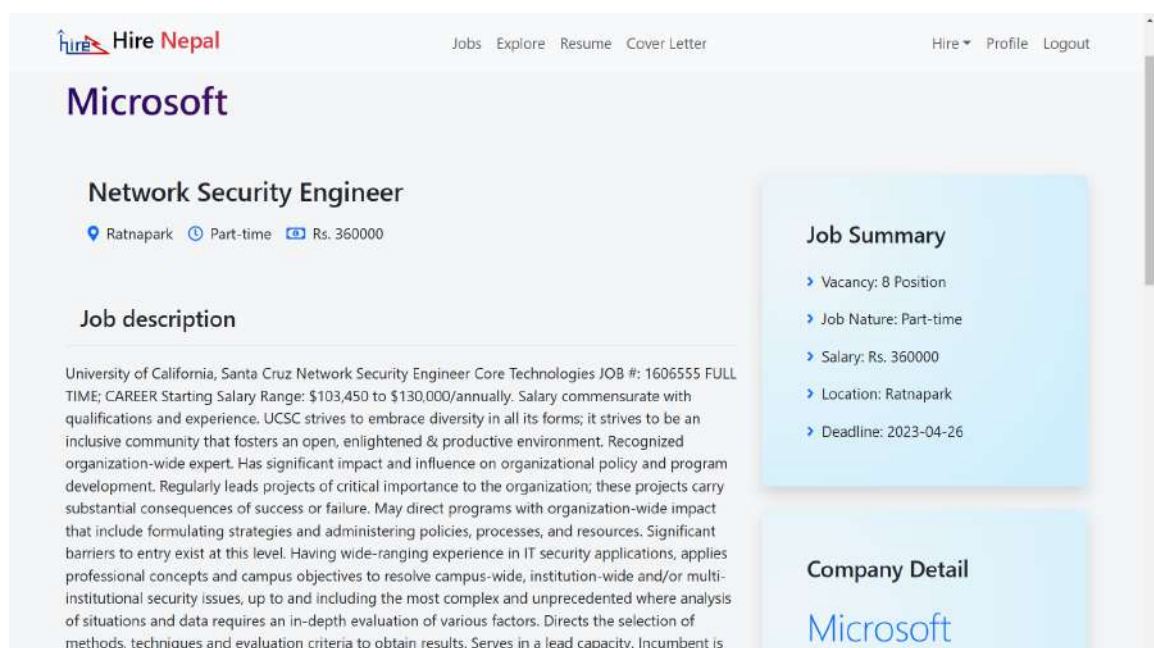
6. RESULT

We have provided below some screenshots of the major features in our job platform system. These images will give you a better idea of the user interface and functionalities that our platform offers.

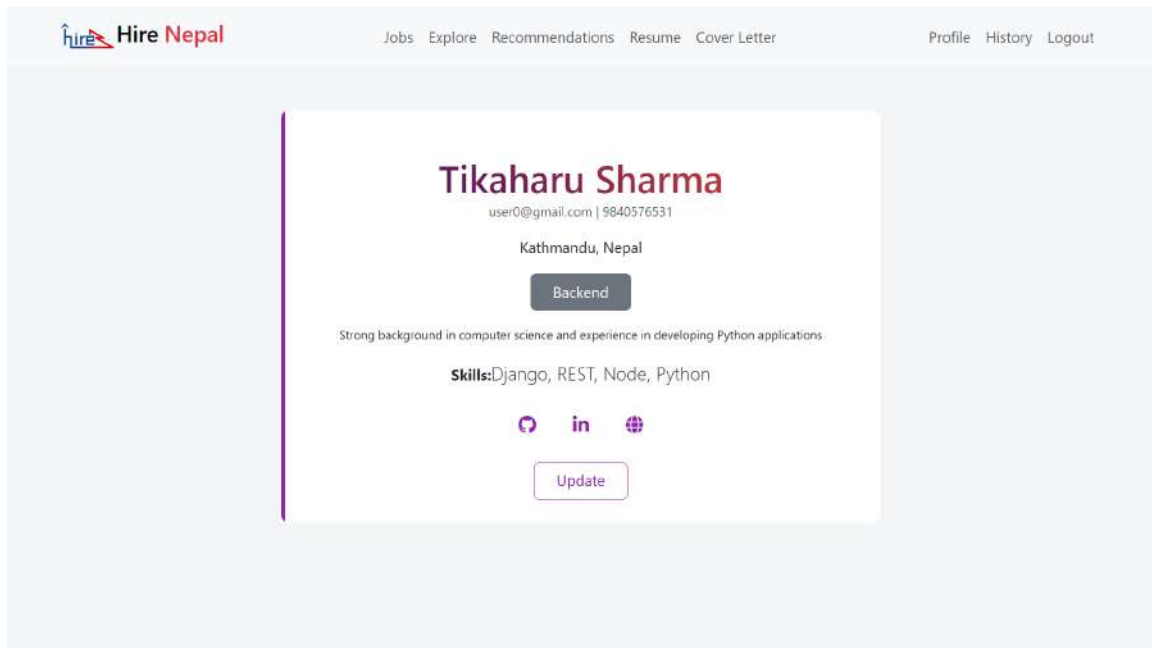
6.1 All Jobs List



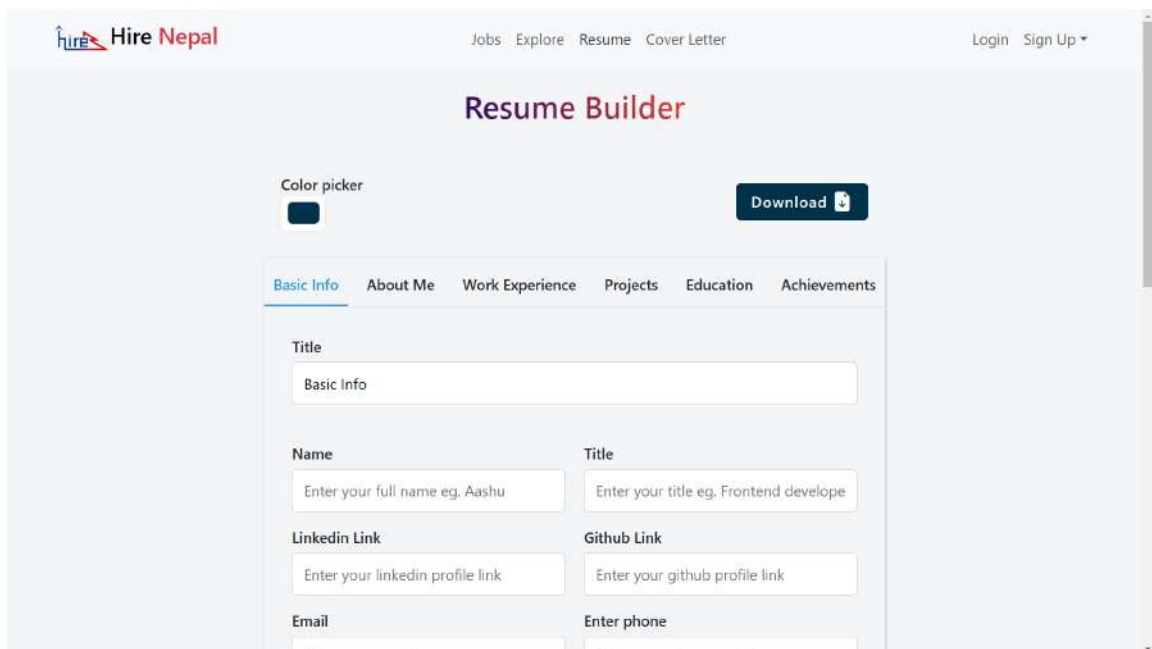
6.2 Job Detail



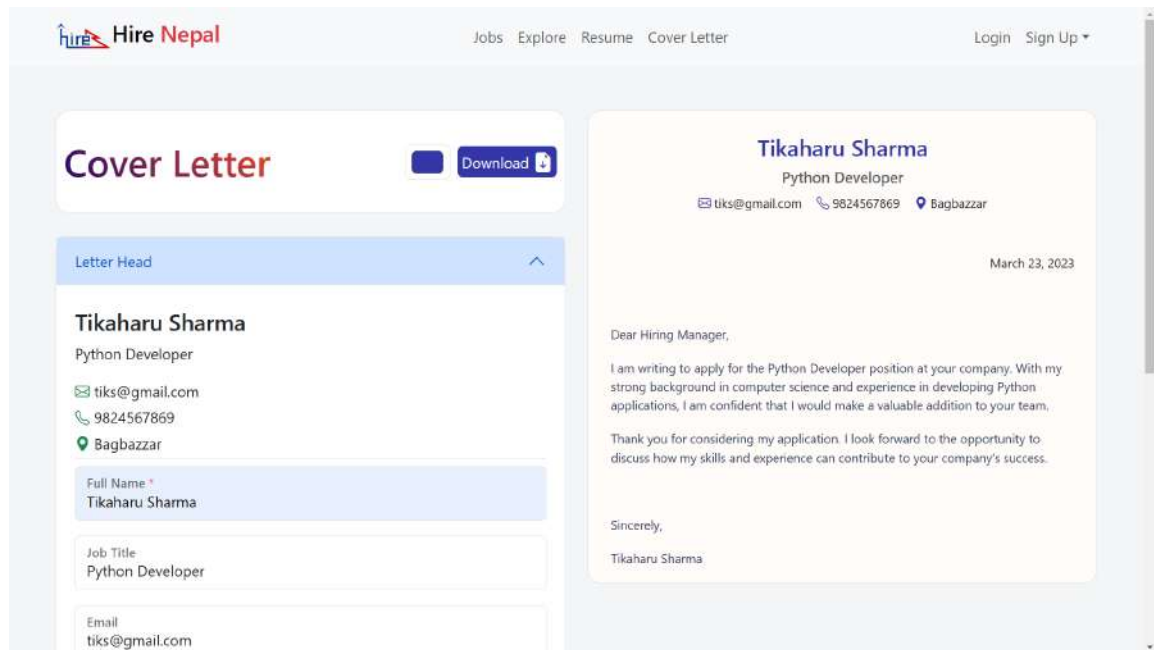
6.3 User Profile



6.4 Resume Builder



6.5 Cover Letter Generator



Cover Letter Download

Letter Head

Tikaharu Sharma
Python Developer
tiks@gmail.com
9824567869
Bagbazzar

Full Name *
Tikaharu Sharma

Job Title
Python Developer

Email
tiks@gmail.com

Tikaharu Sharma
Python Developer
tiks@gmail.com 9824567869 Bagbazzar

March 23, 2023

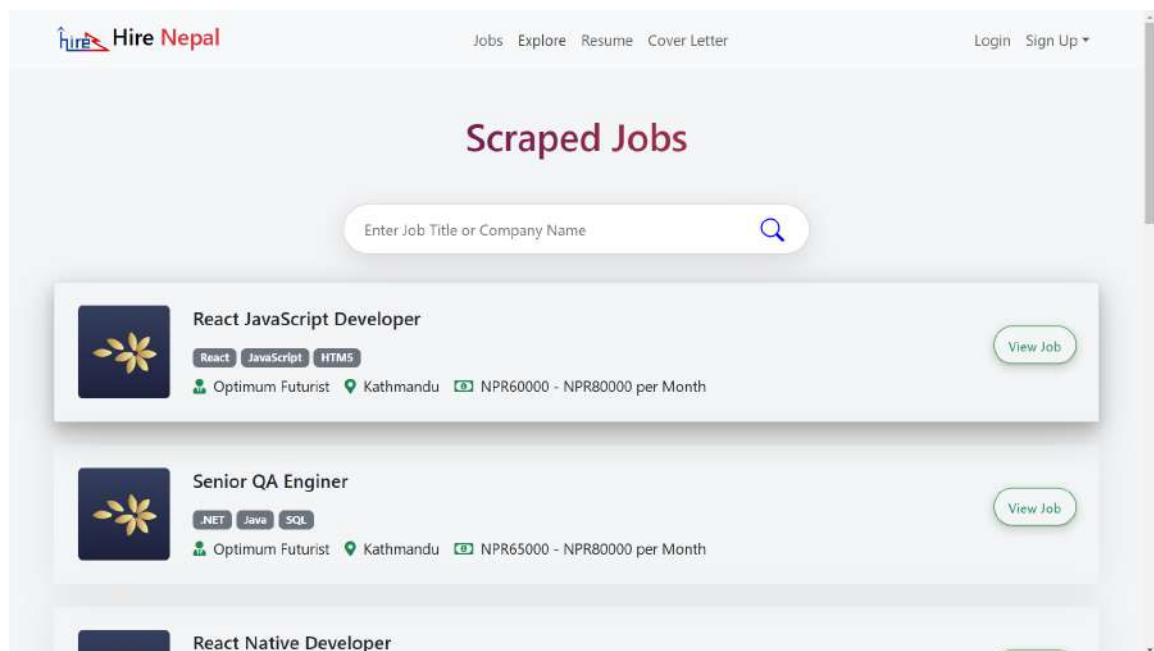
Dear Hiring Manager,

I am writing to apply for the Python Developer position at your company. With my strong background in computer science and experience in developing Python applications, I am confident that I would make a valuable addition to your team.

Thank you for considering my application. I look forward to the opportunity to discuss how my skills and experience can contribute to your company's success.

Sincerely,
Tikaharu Sharma

6.6 Scraped Jobs



Scraped Jobs

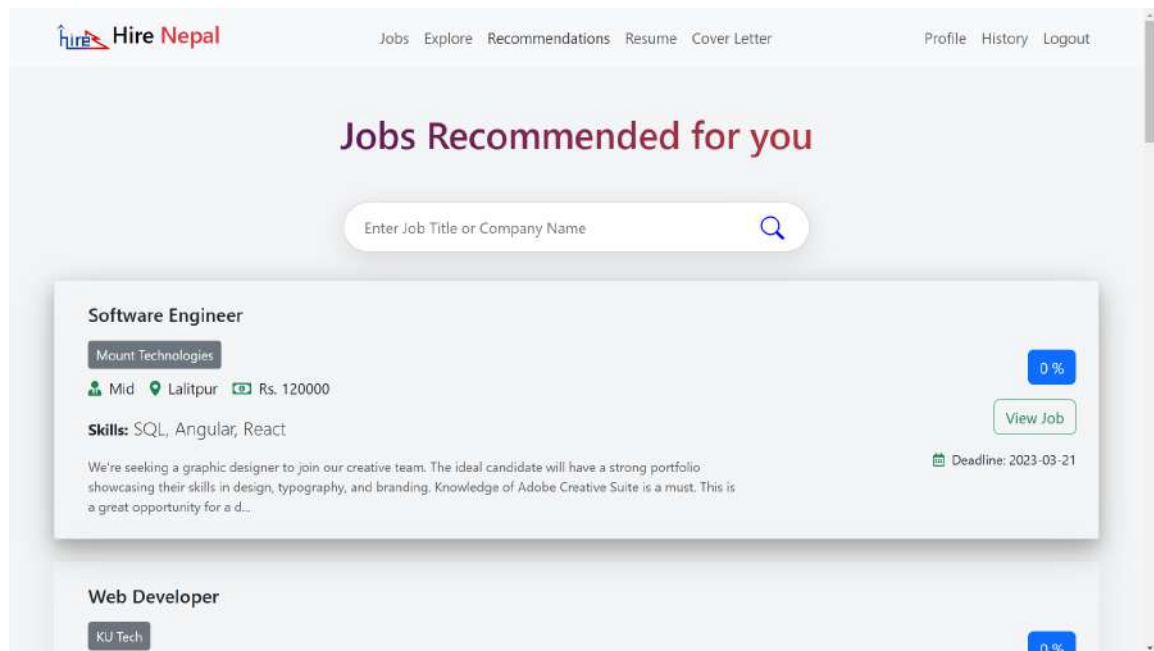
Enter Job Title or Company Name

React JavaScript Developer
React JavaScript HTMS
Optimum Futurist Kathmandu NPR60000 - NPR80000 per Month View Job

Senior QA Engineer
.NET Java SQL
Optimum Futurist Kathmandu NPR65000 - NPR80000 per Month View Job

React Native Developer

6.7 Recommended Jobs



Hire Nepal Jobs Explore Recommendations Resume Cover Letter Profile History Logout

Jobs Recommended for you

Enter Job Title or Company Name

Software Engineer

Mount Technologies

Mid Lalitpur Rs. 120000

Skills: SQL, Angular, React

We're seeking a graphic designer to join our creative team. The ideal candidate will have a strong portfolio showcasing their skills in design, typography, and branding. Knowledge of Adobe Creative Suite is a must. This is a great opportunity for a d...

0 %

View Job

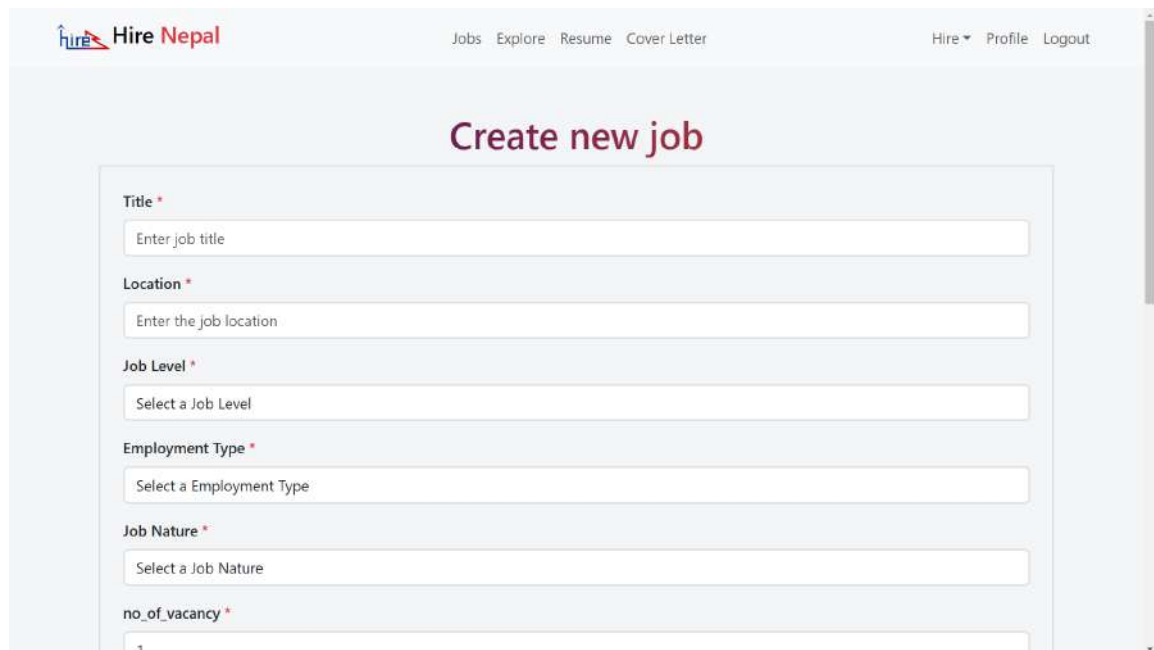
Deadline: 2023-03-21

Web Developer

KU Tech

0 %

6.8 Create New Job



Hire Nepal Jobs Explore Resume Cover Letter Hire Profile Logout

Create new job

Title *

Enter job title

Location *

Enter the job location

Job Level *

Select a Job Level

Employment Type *

Select a Employment Type

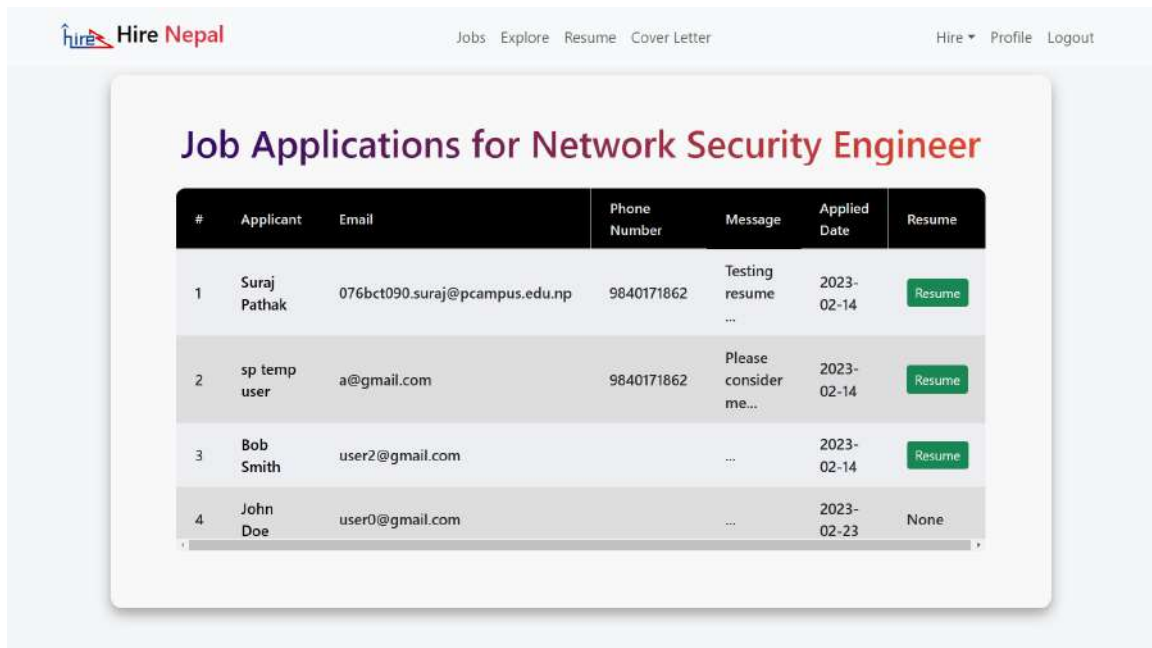
Job Nature *

Select a Job Nature

no_of_vacancy *

1

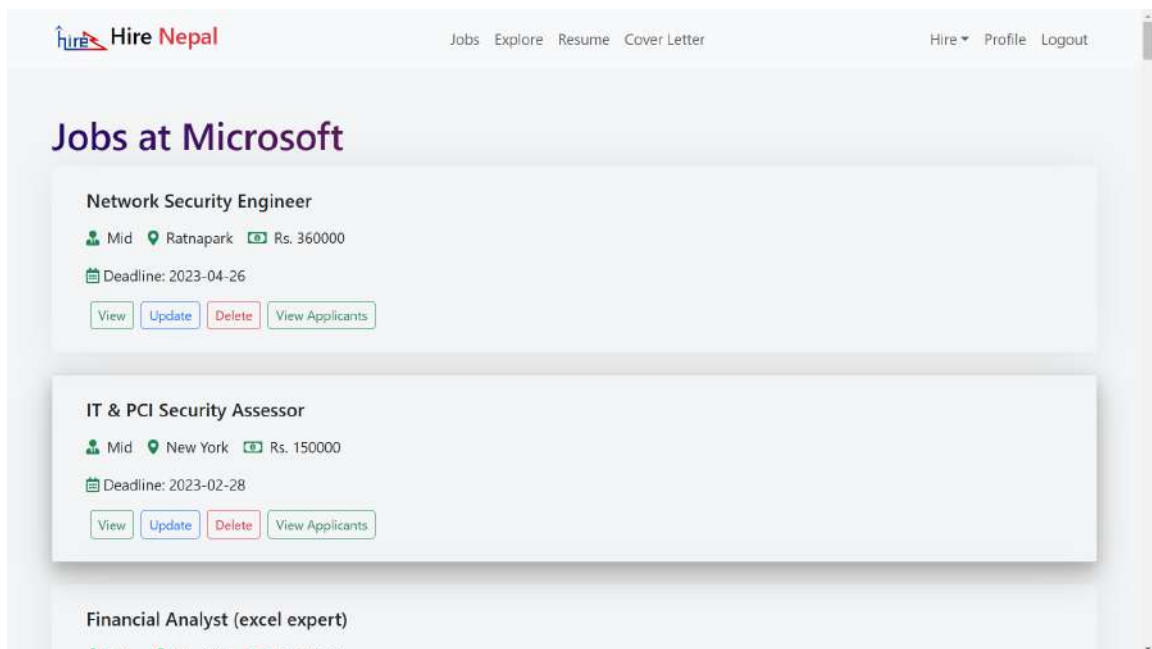
6.9 View Applicants



Job Applications for Network Security Engineer

#	Applicant	Email	Phone Number	Message	Applied Date	Resume
1	Suraj Pathak	076bct090.suraj@pcampus.edu.np	9840171862	Testing resume ...	2023-02-14	Resume
2	sp temp user	a@gmail.com	9840171862	Please consider me...	2023-02-14	Resume
3	Bob Smith	user2@gmail.com		...	2023-02-14	Resume
4	John Doe	user0@gmail.com		...	2023-02-23	None

6.10 View Own's Job listing



Jobs at Microsoft

Network Security Engineer
 Mid | Ratnapark | Rs. 360000
 Deadline: 2023-04-26
[View](#) [Update](#) [Delete](#) [View Applicants](#)

IT & PCI Security Assessor
 Mid | New York | Rs. 150000
 Deadline: 2023-02-28
[View](#) [Update](#) [Delete](#) [View Applicants](#)

Financial Analyst (excel expert)
 Mid | New York | Rs. 30000

7. EPILOGUE

7.1 Project Schedule

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8
Planning								
Research								
Requirement Gathering								
Database Design								
Backend Development								
Frontend Development								
Recommendation system Development								
Integration & Testing								
Documentation								

Figure 7.1: Gantt Chart

7.2 Conclusion

After completing the development of the job portal, we are confident that our project has met the objectives we set out to achieve. The incorporation of a recommendation system using Tfidfvectorizer and cosine similarity has enhanced the user experience of the platform, allowing job seekers to connect with job opportunities that align with their interests and skills.

The job portal offers a wide range of features and tools, including the ability to create and upload resumes, generate cover letters, and receive personalized job recommendations. We believe that these features, combined with the ease of use and customization, will attract a large number of job seekers and employers to the platform.

Overall, we are proud of the job portal we have developed. We would like to extend our gratitude to everyone who has contributed to the success of this project, including our mentors, advisors, and the department of Electronics and Computer Engineering at Institute of Engineering, Pulchowk Campus, for their generous support in providing the necessary resources and facilities.

7.3 Limitation

1. Currently, the website only offers tech-related job postings, limiting its reach to a specific industry and potentially excluding job seekers from other industries.
2. The resume and cover letter templates are limited to only one design, which may not cater to everyone's preferences or needs.
3. The website lacks a bookmark feature, making it difficult for job seekers to save job postings they are interested in or want to revisit later.
4. There is no messaging feature available on the website, making it difficult for job seekers and employers to communicate directly through the platform.
5. The website currently lacks any email notification system, which may result in job seekers and employers missing important updates or communications.
6. The job recommender feature only takes into account a few profile features, potentially resulting in inaccurate or irrelevant job recommendations.
7. The search functionality on the website is limited, making it difficult for job seekers to filter and find job postings that meet their specific requirements.

7.2 Future Enhancement

1. Expansion of job postings to include a wider range of industries, making the website more inclusive and accessible to job seekers across various fields.
2. Addition of multiple resume and cover letter templates, allowing job seekers to choose a design that best fits their needs and preferences.
3. Implementation of a bookmark feature, allowing job seekers to save job postings they are interested in or want to revisit later.
4. Addition of a messaging feature, allowing job seekers and employers to communicate directly through the platform.
5. Implementation of an email notification system, sending job seekers and employers updates and reminders about their job applications and postings.
6. Enhancement of the job recommender feature to consider more profile features and provide more accurate and relevant job recommendations.
7. Improvement of the search functionality on the website, allowing job seekers to filter and find job postings based on specific criteria, such as location or salary.

REFERENCES

- [1] Django Project. (2021). Django documentation. [Online]. Available: <https://docs.djangoproject.com/en/3.2/>
- [2] Django REST framework. (n.d.). Django REST framework. [Online]. Available: <https://www.django-rest-framework.org/>
- [3] Scikit-learn developers. (2021). Scikit-learn documentation. [Online]. Available: <https://scikit-learn.org/stable/documentation.html>
- [4] React. (n.d.). React documentation. [Online]. Available: <https://reactjs.org/docs/getting-started.html>
- [5] W. S. Vincent, "Building a RESTful API with Django Rest Framework," William Vincent, 05-Jan-2021. [Online]. Available: <https://wsvincent.com/django-rest-framework-tutorial/>.
- [6] J. Brownlee, "Building a Recommendation System using Scikit-Learn," Machine Learning Mastery, 10-Sep-2018. [Online]. Available: <https://machinelearningmastery.com/make-recommendations-sckit-learn/>.
- [7] React. (n.d.). React.js Tutorials. [Online]. Available: <https://reactjs.org/tutorial/tutorial.html>
- [8] React Training. (n.d.). React Router: Declarative Routing for React.js. [Online]. Available: <https://reactrouter.com/web/guides/quick-start>