Job Board: A Business and employment-oriented online service.

A Project

Submitted to the

Faculty of Science: Computer Science Department

Menoufia University

By

Mohammed Mohammed Fathi Hamuda
Mohammed Saeed Al-Sayed Mahmoud
Helmy Gamal Mustafa Bahbah

In Partial Fulfilment of the Requirements

for the Degree of

BACHELOR OF SCIENCE

Major Program:

Computer Science

July 2022

Supervised by
D.r/ Mariana Barsoum

ABSTRACT

Business and employment-oriented online service is a type of service where applying for a job or seeking one is conducted over the web and managed almost fully automatically by the system.

The purpose of this website is to Facilite the process of recruitment / job finding for both businesses and people, it's designed to help employers find suitable job seekers for their business and vice versa.

Our website’s main functionalities include: the ability to search for a job as a job seeker or post one as an employer, it also requires clients to register to be able to apply for a job, clients can search for a job by categories depending on their desired filed, the recruiter can then communicate with the client through the provided email or through the integrated chat. Our website was designed using client-side languages such as HTML, SCSS, jQuery, JavaScript and Bootstrap, combined with the server-side language which is mostly Python using Django Framework and Django Template and the Database side was designed using SQL, the server side contains all the implementation related to setting up the database, creating session models for joining different user-interface (UI) pages by mapping the desired category or job ID to the respective IDs stored in the database. The client side is responsible for showing the entire user interface, containing the CSS, HTML, and JavaScript.

TABLES OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Chapter 1 Introduction

In our modern era it's quite difficult to find a job or to start a new business at different scales especially with the competition from the well-established brands and with the current number of people with different talents and skills applying for the same job, even if the quality of the products of the business is so good – due to the lack of advertisement – it becomes another face in the sea and almost goes unnoticeable, to face this issue we decided to build a website that may facilitate the process of searching and applying for a job at different scales of companies and various of categories and fields, not only our website focuses on helping people search for jobs, it also helps businesses looking for employees to quickly post and advertise their need for new employees.

Job Board is an advanced search engine that specializes in displaying jobs opening or post jobs for new positions. Organizations and recruiters may use the Job Board to post open positions and search through the applier's resume databases. Workers may use the same website to search for a job and find a new career opportunity depending on their desired field and their skills across a range of variant fields.

1.1 Motivation

The motivation for us behind designing this website came because we were so much familiar with the digital world facilitating almost every aspect of our life, and slowly becoming the norm in our daily basis, further using the Job Board, instead of going through the hassle of going to the companies physically by travelling if it's located in a different city which can be very expensive and wary, and instead of companies struggling to advertise their need for employees, our website works as a mediator between the two to facilitate the process. Moreover, from the back-end perspective we valued learning a new framework; Django framework that is, whilst we learnt about JS, Bootstrap in the front-end side, seeing how powerful these work together is fascinating, it's even more amazing seeing it come to work. We also had the desire to helping computer science students understand the concepts of web-application designing, it would be very easy to incorporate the idea of using programming techniques from the available visuals to understand how a piece of code appears on a user interface.

1.2 Aim of the web

Our website was designed for the purpose of showcasing the capabilities of modern languages and frameworks and how we managed to make use of them. This website showcases the basics about the appearance of a first web page and how a complete working website can be built from scratch, it also provides the concept of user-integrated graphics and how JavaScript can be embedded into HTML through Django framework, it gives insight about how the client-side languages interact with the server-side language and the database.

1.3 Paper Organization

The rest of the document is divided into three parts: Objectives, Implementation, and Testing. The Objectives chapter lists the need for building the system. It provides use cases to help the business and technical users with their understanding. It also gives a detailed explanation for each use case to help with design and implementation, and outlines the constraints regarding the software. The Implementation chapter contains the detailed design of the system, including the Class Diagram, Activity Diagram, and Component Diagram. This chapter also includes a detailed explanation for each component as well as the interaction of the class and its components with each other when carrying out certain tasks, whilst providing Screenshots from the website and the source code.

CHAPTER 2

OBJECTIVES

Chapter 2 objectives

All the steps required in the software-analysis process related to this project (product function, user characteristics, functional and non-functional requirements, constraints, assumptions, and dependencies for the online shopping cart application) are described in the following sections.

2.1 Requirements analysis

The requirements analysis and gathering processes are critical for the success of any software engineering project. Requirements analysis in software engineering is a process that determines the tasks that are required to determine the needs and conditions to design a new product or to make modifications in any existing product/application. This process considers all the stakeholders' conflicting requirements, and analyses the documentation and validation of the system. The requirements should be actionable, measurable, testable, and related to the defined needs of the system design. From the software-engineering perspective.

In this paper the requirements analysis is written with clearness, completeness and consistency in mind. The analysis also handles any ambiguous requirements that do not clearly state what needs to be implemented, which could create a loss of resources and time if identified later in the development or testing phase.

Stakeholder analysis says that, to clearly gather the requirements of the project, analysts first need to identify the stakeholders. Stakeholders are people or organizations that have a valid interest or use in the system thus the following is included in the paper:

- Anyone who operates the system.
- Anyone who benefits from the system.
- Anyone who is directly or indirectly involved in purchasing the system.
- People or organizations opposed to the system.
- Organizations responsible for the system design.
- Organizations that regulate the financial or safety aspects of the system.

Once the stakeholders are successfully identified, interviews are conducted through

Requirements Documentation: This step involves documenting the requirements in various forms, including summary lists, natural language documents, visual documents, use cases, user stories, or process specifications. A requirement specification document is categorized in different ways according to the stakeholders' need, helping to create a clear contract between development and business. The following sections include the different.

categories of requirements specification document that are essential for designing this application: the functional requirements, constraints, system requirements, etc.

2.2 Product Perspective

The Job Board is a web-based system. It can be accessed using any browser such as: Google chrome, Microsoft edge, Mozilla Firefox or any other Chromium-based browsers.

2.3 Hardware and software interface

The Job Board website's minimum hardware requirements are the following hardware configurations:
For PC:

- **Processor:** Pentium 4 intel processor
- **Minimum disk space:** 100 MB of free hard-drive space
- **Minimum Ram:** 128 MB of RAM
- **Operating system:**
  Windows XP SP2
  OS X 10.5.6
  Ubuntu 10.04
  Fedora Linux 14
- Having Google chrome, Microsoft edge, Mozilla Firefox, or any other Chromium-based browsers installed.

Or any other system that can run a supported browser.

For mobile phones:

Any IOS, Android or any similar operating system that is capable of running a Chromium-based or a similar browser.

2.4 User interface

There are 2 interface types found in the Job Board website which are as follows:

User interface: All users are able to view the home page and browse available jobs, categorize them according to their desired needs, any further action such as posting a job or applying for one will require registration which also divides users into 2 types:

1. **Job Seeker:** Once you register as a job seeker in addition to having access to browsing jobs and categorizing them, you can now apply to jobs posted by the Employers, once you apply, your information is sent to the employer who can later check it and decided to either contact you through the provided email or through the website's integrated chat.

2. **Employer:** Once you register as an employer you are able to post jobs with your desired requirements such as:

1. Job title
2. Job type (Part time / Full time).
3. Description.
4. Available vacancy.
5. Expected salary.
6. Category.

once a job seeker applies to your posted job; you will get notified and you will instantly be able to check their application then later you can either decide to accept or reject it or communicate with the Job Seeker through the website's integrated chat.

1. **Admin interface:** It contains the admin panel; it can only be accessed by the website owners. It contains all the data of all website's users and relation tables. Can be used to easily modify the data on the website.

2.5 Product functions

The Job Board website would have the following basic functions:

1. Display all the Popular categories in the home page.
2. Display the featured candidates.
3. Display the two options of either posting a job or applying for one.
4. Allow any user/visitor to be able to search through and browse the available jobs and categorize them according to their needs.
5. Users need to register for an account to access job posting / applying.
6. If you register as an employer; you gain the access to post a job, view application sent from appliers.
7. If you register as a job seeker, you gain the access to apply for a job.
8. Allow the administrator to add/ remove more jobs.
9. Allow the administrator to add/remove a new job category.
10. Registered users can communicate with each user using the website's integrated chat.

2.6 User Characteristics

The users of the Job Board website are based on their roles which are customers (Employer or Job Seeker) and the administrator (owner). These users are identified based on their experience and technical expertise.

• **Admin:** The administrator is the owner of the website. One must obviously have a basic understanding of computers and the internet, they also need to have prior knowledge for SQl, Python programming language, and Django framework. The administrator is responsible for maintaining all the training documents required for the system. The administrator can perform the following functions:

- Have full access to the database and see all the website registered accounts
- Have the access to add/remove jobs from the website.
- Have the access to add/remove job categories from the website.

• **Users:** The users of the Job Board website are all clients who would browse the website, they can later on be categorized as either employers or job seekers after registering. They must have basic understandings about computers and the internet. using our website users can perform the following functions:

1. Employers:

- Post a job with their specified requirements.
- View the application of the users that previously applied.
- Communicate with Job Seekers through the website's integrated chat.

2. Job Seekers:

- Search for a job according to their desired field.
- Categorize jobs according to their desired needs
- Check their previously sent application.
- Communicate with the Employers through the website's integrated chat.

2.7Constraints

1. Hardware and Software limitation: The system can at least run on a system that meets the requirements specified previously.
2. Accessibility: Initially, the website should be hosted for a limited time just for the showcase.
3. Others: The website should be built using Django framework and Django template in the back-end side, and HTML, CSS, Bootstrap in the front-end side, initially it can be accessible through the Visual studio IDE or simply through window's built-in CMD tool and later be hosted on a server.

2.8 Assumptions and dependencies

The assumptions and dependencies are as follows:

1. Users and the administrator who are previously accustomed to the paper-based system would require training to use the online Job Board website.
2. The system is dependent on the availability of the host Server to run.
3. We assume that system users adhere to the system's minimum software and hardware requirements.

2.9 Specific requirements

This section contains details about all the software that is required for designers to create a system to satisfy the users' requirements and for testers to test the given requirements. This section contains the interface description of each GUI for the different system users. These sections also give descriptions about all the system inputs, all the functions performed by the s system, and all the system output.

2.10 Functional requirements

This section contains the requirements for the Job Board website. The functional requirements, as collected from the users, have been categorized as follows to support the types of user interactions that the system shall have.

1. **Educational Purpose:** The main purpose of this Job Board website is to act as a showcase of our programming capabilities as computer science students to build a website from scratch using multiple programming languages and frameworks.
   FR01: The students and supervisors shall be able to view the source code for the entire website.
   FR02: The students shall be able to, individually, view and understand the code for all pieces on the backend and frontend according to which part they worked on.
   FR03: The students shall be able to debug the application's source code using any debugging tool.

2. **Users:** all users shall be able to view the home page of the Job Board website when they first open the website. The users shall be able to view the different categories, select categories and filter job fields according to their needs, browse through the jobs in each category without needing to register. Once the user registers they can choose to act an employer or a job seeker and gain access to the res of the website's functions .

- **Unregistered Users:**
  FR04: all users shall be able to browse all the available jobs on the website.
  FR05: all users shall be able categorize the search using filters such as: category, title, description and experience.

The website visitors must register from now on to gain access to the rest of the functions, which divides the users into two categories:

- **Job Seekers:**

FR06: The users who aim to apply for a job need to fill an application to apply for the job they desire.
FR07: The users can communicate with the employer they applied their job for through an integrated chat, a notification will be sent once the employer replies back.

- **Employers:**

FR08: The users shall be able to post a job and act as an employer.
FR09: Recruiters can be seen in the "Featured candidates" category, the recruiters who post more frequently are the ones featured.
FR010: The user who post a job can determine their job title, description, vacancy, salary, experience and add an image.

FR11: The users can communicate with whom applied to their job through an integrated chat, a notification will be sent to the employee once the employer sends a message.

3. **Login/User Authentication:**

FR10: The users shall login or register using the user authentication form.
FR12: The users shall not login or register if the information is incomplete or invalid.

4. **Admin: View the website's base:**

FR13: The admin shall be able to view all the registered users on the website through the admin panel.

Admin: Add/Delete Jobs/Jobs categories:

FR14: The admin shall be able to add more jobs to the website through the admin panel.
FR15: The administrator shall be able to add more job categories which can be later accessed by filters by customers through the admin panel.

2.11 Performance requirements

This section lists the performance requirements expected from the Job Board website. Be noted that the results may vary according to the connection type and speed used, in this test a VDSL 15 Mb/s connection with an estimation of an 80 Ms response time used.

1. **PR01:** The users should be able to login in / out in less than 3 seconds.

2. **PR02:** The users should be able to filter jobs in less than 2 seconds.

3. **PR03:** The users should be able to apply/post a job in less than 2 seconds.

4. **PR04:** The navigation between panels should take fewer than 1 second.

CHAPTER 3
IMPLEMENTATION

Chapter 3 Implementation

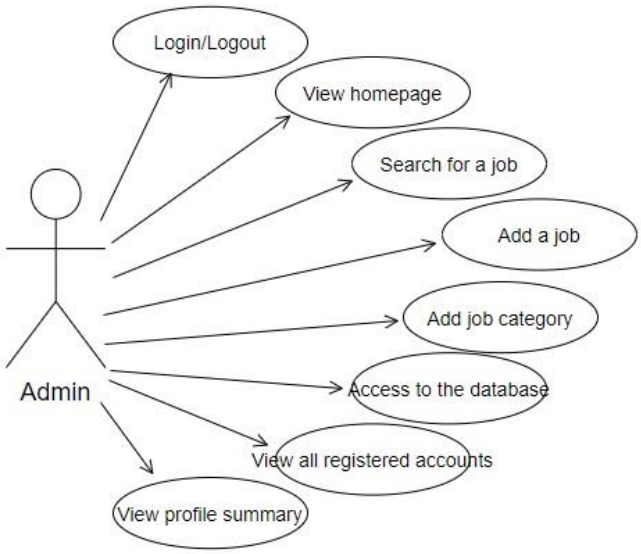
This chapter includes the detailed design used to build the Job Board website. The system's design is used to create the functions and operations of the gathered requirements in detail, including screen layouts, business rules, process diagrams, and other documentation. The output of this chapter describes the new system which is defined as a collection of modules and subsystems. This design stage takes the initial input requirements that were identified in the approved requirements specification document. For each requirement, there is a set of one or more design elements that are produced using the different prototypes. These design elements describe the desired software features, in detail, including functional hierarchy diagrams, screen layouts, activity diagrams, and class diagrams. The intention of these diagrams is to describe the software in detail so that the system can develop the application with less additional design input. The system’s mock screen shots are shown later in this chapter.

3.1 High level use case diagram

This section contains the system use-case diagram for the Job Board website, it also has a detailed explanation for each use case in the system.

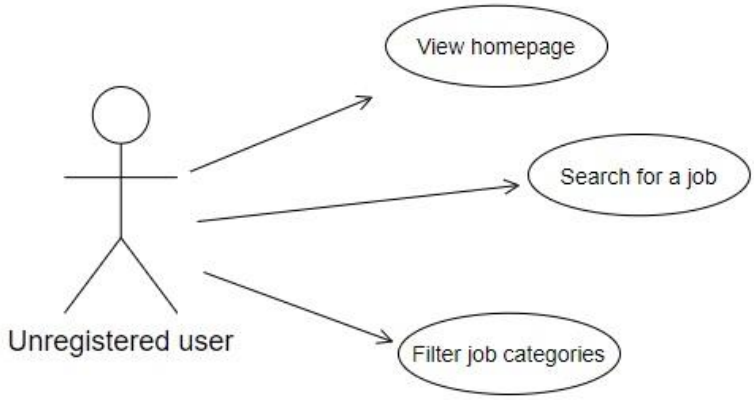
The website’s use case shows the user a simplified and a detailed view of the website and how the actors would interact with each other and with the website. The explanation for each use case is then provided under each figure for helping the user to understand who are the actors as well as giving a brief description for each use case along with its pre - and post - conditions that should be satisfied once the use case is implemented in the website.

Figure 1: demonstrates the use case of for an administrator where they full access to the website. The administrator – in addition to having the user's privilege – they can also access the database and have access to see all the website's registered accounts, have access to add/remove jobs or job categories from the website.



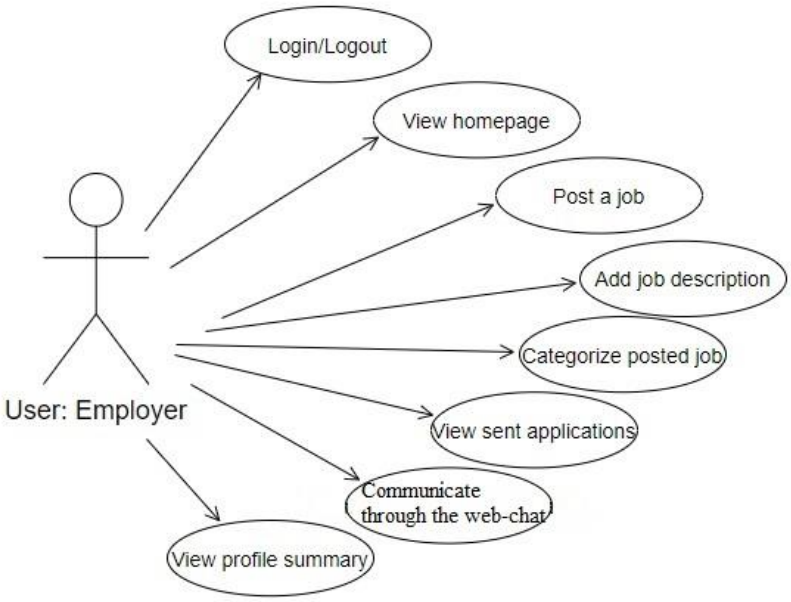
(Figure 1: Use-Case diagram: Admin**).**

Figure 2: demonstrates the use case of an unregistered user, they only have the privilege of viewing the home page, search for a job or filter the search result according to their needs.



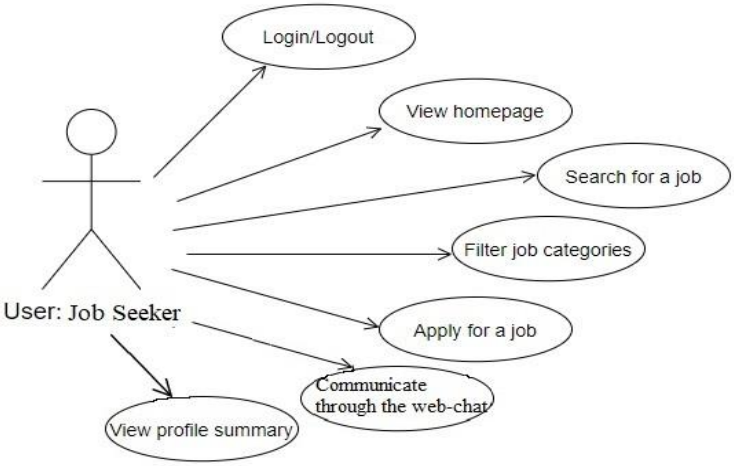
(Figure 2: Use-Case Diagram: Unregistered user**).**

Figure 3: demonstrates the use case for users when they act as an employer, they can access the home page, post a job, define the number of available vacancies and categorize it, communicate with job seekers through the website's integrated chat.



(Figure 3: Use-Case Diagram: User: Employer**).**

Figure 4: user: job seeker demonstrates the use case for users where they have access to the Job Board website and act as a job seeker, they can access the home page, search for a job, filter their search result, apply for a job or communicate with employers through an integrated chat.



(Figure 4: Use-Case Diagram: User: Job Seeker**).**

Below are the different use cases in the system, the use case, and the actors associated with each use case. The description is used for a novice user to better understand the workings of the system and the pre-conditions that should be satisfied before invoking each use case.

1. **Use-Case Number:** US-001:

Use-Case Name: Home page.

Use-Case Description: This use case lets the user/administrator view the home page when they first open the website. It contains jobs categories, featured candidates and testimonial.

Primary Actor: Admin/User.

Precondition: Open the website.

Post-condition: The user/admin successfully opens the website and is able to view the home page.

Basic flow:

- Open the website, the first view will be the home page.

2. **Use-Case Number:** US-002:

Use-Case Name: Search for a job.

Use-Case Description: The user is able to search for a job through the “Browse job” tab.

Primary Actor: User/Admin.

Precondition: Open the website then click on “Browse a job tab”.

Post-condition: The user/admin successfully opens the website and is able to view the available jobs.

Basic flow:

- Open the website.
- Click on “Browse job tab”.

3. **Use-Case Number:** US-003:

Use-Case Name: Filter jobs.

Use-Case Description: The user/admin is able to filter the jobs according to their needs.

Primary Actor: User/Admin.

Precondition: Open the website, click on “Browse job” tab then choose the suitable filter according to their needs.

Post-condition: The User/Admin will only see jobs fitting their entered filters.

Basic flow:

- Open the website.
- Click on “Browse Job” tab.
- Choose a filter suitable for their needs.

4. **Use-Case Number:** US-004:

Use-Case Name: Login.

Use-Case Description: This use case helps the User/Admin to post/apply for a job by logging into the user-authentication form

Primary Actor: User/Admin

Precondition: Open the website, click on Login or if the user is trying to post/apply for a job.

Post-condition: The user is successfully able to log in.

Basic flow:

- Open the website.
- Click on login.
- Enter the username and password.
- Login succeeds.

Exceptional Flow:

- Open the website.
- Click on login.
- Enter an incurrent username or password.
- Login fails.

5. **Use-Case Number:** US-005:

Use-Case Name: Register.

Use-Case Description: This lets users register for the website to be able to post/apply for a job.

Primary Actor: Users.

Precondition: Open the website, Click on “Register”.

Post-condition: The user is now a registered user in the database and is able to login using their credentials.

Basic flow:

- Open the website.
- Click on register
- Fill the requirements (Username, first name, last name, email address, password, password confirmation).
- Click on sign up.
- Successfully sings up.

Exceptional Flow:

- Open the website.
- Click on register
- Don’t to fill one of the fields.
- Receives “Please fill the missing filed” error.
- Sign up fails.

Exceptional Flow 2:

- Open the website.
- Click on register.
- “Password” field and “Password confirmation” field don’t match.
- Receives the Error “The two password fields don’t match”

6. **Use-Case Number:** US-006:

Use-Case Name: Apply for a job

Use-Case Description: Users are able to apply for a job using their registered account.

Primary Actor: Users.

Precondition: Open the website, Login using their username and password, click on “Browse job”, click on apply job on the desired one.

Post-condition: The user now can successfully apply for a job.

Basic flow:

- Open the website.
- Login using user’s credentials.
- Click on “Browse Job”.
- Click on “Apply now” on the desired job.
- Fill the required fileds (Name, Email, Address, Upload CV).
- Click on Apply now.
- Applying succeeds.

Exceptional Flow:

- Open the website.
- Login using user’s credentials.
- Click on “Browse Job”
- Click on “Apply now” on the desired job.
- Don’t fill one of the fields/Don’t upload the CV.
- Receive the Error “Please fill the empty fields”.

Exceptional Flow 2:

- Open the website
- Don’t login.
- Click on “Browse Job”.
- Click on “Apply now” on the desired job.
- Fill the required fileds (Name, Email, Address, Upload CV).
- Click on Apply now.
- Gets redirected to the login page.

7. **Use-Case Number:** US-007:

Use-Case Name: Post a job

Use-Case Description: Users are able to Post a job using their registered account.

Primary Actor: Users.

Precondition: Open the website, Login using their credentials, click on “Post a job”.

Post-condition: The user now is able to post a job.

Basic flow:

- Open the website
- Login using user’s credentials.
- Click on post a job.
- Fill the required fields (Title, Job type, Description, Vacancy, Salary, Experience, Category, Image)
- Click on Post now.
- Successfully post a job.

Exceptional Flow:

- Open the website
- Login using user’s credentials.
- Click on post a job.
- Don’t fill one of the required fields.
- Receive the Error “Please fill the empty fields.

Exceptional Flow 2:

- Open the website
- Don’t login.
- Click on post a job.
- Gets redirected to the login page.

8. **Use-Case Number:** US-008:

Use-Case Name: Post a job category.

Use-Case Description: Admins are able to post a new job category to expand their website.

Primary Actor: Admin.

Precondition: Open the website, click on login, login the website using an admin account, view the admin page.

Post-condition: Admins now are seeing the admin webpage containments.

Basic flow:

- Open the website.
- Login using Admin’s name and password.
- Open the admin page.
- Click on “Add’ next to the “Category” section under the Job’s tab.
- Type the category name.
- Click on save.
- Category is successfully added.

Exceptional Flow:

- Open the website
- Login using Admin’s name and password.
- Open the admin page
- Click on “Add’ next to the “Category” section under the Job’s tab.
- Don’t type the category name.
- Click on save.
- Receive the error “This field is required”.

Exceptional Flow 2:

- Open the website.
- Login using Admin's name and password.
- Enters a wrong username or password.
- Receive a login error and doesn't procced to the admin panel.

9. **Use-Case Number:** US-009:

Use-Case Name: View Database.

Use-Case Description: View the website's Database.

Primary Actor: Admins.

Precondition: Open the website, click on login, login the website using an admin account, view the admin page.

Post-condition: Admins now are seeing the admin webpage containments.

Basic flow:

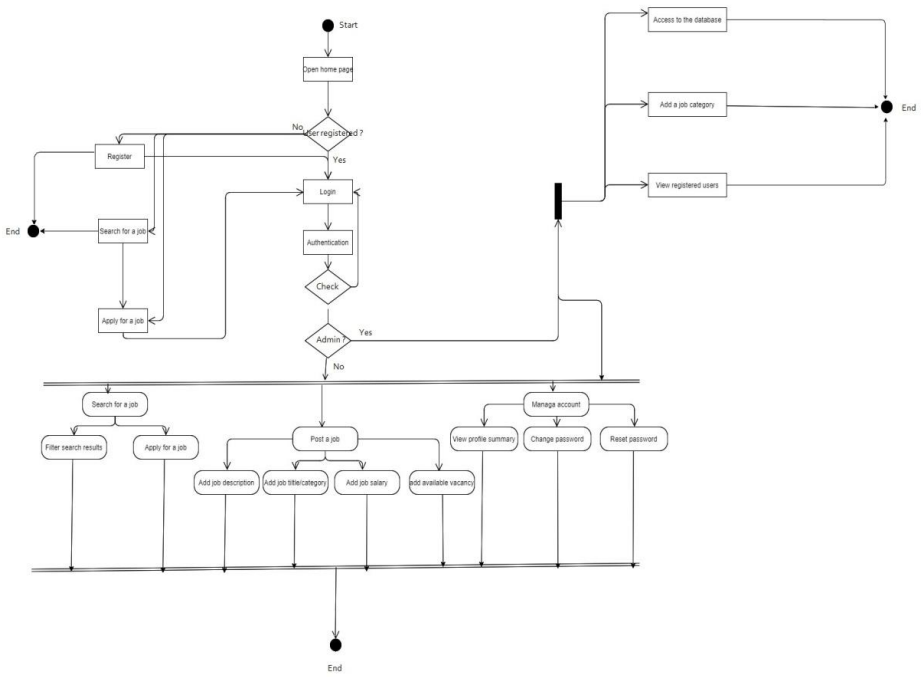
- Open the website.
- Login using admin's username and password.
- Open the admin page.
- Now you can see the Database.

Exceptional Flow:

- Open the website.
- Enter a wrong admin's username and password.
- Receive an error and doesn't procced.

3.2 Activity diagram

This section lists the activity diagram and describes the flow of activities in the system. The figure **(Figure 5)** below demonstrates the activity diagram for this Job board website, the flow is different for registered users, unregistered users and admins. The flow begins when the user opens the website. If they continue without logging on or registering, they get limited access to actions such as searching for a job or filtering them, once they decide to either apply for a job or post a job, they will be redirected to the login page which means that any further action taken requires them to be a registered user, Once the unregistered users register and login, they gain access to applying for a job, posting a job and viewing their profile summary if desired, from now on, they can act as a job seeker by just search for a job or apply for a one according to their need or they can act as an employer which gives them access to posting their job with their desired title, categorize it, add job description, add job salary and provide a number of available vacancies, they also gain access to viewing sent applications. If the user logged in with an admin account; in addition to having access to the normal user's privilege, they also gain access to viewing the database, viewing the registered accounts and adding a new job category if desired.



(Figure 5: Activity Diagram**).**

3.3 Class diagram

1. **User Authentication:** This class is utilized to get user information from the database for their authenticating. The class diagram in (Figure 6) shows the functions that are used in this class and the description of each class is listed below.

- **Authorization:** This function is used to verify what data each user has access to.
- **Register:** This function allows new users to create a new account and register for the website.
- **Check Password:** This function is used to check if the password entered correctly matches the one stored in the database.
- **Login:** This function is used to allow registered users to login using their credentials if it correctly matches ones stored in the database.

2. **Admin:**

- **Login/Logout:** Each of these functions is used to allow the admin to login/logout respectively using their credentials if it matches the ones stored in the database.
- **View Database:** This function allows admins to have full access view the database without compromising users' personal data.
- **Add/Remove/Edit users:** This function allows admins to add/remove user accounts or edit their data respectively.
- **Add Job:** This function allows admins to add more jobs to expand their website.
- **Add Job Category:** This function allows admins to add more job categories to make their website more diverse and to facilitate the searching process.

3. **Job Seeker:**

- **Login/Logout:** Each of these functions is used to allow the users who are registered as Job Seekers to login/logout respectively using their credentials if it correctly matches the ones stored in the database.
- **View Profile:** This function allows users who are registered as Job Seekers to view a summary of their profile containing information such as (Name, First Name, Last name, Email, Phone ...).
- **Apply Application:** This function allows users who are registered as Job Seekers to apply for jobs posted by Employers.

4. **Employer:**

- **Login/Logout:** Each of these functions is used to allow the users who are registered as Job Seekers to login/logout respectively using their credentials if it correctly matches the ones stored in the database.
- **View Profile:** This function allows users who are registered as Employers to view the profile which shows them a summary of their profile containing information such as (Name, First Name, Last name, Email, Phone ...).

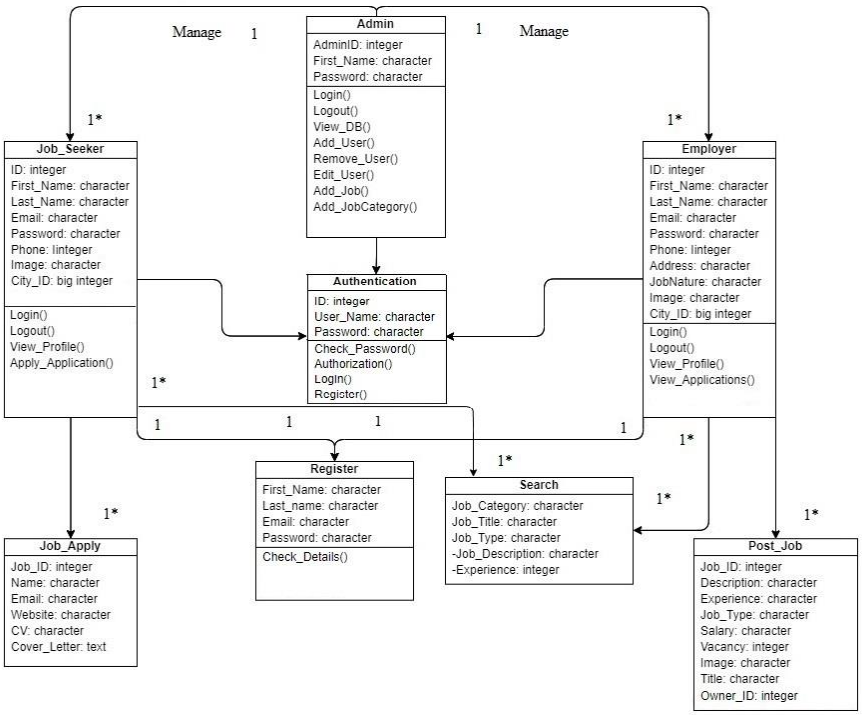
5. **Register:** This function allows new users to sign up for a new account, they can optionally sign up as either Employees or Job Seekers.

- **Check Details:** This function is used to checks if the provided username already exists in the database. If there is an already existing user with the same entered name the user is then prompted to select a different username to create an account.

6. **Search:** This function allows all users either registered or not to search for a job using filters such as (Category, Title contains, Job type, Experience).

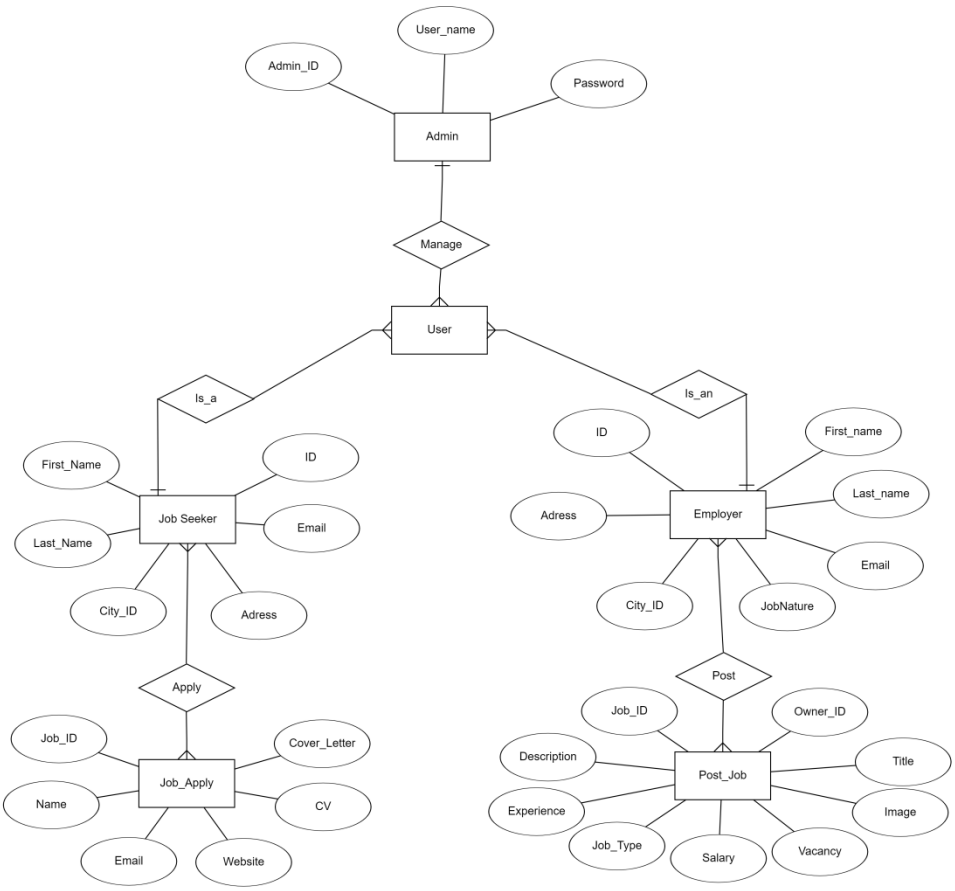
7. **Post Job:** This function allows users who are registered as Employers to post a job application containing its details such as (Job title, Job type, Description, Vacancy, Salary, Experience, Category).

8. **Job Apply:** This function allows users who are registered as Job Seekers to apply for jobs posted by Employers.



(Figure 6: Class Diagram**).**

3.4 Entity–relationship model



(Figure 7: Entity-relationship Diagram**).**

3.5 Job Board website Implementation

This section contains the implementation details for different packages and models of the Job Board website. It also contains code snippets and the whole current database.

The Database & indices (Table 01 to Table 07):

Database

Name	Type	Schema
accounts_city		CREATE TABLE "accounts_city" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "city_name" varchar(50) NOT NULL)
id	integer	"id" integer NOT NULL
city_name	varchar(50)	"city_name" varchar(50) NOT NULL
accounts_company_name		CREATE TABLE "accounts_company_name" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "companyName" varchar(60) NOT NULL)
id	integer	"id" integer NOT NULL
companyName	varchar(60)	"companyName" varchar(60) NOT NULL
accounts_conversationmessage		CREATE TABLE "accounts_conversationmessage" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "content" text NOT NULL, "created_at" datetime NOT NULL, "application_id" bigint NOT NULL REFERENCES "job_apllayjob" ("id") DEFERRABLE INITIALLY DEFERRED, "reciever_id" integer NOT NULL REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED, "sender_id" integer NOT NULL REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED)
id	integer	"id" integer NOT NULL
content	text	"content" text NOT NULL
created_at	datetime	"created_at" datetime NOT NULL
application_id	bigint	"application_id" bigint NOT NULL
reciever_id	integer	"reciever_id" integer NOT NULL
sender_id	integer	"sender_id" integer NOT NULL
accounts_highpriv		CREATE TABLE "accounts_highpriv" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "company_Name" varchar(60) NULL, "phone" varchar(15) NULL)
id	integer	"id" integer NOT NULL
company_Name	varchar(60)	"company_Name" varchar(60)
phone	varchar(15)	"phone" varchar(15)
accounts_namecomapny		CREATE TABLE "accounts_namecomapny" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "Company_Name" varchar(60) NOT NULL)
id	integer	"id" integer NOT NULL
Company_Name	varchar(60)	"Company_Name" varchar(60) NOT NULL
accounts_profile		CREATE TABLE "accounts_profile" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "phone" varchar(15) NULL, "image" varchar(100) NULL, "city_id" bigint NULL REFERENCES "accounts_city" ("id") DEFERRABLE INITIALLY DEFERRED, "user_id" integer NOT NULL UNIQUE REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED, "company_Name" varchar(60) NULL, "User_Type" varchar(30) NOT NULL)

(Table 01: City details, Company name, Profile details and conversation tables).

Name	Type	Schema
id	integer	"id" integer NOT NULL
phone	varchar(15)	"phone" varchar(15)
image	varchar(100)	"image" varchar(100)
city_id	bigint	"city_id" bigint
user_id	integer	"user_id" integer NOT NULL UNIQUE
company_Name	varchar(60)	"company_Name" varchar(60)
User_Type	varchar(30)	"User_Type" varchar(30) NOT NULL
auth_group		CREATE TABLE "auth_group" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "name" varchar(150) NOT NULL UNIQUE)
id	integer	"id" integer NOT NULL
name	varchar(150)	"name" varchar(150) NOT NULL UNIQUE
auth_group_permissions		CREATE TABLE "auth_group_permissions" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "group_id" integer NOT NULL REFERENCES "auth_group" ("id") DEFERRABLE INITIALLY DEFERRED, "permission_id" integer NOT NULL REFERENCES "auth_permission" ("id") DEFERRABLE INITIALLY DEFERRED)
id	integer	"id" integer NOT NULL
group_id	integer	"group_id" integer NOT NULL
permission_id	integer	"permission_id" integer NOT NULL
auth_permission		CREATE TABLE "auth_permission" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "content_type_id" integer NOT NULL REFERENCES "django_content_type" ("id") DEFERRABLE INITIALLY DEFERRED, "codename" varchar(100) NOT NULL, "name" varchar(255) NOT NULL)
id	integer	"id" integer NOT NULL
content_type_id	integer	"content_type_id" integer NOT NULL
codename	varchar(100)	"codename" varchar(100) NOT NULL
name	varchar(255)	"name" varchar(255) NOT NULL
auth_user		CREATE TABLE "auth_user" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "password" varchar(128) NOT NULL, "last_login" datetime NULL, "is_superuser" bool NOT NULL, "username" varchar(150) NOT NULL UNIQUE, "last_name" varchar(150) NOT NULL, "email" varchar(254) NOT NULL, "is_staff" bool NOT NULL, "is_active" bool NOT NULL, "date_joined" datetime NOT NULL, "first_name" varchar(150) NOT NULL)
id	integer	"id" integer NOT NULL
password	varchar(128)	"password" varchar(128) NOT NULL
last_login	datetime	"last_login" datetime
is_superuser	bool	"is_superuser" bool NOT NULL

(Table 02: Authorization & Authentication table).

Name	Type	Schema
username	varchar(150)	"username" varchar(150) NOT NULL UNIQUE
last_name	varchar(150)	"last_name" varchar(150) NOT NULL
email	varchar(254)	"email" varchar(254) NOT NULL
is_staff	bool	"is_staff" bool NOT NULL
is_active	bool	"is_active" bool NOT NULL
date_joined	datetime	"date_joined" datetime NOT NULL
first_name	varchar(150)	"first_name" varchar(150) NOT NULL
auth_user_groups		CREATE TABLE "auth_user_groups" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "user_id" integer NOT NULL REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED, "group_id" integer NOT NULL REFERENCES "auth_group" ("id") DEFERRABLE INITIALLY DEFERRED)
id	integer	"id" integer NOT NULL
user_id	integer	"user_id" integer NOT NULL
group_id	integer	"group_id" integer NOT NULL
auth_user_user_permissions		CREATE TABLE "auth_user_user_permissions" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "user_id" integer NOT NULL REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED, "permission_id" integer NOT NULL REFERENCES "auth_permission" ("id") DEFERRABLE INITIALLY DEFERRED)
id	integer	"id" integer NOT NULL
user_id	integer	"user_id" integer NOT NULL
permission_id	integer	"permission_id" integer NOT NULL
contact_contactus		CREATE TABLE "contact_contactus" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "place" varchar(150) NOT NULL, "phone" varchar(15) NOT NULL, "email" varchar(254) NOT NULL)
id	integer	"id" integer NOT NULL
place	varchar(150)	"place" varchar(150) NOT NULL
phone	varchar(15)	"phone" varchar(15) NOT NULL
email	varchar(254)	"email" varchar(254) NOT NULL
django_admin_log		CREATE TABLE "django_admin_log" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "action_time" datetime NOT NULL, "object_id" text NULL, "object_repr" varchar(200) NOT NULL, "change_message" text NOT NULL, "content_type_id" integer NULL REFERENCES "django_content_type" ("id") DEFERRABLE INITIALLY DEFERRED, "user_id" integer NOT NULL REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED, "action_flag" smallint unsigned NOT NULL CHECK ("action_flag" >= 0))

(Table 03: Authorized user permissions & Admin privileges).

Name	Type	Schema
action_time	datetime	"action_time" datetime NOT NULL
object_id	text	"object_id" text
object_repr	varchar(200)	"object_repr" varchar(200) NOT NULL
change_message	text	"change_message" text NOT NULL
content_type_id	integer	"content_type_id" integer
user_id	integer	"user_id" integer NOT NULL
action_flag	smallint unsigned	"action_flag" smallint unsigned NOT NULL CHECK("action_flag" >= 0)
django_content_type		CREATE TABLE "django_content_type" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "app_label" varchar(100) NOT NULL, "model" varchar(100) NOT NULL)
id	integer	"id" integer NOT NULL
app_label	varchar(100)	"app_label" varchar(100) NOT NULL
model	varchar(100)	"model" varchar(100) NOT NULL
django_migrations		CREATE TABLE "django_migrations" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "app" varchar(255) NOT NULL, "name" varchar(255) NOT NULL, "applied" datetime NOT NULL)
id	integer	"id" integer NOT NULL
app	varchar(255)	"app" varchar(255) NOT NULL
name	varchar(255)	"name" varchar(255) NOT NULL
applied	datetime	"applied" datetime NOT NULL
django_session		CREATE TABLE "django_session" ("session_key" varchar(40) NOT NULL PRIMARY KEY, "session_data" text NOT NULL, "expire_date" datetime NOT NULL)
session_key	varchar(40)	"session_key" varchar(40) NOT NULL
session_data	text	"session_data" text NOT NULL
expire_date	datetime	"expire_date" datetime NOT NULL
job_apllayjob		CREATE TABLE "job_apllayjob" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "name" varchar(50) NOT NULL, "email" varchar(100) NOT NULL, "cv" varchar(100) NOT NULL, "coverLetter" text NOT NULL, "job_id" bigint NOT NULL REFERENCES "job_job" ("id") DEFERRABLE INITIALLY DEFERRED, "portfolio_link" varchar(200) NULL, "created_by_id" integer NOT NULL REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED, "created_at" datetime NOT NULL)
id	integer	"id" integer NOT NULL
name	varchar(50)	"name" varchar(50) NOT NULL
email	varchar(100)	"email" varchar(100) NOT NULL

(Table 04: Session, Job applying & modifying models tables).

Name	Type	Schema
cv	varchar(100)	"cv" varchar(100) NOT NULL
coverLetter	text	"coverLetter" text NOT NULL
job_id	bigint	"job_id" bigint NOT NULL
portfolio_link	varchar(200)	"portfolio_link" varchar(200)
created_by_id	integer	"created_by_id" integer NOT NULL
created_at	datetime	"created_at" datetime NOT NULL
job_category		CREATE TABLE "job_category" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "name" varchar(30) NOT NULL)
id	integer	"id" integer NOT NULL
name	varchar(30)	"name" varchar(30) NOT NULL
job_job		CREATE TABLE "job_job" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "title" varchar(40) NOT NULL, "description" text NOT NULL, "experience" integer NOT NULL, "job_type" varchar(15) NOT NULL, "published_at" datetime NOT NULL, "salary" integer NOT NULL, "vacancy" integer NOT NULL, "category_id" bigint NOT NULL REFERENCES "job_category" ("id") DEFERRABLE INITIALLY DEFERRED, "slug" varchar(50) NULL, "owner_id" integer NOT NULL REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED, "location" varchar(30) NOT NULL, "img" varchar(100) NULL)
id	integer	"id" integer NOT NULL
title	varchar(40)	"title" varchar(40) NOT NULL
description	text	"description" text NOT NULL
experience	integer	"experience" integer NOT NULL
job_type	varchar(15)	"job_type" varchar(15) NOT NULL
published_at	datetime	"published_at" datetime NOT NULL
salary	integer	"salary" integer NOT NULL
vacancy	integer	"vacancy" integer NOT NULL
category_id	bigint	"category_id" bigint NOT NULL
slug	varchar(50)	"slug" varchar(50)
owner_id	integer	"owner_id" integer NOT NULL
location	varchar(30)	"location" varchar(30) NOT NULL
img	varchar(100)	"img" varchar(100)
nnotification_notification		CREATE TABLE "nnotification_notification" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "notification_type" varchar(20) NOT NULL, "is_read" bool NOT NULL, "extra_id" integer NULL, "created_at" datetime NOT NULL, "created_by_id" integer NOT NULL REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED, "to user id" integer NOT NULL REFERENCES "auth user"
		("id") DEFERRABLE INITIALLY DEFERRED)
id	integer	"id" integer NOT NULL
notification_type	varchar(20)	"notification_type" varchar(20) NOT NULL
is_read	bool	"is_read" bool NOT NULL
extra_id	integer	"extra_id" integer
created_at	datetime	"created_at" datetime NOT NULL
created_by_id	integer	"created_by_id" integer NOT NULL
to_user_id	integer	"to_user_id" integer NOT NULL
sqlite_sequence		CREATE TABLE sqlite_sequence(name,seq)
name		"name"
seq		"seq"

(Table 05: Job details, Notification & Auto increment generator table).

Indices

Name	Type	Schema
accounts_conversationmessage_application_id_19e9d341		CREATE INDEX "accounts_conversationmessage_application_id_19e9d341" ON "accounts_conversationmessage" ("application_id")
application_id		"application_id"
accounts_conversationmessage_reciever_id_139f1bc0		CREATE INDEX "accounts_conversationmessage_reciever_id_139f1bc0" ON "accounts_conversationmessage" ("reciever_id")
reciever_id		"reciever_id"
accounts_conversationmessage_sender_id_601787b7		CREATE INDEX "accounts_conversationmessage_sender_id_601787b7" ON "accounts_conversationmessage" ("sender_id")
sender_id		"sender_id"
accounts_profile_city_id_267b3d7f		CREATE INDEX "accounts_profile_city_id_267b3d7f" ON "accounts_profile" ("city_id")
city_id		"city_id"
auth_group_permissions_group_id_b120cbf9		CREATE INDEX "auth_group_permissions_group_id_b120cbf9" ON "auth_group_permissions" ("group_id")
group_id		"group_id"
auth_group_permissions_group_id_permission_id_0cd325b0_uniq		CREATE UNIQUE INDEX "auth_group_permissions_group_id_permission_id_0cd325b0_uniq" ON
		"auth_group_permissions" ("group_id", "permission_id")
group_id		"group_id"
permission_id		"permission_id"
auth_group_permissions_permission_id_84c5c92e		CREATE INDEX "auth_group_permissions_permission_id_84c5c92e" ON "auth_group_permissions" ("permission_id")
permission_id		"permission_id"
auth_permission_content_type_id_2f476e4b		CREATE INDEX "auth_permission_content_type_id_2f476e4b" ON "auth_permission" ("content_type_id")
content_type_id		"content_type_id"
auth_permission_content_type_id_codename_01ab375a_uniq		CREATE UNIQUE INDEX "auth_permission_content_type_id_codename_01ab375a_uniq" ON "auth_permission" ("content_type_id", "codename")
content_type_id		"content_type_id"
codename		"codename"
auth_user_groups_group_id_97559544		CREATE INDEX "auth_user_groups_group_id_97559544" ON "auth_user_groups" ("group_id")
group_id		"group_id"
auth_user_groups_user_id_6a12ed8b		CREATE INDEX "auth_user_groups_user_id_6a12ed8b" ON "auth_user_groups" ("user_id")
user_id		"user_id"
auth_user_groups_user_id_group_id_94350c0c_uniq		CREATE UNIQUE INDEX "auth_user_groups_user_id_group_id_94350c0c_uniq" ON "auth_user_groups" ("user_id", "group_id")
user_id		"user_id"
group_id		"group_id"
auth_user_user_permissions_permission_id_1fbb5f2c		CREATE INDEX "auth_user_user_permissions_permission_id_1fbb5f2c" ON "auth_user_user_permissions" ("permission_id")
permission_id		"permission_id"
auth_user_user_permissions_user_id_a95ead1b		CREATE INDEX "auth_user_user_permissions_user_id_a95ead1b" ON "auth_user_user_permissions" ("user_id")
user_id		"user_id"
auth_user_user_permissions_user_i		CREATE UNIQUE INDEX

(Table 06: Database Index 01).

Name	Type	Schema
d_permission_id_14a6b632_uniq		"auth_user_user_permissions_user_id_permission_id_14a6b632_uniq" ON "auth_user_user_permissions" ("user_id", "permission_id")
user_id		"user_id"
permission_id		"permission_id"
django_admin_log_content_type_id_c4bce8eb		CREATE INDEX "django_admin_log_content_type_id_c4bce8eb" ON "django_admin_log" ("content_type_id")
content_type_id		"content_type_id"
django_admin_log_user_id_c564eba6		CREATE INDEX "django_admin_log_user_id_c564eba6" ON "django_admin_log" ("user_id")
user_id		"user_id"
django_content_type_app_label_model_76bd3d3b_uniq		CREATE UNIQUE INDEX "django_content_type_app_label_model_76bd3d3b_uniq" ON "django_content_type" ("app_label", "model")
app_label		"app_label"
model		"model"
django_session_expire_date_a5c62663		CREATE INDEX "django_session_expire_date_a5c62663" ON "django_session" ("expire_date")
expire_date		"expire_date"
job_apllayjob_created_by_id_aa1dcfc4		CREATE INDEX "job_apllayjob_created_by_id_aa1dcfc4" ON "job_apllayjob" ("created_by_id")
created_by_id		"created_by_id"
job_apllayjob_job_id_214bcf33		CREATE INDEX "job_apllayjob_job_id_214bcf33" ON "job_apllayjob" ("job_id")
job_id		"job_id"
job_job_category_id_555b6898		CREATE INDEX "job_job_category_id_555b6898" ON "job_job" ("category_id")
category_id		"category_id"
job_job_owner_id_d5c16855		CREATE INDEX "job_job_owner_id_d5c16855" ON "job_job" ("owner_id")
owner_id		"owner_id"
job_job_slug_1de62f0e		CREATE INDEX "job_job_slug_1de62f0e" ON "job_job" ("slug")
slug		"slug"
nnotification_notification_created_by_id_e1746cd9		CREATE INDEX "nnotification_notification_created_by_id_e1746cd9" ON "nnotification_notification" ("created_by_id")
created_by_id		"created_by_id"
nnotification_notification_to_user_id_8f49c6c6		CREATE INDEX "nnotification_notification_to_user_id_8f49c6c6" ON "nnotification_notification" ("to_user_id")
to_user_id		"to_user_id"

(Table 07: Database Index 02).

Code snippets:

Backend:

This job model **(Figure 08)** connects the required data to the database according to their type, similar approach is taken for every other model.

```python
from django.db import models
from django.db.models.deletion import CASCADE
from django.db.models.fields import EmailField, TextField
from django.utils.text import slugify
from django.contrib.auth.models import User
from ckeditor.fields import RichTextField
# Create your models here.

'''
What is django model fiels:
    - html widget
    - validations
    - DB Size
'''
JOB_TYPE = (
    ('Full Time','Full Time'),
    ('Part Time','Part Time'),
)

# Customize Image Upload
def image_upload(instance,filename):
    imagename , extension = filename.split('.')
    return "jobs/%s/%s.%s"%(instance.id,instance.id,extension)

class Job(models.Model):            #table
    owner = models.ForeignKey(User,related_name='job_owner',on_delete=CASCADE)
    title = models.CharField(max_length=40)
    #location =
    job_type = models.CharField(max_length=15,choices=JOB_TYPE)
    description = RichTextField()
    #description = models.TextField(max_length=1000)
    published_at = models.DateTimeField(auto_now=True)
    vacancy = models.IntegerField(default=1)
    salary = models.IntegerField(default=0)
    experience = models.IntegerField(default=0)
    category = models.ForeignKey('Category',on_delete=CASCADE)
    img = models.ImageField(upload_to=image_upload)
    slug = models.SlugField(blank=True,null=True)


    #Slug Function
    def save(self,*args, **kwargs):
        ##logic
        self.slug = slugify(self.title)
        super(Job,self).save(*args,**kwargs)

    def __str__(self):
        return self.title

class Category(models.Model):
    name = models.CharField(max_length=30)

    def __str__(self):
        return self.name


class ApllayJob(models.Model):
    job = models.ForeignKey(Job,related_name='ApllayJob',on_delete=CASCADE)
    name = models.CharField(max_length=50)
    email = EmailField(max_length=100)
    website = models.URLField(blank=True,null=True)
    cv = models.FileField(upload_to='apply_job/')
    coverLetter= TextField(max_length=500)
```

(Figure 08: Job model**).**

Language: Django template language

This snippet **(Figure 09)** contains all the main or parent URLS:

```
"""project URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/3.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path , include
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('accounts/', include('django.contrib.auth.urls')),
    path('accounts/',include('accounts.urls',namespace='accounts')),
    path('admin/', admin.site.urls),
    path('',include('job.urls',namespace='jobs')),
    path('contact-us/',include('contact.urls',namespace='contact')),
    path('api-auth/', include('rest_framework.urls')),
    path('ckeditor/',include('ckeditor_uploader.urls')),
]
urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

(Figure 09: Parent URLs**).**

Language: Django template language

This snippet **(Figure 10)** shows the URL “accounts” URL’s child URLS.

```
from django.urls import include,path
from . import views


app_name = 'accounts'

urlpatterns = [
    path('signup',views.signup,name='register'),
    path('profile/',views.profile,name='profile'),
    path('profile/edit',views.profile_edit,name='profile_edit'),
]
```

(Figure 10: Account URL’s Child URLs**).**

Language: Django template language

This code snippet **(Figure 11)** shows the “job” URL’s child URLs.

```python
from django.urls import include,path
from . import views
from . import api

app_name = 'job'

urlpatterns = [
    path('',views.index,name='index'),
    path('jobs/',views.job_list,name='job_list'),
    path('jobs/add',views.add_job,name='add_job'),
    path('jobs/add/done/',views.done_job,name='done_job'),
    path('<str:slug>',views.job_details,name='job_detail'),


    #api urls
    path('api/jobs',api.job_list_api,name='job_list _api'),
    path('api/jobs/<int:id>',api.job_detail_api,name='job_detail_api'),

    # Class Based Views
    path('api/v2/jobs',api.JobListApi.as_view(),name='job_detail_api'),
    path('api/v2/jobs/<int:id>',api.JobDetails.as_view(),name='job_detail_api'),

]
```

(Figure 11: Job URL’s child URLs**).**

Similar Approach is taken for each parent URL.

These views **(Figure 12)** and **(Figure 13)** connect the front with the back end:

```python
from django.contrib.auth import authenticate, login
from django.http.response import HttpResponseRedirect
from django.shortcuts import redirect, render
from django.contrib.auth.models import User, auth
from .forms import SignupForm,ProfileForm,UserForm
from .models import Profile
from django.urls import reverse
# Create your views here.

def signup(request):
    if request.method == "POST":
        form = SignupForm(request.POST)
        if form.is_valid():
            form.save()
            username = form.cleaned_data['username']
            password = form.cleaned_data['password1']
            user = authenticate(username=username,password=password) # if user and email is not none ==> Create new account (Do Login)
            login(request,user)
            return redirect(reverse('accounts:profile'))
    else:
        form = SignupForm()
    return render(request,'registration/signup.html',{'form':form})


def profile(request):
    profile = Profile.objects.get(user=request.user) # هنا بجيب الهوزر الي عامل لوجن
    return render(request,'accounts/profile.html',{'profile':profile})


def profile_edit(request):
    profile_edit = Profile.objects.get(user=request.user) # هنا بجيب الهوزر الي عامل لوجن
    if request.method == 'POST':
        userform = UserForm(request.POST,instance=request.user)
        profileform = ProfileForm(request.POST,request.FILES,instance=profile_edit)
        if userform.is_valid() and profileform.is_valid():
            userform.save()
            myprofile= profileform.save(commit=False) ## commit=false ===> to Get user And Edit Data
            myprofile.save()
            return redirect(reverse('accounts:profile'))
    else:
        userform = UserForm(instance=request.user) # logged user Now
        profileform = ProfileForm(instance=profile_edit) # Data Belong to USer Logged Now
    context={'userform':userform,
             'profileform':profileform,
             }
    return render(request,'accounts/profile_edit.html',context)
```

(Figure 12: View (1) that connect the back end to the front end).

Language: Django template language

```python
from django.db.models.query import QuerySet
from django.http import request
from django.shortcuts import render,redirect
from .models import Job
from django.core.paginator import Paginator
from .form import ApplyForm,add_job_form
from django.urls import reverse
from django.contrib.auth.decorators import login_required
from .filters import JobFilter
# Create your views here.


## Home Page
def index(request):
    job_list = Job.objects.all()
    context={'jobs':job_list}
    return render(request,'job/index.html',context)


## All Jobs
def job_list(request):
    job_list = Job.objects.all()



    ##Filter
    myfilter = JobFilter(request.GET,queryset=job_list)
    job_list = myfilter.qs

    # Adding paginator
    paginator = Paginator(job_list, 3) # Show 1 contacts per page.
    page_number = request.GET.get('page')
    page_obj = paginator.get_page(page_number)

    context = {
        'jobs' : page_obj,       # jobs in Template Name
        'myfilter':myfilter,
    }

    return render(request,'job/job_list.html',context)

## Jobs Details
def job_details(request, slug):
    job_details = Job.objects.get(slug=slug)
    # Apliy Form
    if request.method=='POST':
        form = ApplyForm(request.POST, request.FILES) # request Files --> CV
        if form.is_valid():
            myform = form.save(commit=False) #Don't Save in DB
            myform.job = job_details
            myform.save() #Save in DB
            return redirect(reverse('jobs:done_job'))

    else:
        form = ApplyForm()


    context={
        'job': job_details,
        'form': form
    }
    return render(request,'job/job_details.html',context)

#Done Sending Request Job
@login_required
def done_job(request):
    return render(request,'job/done_job.html')


# Add Job
@login_required
def add_job(request):
    if request.method=='POST':
        form = add_job_form(request.POST,request.FILES)
        if form.is_valid():
            myform= form.save(commit=False)
            myform.owner = request.user
            myform.save()
            return redirect(reverse('jobs:job_list'))
    else:
        form = add_job_form()

    context={
        'form': form,
    }

    return render(request,'job/add_job.html',context)
```

(Figure 13: View (2) that connect the back end to the front end).

Similar approach is taken for every other view.

Language: Django

Create (index , List All Jobs, Job details and description, Apply for a job) Functions

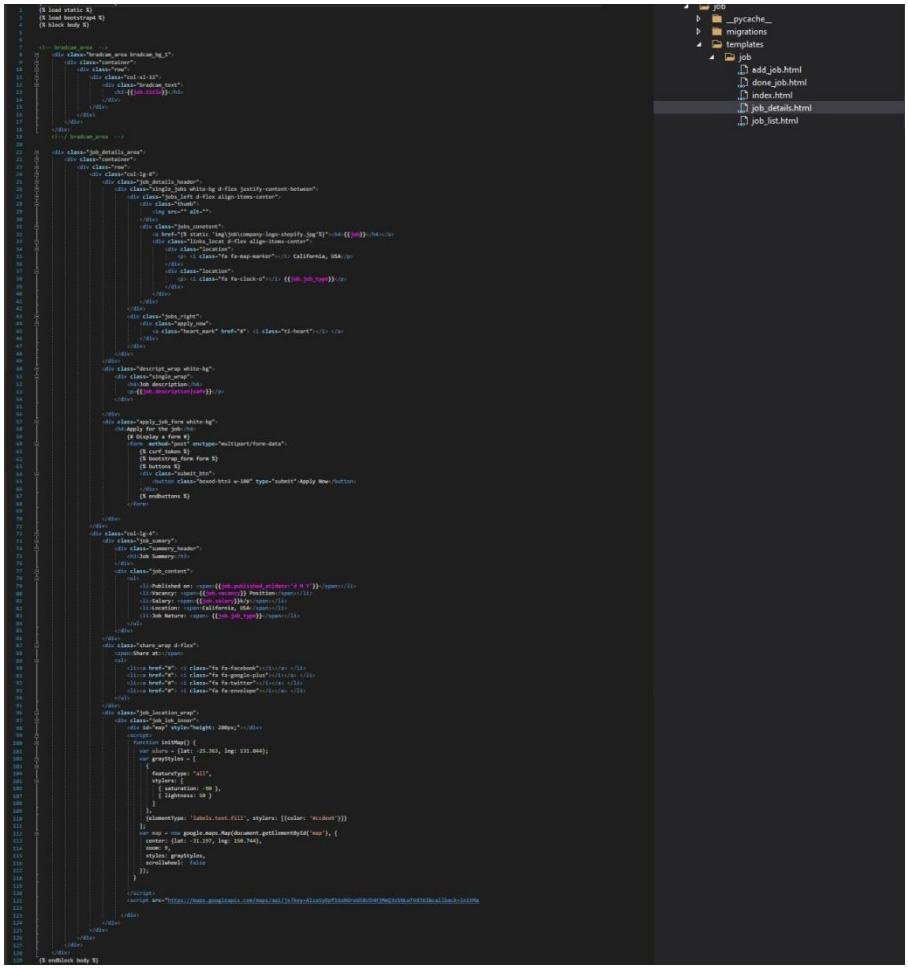
Frontend:

Base HTML **(Figure 14)** file that most of the HTML files using inheritance can inherit from:

```html
{% load static %}
<!doctype html>
<html class="no-js" lang="zxx">

<head>
    <meta charset="utf-8">
    <meta http-equiv="x-ua-compatible" content="ie=edge">
    <title>Job Board</title>
    <meta name="description" content="">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- <link rel="manifest" href="site.webmanifest"> -->
    <link rel="shortcut icon" type="image/x-icon" href="{% static 'img/Favicon.png' %}">
    <!-- Place favicon.ico in the root directory -->

    <!-- CSS here -->
    <link rel="stylesheet" href="{% static 'css/bootstrap.min.css' %}">
    <link rel="stylesheet" href="{% static 'css/owl.carousel.min.css' %}">
    <link rel="stylesheet" href="{% static 'css/magnific-popup.css' %}">
    <link rel="stylesheet" href="{% static 'css/font-awesome.min.css' %}">
    <link rel="stylesheet" href="{% static 'css/themify-icons.css' %}">
    <link rel="stylesheet" href="{% static 'css/nice-select.css' %}">
    <link rel="stylesheet" href="{% static 'css/flaticon.css' %}">
    <link rel="stylesheet" href="{% static 'css/jquery-ui.css' %}">
    <link rel="stylesheet" href="{% static 'css/gijgo.css' %}">
    <link rel="stylesheet" href="{% static 'css/animate.min.css' %}">
    <link rel="stylesheet" href="{% static 'css/slicknav.css' %}">

    <link rel="stylesheet" href="{% static 'css/style.css' %}">
    <!-- <link rel="stylesheet" href="css/responsive.css"> -->
</head>

<body>
    <!--[if lte IE 9]>
            <p class="browserupgrade">You are using an <strong>outdated</strong> browser. Please <a href="https://browsehappy.com/">upgrade your browser</a> to improve your experience and security.</p>
        <![endif]-->

    <!-- header-start -->
    <header>
        <div class="header-area ">
            <div id="sticky-header" class="main-header-area">
                <div class="container-fluid ">
                    <div class="header_bottom_border">
                        <div class="row align-items-center">
                            <div class="col-xl-3 col-lg-2">
                                <div class="logo">
                                    <a href="{% url 'jobs:index' %}">
                                        <img src="{% static 'img/logo.png' %}" alt="">
                                    </a>
                                </div>
                            </div>
                            <div class="col-xl-6 col-lg-7">
                                <div class="main-menu  d-none d-lg-block">
                                    <nav>
                                        <ul id="navigation">
                                            <li><a href="{% url 'jobs:index' %}">Home</a></li>
                                            <li><a href="{% url 'jobs:job_list' %}">Browse Job</a></li>
                                            <li><a href="#">pages <i class="ti-angle-down"></i></a>
                                                <ul class="submenu">
                                                    <li><a href="candidate.html">Candidates </a></li>
                                                    <li><a href="job_details.html">job details </a></li>
                                                    <li><a href="elements.html">elements</a></li>
                                                </ul>
                                            </li>
                                            <li><a href="{% url 'contact:contact' %}">Contact</a></li>
                                        </ul>
                                    </nav>
                                </div>
                            </div>
                            <div class="col-xl-3 col-lg-3 d-none d-lg-block">
                                <div class="Appointment">

                                    {% if request.user.is_authenticated %}
                                    <div class="phone_num d-none d-xl-block">
                                        <a href="{% url 'accounts:profile' %}">Profile</a>
                                    </div>

                                    <div class="phone_num d-none d-xl-block">
                                        <a href="{% url 'logout' %}">Log Out</a>
                                    </div>

                                    {% else %}
                                        <div class="phone_num d-none d-xl-block">
                                            <a href="{% url 'login' %}">Log in</a>
                                        </div>
                                        <div class="phone_num d-none d-xl-block">
                                            <a href="{% url 'accounts:register' %}">  Register </a>
                                        </div>
                                    {% endif %}

                                    <div class="d-none d-lg-block">
```

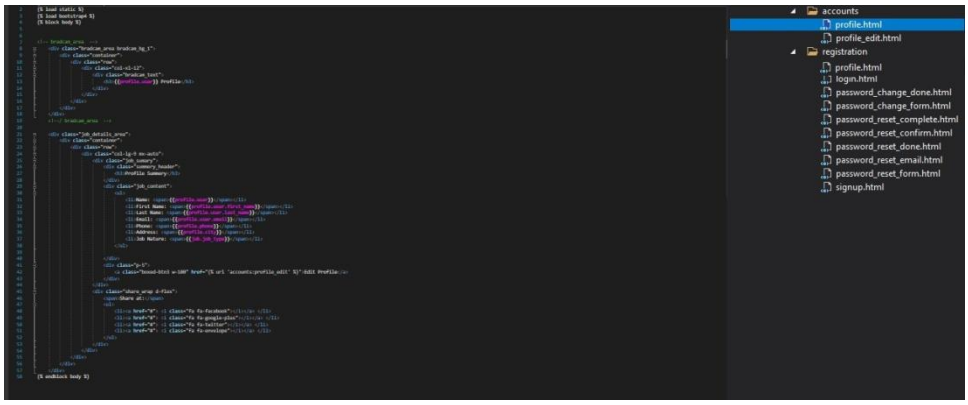
(Figure 14: Base HTML file**).**

Snippet from the job details HTML file **(Figure 15)** that shows the job details when searching for it such as job title, job description, job type, vacancies, job category, job salary, required experience:



(Figure 15: Job Details HTML file**).**

Account HTML **(Figure 16)** file that inherits from the base HTML, it contains front view details about the account such as singing up, logging, password change, password reset ... etc (these snippets **(Figure 16) & (Figure 17)** are taken from just 2 files just for the purpose of showcasing)



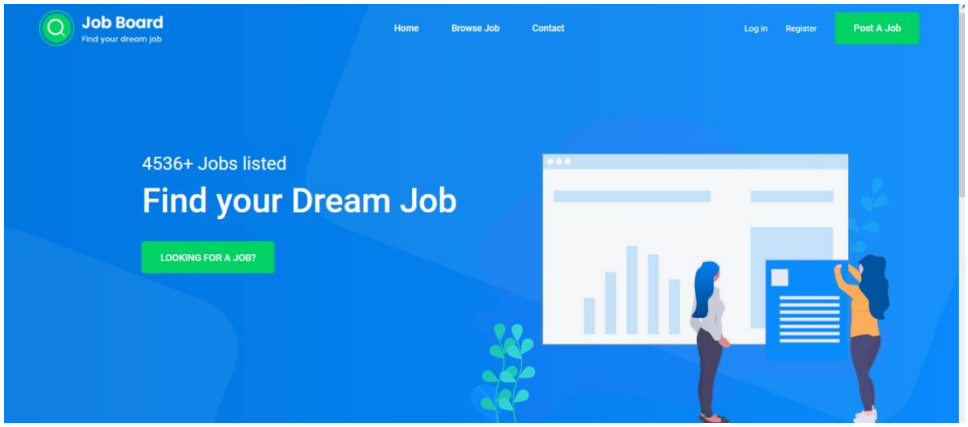
(Figure 16: Account HTML file**).**

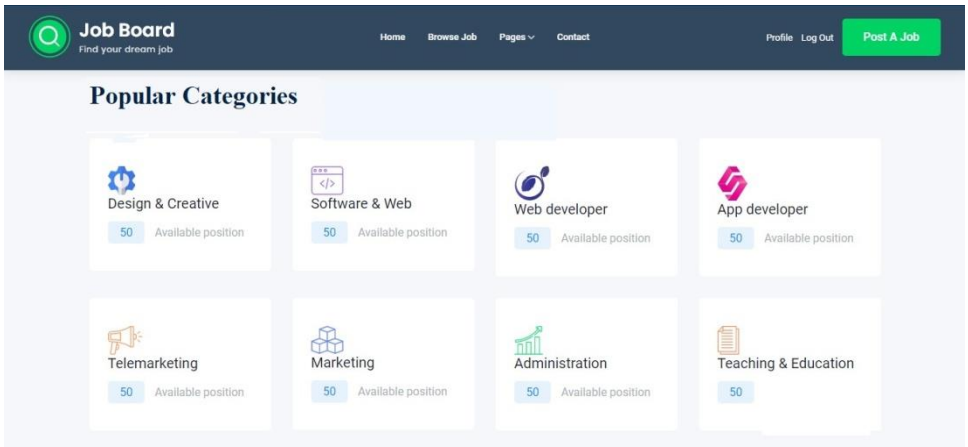
(Figure 17: Registration HTML file**).**

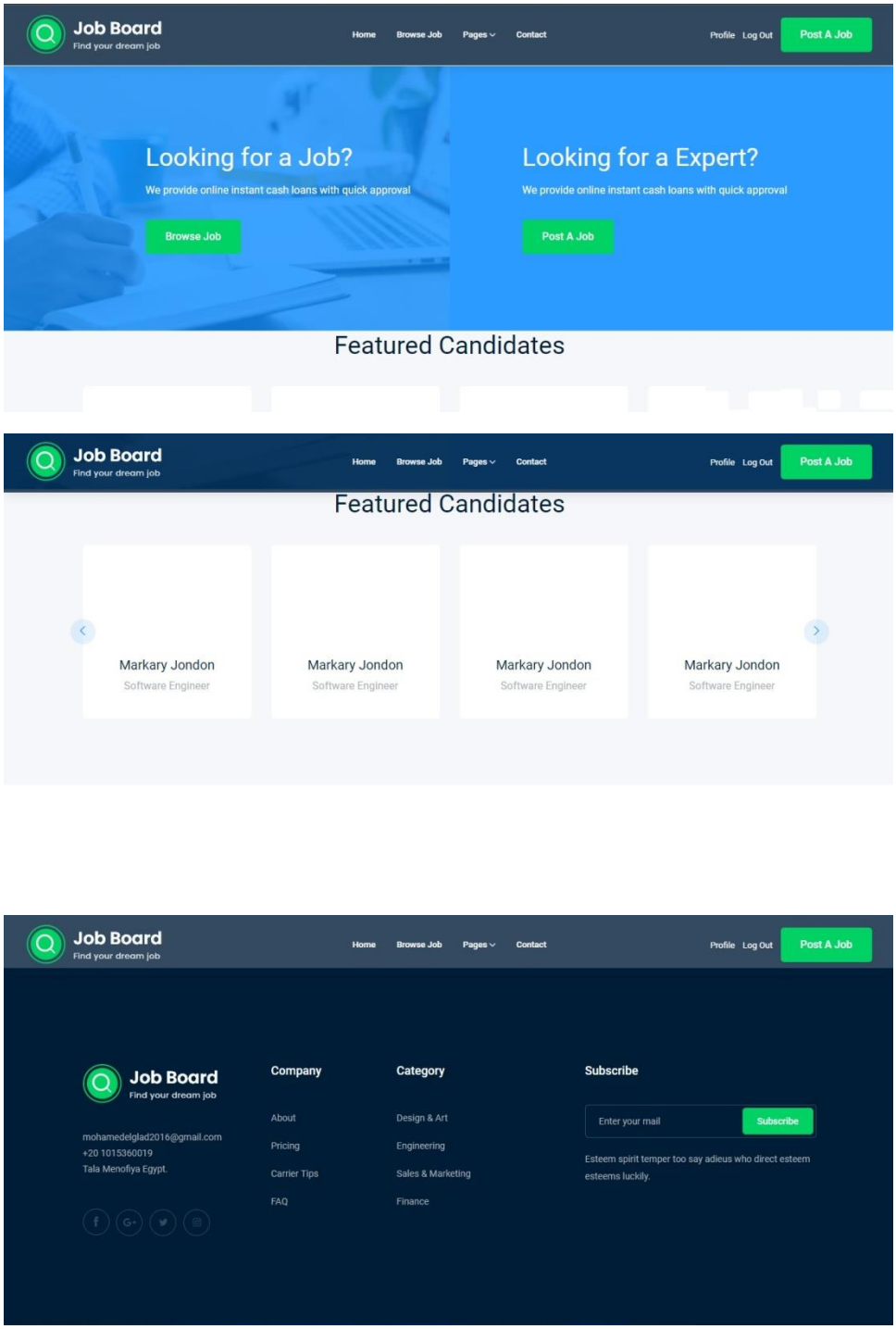
3.6 Job Board Website interface

This section describes the different interfaces for the Job Board website. It contains a detailed description about each interface along with a screen shot of the interface.

- **Home page:** The home page of the application **(Figure 18)** is common to all the system users/administrators. This interface is available through the website, it's first seen once you open the website.

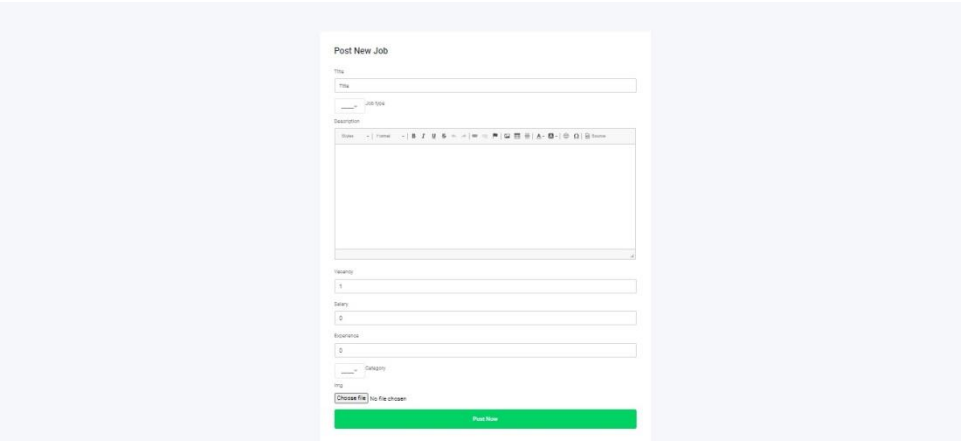






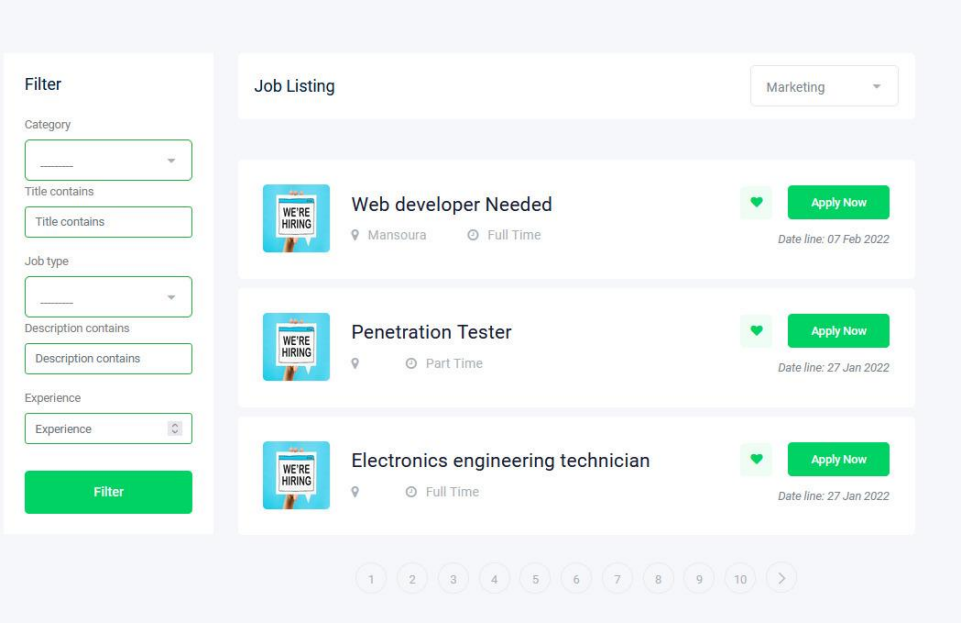
(Figure 18: Screenshot of the home page**).**

- **Post a job page:** this interface allows registered users to act as an employer and post a job **(Figure 19).**



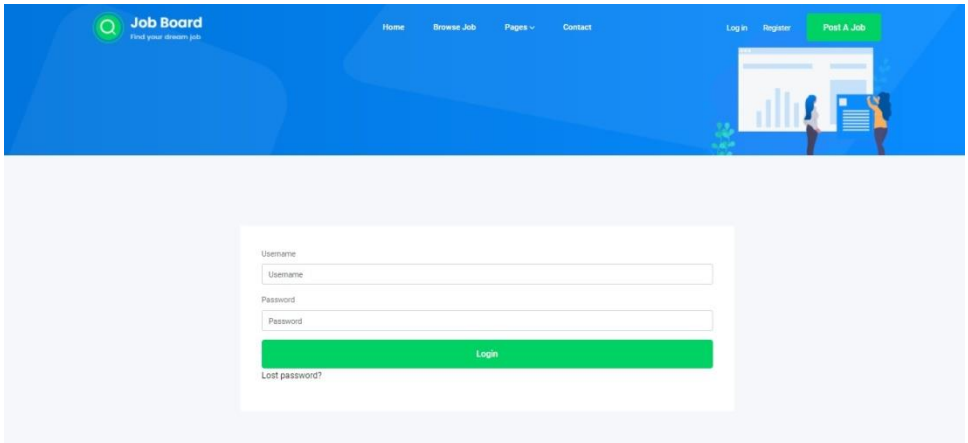
(Figure 19: Screenshot of posting a job interface**).**

- **Browse a job page:** This interface allows all users (even unregistered ones) to browse the available jobs. **(Figure 20)**



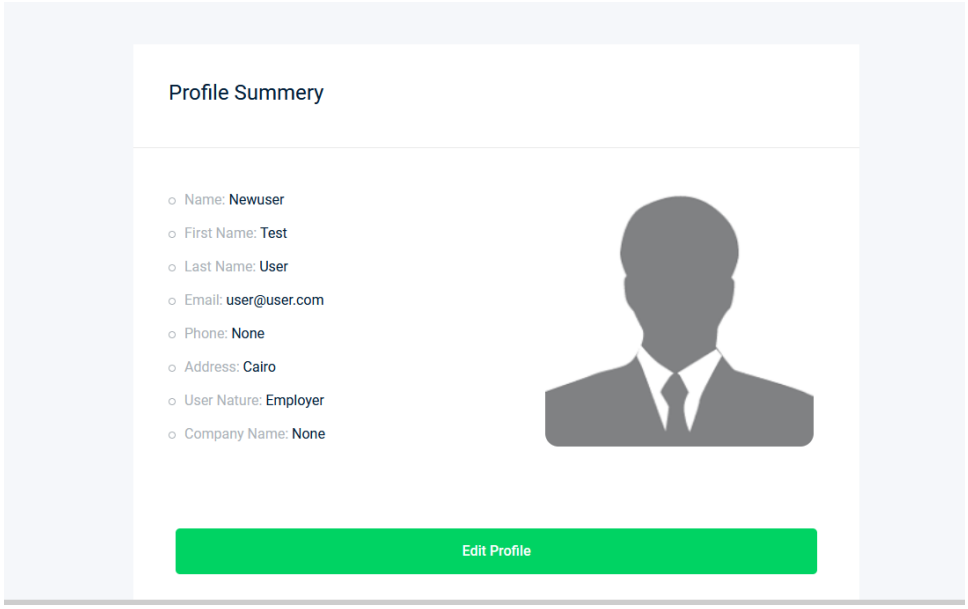
(Figure 20: Screenshot of searching for a job with filtering categories**).**

- **Login page:** This interface allows both users and admin to login their account using their previously registered credentials. **(Figure 21).**



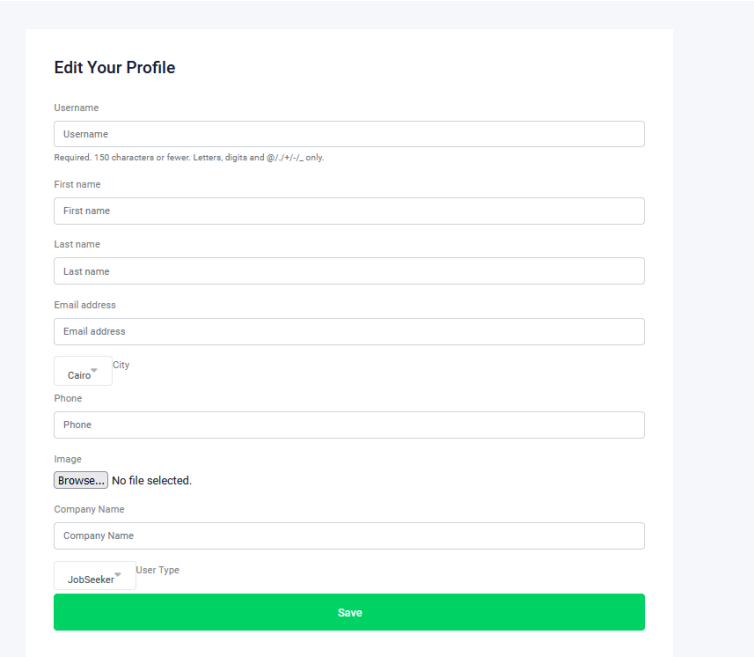
(Figure 21: Screenshot of the login page**).**

- **Profile summary:** This provides a summary for the profile of registered users or admins. **(Figure 22).**



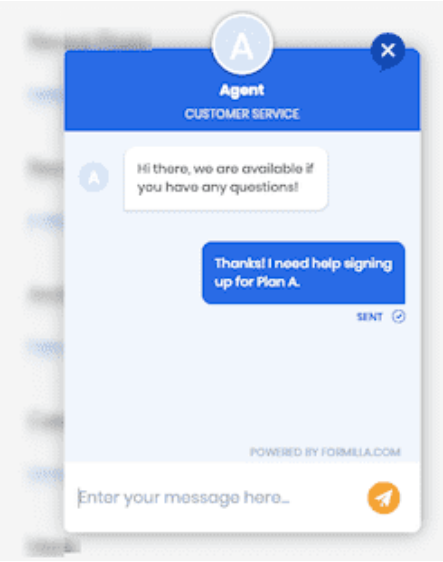
(Figure 22: Screenshot of the profile summary page**).**

- **Edit Profile:** This allows registered users to modify their profile data such as (Username, First name, Last name, Email, City, Phone, Image). (**Figure 23).**



(Figure 23: Screenshot of the edit profile page**).**

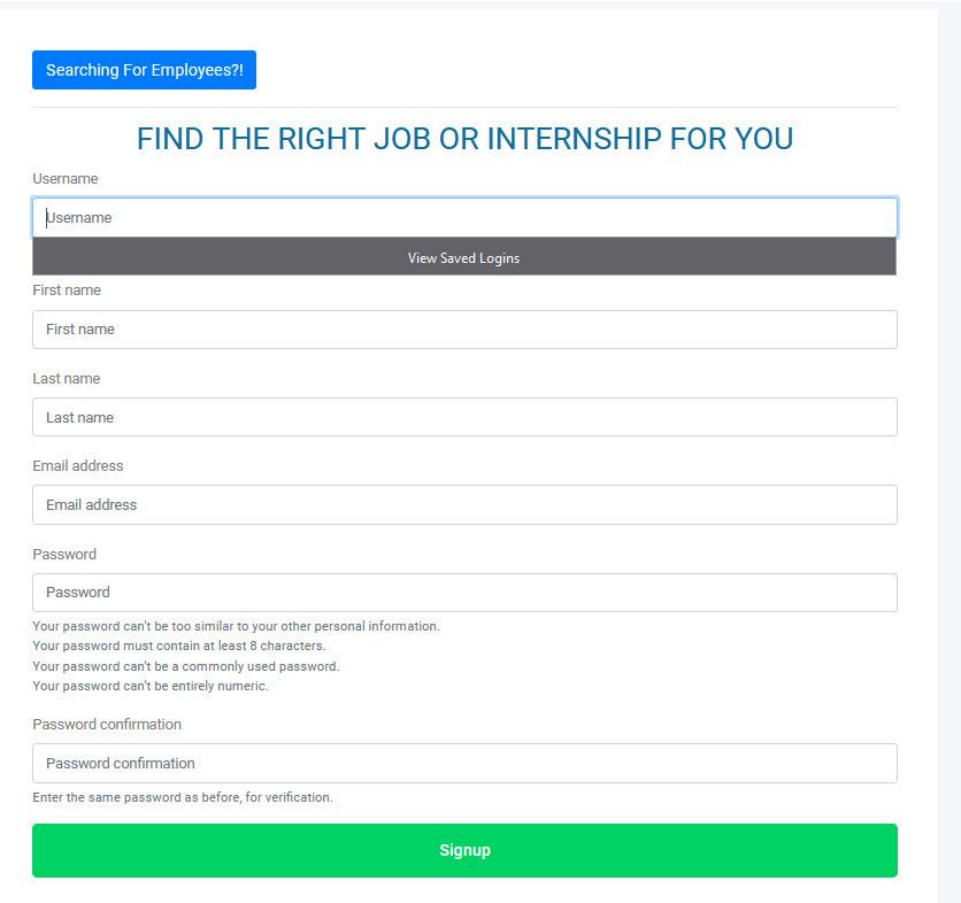
- **Integrated Chat:** The Job Board website has a built-in web chat that facilitates the compunction between job seekers and employees.**(Figure 24).**



(Figure 24: Screenshot of the integrated chat**).**

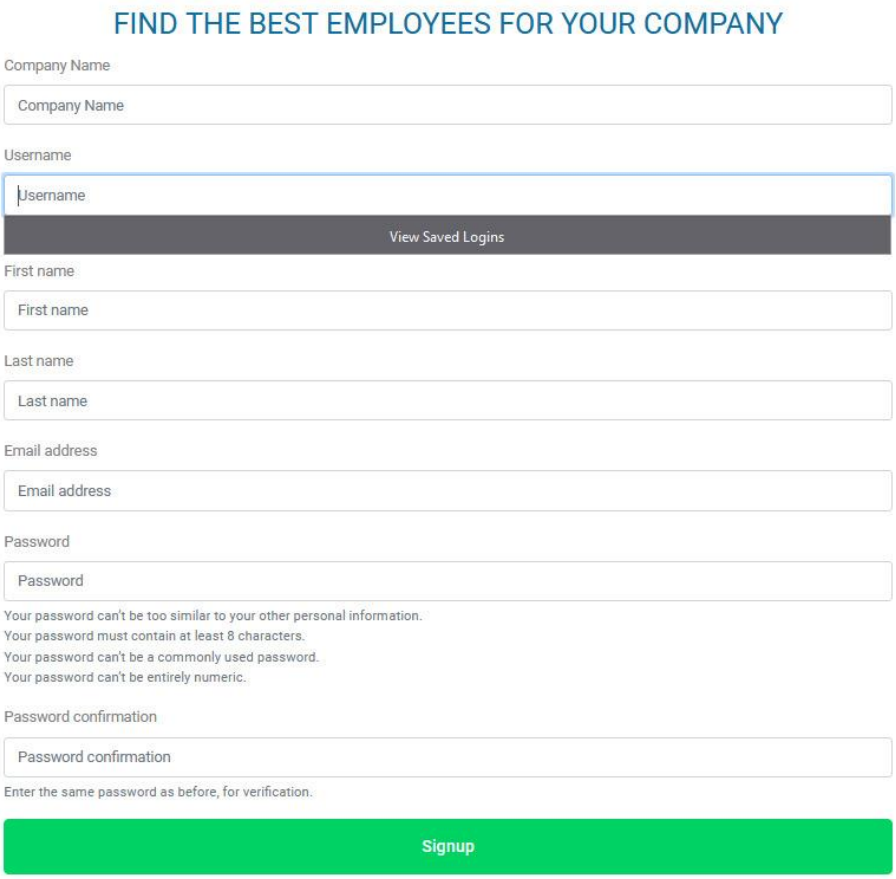
- **Register page:**

1. **Job Seeker:** This allows new users to register for an account as job seekers if they desire to take further action using the website as job seekers such as applying for a job. **(Figure 25).**



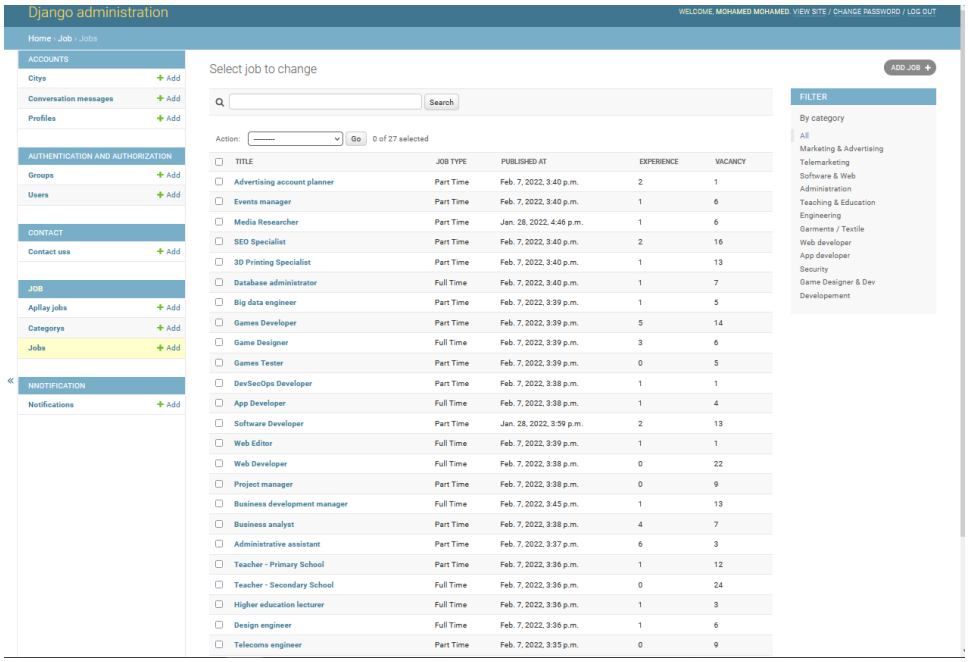
(Figure 25: Job Seekers register page**).**

2. **Employer:** This allows new users to register for an account as employers if they desire to take further action using the website as employers such as posting a job to expand their business and look for more employees. **(Figure 26).**



(Figure 26: Employer register page**).**

- **Admin panel:** This is an only admin exclusive web page; it allows admins to view the database or alter it. **(Figure 27).**



(Figure 27: Screenshot of the admin panel**).**

CHAPTER 4
PROGRAMMING LANGUAGES & SOFTWARES

Backend:

- **Python:**

Python is an interpreted object-oriented high-level general purpose programming language with dynamic semantics, Python's design philosophy is oriented around code readability and reliability.

- **Django:**

Django is a high-level Python-based free open-source web framework that enables rapid development of secure and maintainable website, it also follows the model-template views architectural pattern.

Frontend:

- **HTML:**

The Hypertext Markup Language, or HTML is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by scripting languages such as JavaScript or by technologies such as Cascading Style

- **CSS:**

Cascading Style Sheets is a style sheet language used for describing the presentation of a document written in a markup language such as HTML.

- **JavaScript:**

JavaScript is a text-based programming language used both on the client-side and server-side that allows web pages to be interactive.

- **jQuery:**

jQuery is a free open-source JavaScript library designed for even handling and CSS animation.

API:

- **Rest API**

Rest API is an application programming interface that conforms to the constraints of REST architectural style and allows integration with other websites.

IDE:

- **Visual Studio Code**

Visual Studio Code is a source-code editor made by Microsoft for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.

References

- Learn Python the Hard Way: A Very Simple Introduction to the Terrifyingly Beautiful World of Computers and Code by Zed Shaw. 1st Edition. September 19, 2013
- Software Engineering by Ian Sommerville. 9th Edition. March 13, 2010.
- Database System Concepts 6th Edition by Abraham Silberschatz, Henry Korth, S. Sudarshan. January 28, 2010.
- Python for Dummies Book by Aahz Maruch and Stef Maruch. 1st Edition. September 14, 2006.
- Django for Beginners: Build websites with Python and Django Book by William S. Vincent. March 7, 2018.
- Full Python course by Derek Banas. 20 August 2019
- Python Django Web Framework - Full Course for Beginners by FreeCodeCamp. 2 January 2019
- Python with Django Framework by Maximilian schwarzmüller. 9 June 2021
- Python with Django course by FreeCodeCamp. 1 July 2021