1. Introduction

This document provides an overview of Resume Generator Web, a resume building application. The system allows users to create, download and save resumes in PDF format. At the same time, the front-end of the system is built with HTML, CSS, JavaScript, and Bootstrap 5, and the back-end is made up of Django and MySQL. The system will distinguish between administrators and ordinary users by performing token authentication and role-based access control. Finally, in order to ensure security and high performance, the system will be deployed using Nginx and uWSGI.

2. Data Model

The following describes the core entities in the application's relational database (MySQL).

2.1 Entity-Relationship Diagram (ERD)

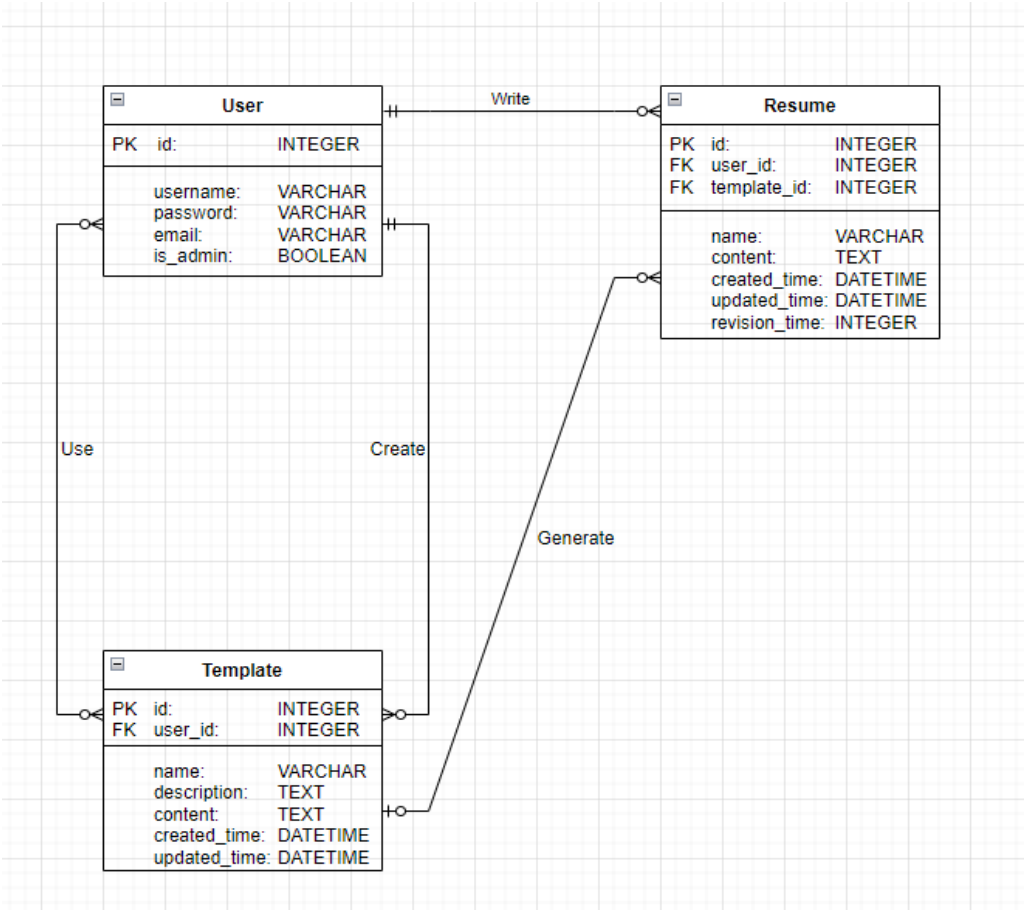

Fig2.1.1 Entity-Relationship Diagram

The key entities include:

- **User**:

•id: Primary Key

•username: Unique username for the user

•password: Encrypted user password

•email: User's email address

•is_admin: Boolean, true if the user is an administrator

- **Resume**:

•id: Primary Key

•user_id: Foreign Key to the User table

•template_id: Foreign Key to the Template table

•content: The resume content

•created_time: Timestamp when the resume was created

•updated_time: Timestamp when the resume was last updated

•revision_time: The revision times of the resume

- **Template**:

•id: Primary Key

•user_id: Foreign Key to the User table

•name: The name of the template

•description: A description of the template layout for summary and keyword search

•content: The template content

•created_time: Timestamp when the template was created

•updated_time: Timestamp when the template was last updated

The relationships between the entities include:

User (1 Mandatory) – Create – Resume (Many Optional):

User writes the Resume. User can write 0 to many Resumes, just depends on their wishes. But Resume must be created by one and only one user.

User (1 Mandatory) – Create – Template (Many Optional):

User creates the Template. User can create 0 to many Templates, just depends on their wishes. But Template must be created by one and only one user.

User (Many Optional) – Use – Template (Many Optional):

User can use 0 to many Templates. Also, Template can be used by 0 to many Users.

Template (1 Optional) – Generate – Resume (Many Optional):

Template can generate 0 to many Resumes. Resume can be generated by Template or not. If so, it could only be generated by 1 Template. If not, it will be written directly without Template.

2.2 Table Definitions

User Table

Field	Data Type	Description
id	INTEGER	Primary Key
username	VARCHAR	Unique, max 50 characters
password	VARCHAR	Encrypted user password
email	VARCHAR	User's email
is_admin	BOOLEAN	User role(admin or regular)

Tbl2.2.1 user table

Resume Table

Field	Data Type	Description
id	INTEGER	Primary Key
user_id	INTEGER	Foreign Key to User
template_id	INTEGER	Foreign Key to Template
name	VARCHAR	Max 50 characters
content	TEXT	Resume content
created_time	DATETIME	Creation timestamp
updated_time	DATETIME	Last updated timestamp
revision_time	INTEGER	Revision times

Tbl2.2.2 resume table

Template Table

Field	Data Type	Description
id	INTEGER	Primary Key

user_id	INTEGER	Foreign Key to User
name	VARCHAR	Max 50 characters
description	TEXT	Descrition of the template
content	TEXT	Template content
created_time	DATETIME	Creation timestamp
updated_time	DATETIME	Last updated timestamp

Tbl2.2.3 Template table

3. Project Structure

3.1 Style Guides

The system will follow a structured project layout with clearly defined components for the **frontend** and **backend**. The project will follow the PEP 8 python coding style for a clear and concise coding format. In addition, we use the github as version control system that each team member could collaborate effectively. For the GitHub Rulesets, we would use the main branch and require a notice through online chat to inform a push request. The force push is not allowed, and the merge request must be applied after checking the diff tree from the main branch GitHub. Regarding the test procedures, the project would use manual testing for different scenarios such web UI and functions since it is necessary to simulate as users to test and find what to improve. About the document standards, the files need to indicate the essentials for the project. The readme file would contain the Project Overview, Features, Demo, Installation, Usage, Technologies Used, Contributing, License and Contact. In addition, the Api documentation would include Endpoints, Request/Response, Error Codes and Authentication. As for the security guidelines, we would consider the input validation, dependencies, data protection, error handling, access control, and monitoring.

3.2 File/Folder layout

3.2.1 Frontend

The frontend will be built using **Bootstrap 5**, **JavaScript** (using fetch for API calls), and **HTML**. It contains a clear separation for different functional areas and a logical

organization for assets and static files. These files are for the webpage layouts for different scenarios. The directory structure is as follows:

3.2.2 Backend

The backend is built with **Django 4.2** and integrates **JWT authentication** for user management. The directory structure is separated into core app functionalities and project-wide configurations. The backend follows this structure:

3.2.3 Configuration Files Management

The configuration files are crucial for managing the environment settings, dependencies, and project configurations. In this project, all configuration files are in the project's main directory or under frontend and backend folders to maintain a centralized and organized structure.

4. Dependencies

The Dependency, also known as coupling, means the connections between those modules mentioned later, ensuring the smooth operation of our web application. In this program, dependencies are the essential equipment and

libraries which needed for both backend and frontend development, and other tools.

4.1 Backend Dependencies

The Backend Dependencies are the core tools which help us with the so-called server-side functions. They enable our staffs to handle user requests, store and manage data, and implement business logic. Besides, these tools allow the frontend and backend to interact with each other while ensuring the users' authentication security and data protection.

4.1.1 Django 4.2.15

Django is the main web framework used for backend development. Django offers a strong framework with built-in tools for handling requests, routing, and templates. Besides, it allows users to keep adding resources to the system, which helps reduce manual coding and improve productivity.

4.1.2 Django REST framework 3.15.2

RESTful API that helps simplify front-end and back-end interactions by using this Django extension. At the same time, Django REST framework 3.15.2 can also help the system with serialization, authentication, and permission control.

4.1.3 Django REST framework SimpleJWT 5.3.1

SimpleJWT can assist the system in handling JSON Web Token (JWT) authentication. This authentication is based on setting a specific transmission code between the front-end and back-end.

4.1.4 MySQL

MySQL is highly scalable and reliable. The system stores and manages all kinds of data and detailed resume information of users through MySQL.

4.2 Frontend Dependencies

In order to ensure the user experience of this system and the security of the working environment of this website, the front-end dependencies are mainly composed of user interface tools that are attractive enough and safe and reliable

4.2.1 Bootstrap 5.3.3

This system uses Bootstrap for web design. It gives a wide range of pre-built UI components and layout systems. While ensuring the fast response of the website, it can help the system to build a more modern user interface

4.2.2 Dotenv 16.4.5

This library is used to manage environment variables. By separating environment-specific variables such as API keys and database credentials, it ensures security and flexibility during the development and deployment.

4.3 Other Tools

There are also some other tools that are essential for improving server performance and ensuring efficient communication. These tools handle Tasks like serving static files and managing reverse proxying, especially when many users are accessing it at the same time. Besides, they keep the application to be stable and reliable when it’s used in production.

4.3.1 Nginx

Nginx acts as the web server for serving static files and handling reverse proxying. It efficiently handles multiple client requests and balances the load,improving the system's overall performance.

4.3.2 UWSGI

This application server is used to run Django in a production environment. It acts as a link between Nginx and Django, efficiently handling requests and serving responses.

5. API Documentation

5.1 User Registration and Authentication

	User Registration	User Login	Check Authentication	User Logout
Endpoint	/api/register/	/api/login/	/api/check-auth/	/api/logout/
Method	POST	POST	GET	POST
Description	Allow users to register for an account by providing essential information such as username, password, and email. This system will send a confirmation email for account verification.	Allows users to register for an account by providing essential information such as username, password, and email. This system will send a confirmation email for account verification.	Verify if the user is authenticated based on the provided JWT token. Returns user details if the token is valid, or an error message if invalid or expired.	Log the user out by invalidating the current JWT token.

Tbl5.1.1 User Registration and Authentication

5.2 Resume Management

	Create Resume	View Resumes	Update Resume	Delete Resume
Endpoint	/api/resumes/	/api/resumes/	/api/resumes/{resume_id}/	/api/resumes/{resume_id}/
Method	POST	GET	PUT	DELETE
Description	Allow the creation of a	Retrieve all resumes	Update the specified	Delete the specified

	new resume. The user can submit data such as personal information, education, work experience, and skills, which are stored in the database	associated with the authenticated user. Returns a list of resumes with basic details for easy identification.	resume with new or modified data. The user can update various sections of the resume, and changes are saved to the database	resume from the database, ensuring the data is permanently removed

Tbl5.2.1 Resume Management

5.3 Template Management

	Get All Templates	Get Template Details
Endpoint	/api/templates/	/api/templates/{template_id}/
Method	GET	GET
Description	Retrieve all available resume templates. Users can browse through various options to select the template that best fits their needs	Fetch the detailed information about a specific template, including layout and design features

Tbl5.3.1 Tamplate Management

5.4 Admin-Specific APIs

	Get All Users	Delete User
Endpoint	/api/admin/users/	/api/templates/{template_id}/

Method	GET	DELETE
Description	Retrieve a list of all registered users. Admins can view user profiles and other relevant data	Delete a specific user and their associated data from the system.

Tbl5.4.1 Admin-Specific APIs

6. Security

The security of the user's data is related to the level of trust in the system, the user experience and the security of their own network The following measures are used to enhance the security of the system.

6.1 JWT authentication:

JWT stands for json web token, which is a token method that directly queries data after passing the key verification. The system will use the commonly used HS256 encoding format and encrypt the HS256 format by setting the security code in the HS256 format. The corresponding data can only be read if the security code is correct. Through JWT, the burden on the server can be reduced, and the security of the system can be increased to a certain extent.At the same time, the system uses the SimpleJWT library under the Django REST Framework (DRF) to seamlessly integrate JWT tokens to achieve secure token-based authentication(Showed in figure 6.1.1).

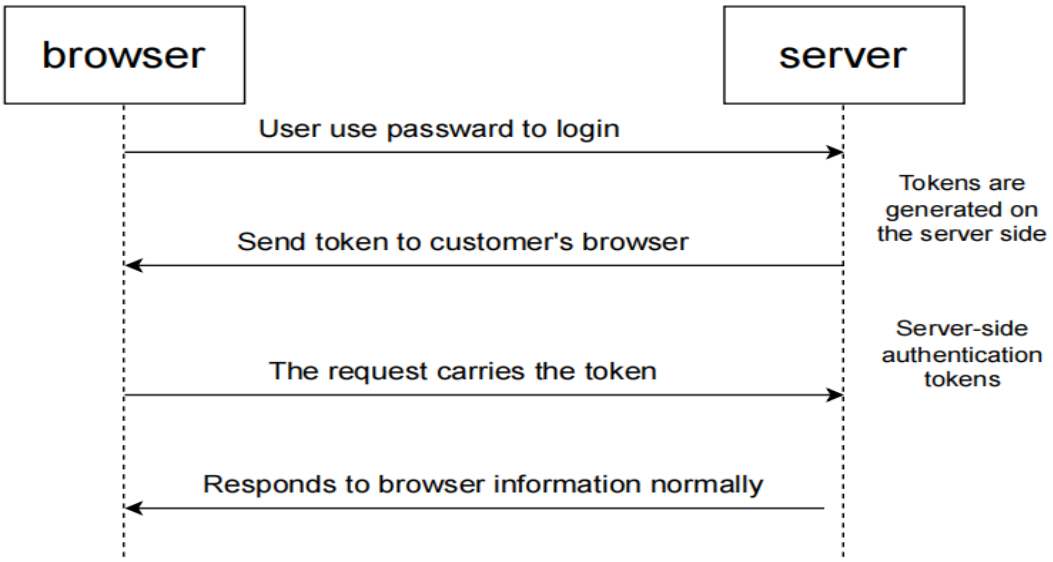

Fig 6.1.1

After the user logs in successfully, the server will generate a time-limited token via SimpleJWT. The token takes effect when the user initiates an API request.

6.2 CSRF protection

Using Django's built-in CSRF protection. Defend against CSRF attacks by using the built-in CSRF protection of the Django framework. When a user accesses the form interface, Django generates a unique token as a hidden field. Only if the token matches will the user's submission form behaviour be considered valid. Thís method will be performed in all forms in our system.

6.3 Role-based access control (RBAC)

By distinguishing the permissions of the roles, the roles are divided into administrators and ordinary users. This prevents unauthorized users from accessing sensitive information and performing important actions. By defining permissions, it is easier to manage website permissions and strengthen the security of websites.

6.4 Password hashing:

The system will use Django's built-in hash password mechanism to encrypt and store the user's key data in the database, and compare it with the user's bad password when logging in, if the two match, the login will be successfully authenticated(Showed in figure 6.1.2). Due to the one-item nature of the hash value, it is difficult to derive the original password even if the hacker steals the hash value. At the same time, Django's hashing mechanism automatically generates a unique corresponding salt for each password, further enhancing reliability.

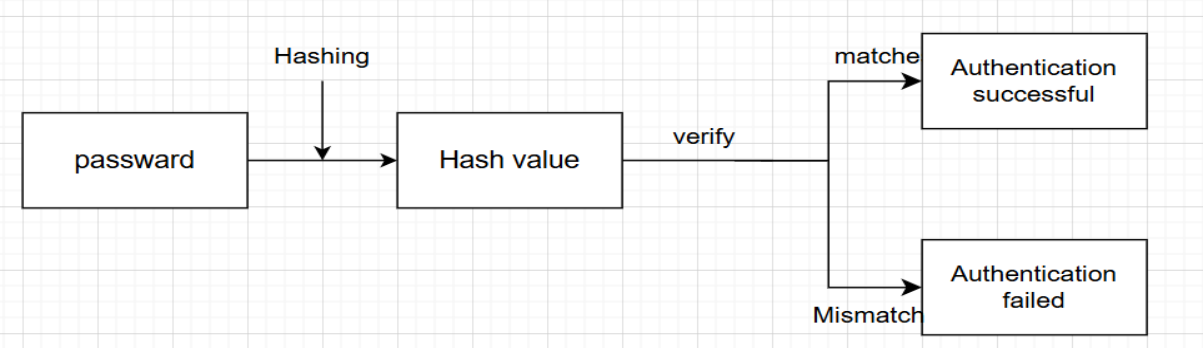

Fig 6.4.1

6.5 HTTPS

Through the HTTPS protocol based on SSL certificates, when the user and the website resume are connected, the server will send an SSL certificate containing the public key to the website for the resume encrypted connection, and it will not be intercepted or eavesdropped during transmission.

6.6 Nginx

By using Nginx as a reverse proxy, the system acts as an intermediarybetween the server and the client, prioritizing and filtering all requests, so asto minimize the direct exposure of the backend server. In this way, hackerscannot directly interact with the backend server, which greatly improves thesecurity and reliability of the system. A schematic diagram of its work is shown in figure 6.1.3.

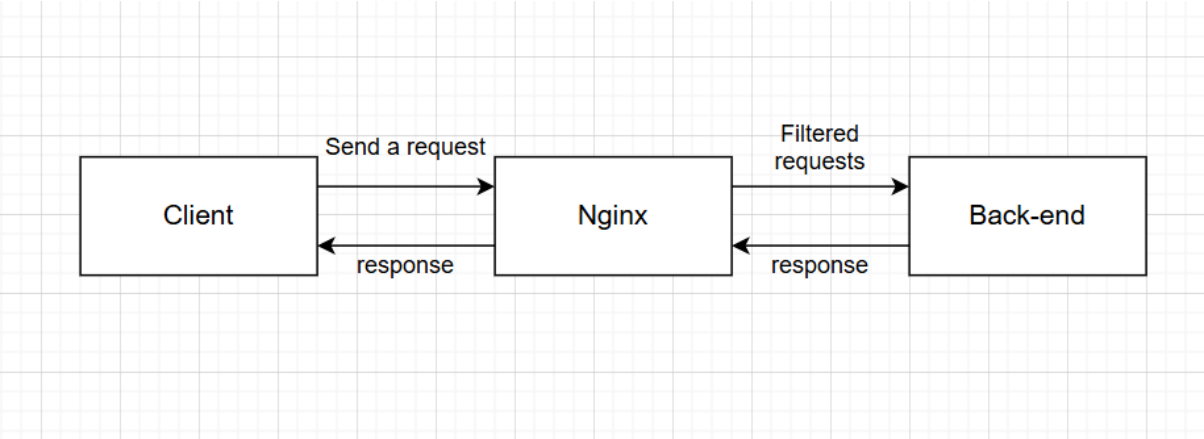

Fig 6.6.1

7. Conclusion

This article systematically describes the content of the design system of this group, specifies the type of variable diagram, and explains the structure of each table in detail through Elder at the same time. At the same time, the front-end and back-end of the system, as well as the key parts of the file management and coding style of the database, are summarized. With a robust architecture, secure authentication, and modular structure, this document provides a foundation for a scalable and maintainable application.