International Journal on Science and Technology (IJSAT)
E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

Online Job Portal Using Django, React, and PostgreSQL: A Full-Stack Employment Solution

K Abhay[1], Kunal Juvvala[2], M Manivardhan[3]

[1,2,3]Anurag University

Abstract

In an increasingly digital job market, the need for seamless and secure employment platforms is greater than ever. This paper presents the design and implementation of a full-stack Online Job Portal developed using Django for the backend, React for the frontend, and PostgreSQL as the relational database. The system enables streamlined interaction among job seekers, recruiters, and administrators. Key functionalities include user authentication, role-based dashboards, job posting and application systems, resume upload, and advanced search features. Through a modern and modular architecture, this project aims to enhance the hiring ecosystem by offering a reliable, user-friendly, and scalable platform that caters to both job seekers and employers.

1. Introduction

As employment processes become increasingly digitized, the role of online job portals has grown immensely. Traditional recruitment methods have evolved from newspaper classifieds to digital platforms that facilitate real-time application, filtering, and tracking. However, many existing job portals still suffer from significant shortcomings such as limited user interactivity, poor UI/UX design, and lack of personalization. Moreover, systems often fail to provide meaningful differentiation in user roles and their corresponding privileges. This project attempts to address these gaps by building a comprehensive and efficient job portal with a modern full-stack architecture.

The portal is designed to cater to three primary user roles—Job Seekers, Recruiters, and Administrators—each having access to specific functionalities. With React enabling a fast and dynamic frontend experience, Django providing a robust backend framework, and PostgreSQL offering secure data persistence, the system is built for real-world scalability and usability. This paper documents the development process, technical considerations, and the broader implications of such a system in today's job market.

2. Problem Statement

Despite the proliferation of job portals, many platforms fall short of providing an efficient and personalized experience. Users often encounter issues such as irrelevant job recommendations, lack of search filters, or outdated job listings. From the recruiter's standpoint, candidate management is often fragmented, with no centralized system for tracking applications and reviewing resumes. Furthermore, administrative oversight and moderation are rarely built-in features, leading to data inconsistencies and spam content.

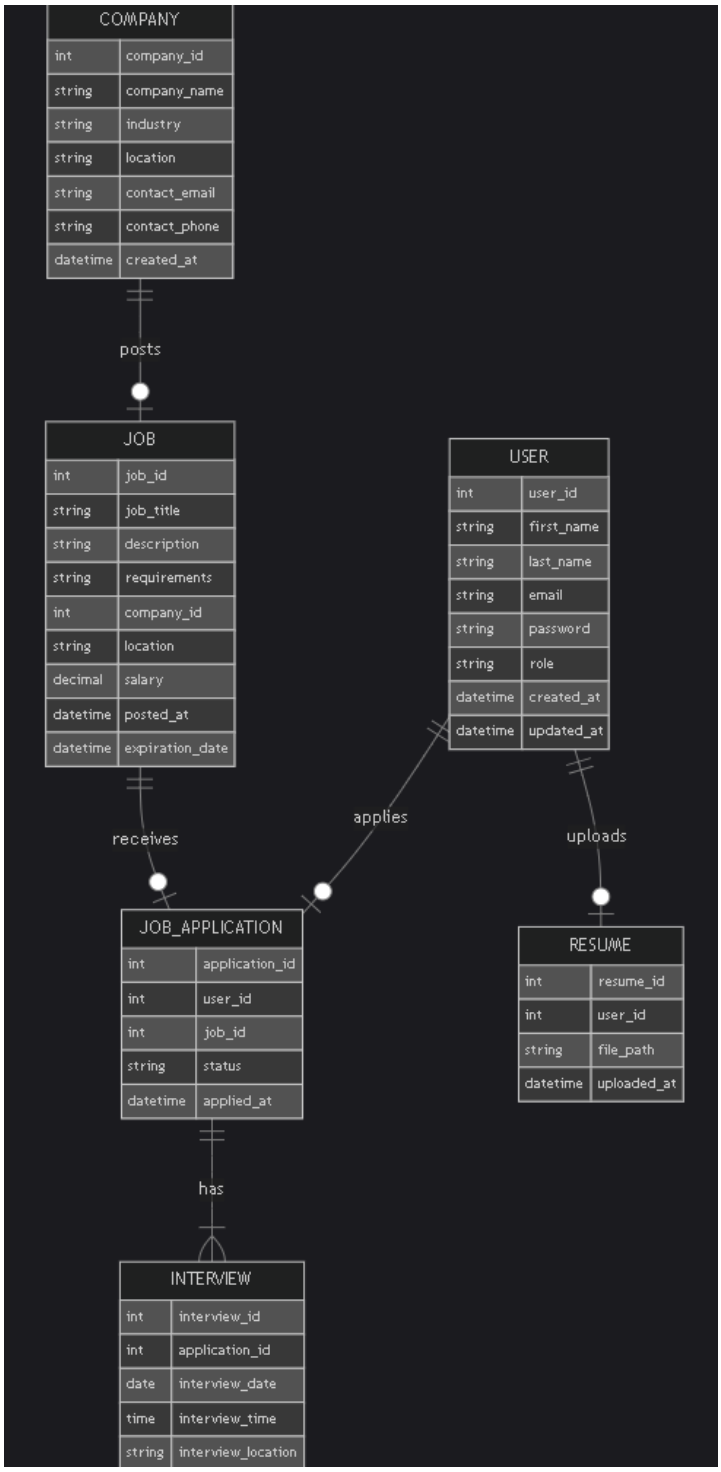
There is a pressing need for a platform that enables seamless job discovery, facilitates communication between job seekers and recruiters, and empowers administrators to manage the ecosystem effectively.

The proposed solution aims to tackle these challenges by building a role-driven architecture that offers differentiated user journeys and a scalable feature set.

3. System Design and Architecture

The Online Job Portal is structured around the Model-View-Controller (MVC) architecture, with Django handling the backend logic and APIs, React rendering the frontend interface, and PostgreSQL managing the relational data. The system is modular, allowing for future extensibility and integration with additional services such as chat modules or analytics dashboards.

React acts as the client-side framework responsible for rendering user interfaces dynamically and managing application state. It communicates with Django's REST APIs for CRUD operations on job listings, user data, applications, and resume uploads. Django, being a Python-based web framework, offers built-in features like ORM, authentication, and admin panel support, which significantly reduces development overhead while ensuring security and consistency. PostgreSQL serves as the database engine, chosen for its robustness, ACID compliance, and support for complex queries, which are essential for the filtering and search functionalities within the platform.

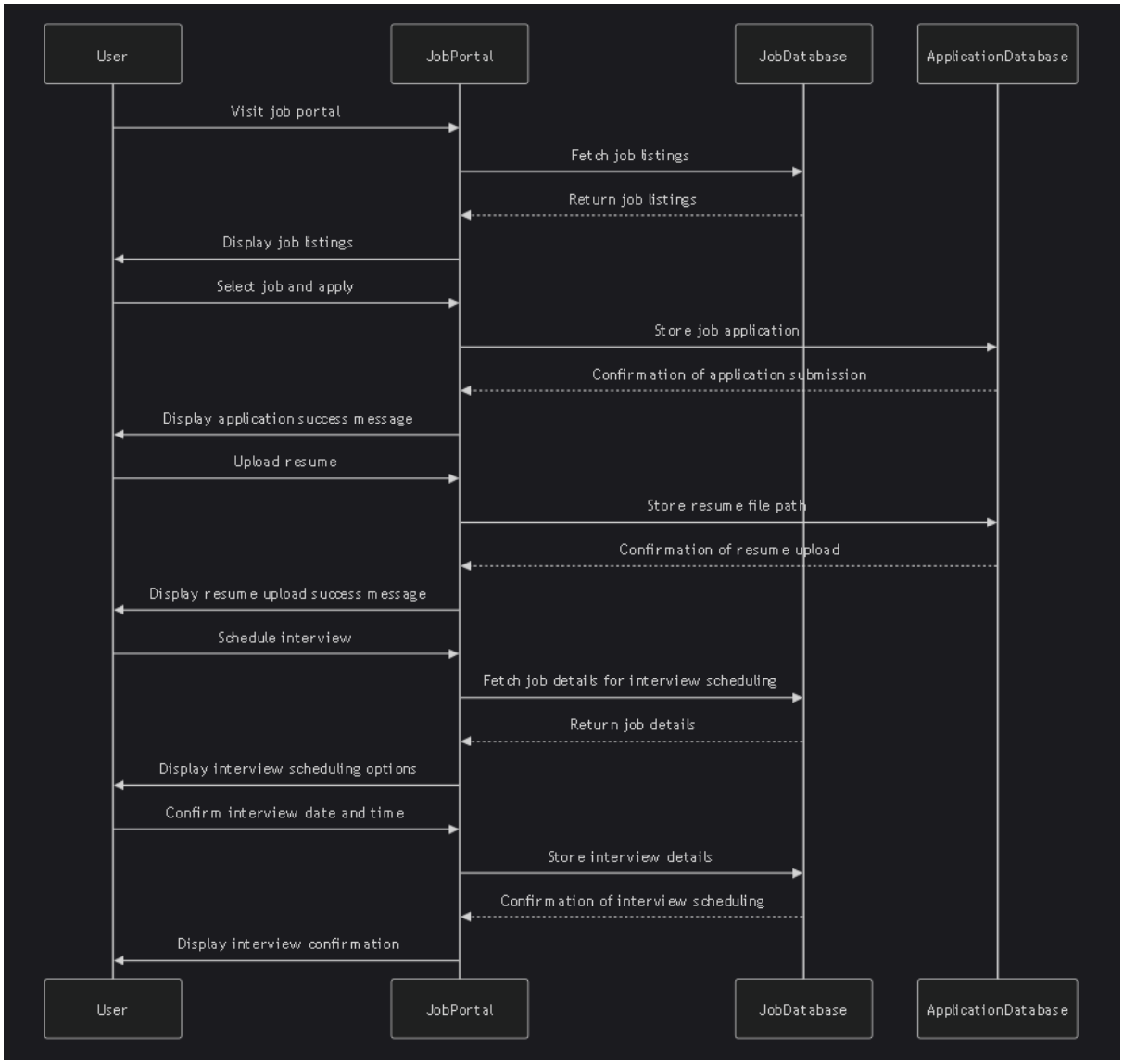


4. Functional Overview

The portal starts with a user registration and login system that identifies each user's role upon authentication. The dashboard is dynamically rendered based on the assigned role. Job seekers have access to a personalized dashboard where they can browse jobs, apply with their uploaded resumes, and track application statuses. The application process is streamlined with a single-click submission model that ensures user convenience.

Recruiters, on the other hand, can post new job listings with complete details such as job title, location, salary, description, and required experience. They can also manage these listings through editing and deletion. Additionally, recruiters are given access to applicant data and resumes, which can be reviewed through an intuitive interface.

Administrators are granted a higher level of control. They can oversee the entire platform, manage users, and moderate job listings to ensure content quality and integrity. The system includes features such as user banning, approval of new listings, and access to platform-wide metrics, giving admins complete control over platform governance.



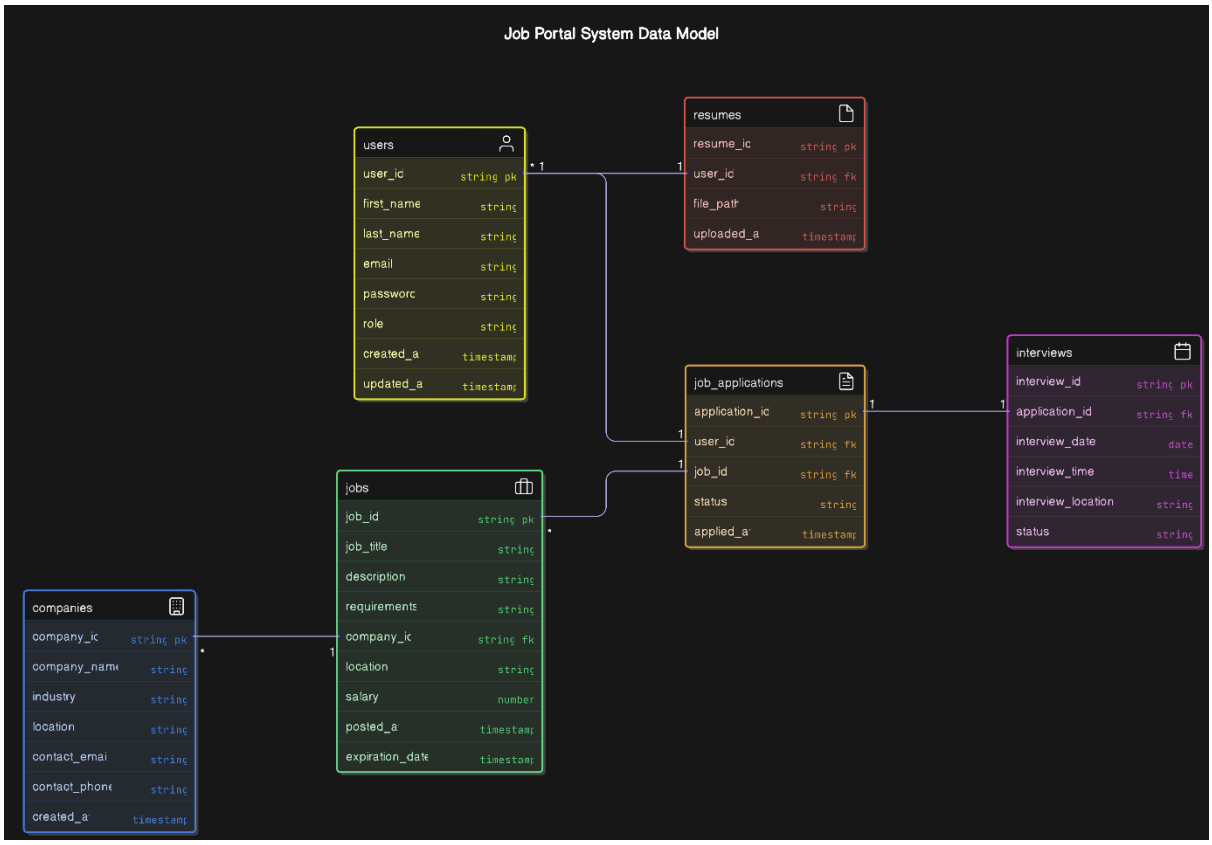
5. Implementation and Key Features

The application relies heavily on Django's ORM for managing relational data models. A custom user model extends the built-in Django user model to incorporate role-based access. Forms are validated on both frontend and backend to ensure data consistency and security. All sensitive actions are protected using CSRF tokens and user sessions, while password hashing is managed using Django's built-in encryption methods.

The job listing module supports both basic and advanced search. Users can filter jobs based on title, company name, location, salary range, and more. Search optimization is handled at the query level using Django's filtering mechanisms and PostgreSQL's indexing features.

Resume upload functionality is implemented using Django's file handling system. Files are stored securely on the server and linked to user profiles. Job seekers can upload resumes in PDF format, and recruiters can download them while reviewing applications.

All interactions are asynchronous and API-driven, ensuring that the frontend remains responsive. Form submissions, dashboard updates, and navigation between components happen without full-page reloads, offering a modern single-page application (SPA) experience.



6. Evaluation and Testing

The system underwent multiple phases of testing, including unit testing, integration testing, and user acceptance testing. Unit tests were written using Django's built-in test suite, particularly for models, views, and forms. REST API endpoints were validated using tools like Postman. React components were tested using React Testing Library to ensure proper rendering and user interaction.

From a usability standpoint, user feedback highlighted the platform's intuitive navigation, fast load times, and effective filtering options as key strengths. Performance testing confirmed that the system scaled well under concurrent user loads, owing to the asynchronous design and efficient data queries.

7. Challenges and Limitations

Developing a full-stack application with role-based access control brought its share of challenges. One primary issue was managing state consistency across the client and server, especially with asynchronous operations. Ensuring that data updates reflected instantly on the React frontend required careful state management and API synchronization.

File handling posed another challenge, particularly around securely managing resume uploads and preventing storage bloat. Security was a constant concern, with input sanitization, form validation, and secure authentication mechanisms requiring thorough implementation.

Although the system performs well under typical usage conditions, it currently lacks real-time communication features, and user analytics are minimal. These limitations present opportunities for future improvement.

8. Conclusion and Future Work

The Online Job Portal project demonstrates the power of full-stack development in addressing real-world problems. Through the integration of Django, React, and PostgreSQL, the portal offers a scalable, user-friendly, and technically sound solution to modern recruitment needs. By introducing role-based workflows, resume management, and job search optimization, the platform significantly reduces friction in the hiring process.

Future enhancements could include the integration of a real-time chat module using Django Channels or WebSockets, AI-driven job recommendations, and dashboard analytics for recruiters. Further, adapting the system for mobile platforms and supporting third-party API integrations for resume parsing or interview scheduling could make it even more versatile.

References

1. Django Documentation – https://docs.djangoproject.com/
2. React Official Docs – https://react.dev/
3. PostgreSQL Documentation – https://www.postgresql.org/docs/
4. Fielding, Roy T. “Architectural Styles and the Design of Network-based Software Architectures.”
5. IEEE Papers on Web Portals and Full-Stack Architectures