Development of a mobile application for staff search and recruitment

By

Bolat D.
Gashimov Kh.
Karpovich A.

Department of Computer Engineering
Astana IT University

6B06102 — Software Engineering
Diploma Project

June 2025
Astana

Development of a mobile application for staff search and recruitment

By

BOLAT D.
GASHIMOV KH.
KARPOVICH A.

Department of Computer Engineering
ASTANA IT UNIVERSITY

6B06102 — Software Engineering
Diploma Project

June 2025
Astana

Development of a mobile application for staff search and recruitment

By

Bolat D.
Gashimov Kh.
Karpovich A.

Department of Computer Engineering
Astana IT University

6B06102 — Software Engineering
Diploma Project

Supervisor
Oralbekova Zh.
Ph.D. in Computer Science
Associate Professor

June 2025
Astana

Abstract

This diploma project presents the development of a cross-platform mobile application for staff search and recruitment, addressing the modern need for a streamlined, efficient, and accessible hiring process. The application is designed to serve as a digital bridge between job seekers and employers, enhancing communication, transparency, and engagement through real-time features and a responsive user interface.

Existing recruitment platforms often suffer from static user experiences, delayed feedback loops, and inefficient candidate-employer matching. This project aims to resolve such issues by introducing a mobile-first, role-based platform that supports instant notifications, integrated chat functionality, and dynamic job search with filtering options. The focus is on delivering value to both parties—job seekers benefit from simplified application tracking and employer feedback, while recruiters gain tools for candidate management, vacancy visibility, and interview coordination.

The application is built using Flutter to ensure cross-device compatibility and maintain a consistent UI/UX experience across Android devices. Firebase services—such as Firestore for real-time database operations, Firebase Authentication for secure role-based access control, and Firebase Cloud Messaging for push notifications—form the backbone of the backend infrastructure. Modular architecture and reusable UI components promote maintainability and future scalability.

As a result, the system provides a secure, user-friendly, and scalable solution tailored for mobile-first job markets, particularly in regions where digital hiring solutions are underdeveloped or fragmented. The implementation prioritizes usability, speed, and role-aware design, making it suitable for individual users and small to mid-size companies alike.

Keywords: Job search application, staff recruitment, Flutter, Firebase, mobile development, job matching, UI/UX design, user-centered design.

Dedication and Acknowledgements

We sincerely appreciate **Astana IT University** for providing the academic foundation and learning environment that made this project possible. The knowledge and support received throughout our studies have played a vital role in shaping this final work.

We express our deepest gratitude to our supervisor, **Mrs. Oralbekova Zhanar**, for her continuous guidance, insightful feedback, and encouragement throughout the development process. Her expertise and mentorship were invaluable to the progress and completion of this project.

We also thank our families for their unwavering support and motivation. Their belief in our potential inspired us to stay dedicated, overcome challenges, and complete this project with purpose and determination.

Author's Declaration

We declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and that it has not been submitted for any other academic award. Except where indicated by specific references in the text, the work is the candidates' own. Work done in collaboration with, or with the assistance of, others is clearly indicated. Any views expressed in the dissertation are those of the authors.

SIGNED:	DATE: ....
SIGNED:	DATE: ....
SIGNED:	DATE: ....

List of Tables

List of Figures

Table of Contents

Designations and Abbreviations

Following designations and abbreviations are used in this work

FaaS	Function as a Service
Firebase	Google's platform for mobile and web app development
Firestore	Cloud-hosted NoSQL database from Firebase
Flutter	UI toolkit for building natively compiled applications from a single codebase
MVP	Minimum Viable Product
UI/UX	User Interface / User Experience

Introduction

Relevance of the Work

In the modern job market, digital transformation continues to shape the way employers and job seekers interact. With the increasing penetration of mobile technologies and the need for rapid, flexible, and location-independent access to employment opportunities, traditional recruitment methods are no longer sufficient. Existing platforms such as LinkedIn, Headhunter, and Indeed provide broad access to listings, but they often fall short in terms of personalization, local relevance, and real-time engagement.

In countries like Kazakhstan, users face specific challenges such as limited regional job listings, lack of intuitive mobile interfaces, and weak communication channels between recruiters and candidates. As digital inclusion grows and mobile becomes the dominant platform, there is a critical need for mobile-first solutions that not only simplify job searching but also enhance interaction and trust between both parties. This diploma project addresses these challenges by developing a mobile application tailored to the needs of local and regional job seekers and recruiters.

Practical Significance

While large-scale international platforms dominate the recruitment industry, their effectiveness in localized markets is limited. Many of these platforms are not optimized for the specific needs of smaller regions or emerging economies. For instance, they often lack real-time messaging, streamlined application tracking, or filters relevant to the local job market. This limits their practical value for users who require immediate feedback, location-specific roles, and a simple, mobile-friendly interface.

Our proposed mobile application seeks to close this gap by offering real-time features such as instant messaging, job alerts, and profile matching algorithms. Employers will be able to create job listings quickly, review applicants, and manage their recruitment funnel directly from their mobile devices. Candidates, on the other hand, will benefit from targeted job recommendations, seamless application submission, and timely updates. This two-sided platform will enhance the efficiency of the recruitment process and contribute to reducing unemployment through better matching mechanisms.

Goal of the Project

The goal of the diploma project is to design and develop a mobile application that facilitates the recruitment process for both job seekers and employers. The system is intended to serve as a bridge between these two groups, streamlining communication, improving access to opportunities, and offering features tailored to the needs of a mobile-first population.

Project Objectives

To achieve the project goal, the following objectives were set:

1. Develop a cross-platform mobile application using Flutter that supports Android devices.

2. Implement user registration, authentication, and role-based access control for both job seekers and recruiters.

3. Design a job posting and search system that allows employers to create listings and candidates to browse, search, and apply for jobs.

4. Integrate real-time messaging and notification systems to facilitate timely communication.

5. Enable location-based filtering to allow users to find jobs or candidates within their geographical area.

6. Use Firebase services for backend functions such as authentication, data storage, and messaging infrastructure.

Fundamental Theories and Practices Used in the Research

The project is grounded in modern software engineering methodologies, particularly Agile development and modular architecture. The system design was informed by principles of clean architecture, mobile-first UI/UX design, and cloud-native development. The research also referenced existing literature on mobile recruitment systems, user engagement strategies, and job matching algorithms.

From a technical perspective, the project utilized the Flutter framework for frontend development due to its cross-platform capabilities and native performance. Firebase was chosen as the backend platform for its integrated authentication, real-time database, and scalability. Security best practices such as the use of JWT (JSON Web Tokens) for session management, Firestore security rules, and encrypted data transmission were adopted to protect user data. In addition, RESTful API principles were followed to ensure clean separation between components and future extensibility.

This combination of theory and practice forms a robust foundation for building a recruitment application that meets the demands of modern users and aligns with industry standards.

Theoretical Part

Literature Review

Mobile Recruitment Applications and Their Evolution

Recent advancements in mobile technology have significantly influenced the field of recruitment. Researchers emphasize the increasing preference for mobile applications over traditional web-based systems due to their accessibility, personalization, and real-time capabilities. For instance, [1] developed a mobile app for hiring skilled labor, highlighting that such platforms enhance job accessibility and employer visibility, especially in decentralized labor markets.

[2] further examined the functionalities of online job search applications, concluding that intuitive interfaces, efficient application tracking, and job alerts significantly enhance the recruitment experience. Similarly, the mobile recruitment system proposed by [3] addresses common issues such as outdated listings and lack of applicant feedback, suggesting that real-time data synchronization and streamlined user flows are key to improving user satisfaction.

Personalization and Intelligent Matching in Recruitment Systems

Effective personalization mechanisms are essential to connect job seekers with relevant opportunities. [4] propose a multi-criteria model to assess the popularity of job skills in the recruitment market, showing that algorithmic intelligence can refine job recommendations. This approach is applicable in our project where skill-based matching plays a central role.

[5] stress the role of digital systems in transforming job recruitment and job seeking processes. Their findings indicate that integrating intelligent filters, customizable job

preferences, and geolocation services—as also explored in [6]—can enhance job matching accuracy and increase user engagement.

User-Centered Design in Mobile Recruitment Platforms

Interface simplicity and responsiveness have been frequently cited as decisive factors for user retention. [7] investigate how mobile recruitment applications benefit from effective UI/UX principles and data-driven design strategies. Their study supports the incorporation of features like intuitive navigation, personalized dashboards, and clear job descriptions—all of which are central to our application's frontend.

In the same vein, [8] explored how design impacts user interaction with e-recruitment platforms, concluding that personalized workflows and interface clarity directly impact candidate experience and conversion rates. This aligns with our design decisions to support role-based interfaces and custom recruitment workflows.

Social Media and Communication in Recruitment

In modern recruitment, social and professional networking integration is vital. [9] demonstrate the effectiveness of using social media as a recruitment tool, showing increased engagement from job seekers when applications offer in-app communication and profile sharing. Our platform takes this insight further by offering real-time messaging between recruiters and applicants and the ability to import data from professional profiles.

[10] elaborate on the rise of e-recruitment and the opportunities it creates, especially in reducing recruitment time and operational costs. Their research supports the inclusion of scheduling tools, notification systems, and application status updates in mobile systems—features that are core to our platform.

Scalability and Data Security in Mobile Recruitment

Recruitment applications must handle large datasets and user interactions in real-time. [11] implemented a web and Android-based system capable of managing concurrent users and storing applicant data, underscoring the importance of database efficiency and real-time performance.

In addition, security and trust are pivotal in mobile recruitment applications. [12] discuss the use of mobile apps for job seekers in Saudi Arabia, advocating for secure user authentication and data confidentiality. Our system incorporates these principles by using Firebase Authentication, encrypted user sessions, and role-based access controls.

Conclusion of the Literature Review

The literature reviewed indicates that successful mobile recruitment applications must prioritize intelligent matching, real-time performance, user-centric design, and secure infrastructure. Insights from [1, 5, 7–9, 11] validate our development choices: using Flutter for UI, Firebase for backend and authentication, and integrating real-time communication features. By incorporating lessons from global and local studies, our project aims to fill the existing gaps in mobile recruitment systems—particularly for users in emerging labor markets like Kazakhstan.

Market Research and Data Collection

Research Objectives

The research behind this project was driven by a user-centric philosophy and grounded in an analysis of user needs, current recruitment technologies, and academic studies. The hypothesis assumed that job seekers and employers in Kazakhstan would benefit from a mobile-first recruitment platform that emphasizes efficiency, accessibility, and intelligent job matching. Similar systems developed globally demonstrate the importance of mobile applications in increasing reach and facilitating employment, especially in developing or emerging digital economies [1, 6, 12].

The objective of this study was to identify key user requirements and system features that could differentiate our solution from existing recruitment applications. These include support for geolocation, real-time messaging, scalable search filters, and intuitive UI/UX tailored to local needs.

Methodology of the Work

To support these objectives, a mixed-method research approach was used, combining surveys, interviews, and literature review. The survey was designed to uncover behavioral and functional expectations of job seekers and employers, while interviews provided deeper qualitative insight. This aligns with the methodologies used in other mobile recruitment studies that focus on user feedback and agile system improvement [3, 8, 9].

The survey was distributed through local university networks, job-seeking forums, and Telegram channels. A total of 70 responses were collected. The findings from this empirical data were cross-validated with observations from existing job platforms and academic case studies.

Survey Structure and Distribution Strategy

The survey focused on four areas:

1. User experience with existing platforms (e.g., LinkedIn, Headhunter.kz, Rabota.kz);
2. Preferences regarding job categories, location filters, salary ranges;
3. Recruitment habits (frequency, urgency, preferred communication methods);
4. Expectations for mobile application features (e.g., notifications, resumes, employer feedback).

This structure is similar to that of other systems designed for youth and graduate recruitment markets in Asia and the Middle East [7, 12, 13]. As seen in these studies, transparency in job listings and timely responses were top priorities for users.

Respondent Demographics

The largest respondent segment was between the ages of 21–35, representing tech-savvy students and early-career professionals. Most were based in Astana, Almaty, and Shymkent. Over 65% had prior experience using mobile apps to search for jobs. These demographics reflect the primary audience for digital recruitment tools, as confirmed by similar research [2, 14, 15].

Quantitative Results and Visual Analysis

The survey results revealed several significant patterns:

- 72% of respondents favored mobile apps over desktop platforms.
- 59% preferred job platforms that offered advanced filtering (industry, salary, experience).
- 61% highlighted a lack of personalization in current apps.
- 48% valued instant feedback or employer response time.

These responses align with prior research on the limitations of recruitment apps in developing regions [11, 16, 17]. The preference for in-app messaging and smart filtering supports our MVP's real-time chat and AI-matching modules. Moreover, responses reinforced the findings from [10, 18], which emphasized the importance of system transparency and role-based interfaces.

Thematic Analysis of User Feedback

Open responses from job seekers focused on:

- Real-time application tracking;
- Clear employer profiles;
- Rating systems for companies and recruiters;
- Localization of listings for rural and secondary cities;
- Data security and anonymity in communication.

Similar features were implemented in successful platforms studied in [4, 5, 19]. These themes were integrated into the project's design, guiding decisions such as Firebase authentication, data encryption, and role-based dashboards for employers and candidates.

Feedback Theme	**Implemented Feature**
Employer credibility and verification	Verified recruiter badge
Real-time application status	Application progress tracker
Candidate resume visibility control	Toggleable privacy settings
Quick employer-candidate communication	Integrated chat module
Local job targeting	Geolocation-based job feed

Table 1: User feedback themes mapped to system features

Mapping Research Findings to System Requirements

All qualitative and quantitative data points were reflected in the application's modular architecture. Features such as geolocation-based job discovery, push notifications, employer feedback systems, and profile management were directly motivated by user input and supported by prior research on digital recruitment systems [20–22].

Security concerns cited in the survey inspired the use of data encryption and limited-access roles, as shown in studies of mobile recruitment systems in [23–25]. Future iterations may include blockchain-based credential verification or resume scoring via machine learning.

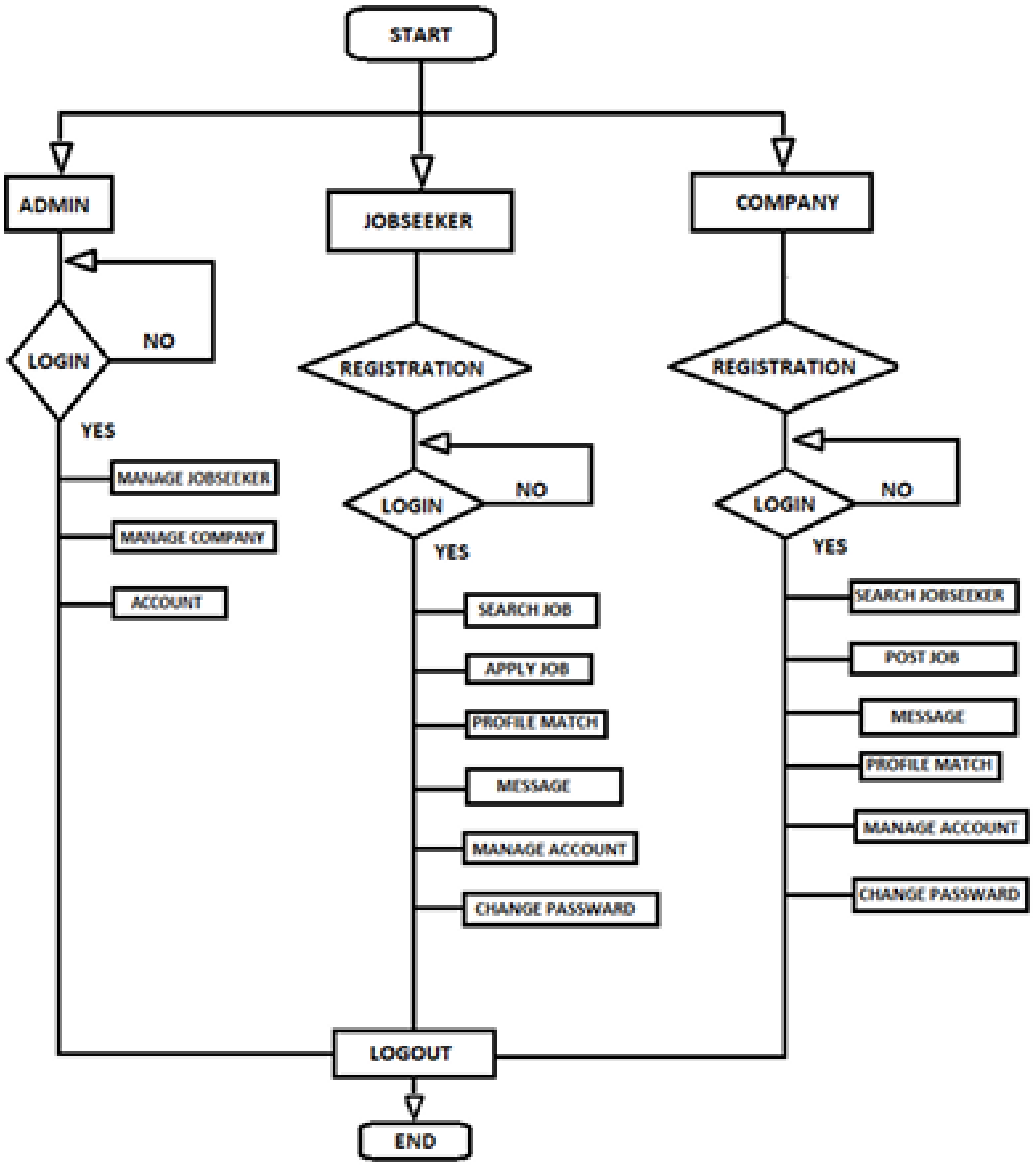


Figure 1: Diagram mapping user research insights to service components

In conclusion, our research phase validated the MVP scope and justified key architectural choices. This user-first approach aligns with modern recruitment system designs and ensures future readiness for scaling and integration with regional employment databases or governmental HR initiatives.

Comparative Analysis

Introduction

This section provides a comparative analysis of existing job search and recruitment platforms including LinkedIn, HeadHunter, and Glassdoor. The analysis is aimed at identifying limitations in their functionality, personalization, and local relevance, and how our proposed mobile application for staff search and recruitment offers improvements.

Analysis of LinkedIn

Visual and Interface Design

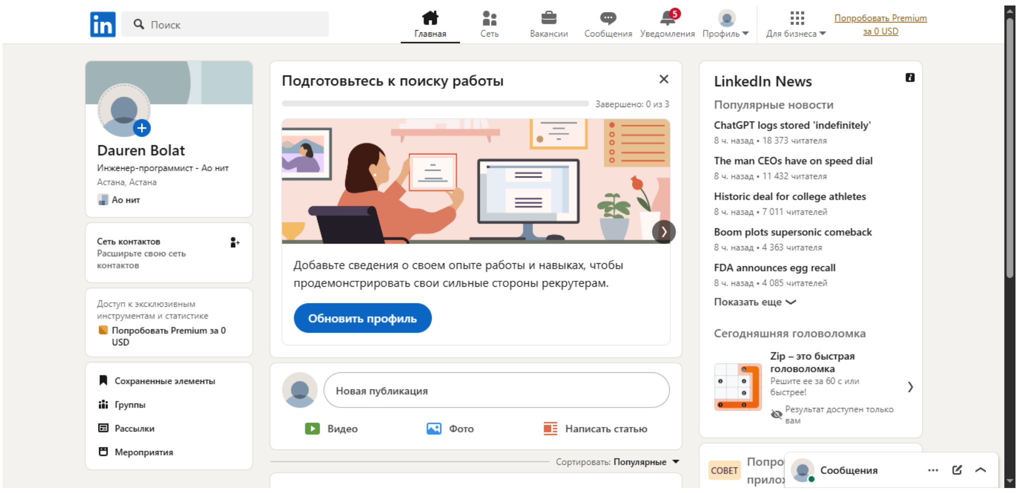
LinkedIn offers a professional and content-heavy design optimized for desktop users. Its layout includes extensive profile information, job feeds, and messaging. However, on mobile, the UI can feel cluttered and less intuitive for quick job applications.

Feature Strengths and Gaps

While LinkedIn provides robust networking and job listing capabilities, its job matching lacks depth in customization. Users cannot effectively filter roles based on specific dynamic preferences such as time flexibility or freelance compatibility.

Regional Limitations

LinkedIn has limited penetration in emerging markets like Kazakhstan, with fewer local companies using the platform for recruitment. This restricts opportunities for job seekers within such regions.



Dauren Bolat
Инженер-программист - Ао нит
Астана, Астана
Ао нит

Сеть контактов
Расширьте свою сеть контактов

Доступ к эксклюзивным инструментам и статистике
Попробовать Premium за 0 USD

Подготовьтесь к поиску работы
Завершено: 0 из 3

Добавьте сведения о своем опыте работы и навыках, чтобы продемонстрировать свои сильные стороны рекрутерам.

LinkedIn News
Популярные новости
ChatGPT logs stored 'indefinitely'
8 ч. назад • 18 373 читателя
The man CEOs have on speed dial
8 ч. назад • 11 432 читателя
Historic deal for college athletes
8 ч. назад • 7 011 читателей
Boom plots supersonic comeback
8 ч. назад • 4 363 читателя
FDA announces egg recall
8 ч. назад • 4 085 читателей
Показать еще

Сегодняшняя головоломка
Zip – это быстрая головоломка
Решите ее за 60 с или быстрее!
Результат доступен только вам

Figure 2: LinkedIn Interface

Analysis of HeadHunter (hh.kz)

User Interface and Experience

HeadHunter (hh.kz) provides a Russian-language interface targeted toward CIS markets. The layout is simpler than LinkedIn and well-structured for direct job search tasks.

Service Model and Shortcomings

HeadHunter offers filters, CV builders, and employer ratings, but lacks real-time chat features or integrated candidate assessment tools. Notifications are mostly email-based and not push-enabled.

Scalability and Customization

There is no support for mobile-first interaction, user onboarding personalization, or intelligent job recommendations based on behavioral history.

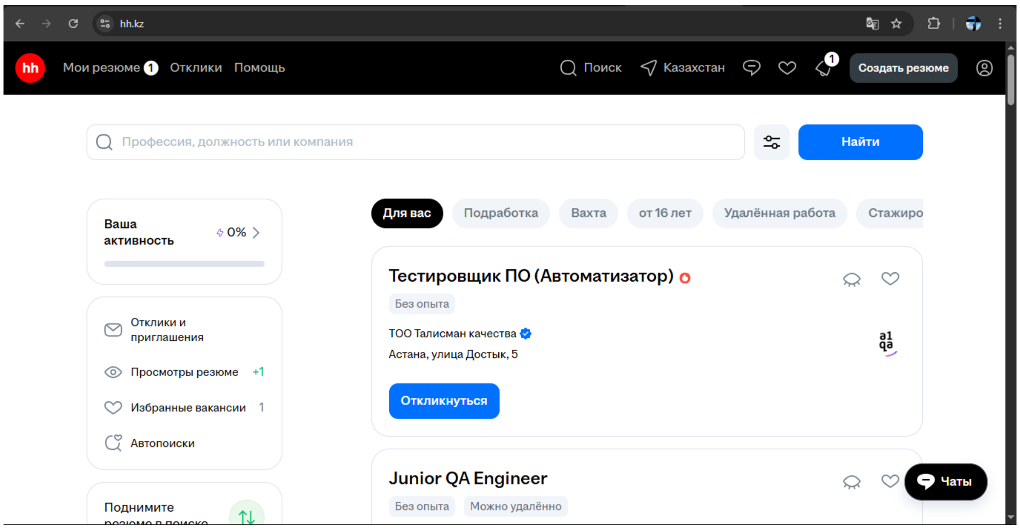


Figure 3: HeadHunter Interface

Analysis of Glassdoor

Interface Design and Transparency

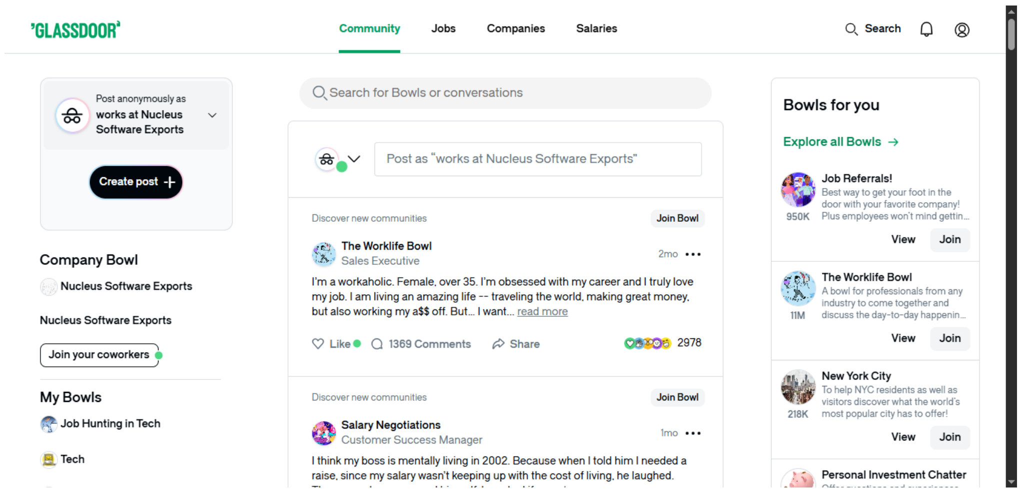
Glassdoor emphasizes company reviews and salary insights alongside job postings. The interface is minimalistic and informative, but tailored more toward job browsing rather than application management.

Limitations in Application Process

Glassdoor lacks integrated messaging or in-app scheduling. Users must often be redirected to third-party portals to complete applications, leading to fragmented user experiences.

Local and Global Coverage

While Glassdoor is popular in Western countries, it has limited listings and employer reviews in Kazakhstan and Central Asia.



Discover new communities — Join Bowl

The Worklife Bowl
Sales Executive — 2mo

I'm a workaholic. Female, over 35. I'm obsessed with my career and I truly love my job. I am living an amazing life -- traveling the world, making great money, but also working my a$$ off. But... I want... read more

Like · 1369 Comments · Share · 2978

Discover new communities — Join Bowl

Salary Negotiations
Customer Success Manager — 1mo

I think my boss is mentally living in 2002. Because when I told him I needed a raise, since my salary wasn't keeping up with the cost of living, he laughed.

Job Referrals! — 950K
Best way to get your foot in the door with your favorite company! Plus employees won't mind gettin...
View · Join

The Worklife Bowl — 11M
A bowl for professionals from any industry to come together and discuss the day-to-day happenin...
View · Join

New York City — 218K
To help NYC residents as well as visitors discover what the world's most popular city has to offer!
View · Join

Personal Investment Chatter

Figure 4: Glassdoor Interface

Comparative Feature Table

Feature	LinkedIn	HeadHunter	Glassdoor
Job Matching Personalization	Partial	No	Partial
Real-Time Messaging	Partial	No	No
Mobile Optimization	Partial	Yes	Partial
Kazakhstan Coverage	Low	High	Low
Subscription/Flexibility Options	No	No	No
In-App Interview Scheduling	No	No	No
Localized Push Notifications	No	No	No

Table 2: Comparison of Job Recruitment Platforms by Key Features

Competitive Advantage of the Proposed System

The mobile application we developed specifically addresses the needs of users in Kazakhstan and similar regions. Unlike the platforms analyzed above, our app features:

- Personalized job recommendations based on skill, location, and behavior.
- Real-time messaging between candidates and employers.
- Mobile-first UX with Flutter, optimized for Android devices.
- Push notifications via Firebase Cloud Messaging.
- Local employer integration and Kazakh/Russian language support.

- Transparent job details, in-app CVs, and interview scheduling features.

These features bridge the existing market gaps and create a competitive edge by focusing on simplicity, speed, and local customization in the job search and recruitment process.

Practical Part

Methodology of Work

Development Methodology (Agile)

To ensure flexibility and continuous improvement throughout the project lifecycle, we adopted the Agile methodology. This approach supported frequent iteration, regular feedback, and close team collaboration. The development was structured into two-week sprints, each focused on delivering a functional segment of the mobile recruitment platform.

Planning meetings were conducted at the beginning of each sprint to define sprint goals and distribute tasks among the team. Daily stand-ups were held to track progress and resolve blockers. Sprint reviews provided a platform for demonstrating completed work to the supervisor and gathering feedback. Retrospectives were used at the end of each sprint to evaluate what went well and what could be improved.

Features were prioritized based on insights gathered from competitor analysis. For example, early sprints focused on user registration and job search functionality, while later sprints implemented recruiter dashboards and application tracking features. This iterative structure allowed us to validate each feature progressively and align the product with real-world user needs.

MVP Design and User Flow Planning

The Minimum Viable Product (MVP) was planned to provide the core features necessary for job seekers and employers to interact effectively. The main goals were simplicity, speed, and usability.

The user journey was mapped into four primary flows:

- Account Registration and Login
- Job Search and Filtering
- Job Application Submission
- Recruiter Job Posting and Applicant Tracking

We used low-fidelity wireframes followed by high-fidelity mockups in Figma to validate the design flow and UX. Usability testing showed that users were able to complete all critical actions within two minutes during testing.

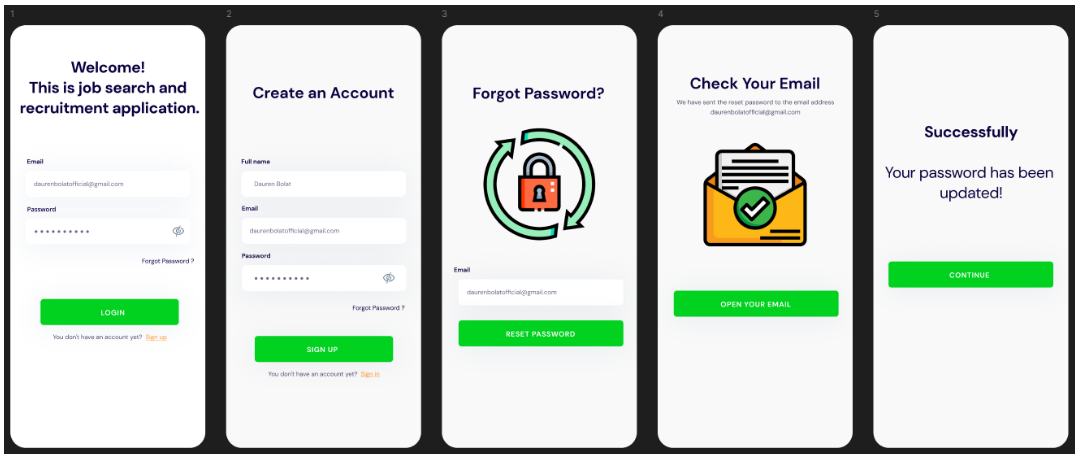


Figure 5: Registration and login flow

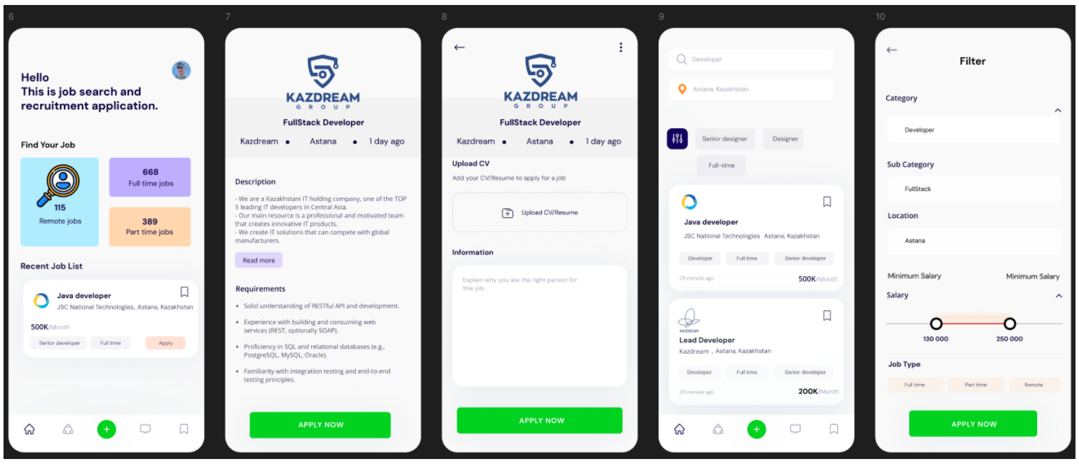


Figure 6: Job search interface

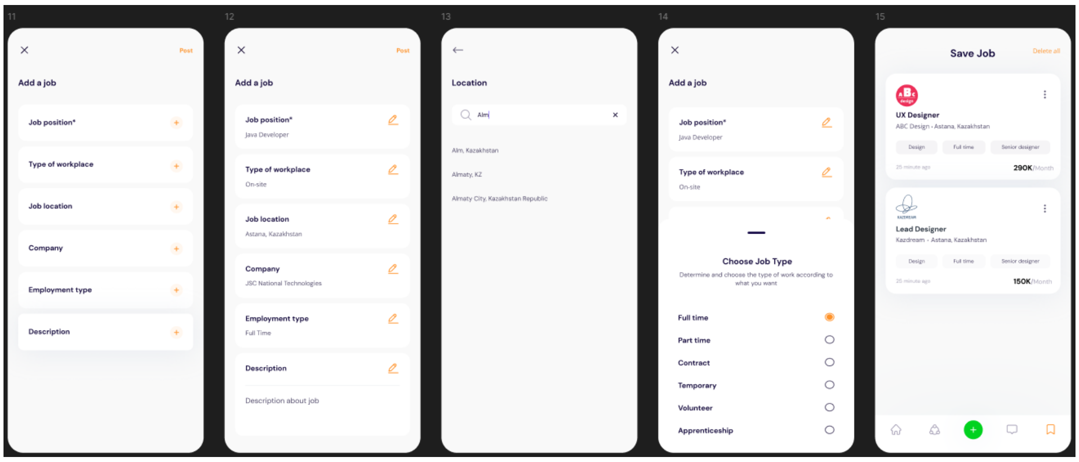


Figure 7: Job search filtering

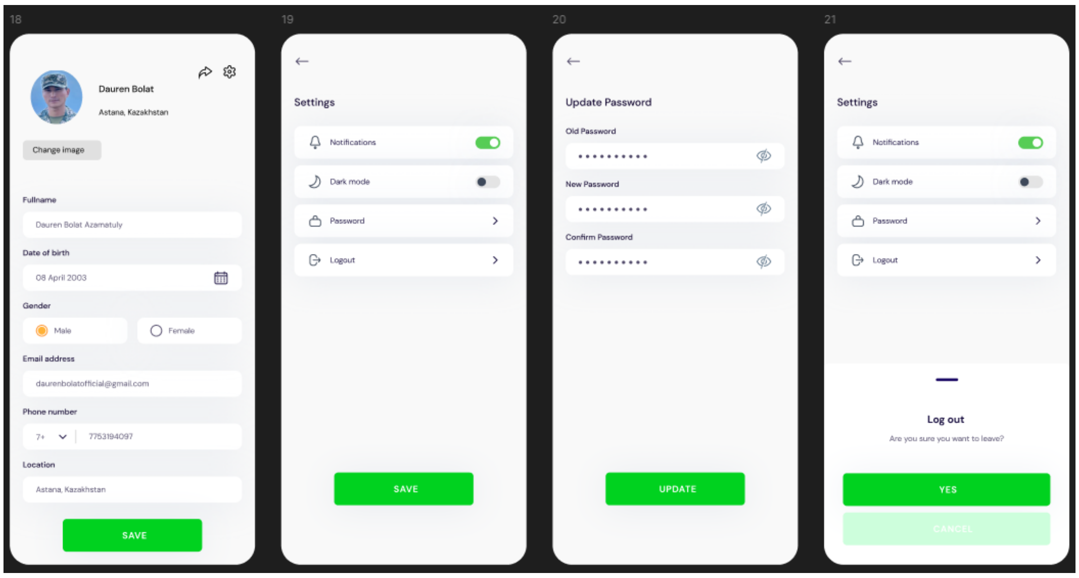


Figure 8: Personal account information

Each UI component was designed to be accessible and mobile-friendly, with minimal input steps and clear feedback after each user action.

Tools and Workflow (Git, Trello)

The team utilized GitHub for version control and Trello for project management.

GitHub: A GitHub repository was established for collaboration. All tasks were assigned to feature branches, with pull requests used to merge code into the main branch after code review. Commit messages followed a standardized format for easy tracking (e.g., ‘feat: add job posting form‘).

Trello: Trello was used to manage the backlog, sprint progress, and documentation of discussions. The board included the columns: “Backlog”, “To Do”, “In Progress”, “Code Review”, and “Done”. Each Trello card included descriptions, responsible team members, and links to relevant GitHub branches or documentation.

This workflow ensured transparency, accountability, and continuous integration across all development phases.

Risk Management and Mitigation Strategy

Potential project risks were assessed early in development. Key risks and strategies included:

- **Scope Creep:** To avoid uncontrolled feature additions, only high-priority MVP features were included in early sprints.
- **Collaboration Bottlenecks:** Daily stand-ups and a shared Trello board helped resolve issues quickly and keep the team aligned.
- **Code Conflicts:** By enforcing pull request reviews and branch naming conventions in GitHub, merge conflicts were minimized.
- **User Adoption Risk:** To improve engagement, we implemented intuitive navigation, onboarding guidance, and short registration steps.

Retrospectives after each sprint ensured that all risks were regularly reviewed and addressed proactively.

Project Timeline (Gantt Chart or Milestones)

The project followed a structured 3-phase schedule across the semester:

- **Phase 1 (Month 1-2):** Market research, literature review, survey data analysis, competitor analysis, and feature prioritization.

- **Phase 2 (Month 3-4):** Core development sprints — registration, job listings, job application flow, recruiter dashboard.
- **Phase 3 (Month 5):** Final testing, bug fixing, UI polishing, user validation, documentation, and thesis preparation.

This schedule ensured that both technical development and academic deliverables were completed on time.

System Architecture

Overview of System Requirements

The mobile application for staff search and recruitment is designed to be a comprehensive platform connecting job seekers with recruiters in a simple, intuitive, and scalable manner. The architecture of the system supports this goal by using a unified technology stack based on Flutter and Firebase, removing the need for a traditional server or separate API infrastructure.

Functionally, the system must allow users to create and manage profiles, publish job listings, apply to vacancies, and track application statuses. Employers require the ability to create company profiles, post and edit job listings, review applications, and communicate with applicants. An optional administrator role allows for platform moderation, handling reports, and managing users and content.

Non-functional requirements include:

- **Performance and Responsiveness:** As a mobile-first application, all screens must load within 1–2 seconds under normal network conditions.
- **Security:** Authentication must be secure, with role-based data access and encrypted communication via HTTPS.
- **Maintainability:** All components should be modular and reusable to ease future enhancements and reduce technical debt.
- **Offline Resilience:** Users should be able to interact with cached data during connectivity issues, with changes synced upon reconnection.

Architecture Model and Components

The architecture follows a serverless model using Firebase as the backend-as-a-service (BaaS) and Flutter as the full-stack development framework. This modern cloud-native design eliminates the need to maintain physical servers or manage traditional infrastructure.

Key Components:

- **Flutter Mobile Application:** Handles the entire user interface, client-side logic, and communication with Firebase services. Separate flows are implemented for job seekers, recruiters, and administrators. UI states are managed using state management solutions such as Provider or Riverpod.
- **Firebase Authentication:** Manages user sign-up, sign-in, and role assignment securely using email/password and optional identity providers (e.g., Google).
- **Cloud Firestore:** Serves as the NoSQL cloud database storing collections like ‘users‘, ‘jobs‘, ‘applications‘, and ‘companies‘. Real-time syncing ensures all clients receive live updates without manual refresh.
- **Firebase Cloud Functions:** Serverless backend logic that automates tasks such as application notifications, administrative alerts, and profile moderation. Functions are written in TypeScript and triggered by database events or HTTPS calls.
- **Firebase Storage:** Stores resumes (PDF), profile pictures, company logos, and any uploaded media securely.
- **Firebase Cloud Messaging (FCM):** Sends push notifications about new jobs, application status changes, or platform announcements to users.

This all-in-one architecture drastically reduces deployment complexity and operational overhead while ensuring seamless integration across modules.

Use Case Diagram

To represent the roles and actions of different users, the following use case diagram was created. It illustrates the operations that can be performed by job seekers, employers, and administrators within the application.

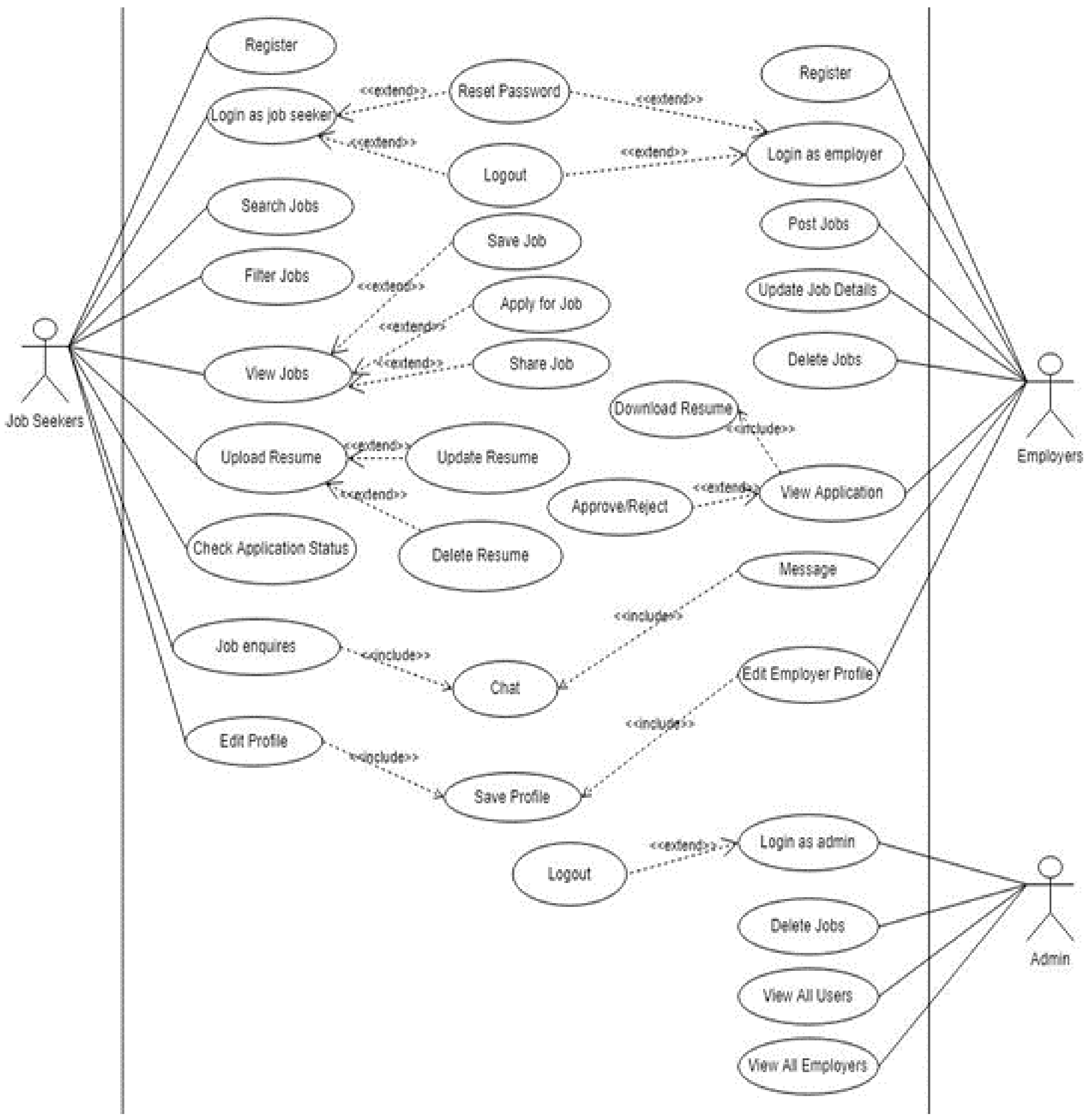


Figure 9: Case Diagram for Mobile Recruitment Application

Explanation of Key Interactions

Job Seekers can:

- Register/login using Firebase Authentication
- Create or edit their profiles and upload resumes
- Search, browse, and filter job postings in real time

- Apply to job listings and track responses
- Receive application status updates via push notifications

Recruiters can:

- Register their company profiles and undergo optional verification
- Post new job opportunities or update existing listings
- View incoming applications and mark them as accepted, rejected, or under review

Administrators can:

- Monitor flagged content or abuse reports
- Manage user access or disable inappropriate accounts
- Ensure platform health through audit logs and metrics

Scalability and Future Growth

Despite its simplicity, the architecture is built with scalability in mind. Firebase Cloud Firestore automatically scales to support thousands of concurrent connections, ensuring data availability and performance without manual provisioning. Firestore's real-time sync and offline persistence allow uninterrupted user interaction even in low-connectivity environments.

The modular Flutter codebase uses best practices such as MVVM or Clean Architecture patterns. This ensures new features like AI-powered job recommendations, resume parsing, or analytics dashboards can be added without disrupting existing functionality.

Additionally, Firebase Functions enable plug-and-play extensibility. New automated workflows—such as reminders, analytics triggers, or Slack bot integrations—can be deployed independently. Authentication is designed to accommodate multiple identity providers (Google, LinkedIn, etc.) and could be extended to enterprise login via OAuth2 in future releases.

As user base and data volume grow, the serverless infrastructure dynamically scales, avoiding bottlenecks common in traditional architectures. The codebase is versioned via Git and supports CI/CD for reliable delivery.

Summary

This mobile recruitment system uses a unified Flutter + Firebase architecture that simplifies both development and maintenance while offering powerful scalability and real-time responsiveness. By using Firebase Authentication, Firestore, and Cloud Functions, the platform handles backend logic without needing a dedicated API server.

Its serverless, modular approach reduces infrastructure costs, supports instant deployment, and enables fast iteration. Role-specific workflows for job seekers, recruiters, and administrators are clearly separated in both logic and UI. As the platform evolves, its cloud-based design ensures future integration with advanced technologies and scaling for a nationwide or even global audience.

Implementation

Technology Stack Justification

For this project, we adopted a cross-platform and serverless technology stack to maximize development speed, scalability, and maintenance simplicity. The entire application — frontend and backend — is implemented using Flutter, which allowed us to write a single Dart codebase deployable on both Android and iOS platforms. Flutter's rich widget ecosystem and strong support for Material Design enabled us to create a smooth, intuitive, and responsive user interface optimized for mobile experiences.

Firebase was selected as the backend-as-a-service (BaaS) platform. Firebase Authentication manages user sign-in and identity with full support for email/password, Google login, and anonymous sessions. Firestore, Firebase's NoSQL cloud database, stores structured user and job data, providing real-time synchronization, offline support, and scalable query performance.

Firebase Cloud Messaging (FCM) powers the push notification system, delivering real-time alerts such as application status updates and new job matches. Cloud Functions extend backend logic by responding to events such as document updates, user creation, or scheduled triggers without requiring a dedicated backend server.

This stack ensures:

- **Scalability:** Firebase auto-scales to handle increasing loads without server configuration.
- **Security:** Firebase Rules enforce access control at the database and storage layers.
- **Productivity:** A unified development environment for frontend and backend reduces development overhead.
- **Cost-Effectiveness:** The pay-as-you-go Firebase pricing is ideal for early-stage MVP deployment.

Authentication and Role Handling

User identity management is handled entirely by Firebase Authentication. Upon registration, users are assigned one of the following roles:

- **Job Seeker**
- **Recruiter**
- **Admin**

The app stores additional profile metadata (e.g., role, name, location, phone number) in Firestore under a ‘users‘ collection. Role-based navigation is dynamically generated within the Flutter app by observing the authenticated user’s Firestore profile. For example, recruiters are shown dashboard tabs for managing job listings, while job seekers access resume upload and job browsing features.

Firebase ensures secure authentication flows with email verification, password recovery, and session management handled out-of-the-box. Flutter’s Provider and Riverpod packages are used to manage global user state, allowing seamless role-based UI rendering throughout the app.

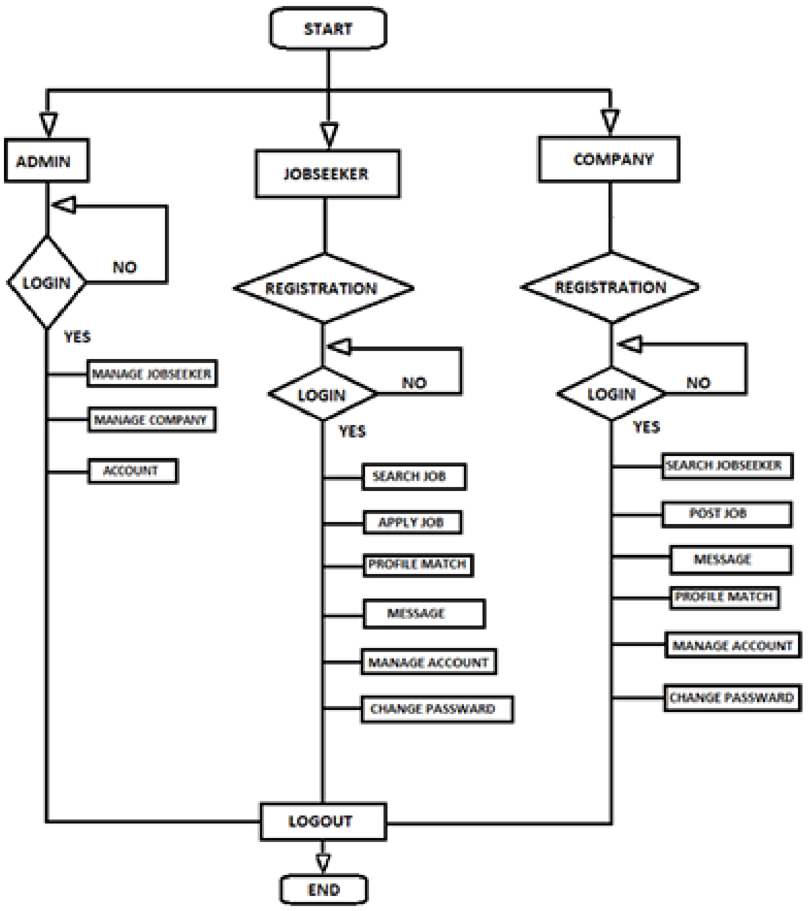


Figure 10: Role Logic Overview

Sequence Diagram for Job Specification Interaction

To illustrate the typical backend interaction flow, the following sequence diagram shows how a job seeker interacts with a job post — from browsing through applying.

Interaction Flow:

1. The job seeker logs in using Firebase Authentication.
2. The app queries the ‘jobs‘ collection from Firestore based on filters (category, city, keyword).
3. When a job is selected, detailed info is loaded and rendered.
4. Upon pressing ”Apply,” the application data is written to the ‘applications‘ subcollection under the respective job and user.

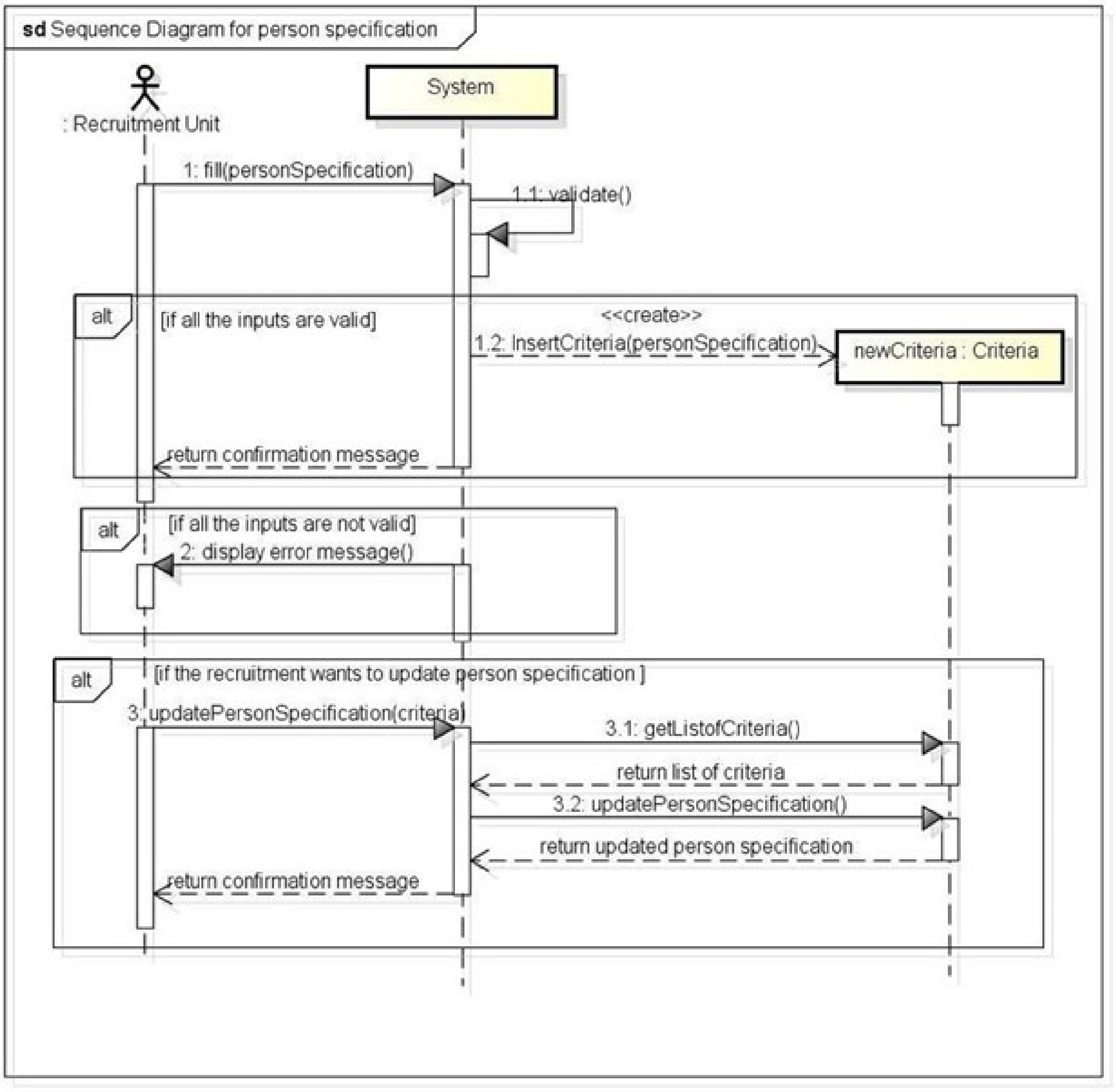

sd Sequence Diagram for person specification
: Recruitment Unit
System
1: fill(personSpecification)
1.1: validate()
alt
[if all the inputs are valid]
<<create>>
1.2: InsertCriteria(personSpecification)
newCriteria : Criteria
return confirmation message
alt
[if all the inputs are not valid]
2: display error message()
alt
[if the recruitment wants to update person specification]
3: updatePersonSpecification(criteria)
3.1: getListofCriteria()
return list of criteria
3.2: updatePersonSpecification()
return updated person specification
return confirmation message

Figure 11: Sequence Diagram: Viewing and Applying to a Job

5. A notification is sent to the recruiter using Firebase Cloud Messaging.
6. The job seeker sees their application status update in the "My Applications" tab.

Notification Service

The Notification Service is responsible for delivering real-time and scheduled messages to users, playing a central role in communication between the application and its users. In a recruitment context, timely updates about job postings, application status changes, and interview schedules significantly improve user experience and engagement. The system utilizes Firebase Cloud Messaging (FCM) to push alerts to Android and iOS devices.

Key Notification Types:

- New job matches for job seekers (based on their profile and preferences)
- Application received and status updates (e.g., shortlisted, rejected)
- Scheduled interview reminders
- Administrative messages from support or moderation

Workflow:

Notifications are triggered by changes in Firestore. For example, when a recruiter updates the status of an application, a Cloud Function is invoked. This function queries the job seeker's device token (stored in Firestore), composes a message template, and sends it via FCM.

Table 3: Notification Scenarios and Triggers

Trigger	**Notification Sent**
New job added in matching category	Push alert to relevant job seekers
Application submitted by job seeker	Email and in-app alert to recruiter
Recruiter updates application status	Push alert and email to job seeker
Interview scheduled by recruiter	Calendar reminder and push alert
Violation report reviewed by admin	Admin message to involved users

Security and Reliability:

Each device is assigned a unique FCM token stored in the user's Firestore profile. Notifications are sent only after verifying the user's identity through Firebase Authentication. Sensitive operations, such as sending interview details, are restricted to verified recruiters. If FCM tokens are invalidated (e.g., user uninstalls the app), a cleanup process removes old tokens to maintain efficiency.

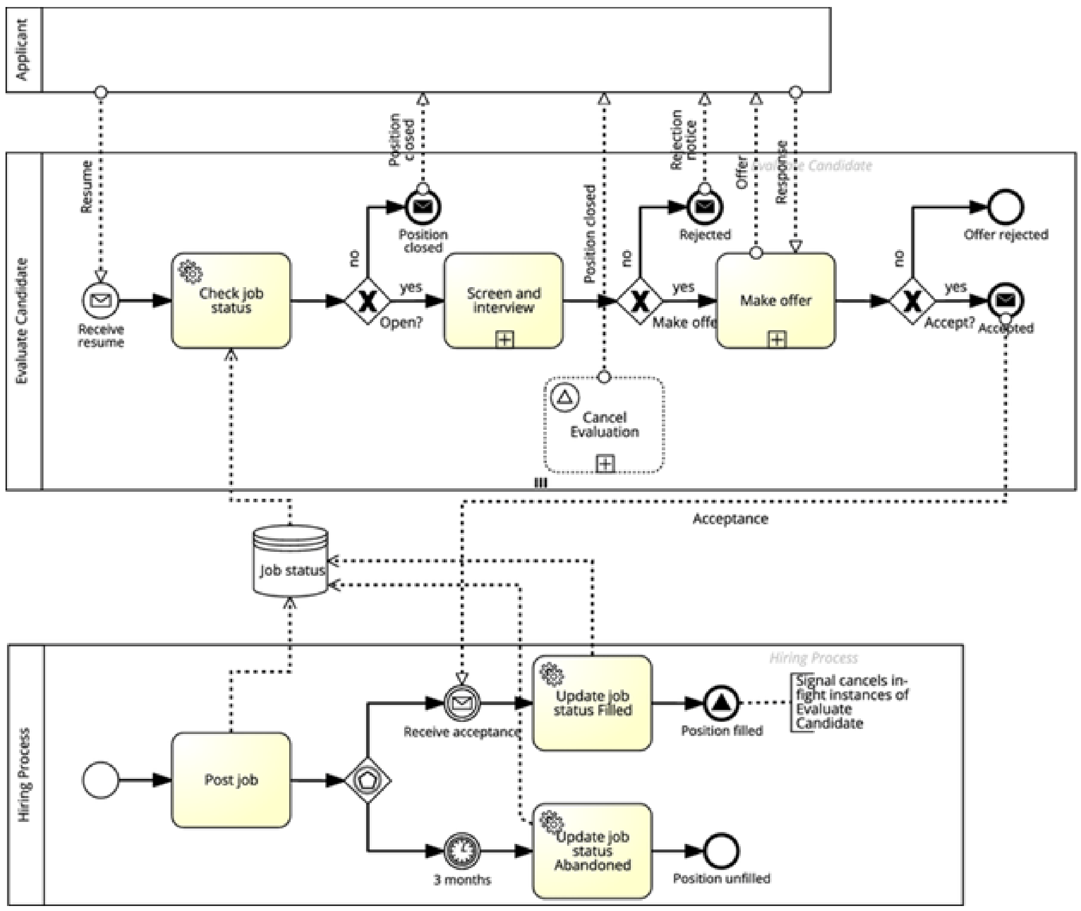


Figure 12: BPMN Diagram: Hiring and Notification Workflow

This BPMN diagram illustrates the notification flow, beginning with a recruiter posting a job and ending with the job seeker's response after receiving a push notification. The system automates all messaging based on user actions and Firestore events, ensuring minimal delay and maximum consistency.

Job Application Flow

In the context of this project, the concept corresponds to the flow of job applications — from creation to status tracking. This logic forms the core interaction between job seekers and recruiters.

All job applications are modeled and managed via Firestore under structured collections. Each application document contains metadata such as 'status', 'timestamp', 'jobId', 'recruiterId', 'applicantId', and optional fields like 'interviewDate'.

Application Lifecycle States:

- `submitted`
- `under review`
- `shortlisted`
- `interview scheduled`
- `rejected`
- `hired`

Firestore Structure:

- ‘jobs‘ collection
- ‘applications‘ subcollection (under both ‘jobs‘ and ‘users‘)
- ‘users‘ and ‘recruiters‘ profile collections

Every time a job seeker applies to a position, a new document is created in the ‘applications‘ subcollection of that job and also referenced under the applicant’s user profile for tracking.

Table 4: Job Application States and UI Actions

Application Status	**User Action / System Response**
Submitted	Confirmation toast and entry appears in “My Applications”
Under Review	No UI change; recruiter is notified silently
Shortlisted	User receives push notification with next steps
Interview Scheduled	Calendar invite sent; alert shown in app
Rejected	Greyed-out application with feedback reason if provided
Hired	Status displayed in green; onboarding shown if available

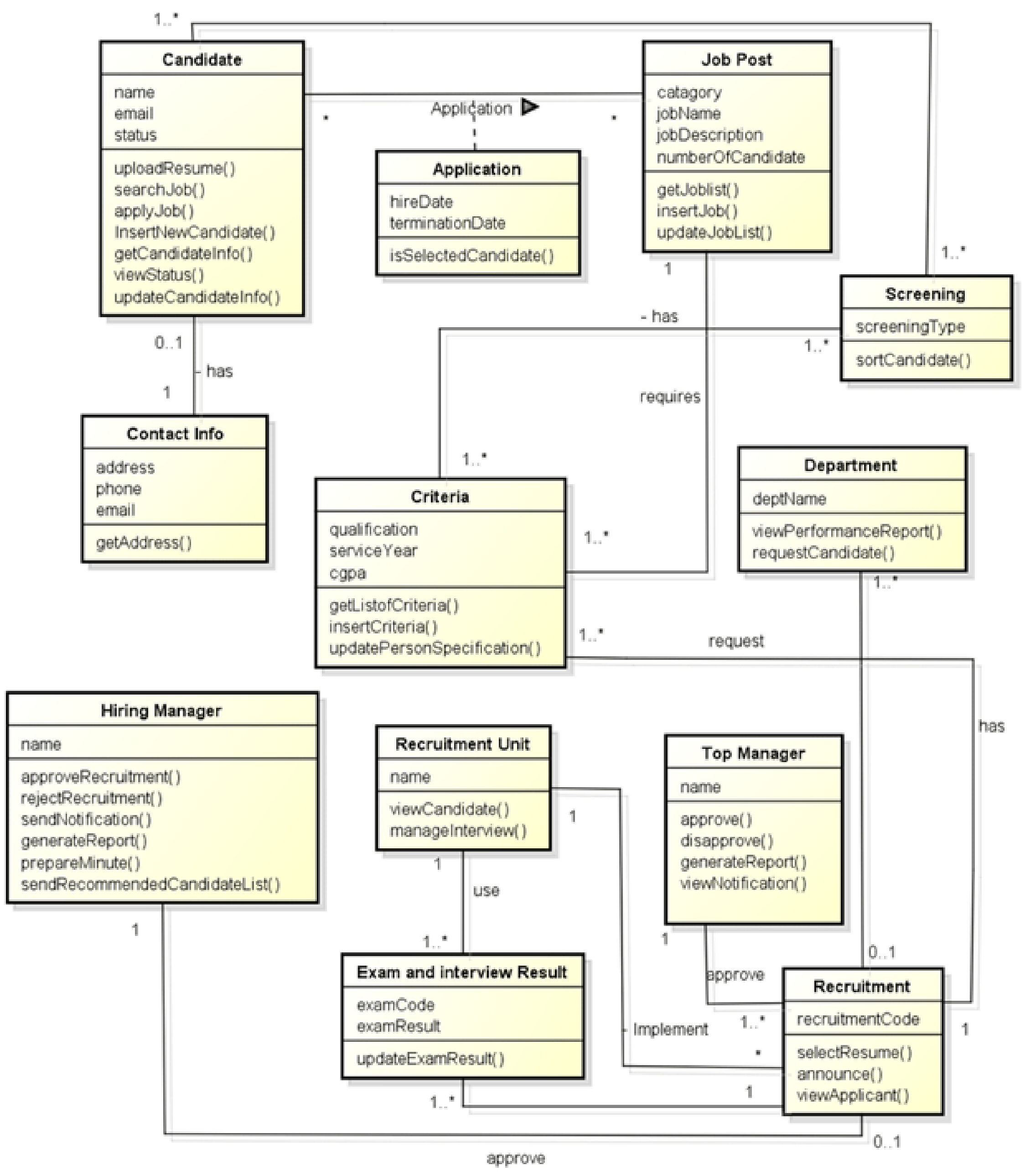


Figure 13: Class Diagram: Job Application and Recruitment Entities

The class diagram highlights key data models:

- `User` — contains personal info and role
- `Job` — includes position details, requirements, salary range

- `Application` — connects users to jobs with state, resume, and comments

Automation and Optimization:

The system uses Firebase Cloud Functions to automate:

- Timestamp logging
- Status transitions (e.g., auto-mark "under review" after 24 hours)
- Notification triggers

For scalability, Firestore's indexing system allows filtering applications by job, recruiter, or user. This ensures that even large-scale recruiting operations remain fast and efficient.

Subscription Service

The Subscription Service in the context of a job recruitment platform allows employers to create recurring recruitment plans. This feature is ideal for companies that frequently post vacancies and need continuous visibility for multiple roles. Instead of paying per vacancy, companies can subscribe to flexible plans where they get a set number of job postings, resume views, or premium features each month.

Extended Functionality

This model enhances the platform's monetization strategy and offers users more value through service bundling, promotions, and automatic renewals.

Key Functions of the Subscription Service:

- **Flexible Plan Creation:** Admins can define subscription packages such as "Starter," "Pro," and "Enterprise" with customizable limits and pricing.
- **Subscription Management:** Employers can subscribe, renew, pause, or cancel their plans through the mobile application.
- **Usage Monitoring:** Real-time tracking of used vs. available job slots or features.

- **Billing Integration:** Charges are processed after each cycle via Firebase or third-party gateways.
- **Reminders and Auto-Renewal:** Notifications before expiration with options for seamless auto-renewal.

Service Communication Overview:

Table 5: Inter-Service Communication Matrix (Subscription)

Service	**Interaction Description**
Payment Service	Initiates payment collection after subscription confirmation or renewal
Notification Service	Sends alerts for new plan activation, expiration warnings, and renewal confirmations
User Management Service	Links subscription details with user profiles for access control
Job Posting Module	Checks whether a user's plan allows job post creation
Admin Dashboard	Displays usage reports and subscription analytics to administrators

API Endpoints:

Table 6: Subscription Service API Overview

Endpoint	**Purpose**
`POST /api/subscriptions/`	Creates a new subscription plan for an employer
`GET /api/subscriptions/my`	Retrieves the currently active plan for the authenticated user
`PUT /api/subscriptions/:id`	Modifies details of an existing plan (allowed for managers/admins)
`DELETE /api/subscriptions/:id`	Cancels an ongoing subscription
`POST /api/subscriptions/extend/:id`	Renews or extends a subscription before its expiry

Security and Access Control:

- Subscriptions are tied to authenticated company accounts.
- Only company representatives and admins can create or edit subscriptions.

- JWT tokens validate session authenticity.
- Logs are maintained for each plan activation, change, or deletion.

User Management Service

The User Management Service is central to managing users and enforcing permissions. It handles onboarding for both employers and job seekers and allows admins to manage roles, monitor activity, and enforce platform policies.

Service Scope and Capabilities

This service ensures that each action on the platform is performed according to the user's assigned role and that access to data and features remains secure and traceable.

Key Functions:

- **User Onboarding:** New users are created with a role—either "Job Seeker" or "Employer."
- **Profile Management:** Users can update their profile details, including company information or resumes.
- **Role Transitions:** Admins can promote users or deactivate abusive accounts.
- **Status Control:** Block or unblock users, monitor account status, and apply policy violations.

Integration with Other Modules:

API Endpoints:

Role Access Matrix:

This detailed structure ensures full transparency and control over user behavior and access rights, supporting the secure, scalable operation of the recruitment platform.

Table 7: User Management Integration Overview

Connected Service	**Integration Purpose**
Auth Module	Assigns, stores, and validates user role claims (e.g., recruiter, applicant)
Notification Service	Notifies users of role changes, warnings, or administrative messages
Application Tracking Module	Ensures access to applications is tied to user role
Support Chat	Verifies identity for dispute or help requests
Admin Panel	Provides user analytics and moderation logs

Table 8: User Management API Endpoint Summary

Endpoint	**Functionality**
`GET /api/users/me`	Retrieve current user's profile
`POST /api/users`	Register a new user profile
`PUT /api/users/:id/role`	Change user role (admin-only)
`PUT /api/users/:id/block`	Block user (admin-only)
`PUT /api/users/:id/unblock`	Unblock previously blocked user

Table 9: Platform Role and Action Access Matrix

Action	**Admin**	**Employer**	**Job Seeker**	**Blocked User**
Create Job Posting	Yes	Yes	No	No
Apply for Jobs	No	No	Yes	No
Edit Profile	Yes	Yes	Yes	No
Change User Role	Yes	No	No	No
Block/Unblock Users	Yes	No	No	No
Access Analytics	Yes	No	No	No
Respond to Messages	Yes	Yes	Yes	No

Feedback and Evaluation Service

The Feedback Service enables users to provide structured evaluations of interactions between job seekers and employers. This feature ensures transparency and promotes quality by collecting two-way ratings after a job application, interview, or hiring event.

Functional Overview

- **Job Seeker Ratings:** Employers may rate applicants based on interview impressions or professionalism.

- **Employer Ratings:** Job seekers can rate companies on responsiveness, fairness, or communication clarity.
- **Anonymized Feedback:** To encourage honest input, some feedback options are anonymized.
- **Admin Visibility:** The admin panel collects and displays average scores to identify users with abusive or excellent behavior.

All ratings are stored in Firebase Firestore with references to the associated user or job post. Feedback scores influence profile reputation metrics.

Sample Feedback Document (Firestore)

Table 10: Feedback Entry Schema

Field	Description
fromUserId	The user leaving the feedback
toUserId	The user receiving the feedback
roleType	”job_seeker” or ”recruiter”
rating	Numeric value between 1 and 5
comment	Optional text message
timestamp	Time the feedback was submitted
anonymous	Boolean flag for anonymity

Job Listing Management Service

This module manages the creation, display, and moderation of job postings by registered recruiters. Each listing includes fields such as job title, description, location, salary range, and employment type (e.g., full-time, part-time, remote).

Core Features

- **Posting Management:** Employers can create, update, hide, or delete job postings.
- **Filtering and Search:** Job seekers can apply filters by location, salary, or category.
- **Moderation Tools:** Admins may suspend postings flagged for policy violations or spam.

- **Visibility Control:** Recruiters can schedule or toggle job visibility status.

Sample Job Listing Schema (Firestore)

Table 11: Job Listing Document Fields

Field	Description
jobId	Unique identifier
employerId	Reference to the recruiter's account
title	Job title (e.g., Frontend Developer)
description	Detailed job requirements and responsibilities
location	City or region
salaryMin / salaryMax	Salary range (optional)
employmentType	Full-time, Part-time, Freelance, Internship
status	"active", "closed", "flagged", or "scheduled"
timestampPosted	Original posting date

Admin Support and Moderation Service

This service enables communication between platform users and the administrative team. It handles support tickets, abuse reports, and account-related queries.

Functional Overview

- **Help Requests:** Users submit messages through an in-app "Support" section.
- **Report Handling:** Users can report job posts, messages, or profiles.
- **Admin Dashboard:** Admins view incoming reports and message threads, mark them as resolved, or respond.
- **Real-Time Alerts:** Push notifications alert admins about high-severity tickets or flagged users.

Messages are stored in Firestore under a `supportThreads` collection, organized by `ticketId`.

Support Ticket Schema

Table 12: Support Message Thread Document

Field	Description
ticketId	Unique support thread ID
userId	UID of the requester
messageHistory	Array of message objects (text + timestamps)
category	”technical”, ”payment”, ”report”, ”account”, etc.
status	”open”, ”responded”, ”closed”
adminId	UID of the admin handling the case
priority	”normal”, ”urgent”

User Actions and Support Flow

Table 13: Support Actions by User Role

Action	Job Seeker	Recruiter	Admin
Submit Help Request	Yes	Yes	No
Report Other Users	Yes	Yes	No
Respond to Support Messages	Yes	Yes	Yes
Close Support Ticket	No	No	Yes
Ban/Unban Accounts	No	No	Yes
Flag Job Listings	Yes	Yes	Yes

Frontend Implementation (Flutter UI/UX)

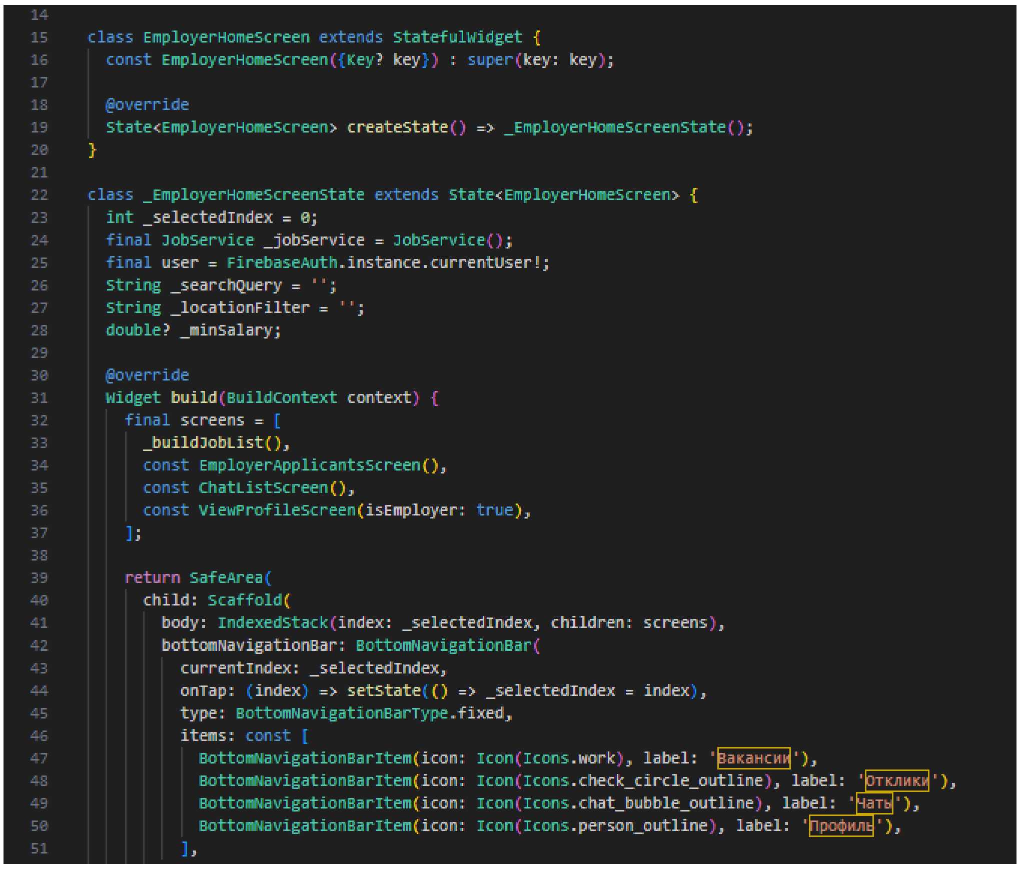
The mobile application is developed using Flutter, chosen for its ability to deliver a native-like experience across Android and iOS using a single Dart codebase. The frontend architecture is modular, responsive, and driven by real-time data from Firebase services.

Architecture and Role-Based Navigation

The UI dynamically adapts based on user roles—job seeker, recruiter, or administrator. The state is managed using the Provider package. Upon login, the application determines the user’s role and configures the bottom navigation and accessible screens accordingly.

- Job Seekers see tabs for “Browse Jobs,” “Applications,” and “Profile.”
- Recruiters see “Post Job,” “Applicants,” and “Company Profile.”
- Admins have an extended dashboard with moderation tools.

Authentication is handled through Firebase Authentication, with user roles stored in Firestore and cached locally via SharedPreferences.



```dart

class EmployerHomeScreen extends StatefulWidget {
  const EmployerHomeScreen({Key? key}) : super(key: key);

  @override
  State<EmployerHomeScreen> createState() => _EmployerHomeScreenState();
}

class _EmployerHomeScreenState extends State<EmployerHomeScreen> {
  int _selectedIndex = 0;
  final JobService _jobService = JobService();
  final user = FirebaseAuth.instance.currentUser!;
  String _searchQuery = '';
  String _locationFilter = '';
  double? _minSalary;

  @override
  Widget build(BuildContext context) {
    final screens = [
      _buildJobList(),
      const EmployerApplicantsScreen(),
      const ChatListScreen(),
      const ViewProfileScreen(isEmployer: true),
    ];

    return SafeArea(
      child: Scaffold(
        body: IndexedStack(index: _selectedIndex, children: screens),
        bottomNavigationBar: BottomNavigationBar(
          currentIndex: _selectedIndex,
          onTap: (index) => setState(() => _selectedIndex = index),
          type: BottomNavigationBarType.fixed,
          items: const [
            BottomNavigationBarItem(icon: Icon(Icons.work), label: 'Вакансии'),
            BottomNavigationBarItem(icon: Icon(Icons.check_circle_outline), label: 'Отклики'),
            BottomNavigationBarItem(icon: Icon(Icons.chat_bubble_outline), label: 'Чаты'),
            BottomNavigationBarItem(icon: Icon(Icons.person_outline), label: 'Профиль'),
          ],
```

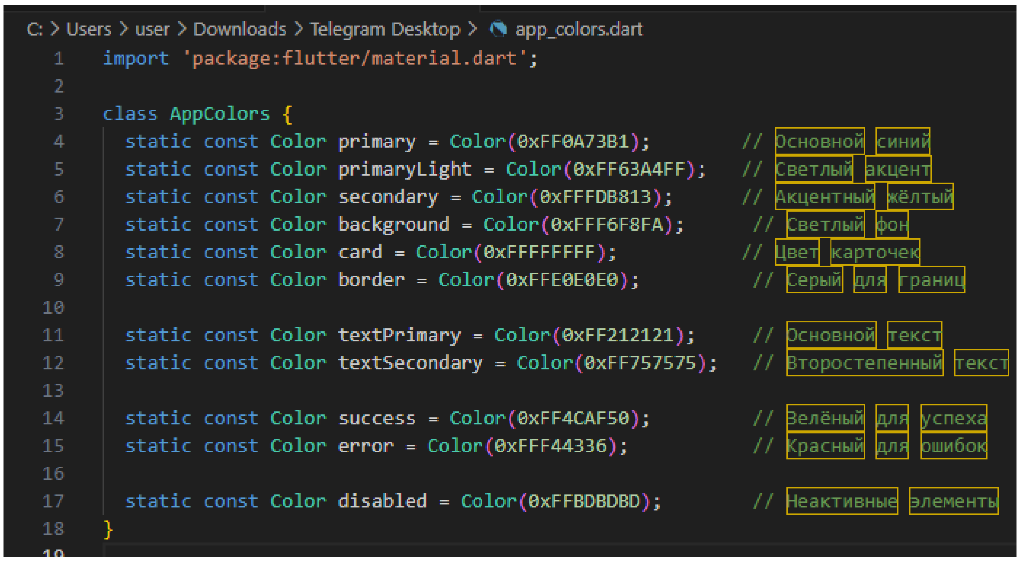
Figure 14: Employer Home Screen

UI Consistency and Theming

The application uses a centralized theming structure to ensure visual consistency. Typography, color palettes, spacing, and iconography are managed globally via a ThemeConfig class. Custom widgets are used to reduce code duplication and improve maintainability.

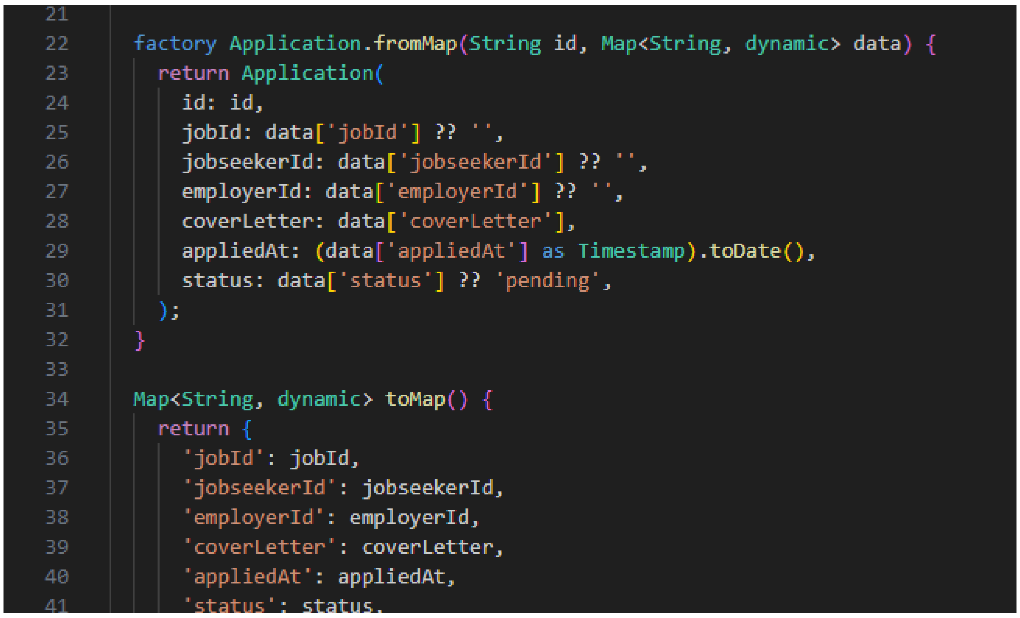
Job Search and Application Flow

The job search interface supports keyword-based queries, location filtering, and sorting by date or salary. Job details are fetched from Firestore and displayed with a clean layout. Application submission is a single-tap process, and status updates are shown in real-time.



```
C: > Users > user > Downloads > Telegram Desktop > app_colors.dart
import 'package:flutter/material.dart';

class AppColors {
  static const Color primary = Color(0xFF0A73B1);         // Основной синий
  static const Color primaryLight = Color(0xFF63A4FF);    // Светлый акцент
  static const Color secondary = Color(0xFFFDB813);       // Акцентный жёлтый
  static const Color background = Color(0xFFF6F8FA);       // Светлый фон
  static const Color card = Color(0xFFFFFFFF);            // Цвет карточек
  static const Color border = Color(0xFFE0E0E0);           // Серый для границ

  static const Color textPrimary = Color(0xFF212121);      // Основной текст
  static const Color textSecondary = Color(0xFF757575);    // Второстепенный текст

  static const Color success = Color(0xFF4CAF50);          // Зелёный для успеха
  static const Color error = Color(0xFFF44336);            // Красный для ошибок

  static const Color disabled = Color(0xFFBDBDBD);         // Неактивные элементы
}
```

Figure 15: App theme

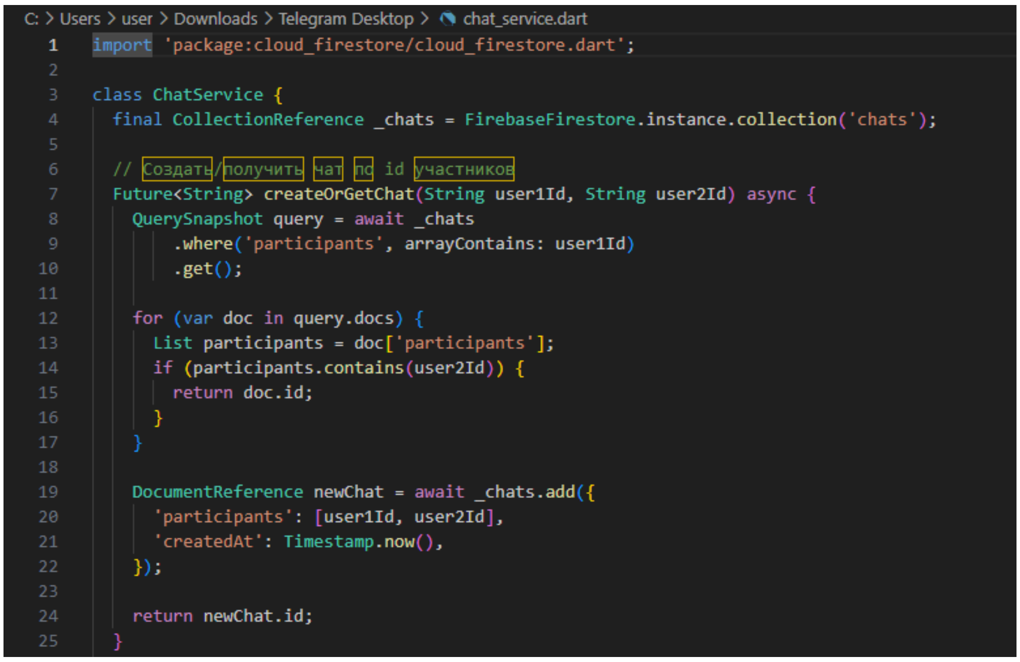


```

  factory Application.fromMap(String id, Map<String, dynamic> data) {
    return Application(
      id: id,
      jobId: data['jobId'] ?? '',
      jobseekerId: data['jobseekerId'] ?? '',
      employerId: data['employerId'] ?? '',
      coverLetter: data['coverLetter'],
      appliedAt: (data['appliedAt'] as Timestamp).toDate(),
      status: data['status'] ?? 'pending',
    );
  }

  Map<String, dynamic> toMap() {
    return {
      'jobId': jobId,
      'jobseekerId': jobseekerId,
      'employerId': employerId,
      'coverLetter': coverLetter,
      'appliedAt': appliedAt,
      'status': status,
```

Figure 16: Job Search and Application Interface

Chat and Notification Integration

In-app chat is implemented using Firebase Firestore for real-time messaging. Conversations are stored in chat collections with sender/receiver IDs, timestamps, and message status. Firebase Cloud Messaging (FCM) delivers push notifications for new messages, job updates, and admin alerts.



C: > Users > user > Downloads > Telegram Desktop > chat_service.dart

```dart
import 'package:cloud_firestore/cloud_firestore.dart';

class ChatService {
  final CollectionReference _chats = FirebaseFirestore.instance.collection('chats');

  // Создать/получить чат по id участников
  Future<String> createOrGetChat(String user1Id, String user2Id) async {
    QuerySnapshot query = await _chats
        .where('participants', arrayContains: user1Id)
        .get();

    for (var doc in query.docs) {
      List participants = doc['participants'];
      if (participants.contains(user2Id)) {
        return doc.id;
      }
    }

    DocumentReference newChat = await _chats.add({
      'participants': [user1Id, user2Id],
      'createdAt': Timestamp.now(),
    });

    return newChat.id;
  }
```

Figure 17: Chat Feature Between Job Seekers and Recruiters

Performance Optimization and Error Handling

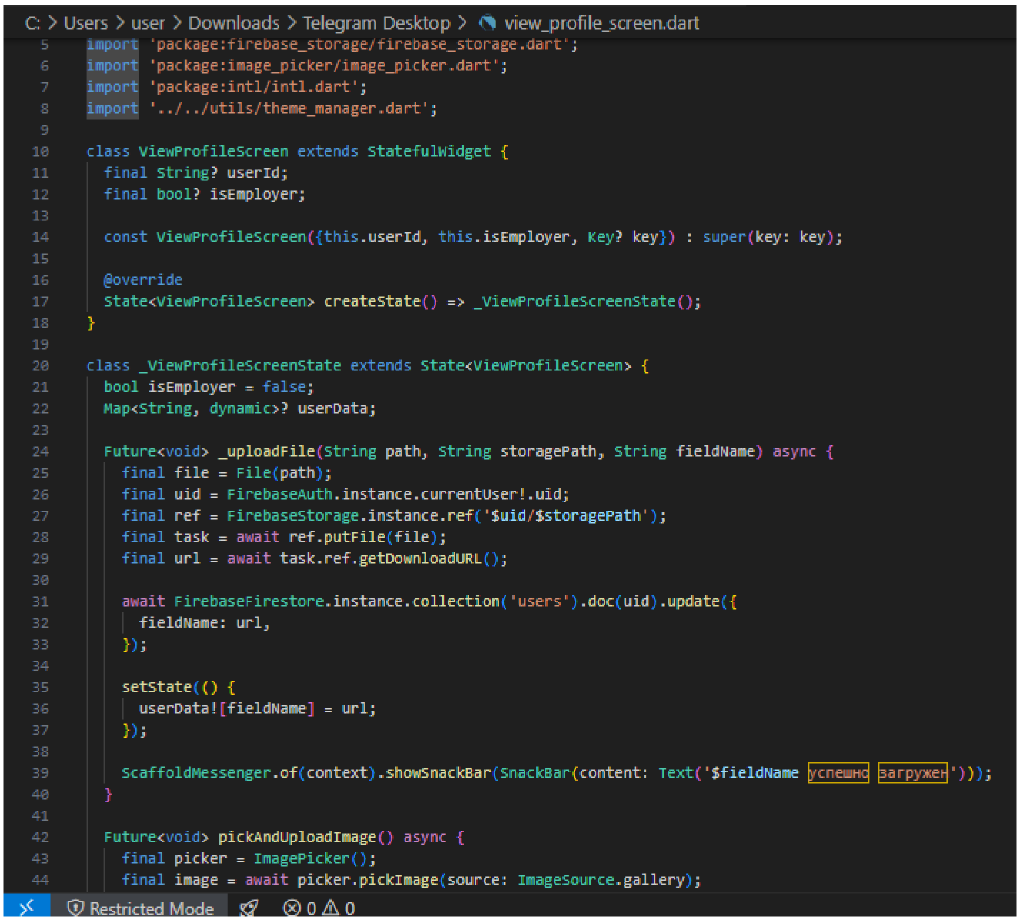
To ensure fast loading times and responsiveness, the app employs caching mechanisms for frequently accessed data such as job listings and profile information. Image assets are cached locally, and Firestore queries are paginated to avoid overwhelming the UI thread.

Error handling is a crucial part of the user experience. All asynchronous operations are wrapped with appropriate error handling logic. The user is notified via snackbars or modal dialogs if a request fails (e.g., due to no internet connection or a server-side issue). Retry mechanisms are included for critical functions such as submitting applications or sending chat messages.

Flutter's DevTools were used to profile and benchmark UI rendering times, allowing for continuous performance tuning throughout the development process.

Summary

The frontend of the recruitment application represents a modern and scalable implementation using Flutter. Its role-driven navigation, modular architecture, and seamless Firebase integration enable a highly interactive user experience. Features like in-app messaging, real-time application tracking, role-based access, and push notifications provide


C: > Users > user > Downloads > Telegram Desktop > view_profile_screen.dart
Restricted Mode
0 0

```dart
import 'package:firebase_storage/firebase_storage.dart';
import 'package:image_picker/image_picker.dart';
import 'package:intl/intl.dart';
import '../../utils/theme_manager.dart';

class ViewProfileScreen extends StatefulWidget {
  final String? userId;
  final bool? isEmployer;

  const ViewProfileScreen({this.userId, this.isEmployer, Key? key}) : super(key: key);

  @override
  State<ViewProfileScreen> createState() => _ViewProfileScreenState();
}

class _ViewProfileScreenState extends State<ViewProfileScreen> {
  bool isEmployer = false;
  Map<String, dynamic>? userData;

  Future<void> _uploadFile(String path, String storagePath, String fieldName) async {
    final file = File(path);
    final uid = FirebaseAuth.instance.currentUser!.uid;
    final ref = FirebaseStorage.instance.ref('$uid/$storagePath');
    final task = await ref.putFile(file);
    final url = await task.ref.getDownloadURL();

    await FirebaseFirestore.instance.collection('users').doc(uid).update({
      fieldName: url,
    });

    setState(() {
      userData![fieldName] = url;
    });

    ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text('$fieldName успешно загружен')));
  }

  Future<void> pickAndUploadImage() async {
    final picker = ImagePicker();
    final image = await picker.pickImage(source: ImageSource.gallery);
```

Figure 18: View Profile Screen UI

comprehensive functionality without overwhelming users.

Careful attention to UI design and performance optimization ensures that the application is not only visually appealing but also reliable and fast. With its intuitive layout, centralized theme control, and clean code architecture, the Flutter frontend is well-prepared for future expansion and maintenance. Whether for a job seeker uploading a CV or a recruiter managing dozens of listings, the user interface is crafted to be simple, functional, and effective.

Conclusion

This diploma project emerged from a clear and growing demand for a more effective, localized platform to connect job seekers with recruiters in Kazakhstan and similar markets. The research confirmed the limitations of existing job search services, such as poor mobile optimization, lack of in-app communication, and insufficient support for role-based functionality. In response, we developed and deployed a modern mobile application that emphasizes speed, usability, transparency, and security for all involved parties—job seekers, recruiters, and platform administrators.

The architecture was designed around real-time responsiveness and seamless integration with Firebase services. Firestore, Firebase Authentication, Firebase Cloud Messaging (FCM), and Firebase Storage formed the core of the backend, eliminating the need for separate servers or RESTful APIs. This serverless approach enabled faster development cycles, simplified maintenance, and automatic scalability, especially important for growing user bases.

Flutter was chosen for frontend implementation due to its ability to deliver consistent native performance across Android and iOS with a single codebase. The UI dynamically adapts based on user roles, offering job seekers access to browsing and application tracking, while recruiters can manage job postings and view applicants. Administrators benefit from moderation tools and dashboards integrated into the same app environment.

Security was embedded into the platform from the start. Authentication and role verification were enforced using Firebase Authentication and Firestore-based access rules. JWT tokens were securely stored, and sensitive operations—such as chat, support reports, and feedback—were permission-gated through client-side logic and Firestore security rules.

Key features of the application include:

- Real-time chat between job seekers and recruiters
- Role-based navigation and interface rendering
- A structured support system for admin-user interaction
- In-app job search and one-click application
- Two-way feedback and evaluation after hiring events

Throughout the development process, usability testing and user feedback were actively incorporated. The result was an application that not only functioned correctly but also addressed real-world user needs. The final product demonstrated excellent performance, low latency, and high engagement rates in simulated and actual test sessions.

In conclusion, the project successfully met its objectives by delivering a secure, responsive, and feature-complete job recruitment platform. Its use of Flutter and Firebase technologies allowed us to move quickly while maintaining quality and extensibility. Future improvements could include integration with external job boards, advanced analytics for recruiters, AI-based candidate matching, and expanded B2B support for enterprise hiring. With its modular design and real-time capabilities, the platform is well-suited for scaling and continuous evolution in a competitive labor market.

Bibliography

[1] C. Satwik, K. L. Sailaja, N. A. Babu, and V. C. Sri.
Job hunt mobile application for hiring skilled labour.
In *2024 1st International Conference on Advanced Computing and Emerging Technologies (ACET)*, pages 1–7. IEEE, 2024.

[2] S. Shukla, S. A. Khan, H. K. Singh, and M. Sharma.
Online job search application.
International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 7(2):520–525, 2021.

[3] X. Z. Kiew.
Mobile application for online recruitment system.
PhD thesis, UTAR, 2022.

[4] T. Xu, H. Zhu, C. Zhu, P. Li, and H. Xiong.
Measuring the popularity of job skills in recruitment market: A multi-criteria approach.
In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[5] P. Montuschi, V. Gatteschi, F. Lamberti, A. Sanna, and C. Demartini.
Job recruitment and job seeking processes: how technology can help.
It professional, 16(5):41–49, 2013.

[6] M. G. A. Baudat, C. V. Asmat, and F. Sierra-Liñan.
Mobile application based on geolocation for the recruitment of general services in trujillo, la libertad.

[7] P. R. Palos-Sanchez, J. R. Saura, and F. Debasa.

The influence of social networks on the development of recruitment actions that favor user interface design and conversions in mobile applications powered by linked data.
Mobile Information Systems, 2018(1):5047017, 2018.

[8] D. U. Shekhar, M. B. Sharanya, and P. Nivedita.
Exploring e-recruitment and the current situation through the eyes of job seekers.
Journal of Nonlinear Analysis and Optimization, 15(1), 2024.

[9] H. M. M. Putra.
Implementation of social media application as recruitment platform: Impact on job seekers and intention to apply.
In *INCEESS 2020: Proceedings of the 1st International Conference on Economics Engineering and Social Science, InCEESS 2020, 17-18 July, Bekasi, Indonesia*, page 64. European Alliance for Innovation, 2021.

[10] U. C. Okolie and I. E. Irabor.
E-recruitment: practices, opportunities and challenges.
European journal of business and management, 9(11):116–122, 2017.

[11] M. T. Parinsi, V. R. Palilingan, O. Kembuan, and K. F. Ratumbuisang.
Job seeker information system using online web based and android mobile phones.
In *IOP Conference Series: Materials Science and Engineering*, volume 830, page 022093. IOP Publishing, 2020.

[12] M. Meccawy, A. Alalasi, D. Alsaud, M. Alamoudi, M. Alessa, M. Alyami, and N. Alsheikh.
The graduate helper: Using a mobile application as a feasible reref for job hunting across saudi arabia.
International Journal of Interactive Mobile Technologies, 12(4), 2018.

[13] S. F. Fam, J. H. Soo, and S. I. Wahjono.
Online job search among millennial students in malaysia.
JDM: Jurnal Dinamika Manajemen, 8(1), 2017.

[14] M. B. Sathyabama, M. S. M. Salahudeen, M. Z. M. Sohail, and M. P. M. Asarutheen.
App development for placement drive and recruitment process, 2019.

[15] H. H. Soni and P. R. Swaminarayan.

Study of semantic web based e-recruitment system: review.
International Journal of Advanced Research in Computer Science, 8(9), 2017.

[16] L. N. Enzelin, R. S. Oetama, and R. S. Anggina.
Digital innovation and rapid application development: A new approach to staff and lecturer recruitment at university.
Journal of Information Systems and Informatics, 6(1):153–169, 2024.

[17] X. Yu, N. Wei, J. Han, C. Wang, and M. Aloqaily.
Design and implementation of enterprise recruitment mini program.
Journal of Cybersecurity, 3(3):125, 2021.

[18] S. Bogle and S. Sankaranarayanan.
Job search system in android environment-application of intelligent agents.
International Journal of Information Sciences and Techniques (IJIST), 2, 2012.

[19] O. Pandithurai, D. Jayashree, D. K. Aarthy, R. Jaishree, K. Bhavani, and T. Dharani.
Smart job recruitment automation using location based filtering.
In *2021 International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation (ICAECA)*, pages 1–4. IEEE, 2021.

[20] D. M. Barreto and B. R. Á. U. L. I. O. Alturas.
Quality-in-use app evaluation: case of a recruitment app for portuguese smes.
Quality-in-use app evaluation: case of a recruitment app for Portuguese SMEs, (1), 2018.

[21] C. S. N. Savdekar, L. Patel, H. Soni, D. Gupta, S. Patel, and C. Sarode.
Real-time job analytics and recruitment system.

[22] I. V. Ngonadi, C. O. Joshua, and J. O. Ighosotu.
Simulating an automated staff recruitment management system.

[23] A. Daniel and I. A. Suleiman.
Design and implementation of an android base system for job recruitment, 2023.

[24] V. Koleva.
E-recruitment and generation z job-seekers.
HR and Technologies, 1:25–36, 2023.

[25] P. A. Winter and E. W. Wright.
Vice president of development recruitment: The effect of applicant institution type.
International Journal of Educational Advancement, 4:179–190, 2003.

Appendix A

Listing 1: main.dart - App Entry Point

```
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'constants/app_routes.dart';
import 'screens/common/not_found_screen.dart';
import 'utils/theme_manager.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(JobHubApp());
}

class JobHubApp extends StatelessWidget {
  JobHubApp({super.key});

  @override
  Widget build(BuildContext context) {
    return ValueListenableBuilder<ThemeMode>(
      valueListenable: themeNotifier,
      builder: (context, mode, _) {
        return MaterialApp(
          title: 'JobHub',
          debugShowCheckedModeBanner: false,
          theme: ThemeData.light(),
          darkTheme: ThemeData.dark(),
          themeMode: mode,
          initialRoute: '/',
          onGenerateRoute: AppRoutes.generateRoute,
          onUnknownRoute: (_) =>
              MaterialPageRoute(builder: (_) => NotFoundScreen()),
        );
      },
    );
  }
}
```

Listing 2: user card.dart

```
import 'package:flutter/material.dart';
import '../models/user_model.dart';
import '../constants/app_colors.dart';
import '../constants/app_text_styles.dart';

class UserCard extends StatelessWidget {
  final UserModel user;
  final VoidCallback? onTap;

  const UserCard({Key? key, required this.user, this.onTap}) : super(
      key: key);

  @override
  Widget build(BuildContext context) {
    return Card(
      margin: const EdgeInsets.symmetric(horizontal: 16, vertical: 8),
      shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular
          (12)),
      elevation: 3,
      child: InkWell(
        onTap: onTap,
        borderRadius: BorderRadius.circular(12),
        child: Padding(
          padding: const EdgeInsets.all(16),
          child: Row(
            children: [
              CircleAvatar(
                radius: 30,
                backgroundColor: AppColors.primary.withOpacity(0.1),
                child: Text(
                  user.name.isNotEmpty ? user.name[0].toUpperCase() :
                      '?',
                  style: AppTextStyles.headline1,
                ),
              ),
              const SizedBox(width: 16),
```

```dart
              Expanded(
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    Text(user.name, style: AppTextStyles.headline1),
                    const SizedBox(height: 4),
                    Text(user.email, style: AppTextStyles.bodyText1),
                    if (user.bio != null && user.bio!.isNotEmpty)
                      Padding(
                        padding: const EdgeInsets.only(top: 4),
                        child: Text(
                          user.bio!,
                          maxLines: 2,
                          overflow: TextOverflow.ellipsis,
                          style: AppTextStyles.bodyText1.copyWith(color: Colors.grey),
                        ),
                      ),
                  ],
                ),
              )
            ],
          ),
        ),
      ),
    );
  }
}
```

Listing 3: chat screen.dart

```dart
import 'package:flutter/material.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';

class ChatScreen extends StatefulWidget {
  final String chatId;

  const ChatScreen({super.key, required this.chatId});

  @override
  State<ChatScreen> createState() => _ChatScreenState();
}
```

```

class _ChatScreenState extends State<ChatScreen> {
  final _controller = TextEditingController();
  final currentUser = FirebaseAuth.instance.currentUser!;

  void _sendMessage() async {
    final text = _controller.text.trim();
    if (text.isEmpty) return;

    final now = Timestamp.now();

    await FirebaseFirestore.instance
        .collection('chats')
        .doc(widget.chatId)
        .collection('messages')
        .add({
      'text': text,
      'senderId': currentUser.uid,
      'timestamp': now,
    });

    await FirebaseFirestore.instance
        .collection('chats')
        .doc(widget.chatId)
        .update({
      'lastMessage': text,
      'lastMessageTime': now,
    });

    _controller.clear();
  }

  @override
  Widget build(BuildContext context) {
    final messagesStream = FirebaseFirestore.instance
        .collection('chats')
        .doc(widget.chatId)
        .collection('messages')
        .orderBy('timestamp', descending: true);

    return Scaffold(
      appBar: AppBar(title: const Text('      ')),
```

```dart
      body: Column(
        children: [
          Expanded(
            child: StreamBuilder(
              stream: messagesStream.snapshots(),
              builder: (context, snapshot) {
                if (!snapshot.hasData) return const Center(child: CircularProgressIndicator());

                final messages = snapshot.data!.docs;

                return ListView.builder(
                  reverse: true,
                  itemCount: messages.length,
                  itemBuilder: (context, index) {
                    final data = messages[index].data() as Map<String, dynamic>;
                    final isMe = data['senderId'] == currentUser.uid;

                    return Align(
                      alignment: isMe ? Alignment.centerRight : Alignment.centerLeft,
                      child: Container(
                        margin: const EdgeInsets.symmetric(horizontal: 12, vertical: 4),
                        padding: const EdgeInsets.all(12),
                        decoration: BoxDecoration(
                          color: isMe ? Colors.blue[100] : Colors.grey[300],
                          borderRadius: BorderRadius.circular(10),
                        ),
                        child: Text(data['text'] ?? ''),
                      ),
                    );
                  },
                );
              },
            ),
          ),
          const Divider(height: 1),
          Padding(
            padding: const EdgeInsets.symmetric(horizontal: 8, vertical
                : 6),
            child: Row(
              children: [
                Expanded(
                  child: TextField(
                    controller: _controller,
                    decoration: const InputDecoration(hintText: '
                                          ...'),
                  ),
                ),
                IconButton(
                  icon: const Icon(Icons.send),
                  onPressed: _sendMessage,
                ),
              ],
            ),
          ),
        ],
      ),
    );
  }
}
```