

Applynix: A Cross-Platform Job Application Management System

Praharsh Agrawal^{1†}, Jagdish Bajaj¹, Dr. Somesh Kumar Dewangan²

¹ UG at Department of Computer Science and Engineering

Shri Shankaracharya Technical Campus, Junwani, Bhilai

Chhattisgarh, India. 490020

² Project Guide

† Corresponding Author and Lead Researcher

Abstract

The rapid digitization of recruitment has led to a surge in online job portals and application tracking systems, yet many existing solutions remain platform-dependent, backend-heavy, and difficult for small institutions or teams to deploy and maintain. This paper presents APPLYNIX, a cross-platform job application management system built using the Flutter framework with Firebase as a Backend-as-a-Service (BaaS) stack. APPLYNIX unifies user authentication, resume storage, application tracking, and interactive user experience across Android, iOS, web, and desktop from a single codebase. The proposed architecture combines modular Flutter widgets, Provider-based state management, and Firebase services including Authentication, Cloud Firestore, and Storage. We describe the system requirements, layered architecture, and implementation pipeline from onboarding to application submission. A prototype implementation and qualitative evaluation demonstrate that this approach significantly simplifies development and provides a consistent user experience across devices, while keeping the backend configuration lightweight. The work illustrates how modern cross-platform and cloud-native technologies can be composed into an extensible blueprint for academic and production grade job application platforms.

Keywords: Job portal, cross-platform, Flutter, Firebase, application tracking, mobile computing, BaaS.

1. INTRODUCTION

The recruitment landscape is increasingly mediated by digital platforms that connect job seekers with employers through online applications, resume submissions, and automated screening workflows. Universities and training institutions similarly rely on placement portals to coordinate campus hiring, internships, and student profiling. Despite this shift, many systems are built as monolithic web applications or platform-specific mobile apps, leading to duplicated development effort, inconsistent user experience, and complex backend administration.

Cross-platform frameworks such as Flutter have emerged as a powerful response to this fragmentation by allowing developers to target multiple platforms from a single codebase, while cloud-hosted Backend-as-a-Service (BaaS) offerings such as Firebase remove much of the operational burden associated with authentication, data storage, and file handling. When combined, these technologies can significantly reduce time-to-deployment for full-stack systems while preserving scalability and security. However, documented design patterns that integrate Flutter and Firebase into cohesive job application management platforms remain limited in academic literature.

APPLYNIX is designed to address this gap by providing a unified, cross-platform job application management system that demonstrates how modern mobile-first UI paradigms, reactive state management, and cloud-native services can be orchestrated into an extensible architecture. This paper formalises the design and implementation of APPLYNIX, distilling lessons learned from its development into a reusable blueprint for similar systems.

1.1 Problem Statement and Motivation

Existing job portals and application tracking systems often suffer from one or more of the following issues: strong platform dependency that necessitates separate native apps, complex self-hosted backends requiring dedicated administration, limited offline capabilities, and fragmented handling of user identity, resumes, and application status. These limitations hinder smaller organisations, academic institutions, and individual developers from deploying robust application platforms.

From a student and early-career perspective, managing multiple resumes, cover letters, and application records across email accounts, cloud drives, and proprietary portals introduces friction and increases the likelihood of errors. A centralised, device-agnostic system that collates user profiles, documents, and application history can significantly streamline this process, particularly when it is accessible on both mobile and desktop environments.

1.2 Objectives

- Design a cross-platform application management system for job applications using Flutter and Firebase.
- Implement secure user authentication, profile management, and session handling with Firebase Authentication.
- Provide upload, storage, and retrieval of resumes and related documents via Firebase Storage, with metadata stored in Cloud Firestore.
- Implement interactive, swipe-based user interfaces for browsing opportunities and submitting applications.
- Demonstrate a modular architecture that separates UI, state management, and backend service access for maintainability.
- Evaluate the prototype qualitatively in terms of usability, responsiveness, and ease of deployment.

2. SYSTEM OVERVIEW AND CONTRIBUTIONS

APPLYNIX is conceived as a full-stack application that supports the end-to-end lifecycle of job applications: user onboarding, profile creation, resume upload, opportunity browsing, application submission, and status tracking. The system targets Android, iOS, web, and desktop platforms through a single Flutter codebase and relies on Firebase for cloud-hosted services.

The main contributions of this work are summarised as follows:

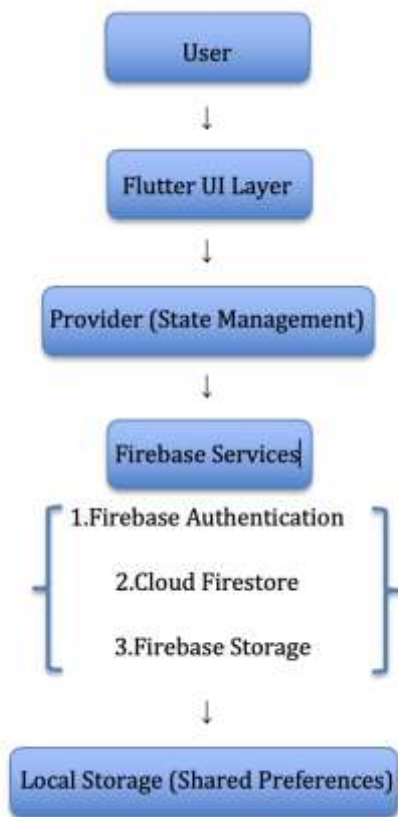
- Cross-platform architecture: A client-side architecture that leverages Flutter and Provider to deliver a consistent user experience across mobile and desktop environments.
- Cloud-native backend integration: A practical integration of Firebase Authentication, Cloud Firestore, and Storage to minimise backend administration while maintaining security and scalability.
- Modular application pipeline: A well-defined pipeline from onboarding to application submission, decomposed into modules for authentication, data management, file handling, and UI interaction.
- Prototype implementation: A working prototype that demonstrates feasibility and documents configuration steps, design decisions, and integration challenges.
- Qualitative evaluation: An initial evaluation of usability and responsiveness, along with a discussion of limitations and directions for future enhancement.

3. SYSTEM ARCHITECTURE

The APPLYNIX platform follows a layered architecture that decouples presentation, state management, and backend integration. At the top layer, Flutter screens and widgets implement the user interface for onboarding, login, dashboards, resume management, and application browsing. State management is handled by the Provider package, which exposes reactive models for authentication status, user profiles, and application lists.

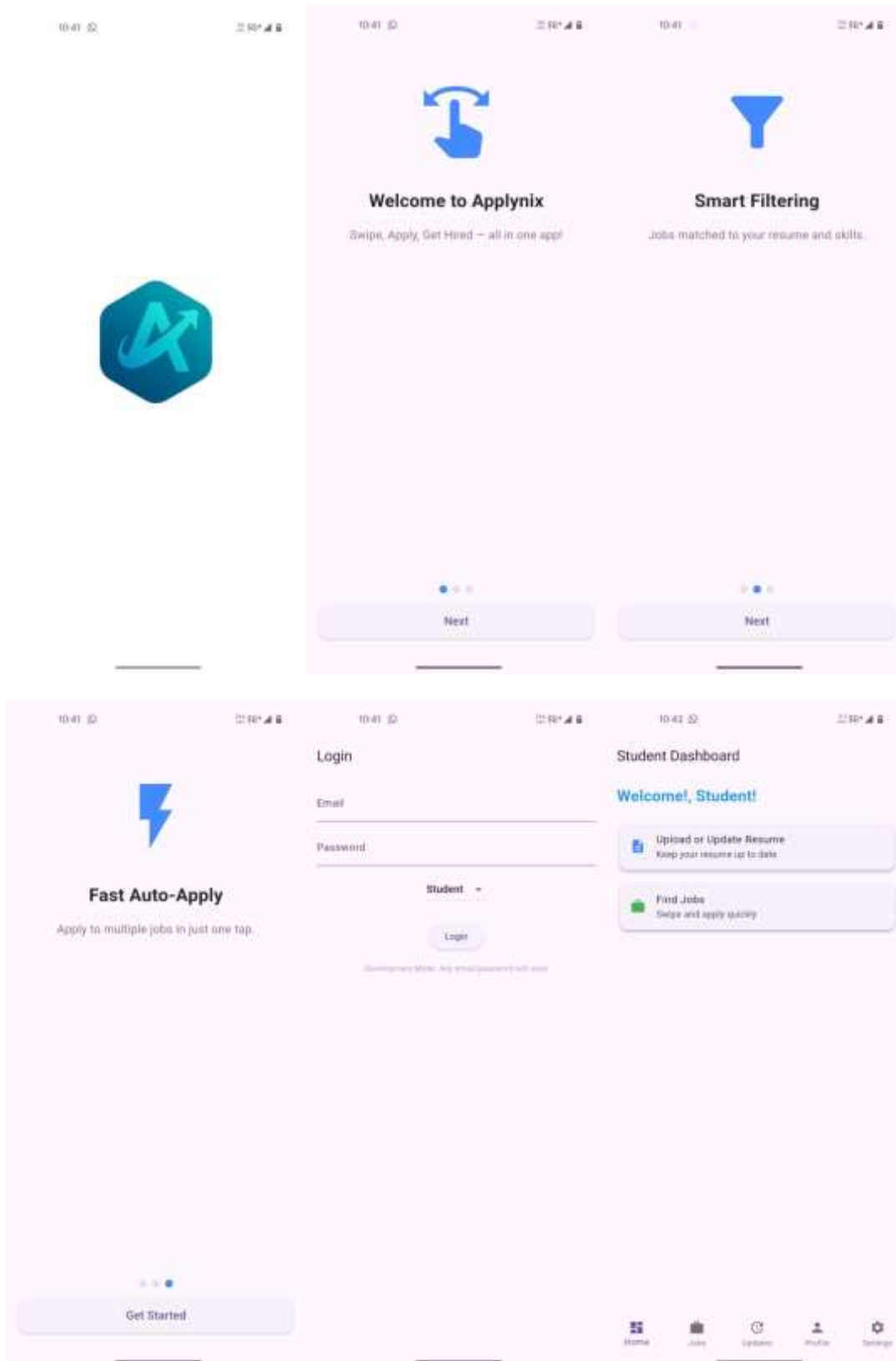
Below the state layer, service classes encapsulate access to Firebase Authentication, Cloud Firestore, and Firebase Storage. This separation isolates platform-specific configuration and network logic from UI code, simplifying unit

testing and future migration to alternative backend providers if required. Shared preferences are used as a local storage mechanism for lightweight data such as login flags and cached user attributes.



3.1 Functional Layers

- **Client Layer:** A Flutter-based front-end delivering responsive UIs for mobile and web, including forms, lists, and swipeable cards.
- **State Management Layer:** Provider-based models that hold authentication state, user data, and collections of opportunities and applications.
- **Service Layer:** Abstractions over Firebase Authentication, Firestore, and Storage, responsible for data retrieval, persistence, and error handling.
- **Persistence Layer:** Cloud Firestore collections and subcollections representing users, resumes, and applications, complemented by local shared preferences caching.



4. METHODOLOGY

The development of APPLYNIX followed an iterative, incremental methodology aligned with common software engineering practices. Requirements were gathered by analysing typical workflows of students applying for jobs and recruiters managing candidate information. These requirements informed the design of use cases and user stories that guided implementation.

The methodology can be summarised in four main phases: requirement analysis, architectural design, implementation, and evaluation. Requirement analysis established functional and non-functional needs. Architectural design produced diagrams for the system layers, authentication flow, resume upload, and application processing. Implementation translated these designs into Flutter code and Firebase configurations. Evaluation focused on functional correctness, usability, and initial performance observations.

4.1 Use Case Definition

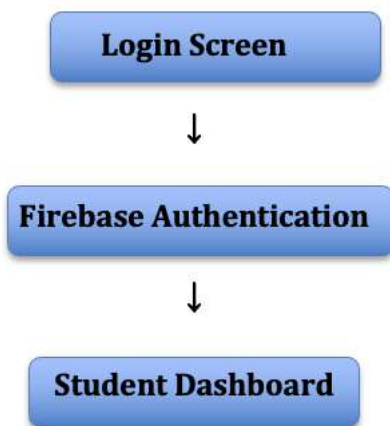
Key use cases include user registration, user login, profile editing, resume upload, opportunity browsing, application submission, and application status tracking. For example, in the resume upload use case, the user navigates to the resume screen, selects a file via a file picker, and initiates an upload. The system stores the file in Firebase Storage and writes a corresponding metadata entry in Firestore, updating the UI upon success or failure.

5. IMPLEMENTATION AND MODULES

The APPLYNIX prototype is implemented in Dart using the Flutter SDK. Project structure follows a feature-oriented layout with separate directories for models, providers, screens, services, and reusable widgets. Dependency management is handled via the pubspec configuration, which includes `firebase_core`, `firebase_auth`, `cloud_firestore`, `firebase_storage`, `provider`, `shared_preferences`, `file_picker`, `dio`, `http`, and `flutter_html`.

5.1 Authentication Module

The authentication module integrates Firebase Authentication to support email/password registration and login. Flutter screens collect user credentials, perform basic client-side validation, and then invoke service methods that communicate with Firebase. Successful authentication results in persisting the user identifier and updating Provider state, which in turn triggers a navigation change to the dashboard. Error messages for invalid credentials or network issues are surfaced through dialog boxes and inline validation messages.



5.2 Data and File Management Modules

User profiles and application metadata are stored in Cloud Firestore collections. Each user document contains fields for personal details, links to uploaded resumes, and configuration preferences. Application entries include titles, descriptions, and optional HTML content for richer formatting. The file management module uses Firebase Storage to upload resumes and other documents, while saving download URLs and metadata to Firestore for later retrieval.

5.3 UI Interaction Module

The user interface is composed of reusable widgets for buttons, cards, and forms, as well as full-screen views for onboarding, login, dashboards, and detail pages. Swipe-based navigation is implemented using Flutter widgets that

enable horizontal card swiping to apply or skip opportunities. The flutter_html package renders HTML content within detail screens, allowing structured descriptions, bullet lists, and formatted text without custom parsing.

6. EVALUATION AND DISCUSSION

Evaluation of APPLYNIX focused on verifying functional requirements and assessing initial usability and responsiveness. Functional testing covered core flows including registration, login, profile update, resume upload, opportunity browsing, application submission, and status viewing. Manual exploratory testing on Android devices and web browsers confirmed that data remained consistent across platforms via Firestore synchronisation.

Static analysis using Flutter tooling was used to identify code smells and potential runtime issues, while informal user feedback from peers highlighted strengths in interface clarity and the convenience of swipe-based browsing. Limitations observed include the absence of advanced search and filtering features, lack of fine-grained offline control, and the reliance on Firebase-specific APIs, which may complicate future migration to alternative cloud providers.

7. FUTURE WORK

Future iterations of APPLYNIX will focus on enhancing the data model and user experience. Planned features include search and filtering of opportunities by skills, location, and keywords; analytics dashboards for users and administrators; and more sophisticated offline-first behaviour with conflict resolution strategies for concurrent edits.

Another direction is the integration of machine learning components for resume parsing and opportunity recommendation, potentially leveraging cloud-hosted natural language processing services. Additionally, exploring multi-tenant configurations could allow institutions to host customised instances of APPLYNIX under a shared infrastructure, broadening its applicability beyond individual users.

8. CONCLUSION

This paper has presented APPLYNIX, a cross-platform job application management system that combines the Flutter framework with Firebase services to deliver a unified experience across mobile and desktop devices. By structuring the system into modular layers for UI, state management, and backend integration, the design achieves a balance between developer productivity and architectural clarity. The prototype implementation demonstrates that a relatively small team can assemble a feature-complete application using contemporary cross-platform and cloud-native tools.

While the current evaluation is primarily qualitative and limited in scale, the architectural patterns and implementation details documented here provide a practical reference for researchers and practitioners seeking to build similar systems. As digital recruitment continues to evolve, platforms like APPLYNIX can serve as a foundation for more intelligent, analytics-driven, and user-centric job application ecosystems.

REFERENCES

1. Flutter Documentation, <https://flutter.dev>
2. Firebase Documentation, <https://firebase.google.com>
3. FlutterFire CLI Documentation, <https://firebase.flutter.dev>
4. Cross-platform feature matching for web applications - <https://dl.acm.org/doi/abs/10.1145/2610384.2610409>
5. JobCenter: an open source, cross-platform, and distributed job queue management system optimized for scalability and versatility - <https://link.springer.com/article/10.1186/1751-0473-7-8>