FreelanceNest

SAMANA HASSAN
01-135221-080
IQRA SHAIKH
01-135221-101

Supervisor: Junaid Hussain

Bachelor of Science in Information Technology

Department of Computer Science
Bahria University, Islamabad

December 2025

Certificate

We accept the work contained in the report titled “FreelanceNest”, written by Miss Samana Hassan AND Miss Iqra Shaikh as a confirmation to the required standard for the partial fulfillment of the degree of Bachelor of Science in Information Technology.

Approved by :

Supervisor:

_____

Internal Examiner:

_____

External Examiner:

_____

Project Coordinator:

_____

Head of the Department:

_____

Abstract

FreelanceNest is an AI-driven freelancing marketplace designed to overcome the challenges of traditional platforms, such as unfair job allocation, high service fees, poor communication tools, and inadequate skill verification. By leveraging intelligent recommendation models, the platform ensures fair, skill-based job matching, providing equal opportunities for both new and experienced freelancers. To facilitate smooth collaboration, FreelanceNest offers real-time messaging, notifications, and secure payment processing via Stripe, fostering transparency and trust between clients and freelancers.

The platform is built with React.js for the frontend, Node.js for the backend, and Firebase for real-time database management, authentication, and notifications. Additional features include an analytics dashboard and admin monitoring tools to enhance user experience and system oversight. By combining AI-powered decision-making with a robust system architecture, FreelanceNest delivers a reliable, fair, and scalable freelancing ecosystem that enhances hiring accuracy, builds user trust, and elevates the freelancing experience.

Acknowledgments

In the name of Allah, the Most Beneficent and the Most Merciful. We are highly grateful to the One who created us and blessed us with a privileged life. We would like to thank our parents, who have always been a pillar of strength and support. We are thankful to them for teaching us how hard work, devotion, and working towards your goal with pure intention can take you to great places.

Furthermore, we would like to thank and appreciate our supervisor, Sir Junaid, who has given us the chance to work on a project that can help in creating value for our beloved country. His professional guidance, time, and effort will always be remembered.

Last but not least, we would like to appreciate our friends who make studying and hard times less challenging. May Allah (SWT) guide us to the right path.

SAMANA HASSAN & IQRA SHAIKH
Islamabad, Pakistan

December 2025

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
CRUD	Create, Read, Update, Delete
DB	Database
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
DL	Deep Learning
NLP	Natural Language Processing
NoSQL	Not Only SQL
REST	Representational State Transfer
SQL	Structured Query Language
UI	User Interface
UML	Unified Modeling Language
UX	User Experience
XML	Extensible Markup Language
JWT	JSON Web Token
IDE	Integrated Development Environment
QA	Quality Assurance
SQA	Software Quality Assurance
RBAC	Role-Based Access Control
FCM	Firebase Cloud Messaging
PCI-DSS	Payment Card Industry Data Security Standard
CI/CD	Continuous Integration / Continuous Deployment
UAT	User Acceptance Testing
GraphQL	Graph Query Language
CORS	Cross-Origin Resource Sharing
Firebase	Google Backend-as-a-Service Platform
Firestore	NoSQL Cloud Database by Firebase
ReactJS	JavaScript Library for Building User Interfaces
Node.js	JavaScript Runtime Environment
Stripe	Online Payment Processing Platform

Chapter 1

Introduction

1.1 Overview

FreelanceNest is an AI-driven freelancing marketplace designed to transform how freelancers and clients interact by providing automation, fairness, and secure collaboration. The freelancing industry has grown significantly over the past decade, driven by digital transformation and global remote work trends [1]. However, existing platforms still suffer from problems such as biased job allocation, high service fees, and ineffective communication tools [2].

FreelanceNest addresses these challenges by using AI-based recommendation models that match freelancers to projects based on skills, expertise, and performance rather than ratings alone [3]. The system includes separate dashboards for clients, freelancers, and administrators featuring analytics, project management, messaging, and real-time notifications.

The platform uses modern technologies such as React.js for the frontend [4], Node.js for backend logic, Firebase for authentication and real-time communication [5], and Stripe for secure financial transactions [6]. Its mission is to create a fair and transparent ecosystem where both new and experienced freelancers have equal opportunities, and clients can make informed hiring decisions.

1.2 Objective

The primary objective of FreelanceNest is to deliver an intelligent, fair, and transparent freelancing platform powered by machine learning. Traditional freelancing systems rely heavily on popularity-based ranking, which disadvantages new freelancers and skews job

allocation [7]. FreelanceNest counters this by using smart recommendation engines that analyze skills, expertise, and work quality to ensure fair project distribution.

Secure transaction processing is enabled through Stripe, ensuring payments are only completed when milestones or full projects are successfully delivered [6]. The platform includes dashboards for clients, freelancers, and administrators, along with analytics, performance tracking, and real-time communication powered by Firebase [5].

FreelanceNest aims to deliver a scalable, user-friendly, and merit-based environment supporting freelancer growth and improving the global hiring experience.

1.3 Problem Description

The freelancing market continues to expand rapidly, yet many mainstream platforms still face major shortcomings [1]. One persistent issue is unfair job distribution new or lower-rated freelancers struggle to secure work due to algorithms prioritizing highly rated profiles [2]. High service fees further reduce freelancer earnings and satisfaction.

Another challenge is inadequate skill verification, leading to mismatches between client expectations and freelancer capabilities. Many platforms also lack integrated communication systems, forcing users to rely on third-party apps, which causes delays and inefficiencies.

Security risks, including non-payments, and fraudulent activities, contribute to an unsafe environment for new users [8]. These problems result in poor user experience, reduced trust, and unequal opportunities.

FreelanceNest addresses these issues through AI-powered job matching [3], Stripe-based secure financial workflows [6], enhanced skill verification, and real-time messaging via Firebase [5]. This creates a fairer, more reliable, and more collaborative freelancing ecosystem.

1.4 Methodology

FreelanceNest was developed using a structured Software Development Life Cycle (SDLC) approach to ensure accuracy, scalability, and user-centered design [9, 10]. The process includes requirement gathering, analysis, system design, development, testing, and maintenance.

1.4.1 Collecting Requirements

Requirements were gathered from freelancers, clients, and administrators by analyzing existing platforms and identifying recurring problems such as unfair job assignment, weak

skill verification, and ineffective communication [7]. These findings shaped core features such as AI-based matching, secure payment integration, and real-time messaging.

1.4.2 Analysis

The collected requirements were analyzed to evaluate feasibility and system functionality. This included assessing integration constraints related to AI models, Firebase real-time capabilities, and Stripe transaction flows. User needs, workflows, and major system components were identified to support a cohesive system design [9].

1.4.3 Design

The design phase defined the overall system architecture, UI/UX, database schema, and module interactions. High-level and detailed designs outlined navigation flow, dashboard interfaces, and backend integration for AI-based matching, messaging, notifications, and payment processing [11, 12].

1.4.4 Development

The development stage implemented the designed system. The frontend was built with React.js [4], while backend logic relied on Node.js. Firebase provided authentication, real-time database, and notifications [5]. AI/Deep Learning models powered job recommendations [3], and Stripe ensured secure payment processing [6].

1.4.5 Maintenance / Testing

This phase ensured system reliability, security, and usability through comprehensive functional, usability, graphical interface, and performance testing [13]. Bugs were identified and resolved, and continuous updates improved long-term stability and scalability.

1.5 Project Scope

The project scope defines system boundaries, included features, and excluded functionalities.

Included in Scope

- **AI-Assisted Job Matching:** Deep learning models match freelancers to projects based on their skills and performance [3]. Dedicated dashboards include analytics, project management tools, and notifications.

- **Secure Payment Integration (Stripe):** Built-in payment processing using Stripe ensures secure and transparent transactions [6].
- **Real-Time Messaging and Notifications:** Firebase enables real-time chat and instant alerts for seamless communication [5].

Excluded from Scope

- **Mobile Application:** Not included in the current development phase.
- **Third-Party Course Marketplace Integration:** External learning platforms are outside the project scope.

1.6 Feasibility Study

The feasibility study evaluates whether the system can be developed within constraints of technology, time, and resources. It assesses risks, required resources, and technical feasibility regarding AI models, real-time processing, and secure payment workflows [10].

1.7 Risks Involved

- **User Adoption and Competition:** Competing with platforms like Fiverr and Upwork is challenging. FreelanceNest offers competitive advantages through lower fees, AI-driven fairness, and improved skill validation [7].
- **Technical Scalability:** The platform must scale as the user base grows. Node.js supports scalable backend operations, and Firebase enables real-time data handling efficiently [5].
- **Security and Fraud Prevention:** Freelance platforms face risks such as fraud and non-payment. FreelanceNest leverages Stripe for secure transactions [6] and applies OWASP security principles [8].

1.8 Resource Requirement

- **Computing Resources:** Firebase, Stripe, AI/DL tools, Node.js, React.js [5, 6, 4].
- **Infrastructure:** Deep learning engineers, full-stack developers, UI/UX designers, and security specialists are required to ensure a complete development lifecycle [9, 10].
- **Financial Resources:** API service charges, deployment costs, marketing expenses, hosting, and ongoing maintenance.

1.9 Solution Application Areas

FreelanceNest provides value across sectors requiring remote collaboration, skill-based recruitment, and efficient digital talent sourcing. The platform is adaptable to environments that demand fairness, transparency, and automation.

1.9.1 Freelancing Platforms

FreelanceNest enhances traditional freelance marketplaces by integrating an AI-based job recommendation system to reduce bias toward highly rated users [7]. Real-time messaging, secure payments, and analytics dashboards improve user experience, making it suitable for modern freelancing ecosystems.

1.9.2 Remote Work Solutions / Corporate Hiring

As remote work becomes increasingly common, FreelanceNest helps organizations find skilled professionals efficiently. AI algorithms and verified profiles assist with talent acquisition and team formation [3]. Integrated dashboards, secure payment workflows, and real-time notifications make it a strong fit for remote project management and temporary staffing.

1.9.3 Other Applications in the Profession

- **Small Businesses:** Enables quick access to verified, skilled talent.
- **Educational or Training Programs:** Facilitates performance monitoring and skill validation.

In all these applications, FreelanceNest prioritizes fairness, transparency, and automated decision-making powered by AI [2].

1.10 Tools and Technology

FreelanceNest leverages a modern and scalable technology stack that supports high performance, real-time communication, secure payments, and intelligent automation.

1.10.1 React.js

React.js is used to build the platform's user interface due to its component-based structure, performance optimization, and scalability [4]. CSS is integrated for responsive and modern UI design.

1.10.2 Node.js

Node.js powers the backend operations, enabling fast request handling, scalable APIs, and real-time operations thanks to its event-driven architecture. This makes it suitable for messaging, notifications, and high concurrency applications.

1.10.3 Firebase

Firebase provides real-time database services, authentication, cloud messaging, and hosting. Firestore efficiently stores user data, projects, activities, and communication logs [5]. It ensures seamless updates and low-latency interactions between freelancers and clients.

1.10.4 Stripe API

Stripe ensures secure payment processing, milestone-based fund release, invoicing, and fraud monitoring [6]. This builds transparency and trust across the platform's financial activities.

1.10.5 AI/DL Tools

AI-based recommendations leverage tools such as TensorFlow and Scikit-learn. These support skill-matching models, performance analytics, and intelligent recommendations [3, 7]. AI models also assist in skill verification and detection of unreliable freelancer patterns.

1.11 Proficiency of the Team Members

The FreelanceNest development team possesses technical and analytical expertise required to build a secure, scalable, and AI-driven marketplace.

- **Full-Stack Development Skills:** Proficiency in React.js, Node.js, and Firebase ensures efficient frontend and backend integration [4, 5].
- **AI/DL Competency:** Team members are skilled in deploying AI algorithms and recommender systems using TensorFlow and Scikit-learn [3].
- **UI/UX Design Skills:** Developers create intuitive, user-friendly dashboards for freelancers, clients, and administrators.

Database Management and Security The team manages real-time data using Firebase Firestore and ensures integrity through industry-level authentication mechanisms and security practices [5, 8]. Stripe-based secure payment integration further enhances the

platform's trustworthiness [6]. Proper API integration ensures seamless communication between all system components.

These combined competencies ensure that FreelanceNest meets high standards of performance, usability, and security.

1.12 Milestones

Milestone	Duration
Requirement Gathering & Research	2 weeks
System Architecture & Design	2 weeks
AI Model Development	3 weeks
Frontend Development	3 weeks
Backend Development	3 weeks
Integration & Testing	3 weeks
User Acceptance Testing & Debugging	2 weeks
Deployment & Final Review	2 weeks

Figure 1.1: Milestones

Chapter 2

Literature Review

In the last ten years, the freelancing market has experienced significant global growth, with millions of freelancers and clients relying on online platforms for project collaboration [1]. Although several platforms have gained international popularity by offering services such as gig posting, bidding, messaging, and payment processing, many still struggle with issues such as preferential job assignment, high service charges, and limited intelligent recommendation capabilities [2, 3].

This chapter reviews existing platforms, their limitations, and how FreelanceNest aims to address these challenges using AI-based automation, secure payment infrastructure, and fairness-oriented design principles.

2.1 Related Work

This section evaluates leading freelancing platforms such as Fiverr, Upwork, and Freelancer.com to understand their strengths, limitations, and influence on the development of FreelanceNest.

2.1.1 Fiverr

Fiverr is one of the largest gig-based marketplaces that allows freelancers to sell predefined services. It offers ease of use, buyer protection, and a vast global marketplace. However, Fiverr's ranking model is heavily dependent on ratings and historical performance, making it difficult for new freelancers to gain visibility [1]. High service fees and strict withdrawal policies further affect freelancer earnings. Additionally, Fiverr relies primarily on manual search and keyword filtering and lacks advanced AI-driven job recommendation systems [2].

2.1.2 Upwork

Upwork operates on a bidding-based model where clients post projects and freelancers submit proposals. The platform supports hourly and fixed-price contracts, milestone payments, and time-tracking tools. Nevertheless, the talent search algorithm often favors profiles with high job success scores, making it challenging for newcomers or low-rated freelancers to compete [7]. Upwork also charges high commission fees, discouraging both freelancers and clients. Furthermore, Upwork does not leverage modern AI-driven skill-based matching approaches that could improve relevance and performance prediction [14].

2.1.3 Freelancer.com

Freelancer.com combines bidding, contest-based hiring, and long-term project collaboration while offering features such as milestone payments and project tracking. However, the platform faces issues such as spam proposals, inconsistent bid quality, and unreliable interactions between clients and freelancers [3]. Skill verification mechanisms are limited, and the platform does not prioritize fairness or intelligent matchmaking. Communication tools are basic, and the system lacks sophisticated recommendation infrastructure found in modern intelligent platforms [2].

2.2 Comparative Analysis

Table 2.1: Comparative Analysis

Feature / Platform	**Fiverr**	**Upwork**	**Freelancer.com**	**FreelanceNest (Proposed)**
Job Matching	Manual search, keyword-based	Basic suggestions	Limited automation	AI-driven skill-based matching [2, 3]
Fairness for New Freelancers	Low visibility	Rating-biased	Competitive & crowded	Equal opportunity using AI-driven profiling [7]
Service Fees	High commission	High commission	Moderate	Lower processing fees
Skill Verification	Limited	Moderate	Weak	AI-based verification test[14]
Communication Tools	Basic chat	Chat & time tracker	Chat	Real-time messaging & notifications (Firebase) [5]
Payment Security	Strong	Strong	Moderate	Stripe-based escrow payments [6]

The comparative analysis shows that while existing platforms offer mature ecosystems, they lack intelligent automation, fairness mechanisms, and high-quality communication systems. FreelanceNest addresses these gaps through AI-based recommendations [2], real-time collaboration tools, secure payment workflows using industry security guidelines [8], and transparent user-focused design.

Chapter 3

Requirement Specifications

3.1 Proposed System

FreelanceNest is an AI-driven freelancing platform designed to enhance fairness, transparency, and efficiency in freelancer–client interactions. Unlike traditional platforms that rely heavily on ratings and popularity, FreelanceNest utilizes intelligent job-matching models based on competence, skill relevance, and project demands [3, 2].

Key Features of FreelanceNest:

- **AI-Powered Job Recommendations:** Leveraging machine learning and recommender system principles, the platform suggests the most suitable freelancers for projects based on skills, experience, and historical performance [7].
- **User-Specific Dashboards:** Tailored dashboards for clients, freelancers, and administrators provide real-time analytics, project status, and intuitive navigation for improved productivity.
- **Secure Stripe Payment System:** Enables milestone-based fund release, encrypted transactions, and fraud-resistant payment flows supported by Stripe's security model [6].
- **Real-Time Chat & Notifications:** Implemented using Firebase to support instant messaging, activity alerts, bidding updates, and milestone notifications [5].

FreelanceNest aims to develop an equitable freelancing environment by integrating automation, security, and transparent workflows.

3.2 Existing System

Current freelancing platforms such as Fiverr, Upwork, and Freelancer.com dominate the market but share several limitations [1]:

- **Bias Toward Highly Rated Freelancers:** Their ranking algorithms prioritize top-rated profiles, making it difficult for new freelancers to gain visibility [2].
- **High Commission Charges:** Excessive service fees reduce freelancer earnings and discourage new users.
- **Inadequate Skill Verification:** Reliance on ratings and brief profiles often leads to mismatches between client expectations and freelancer capabilities [7].
- **Weak Communication Tools:** Many platforms lack integrated real-time communication, causing users to use external apps, resulting in delays and inefficiencies.
- **High Risk of Scam and Fraud:** Issues such as fake clients, non-payment, and spam proposals reduce platform trustworthiness.

These shortcomings highlight the need for an intelligent, fairness-oriented system with improved security, automation, and communication capabilities leading to the development of **FreelanceNest**.

3.3 Functional Requirements

Functional requirements describe what the system should do. The requirements outlined for **FreelanceNest** are based on the needs of clients, freelancers, and administrators following established requirement engineering principles [15].

User Registration & Authentication

- Supports email/password and third-party authentication.
- Firebase Authentication ensures secure login, logout, and password reset workflows [5].

Profile Management

- Freelancers can edit skills, portfolios, and experience.
- Clients can manage company profiles and hiring preferences.

Project Posting & Bidding

- Clients can post projects with full descriptions, budgets, and timelines.

- Freelancers can browse listings and submit proposals.

AI-Based Job Matching

- Clients receive ranked recommendations of suitable freelancers.
- Freelancers receive personalized job suggestions based on their skills [3, 7].

Messaging System

- Real-time messenger for smooth communication.
- Notifications for messages, bids, approvals, and payments [5].

Checkout / Payment Handling

- Clients can pay through Stripe's secure payment gateway.
- Funds are held in escrow until the project is completed.
- Payment records are logged for full transparency [6].

Administrator Management

- Admins can manage users, bids, projects, and categories.
- Admins detect suspicious activity and resolve disputes using platform analytics.

Analytics Dashboard

- Freelancers can view performance metrics, earnings, and job success rates.
- Clients can analyze spending history and project performance overtime.

3.4 Non-Functional Requirements

Non-functional requirements (NFRs) describe how the system behaves rather than what it does. These follow standard NFR classifications in software engineering [9, 10].

Performance Requirements

- The platform must support concurrent usage without performance degradation.
- Real-time chat and notifications must maintain minimal latency.
- API response times should remain under acceptable thresholds during peak load.

Security Requirements

- All sensitive data should be encrypted at rest and in transit.

- Stripe is used for secure financial transactions [6].
- Firebase Authentication prevents unauthorized login attempts [5].
- Role-based access control (RBAC) must be implemented to restrict sensitive operations.

Usability Requirements

- Dashboards must be intuitive and easy to navigate.
- The UI should be responsive across devices and screen sizes.
- User flows should minimize the number of steps required to complete key tasks.
- Clear feedback messages should be provided for user actions and errors.

Reliability Requirements

- Firebase ensures real-time updates and reliable data synchronization [5].
- System must implement recovery mechanisms to minimize downtime.
- Application should perform regular automated backups to prevent data loss.
- Error handling and logging mechanisms should be implemented to detect failures promptly.

Scalability Requirements

- System architecture must scale with increased user activity.
- Backend APIs should handle increased traffic without bottlenecks.
- Database queries must be optimized to maintain performance under high load.
- Cloud infrastructure should support horizontal scaling to accommodate future growth.

Maintainability Requirements

- Codebase must be modular to support easy updates and extension.
- Comprehensive documentation must exist for onboarding new developers.
- System should follow consistent coding standards to ease long-term maintenance.
- Automated testing should be used to ensure system stability after updates.

3.5 Use Case Diagram

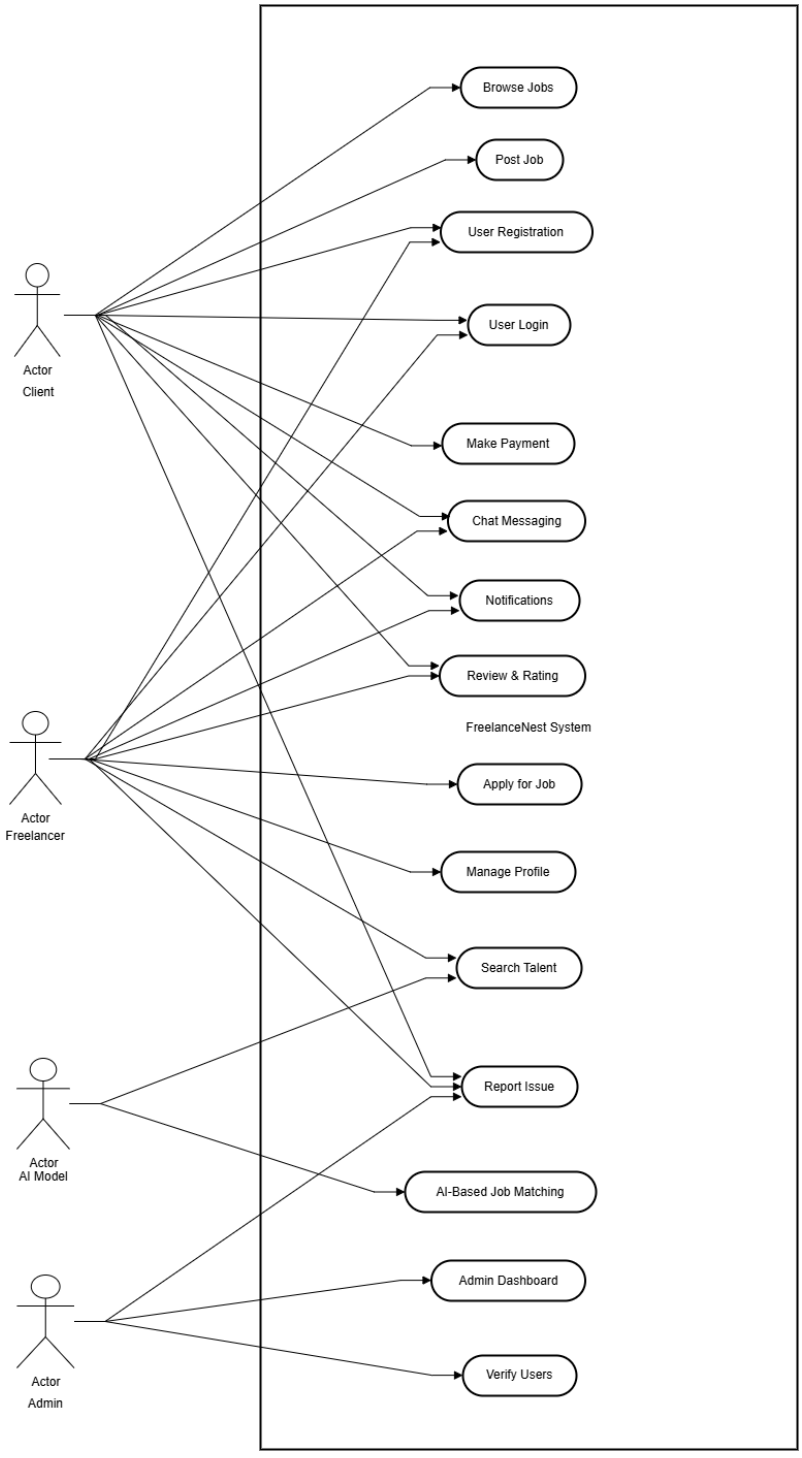

Figure 3.1: Use Case diagram

3.5.1 User Registration

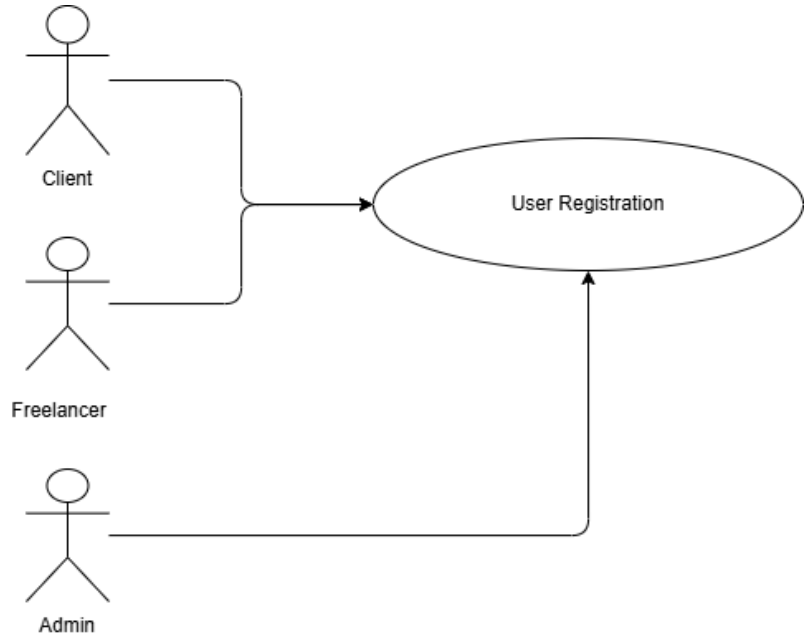

Figure 3.2: Use Case diagram for User Registration

Table 3.1: Use Case: User Registration

Field	**Details**
Use Case ID	UC01
Title	User Registration
Description	The user creates a new account on the platform.
Actor	Client, Freelancer, Admin
Pre-Condition	User must not already be registered.
Basic Flow	1. User opens registration page. 2. Enters required details. 3. System validates fields. 4. Firebase Auth creates account. 5. Confirmation appears.
Alternate Flow	– Email already exists → Show error. – Invalid inputs → Prompt correction.
Post-Condition	Account successfully created.

3.5.2 Login

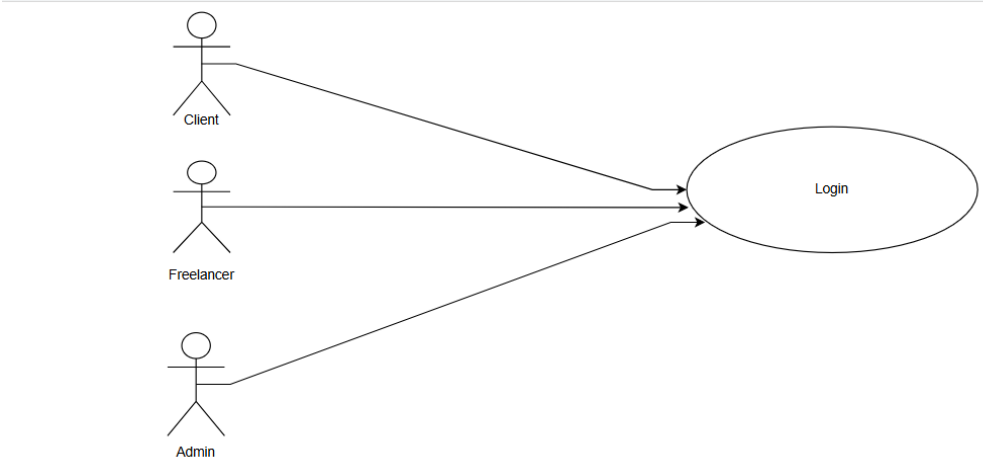

Figure 3.3: Use Case diagram for Login

Table 3.2: Use Case: Login

Field	**Details**
Use Case ID	UC02
Title	Login
Description	Logs the user into the system.
Actor	Client, Freelancer, Admin
Pre-Condition	User must have an existing account.
Basic Flow	1. User enters credentials. 2. Firebase authenticates. 3. User is redirected to the dashboard.
Alternate Flow	– Wrong password → Error. – Invalid email → Error. – Disabled account → Notify.
Post-Condition	User successfully logged in.

3.5.3 Forgot Password

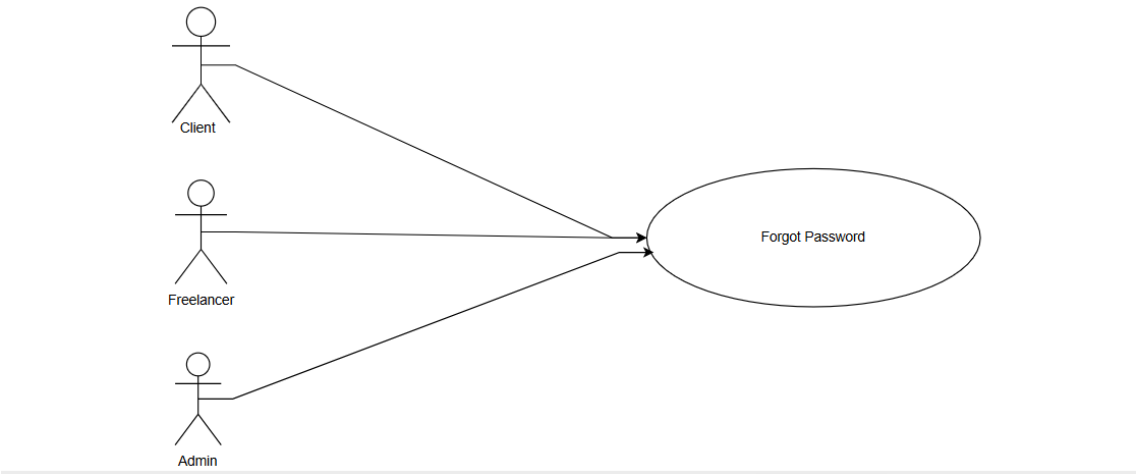

Figure 3.4: Use Case diagram for Forget Password

Table 3.3: Use Case: Forgot Password

Field	**Details**
Use Case ID	UC03
Title	Forgot Password
Description	User resets password via email link.
Actor	Client, Freelancer, Admin
Pre-Condition	User must have registered email.
Basic Flow	1. User clicks “Forgot Password”. 2. Enters email. 3. Firebase sends reset link. 4. User sets new password. 5. Logs in with new password.
Alternate Flow	– Email not found → Error. – Reset failed → Try again.
Post-Condition	Password reset successfully.

3.5.4 Create Gig (Freelancer)

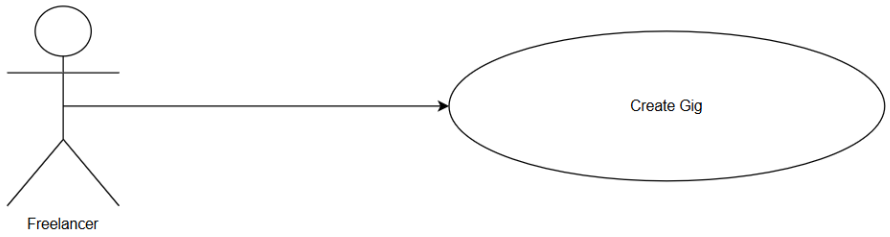

Figure 3.5: Use Case diagram for Create Gig

Table 3.4: Use Case: Create Gig

Field	**Details**
Use Case ID	UC04
Title	Create Gig
Description	Freelancer creates a new gig/service offering.
Actor	Freelancer
Pre-Condition	Freelancer must be logged in.
Basic Flow	1. Freelancer opens “Create Gig”. 2. Enters title, skills, price, description. 3. System validates. 4. Gig stored in Firestore.
Alternate Flow	– Missing information → Error.
Post-Condition	Gig successfully created.

3.5.5 Post Project (Client)

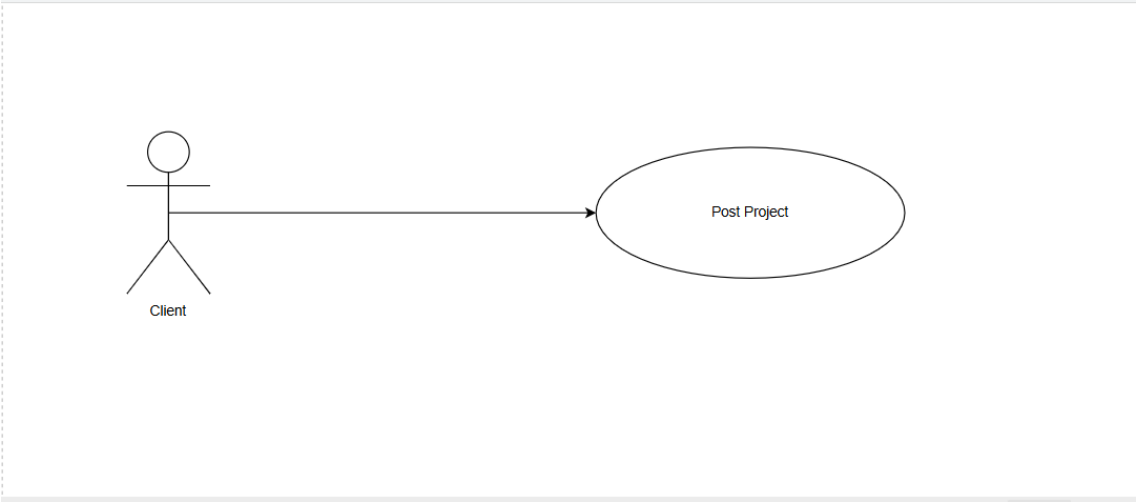

Figure 3.6: Use Case diagram for Post Project (Client)

Table 3.5: Use Case: Post Project

Field	**Details**
Use Case ID	UC05
Title	Post Project
Description	Client creates a new project for freelancers.
Actor	Client
Pre-Condition	Client must be logged in.
Basic Flow	1. Client opens project form. 2. Enters project details. 3. System validates. 4. Firestore stores project. 5. AI indexes the project.
Alternate Flow	Missing required fields → Error.
Post-Condition	Project posted successfully.

3.5.6 Browse Freelancers

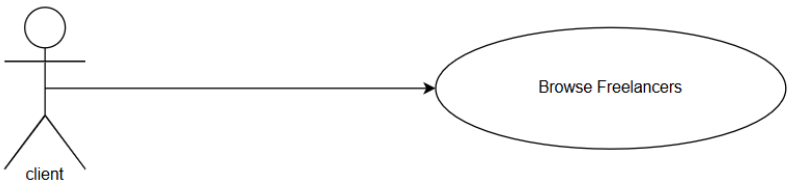

Figure 3.7: Use Case diagram for Browse Freelancers

Table 3.6: Use Case: Browse Freelancers

Field	**Details**
Use Case ID	UC06
Title	Browse Freelancers
Description	Client views and filters freelancers.
Actor	Client
Pre-Condition	Client must be logged in.
Basic Flow	1. Client opens list. 2. Applies filters. 3. System retrieves matches. 4. Displays results.
Alternate Flow	No match → Show “No Results”.
Post-Condition	Freelancer list shown.

3.5.7 AI-Based Job Matching

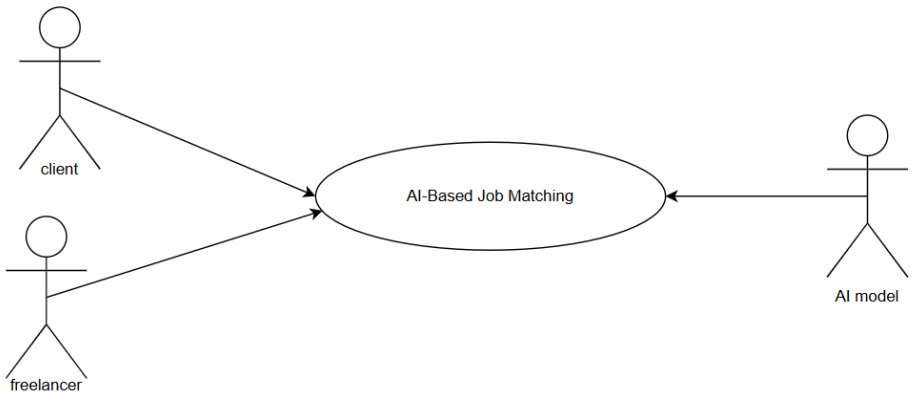

Figure 3.8: Use Case diagram for AI-Based Job Matching

Table 3.7: Use Case: AI-Based Job Matching

Field	**Details**
Use Case ID	UC07
Title	AI-Based Job Matching
Description	AI recommends freelancers and jobs.
Actor	Client, Freelancer, AI Model
Pre-Condition	Projects & profiles must exist.
Basic Flow	1. AI analyzes project & skills. 2. Ranks matches. 3. Sends recommendations.
Alternate Flow	No match → Show fallback.
Post-Condition	AI suggestions displayed.

3.5.8 Messaging / Chat

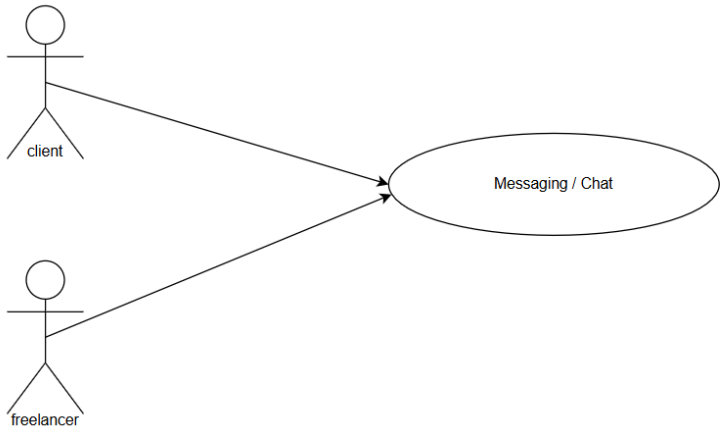

Figure 3.9: Use Case diagram for Messaging / Chat

Table 3.8: Use Case: Messaging / Chat

Field	**Details**
Use Case ID	UC08
Title	Messaging / Chat
Description	Real-time communication between users.
Actor	Client, Freelancer
Pre-Condition	Both users logged in.
Basic Flow	1. User opens chat. 2. Sends message. 3. Firestore stores it. 4. Receiver notified.
Alternate Flow	Network failure → Retry.
Post-Condition	Message delivered.

3.5.9 Freelancer View Jobs/Gigs

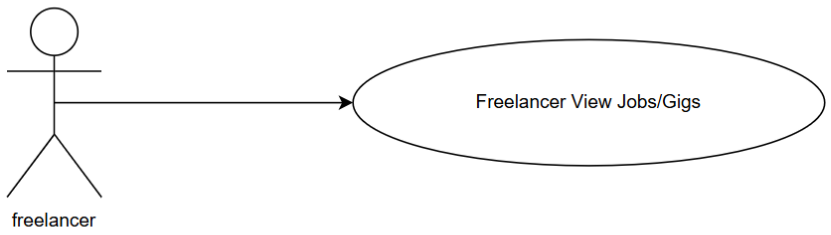

Figure 3.10: Use Case diagram for Freelancer View Jobs/Gigs

Table 3.9: Use Case: Freelancer View Jobs/Gigs

Field	**Details**
Use Case ID	UC09
Title	Freelancer View Jobs/Gigs
Description	Freelancer views jobs and gig recommendations.
Actor	Freelancer
Pre-Condition	Freelancer logged in.
Basic Flow	1. Opens job list. 2. System filters jobs. 3. Displays list & AI suggestions.
Alternate Flow	No jobs found → Show alternatives.
Post-Condition	Jobs/gigs displayed.

3.5.10 Client View Freelancers

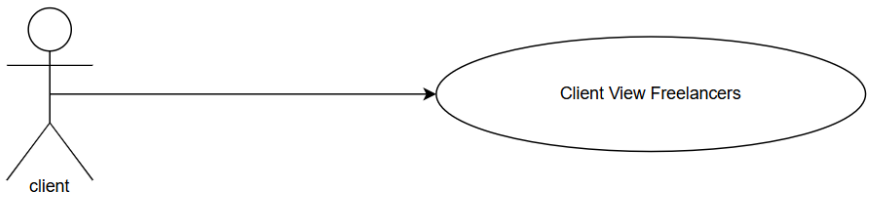

Figure 3.11: Use Case diagram for Client View Freelancers

Table 3.10: Use Case: Client View Freelancers

Field	**Details**
Use Case ID	UC10
Title	Client View Freelancers
Description	Client views freelancer profiles for hiring.
Actor	Client
Pre-Condition	Must be logged in.
Basic Flow	1. Client opens list. 2. Selects freelancer. 3. System shows profile.
Alternate Flow	Profile unavailable → Error.
Post-Condition	Client views freelancer.

3.5.11 Freelancer Search Projects/Gigs

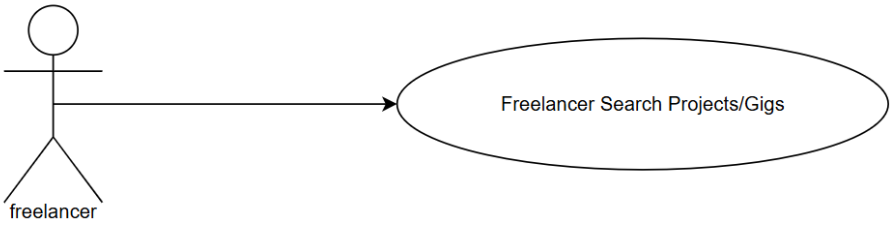

Figure 3.12: Use Case diagram for Freelancer Search

Table 3.11: Use Case: Freelancer Search Projects/Gigs

Field	**Details**
Use Case ID	UC11
Title	Freelancer Search Projects/Gigs
Description	Freelancer searches projects via filters/keywords.
Actor	Freelancer
Pre-Condition	Freelancer logged in.
Basic Flow	1. Enter search text. 2. System retrieves results. 3. Display list.
Alternate Flow	No projects → Show suggestions.
Post-Condition	Results shown.

3.5.12 Admin Manage Users

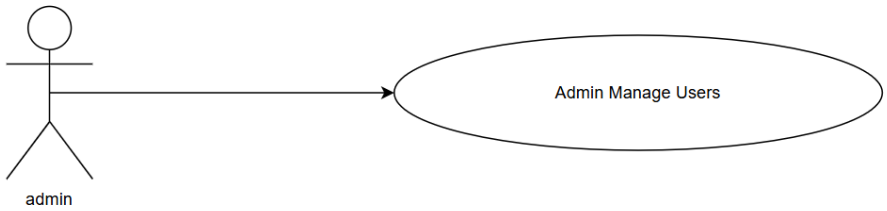

Figure 3.13: Use Case diagram for Admin Manage Users

Table 3.12: Use Case: Admin Manage Users

Field	**Details**
Use Case ID	UC12
Title	Admin Manage Users
Description	Admin manages all user accounts.
Actor	Admin
Pre-Condition	Admin must be authenticated.
Basic Flow	1. Admin opens Users List. 2. Edits/Deletes user.
Alternate Flow	Invalid action → Error.
Post-Condition	User updated.

3.5.13 Admin Search Projects/Gigs

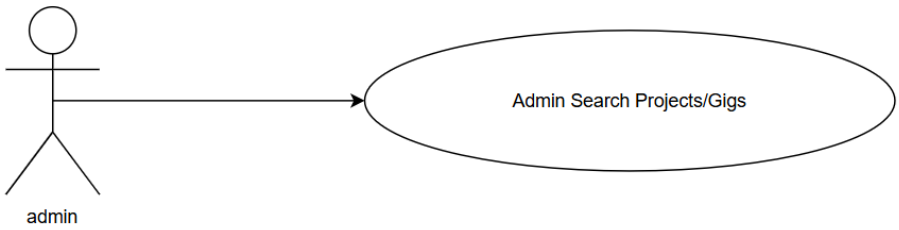

Figure 3.14: Use Case diagram for Admin Search Projects/Gigs

Table 3.13: Use Case: Admin Search Projects/Gigs

Field	**Details**
Use Case ID	UC13
Title	Admin Search Projects/Gigs
Description	Admin views all projects/gigs.
Actor	Admin
Pre-Condition	Admin logged in.
Basic Flow	1. Enter filters. 2. System retrieves data. 3. Display results.
Alternate Flow	No results → Notify admin.
Post-Condition	Projects/gigs displayed.

3.5.14 Secure Payment

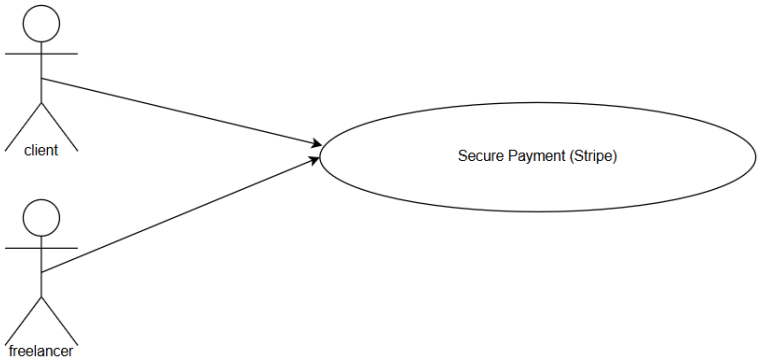

Figure 3.15: Use Case diagram for Secure Payment

Table 3.14: Use Case: Secure Payment

Field	**Details**
Use Case ID	UC14
Title	Secure Payment
Description	Handles transactions through Stripe.
Actor	Client, Freelancer
Pre-Condition	Valid payment method.
Basic Flow	1. Client initiates payment. 2. Stripe processes. 3. Funds held. 4. Released after approval.
Alternate Flow	Payment failed → Retry.
Post-Condition	Payment completed.

3.5.15 Submit Review

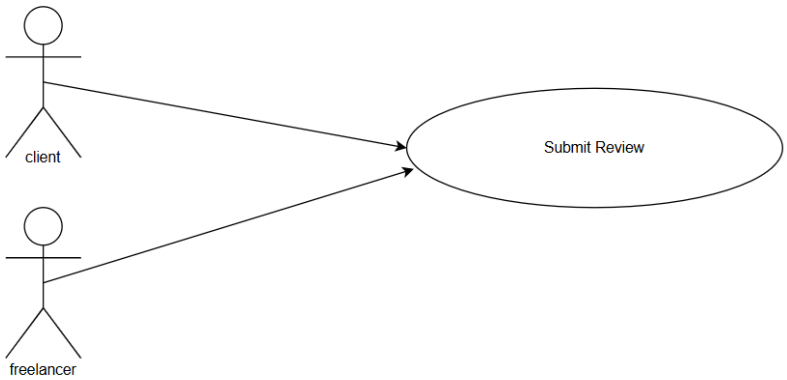

Figure 3.16: Use Case diagram for Submit Review

Table 3.15: Use Case: Submit Review

Field	**Details**
Use Case ID	UC15
Title	Submit Review
Description	User submits a rating & review after project completion.
Actor	Client or Freelancer
Pre-Condition	Completed order/contract.
Basic Flow	1. User opens review form. 2. Submits rating/comment. 3. System stores review.
Alternate Flow	Invalid rating → Error.
Post-Condition	Review saved.

3.5.16 View Orders / Contracts

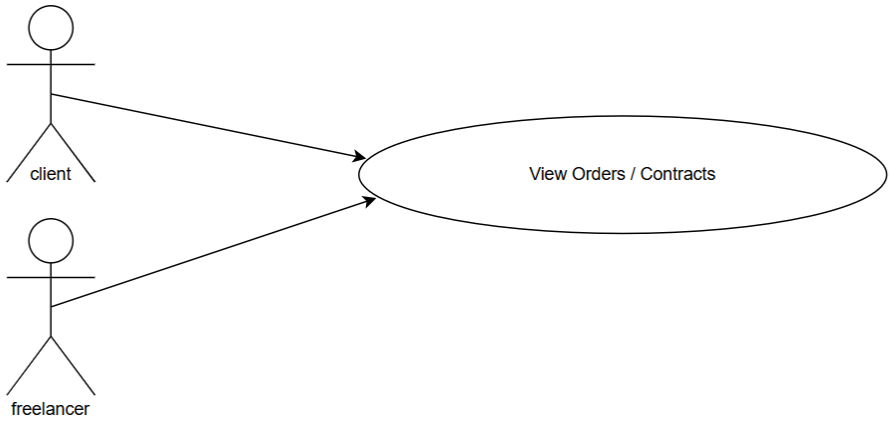

Figure 3.17: Use Case diagram for View Orders / Contracts

Table 3.16: Use Case: View Orders / Contracts

Field	**Details**
Use Case ID	UC16
Title	View Orders / Contracts
Description	Users view their active & completed orders.
Actor	Client, Freelancer
Pre-Condition	Logged in.
Basic Flow	1. Open “Orders”. 2. System retrieves data. 3. Displays order details.
Alternate Flow	No orders found.
Post-Condition	Orders/Contracts displayed.

3.5.17 Change Project/Gig Status

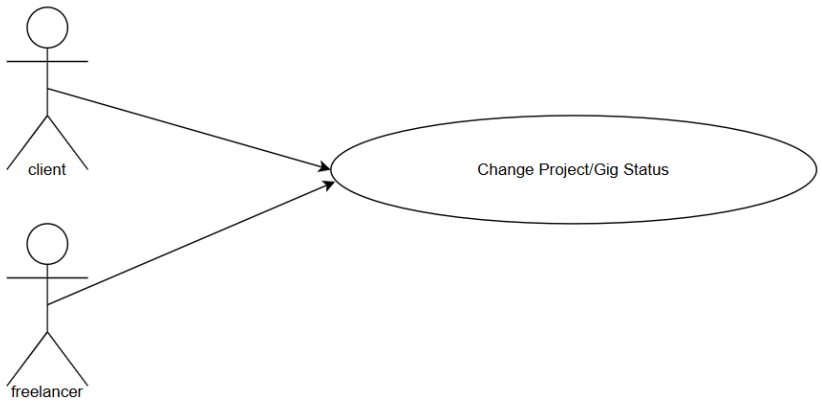

Figure 3.18: Use Case diagram for Change Project/Gig Status

Table 3.17: Use Case: Change Project/Gig Status

Field	**Details**
Use Case ID	UC17
Title	Change Project/Gig Status
Description	User updates project/gig status (pending, working, completed).
Actor	Client, Freelancer
Pre-Condition	Must be project owner.
Basic Flow	1. User selects new status. 2. System updates Firestore. 3. Notify other users.
Alternate Flow	Unauthorized $\rightarrow$ Error.
Post-Condition	Status updated.

3.5.18 View Logged-in Users

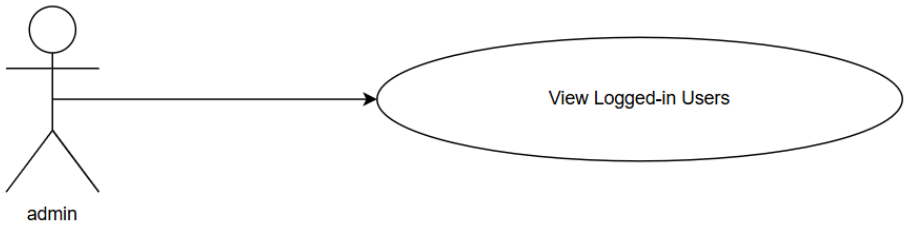

Figure 3.19: Use Case diagram for View Logged-in Users

Table 3.18: Use Case: View Logged-in Users

Field	**Details**
Use Case ID	UC18
Title	View Logged-in Users
Description	Admin views all active users.
Actor	Admin
Pre-Condition	Admin logged in.
Basic Flow	1. Admin opens Active Users. 2. The system displays online users.
Alternate Flow	None.
Post-Condition	Active user list shown.

3.5.19 Assign User Role

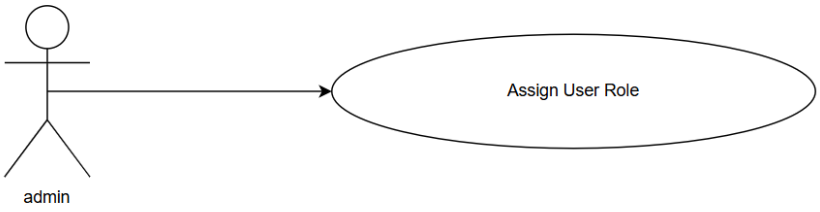

Figure 3.20: Use Case diagram for Assign User Role

Table 3.19: Use Case: Assign User Role

Field	Details
Use Case ID	UC19
Title	Assign User Role
Description	Admin assigns or changes user roles.
Actor	Admin
Pre-Condition	Valid users must exist.
Basic Flow	1. Admin selects user. 2. Modifies role. 3. System updates profile.
Alternate Flow	Invalid role → Error.
Post-Condition	User role updated.

3.5.20 View User Queries

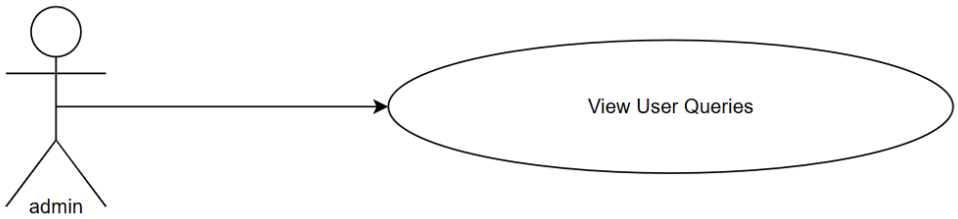

Figure 3.21: Use Case diagram for View User Queries

Table 3.20: Use Case: View User Queries

Field	Details
Use Case ID	UC20
Title	View User Queries
Description	Admin views complaints/queries submitted by users.
Actor	Admin
Pre-Condition	Query must exist.
Basic Flow	1. Admin opens Queries Panel. 2. System loads queries. 3. Admin views or resolves.
Alternate Flow	No queries → Show “No queries found”.
Post-Condition	Queries displayed.

3.5.21 Logout

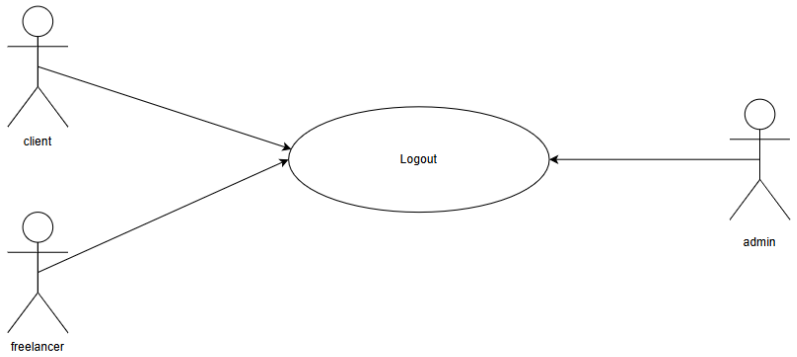

Figure 3.22: Use Case diagram for Logout

Table 3.21: Use Case: Logout

Field	**Details**
Use Case ID	UC21
Title	Logout
Description	User logs out of the platform.
Actor	Client, Freelancer, Admin
Pre-Condition	User must be logged in.
Basic Flow	1. User clicks “Logout”. 2. System invalidates session/JWT. 3. Redirect to homepage.
Alternate Flow	None.
Post-Condition	User successfully logged out.

Chapter 4

Design

The system design provides the information on how the elements of **FreelanceNest** are connected to form a secure, scalable and intelligent freelancing site. It addresses the architectural design, backend architecture, information flow, and ways to integrate AI, messaging, and payment modules. This chapter will give a much needed insight of the internal workings of the platform and the way each module will be connected to produce a seamless software solution.

The design strategy is based on:

- **Scalability:** has the potential to serve thousands of users with ease.
- **High Performance:** Designed to deliver high-performance in terms of accessing data and real-time communication.
- **Security:** Has features of secure authentication, role-based access control and encrypted payment processing.
- **Modularity:** Components that can be maintained and easily updated and upgraded.
- **AI Integration:** MAKES sure that there is no friction between the machine learning models and the backend.

The design will help **FreelanceNest** to offer a powerful, convenient, and scalable platform.

4.1 System Architecture

The system architecture provides the general structure of FreelanceNest, including the interaction between the frontend and backend, AI models, database, authentication, and third-party integration, such as Stripe. This architecture is done to help facilitate smooth flow of information and guarantee safe and effective processing of data by all the modules.

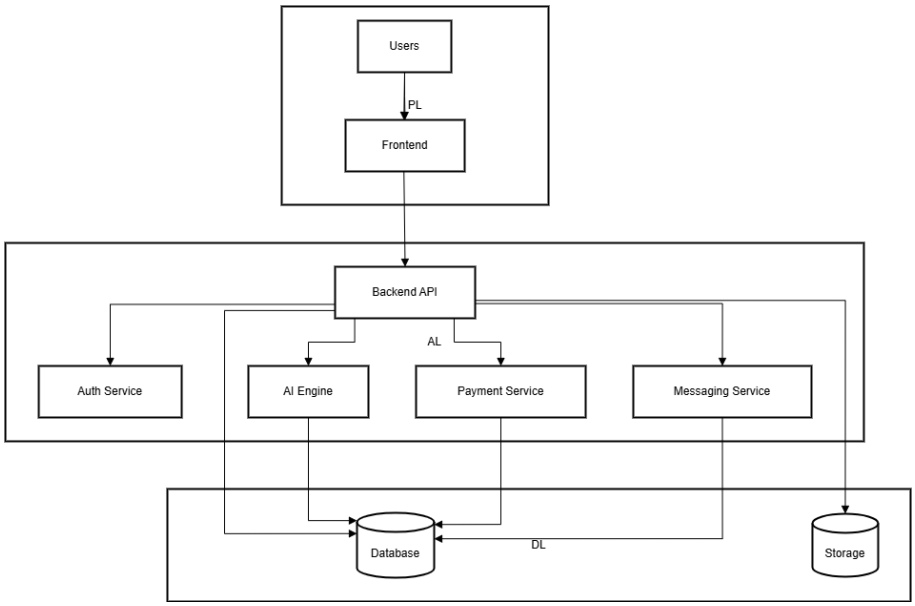

Figure 4.1: System Architecture Diagram

4.1.1 Architecture Overview

1. Frontend Layer (React) Provides dynamic UI for clients, freelancers, and admins. Communicates with the backend via secure REST APIs. Follows UI architecture guidelines described in industry standards [16].

2. Backend Layer (Node.js) Handles business logic, security, and integration with external systems. Backend modularity aligns with best practices in software design [17].

3. Database Layer (Firebase Firestore) Provides real-time updates, flexible NoSQL document storage, scalable reads/writes, and sub-collection structures. Firestore best practices follow database design principles [18]. Technical implementation follows Firebase documentation [19].

4. Authentication Layer (Firebase Auth + JWT) Manages identity, session management, and secure access control following recommended authentication standards [8].

5. AI Engine (Deep Learning Models) Implements job recommendations and skill matching using established methods in recommender systems [20, 21].

6. Payment Gateway (Stripe Integration) Secure payments, escrow, and fund release are implemented using Stripe's recommended APIs and standards.Escrow functionality ensures that payments are securely held until job completion, thereby protecting both clients and freelancers, while Stripe's built-in security mechanisms help prevent fraud and unauthorized transactions [22].

7. Real-Time Messaging (Firestore + Cloud Messaging) The real-time messaging

module enables instant communication between platform users using Firebase Firestore for real-time data synchronization and Firebase Cloud Messaging (FCM) for push notifications. This component supports features such as live chat, message delivery status, and notification alerts for important events like new messages or job updates. The use of Firebase's messaging infrastructure ensures low-latency communication, reliable message delivery, and seamless scalability as user activity increases [19].

4.1.2 System Architecture Workflow

Figure 4.2: System Architecture Workflow Diagram

4.1.3 Architecture Strengths

- **Very Scalable:** Firebase and React were used to support the increasing user demands.
- **Safe:** Has a strong authentication and coded messages.
- **Real-Time Updates:** It also has instant messaging features and notifications.
- **AI-Guided Matching:** Intelligent algorithms are applied to the job and freelancer matching.
- **Modular Backend:** This is made to be easily maintained and updated.

- **Trustworthy Payments:** Stripe-based transactions are uncomplicated and safe [23].
- **High Availability:** Cloud-based services ensure minimal downtime and continuous system accessibility.
- **Cross-Platform Compatibility:** The architecture supports consistent performance across web and mobile devices.

4.2 Design Constraints

Technical Constraints

- The application of FreelanceNest is created based on React.js, Node.js, Firebase, and Stripe with certain limitations. Firestore has a limit on document size and rate of reads/writes [19], Stripe lacks intrinsic support of escrow, and AI models cannot be served as native microservices. [22]
- Firebase real-time listeners and Firebase Cloud Messaging (FCM) are used in real-time messaging and notifications. Performance may be affected by network delays or network multiplicity.
- Cloud based services like Firestore, FCM and AI microservices create dependency to network connectivity and can be expensive to operate once user traffic increases.

Integration Constraints

- Stripe is strict in compliance, region support, and API workflow, which constrains the manner in which payments, payouts, and card information can be managed. The system cannot store sensitive information.
- AI models demand structured data to make appropriate suggestions. The interaction of Node.js and AI microservices also brings in latency, restricting real-time recommendations.

Security and Compliance Issues

- The system should be in line with Firebase security, JWT authentication, and Stripe standards based on PCI-DSS.[22] These limit access, storage, and processing of data.
- Role-Based Access Control (RBAC) restricts administrative operations, payment operations, and user management, requiring strict database read/write rules.

Scalability Limitations and Performance

- The pay-per-read/write mechanism in Firestore needs a cost-efficient data structuring to prevent unjustified cost increase.
- Node.js is single-threaded and cannot compute heavy AI procedures in-memory, requiring external microservice use for model inference.
- Live updates (chat, notifications, status change) demand continuous listeners, which negatively affects scalability.

Design Trade-Offs

- **Real-time vs Cost:** Firestore listeners add cost, whereas real-time communication was prioritized to improve the user experience.
- **AI Accuracy vs Response Speed:** Lightweight DL models (TF-IDF, cosine similarity) were selected over complex deep-learning models to reduce processing time.
- **Firebase vs SQL Databases:** Firebase was chosen for real-time capabilities despite SQL databases offering more robust querying.
- **Modular Architecture vs Development Time:** Modular architecture increases maintainability but expands development workload; trade-off accepted for scalability.

Assumptions

- Reliable internet connectivity for users to support real-time messaging and notifications.
- Clients and freelancers have valid payment methods compatible with Stripe.
- AI model accuracy will improve with additional training data.
- Initial system traffic will be moderate, allowing gradual scaling.
- Users provide accurate profile and project information necessary for reliable AI matching.

These assumptions, trade-offs, and constraints defined the system design and provided a sensible balance between performance, scalability, security, and user experience.

4.3 Design Methodology

4.3.1 Agile Software Development Methodology

FreelanceNest has adhered to the Agile approach because it is flexible, iterative, and supports changing requirements. Agile enables constant stakeholder engagement, fast feature development, and quick adjustment to feedback.[24] The project is divided into small steps (sprints) where each sprint delivers a working module.

Significant Agile Characteristics Used

1. **Refinement of the Development Process:**
   System was developed through multiple iterations focusing on modules: user authentication, posting projects, bidding by freelancers, messaging, payment management. Each release produced a fully working partial system.

2. **Constant Stakeholder Engagement:**
   Frequent meetings ensured requirements were validated early, misalignments were minimized, and features were refined based on feedback.

3. **Sprint Planning:**
   Priorities were set, tasks divided into actionable work items, and workload estimated in story points.

4. **Daily Standups (Internal):**
   Regular check-ins were held, obstacles tracked, and tasks reassigned as needed.

5. **Incremental Testing:**
   At the end of each sprint: unit testing, UI testing, integration testing (database and API), and user acceptance testing (UAT) were performed.

6. **Sprint Review & Retrospective:**
   Features were demonstrated, feedback recorded, and improvements incorporated into the next sprint.

7. **Continuous Integration / Continuous Deployment (CI/CD):**
   Automated build pipelines were used to ensure quick integration, early bug detection, and rapid deployment of new features.

8. **Backlog Grooming:**
   Requirements were regularly reviewed, refined, and prioritized to keep the product backlog organized and aligned with project goals.

This cycle continued until the entire platform was completed.

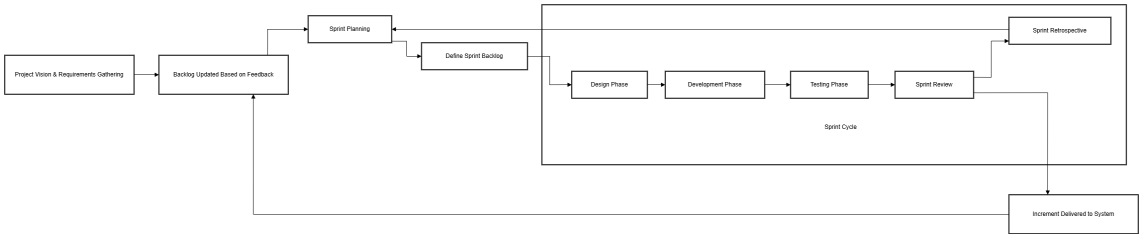

Figure 4.3: Methadology Diagram

4.4 High Level Design

4.4.1 Conceptual View

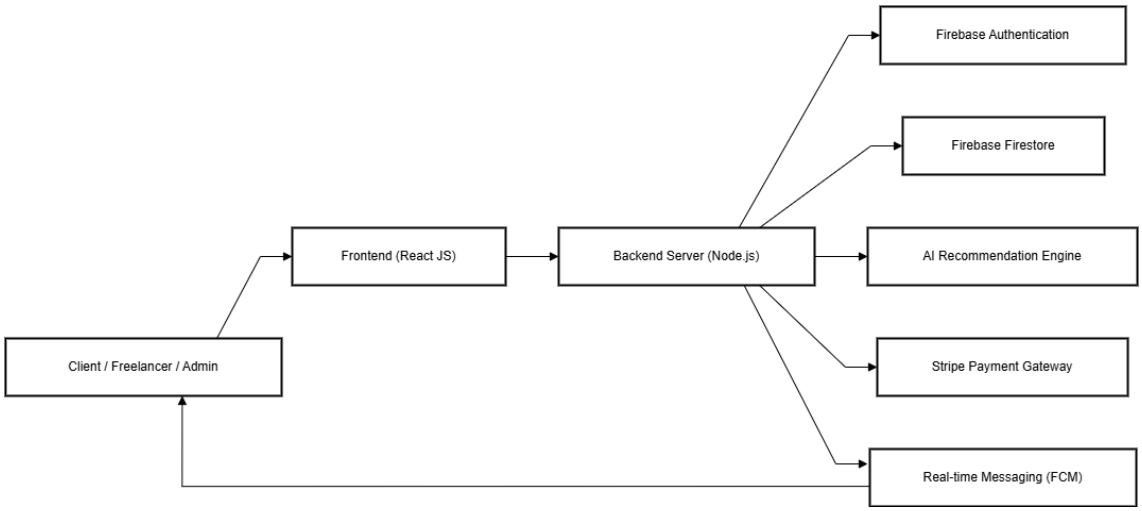

Figure 4.4: Conceptual View Diagram

4.4.2 Process View

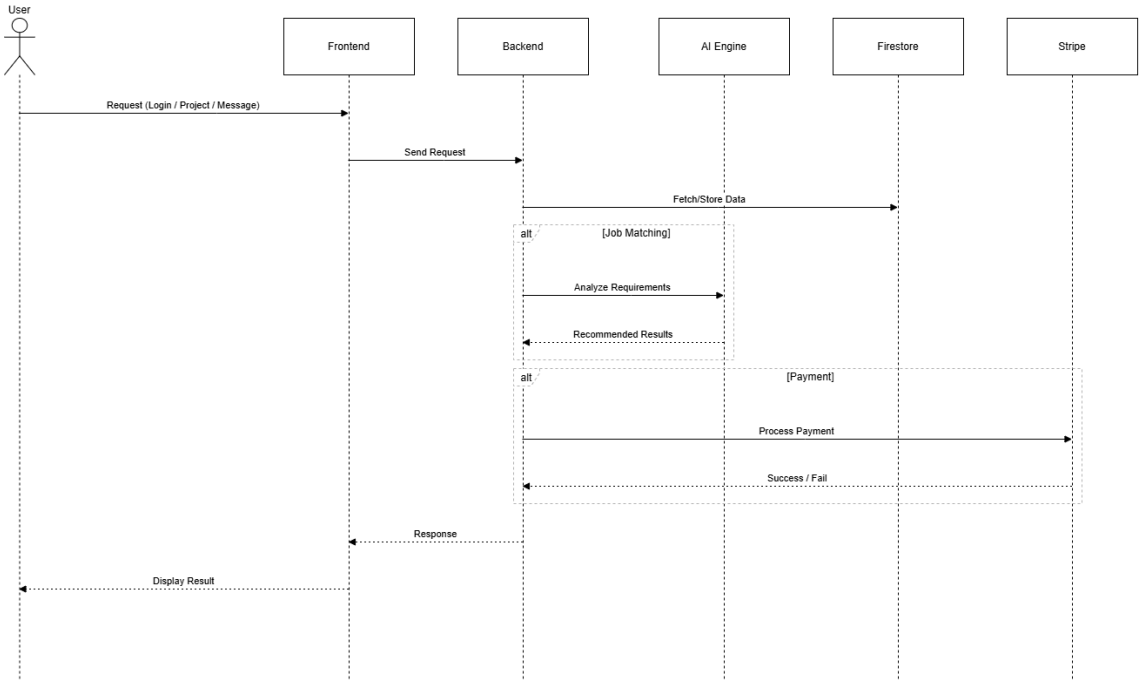

Figure 4.5: Process View Diagram

4.4.3 Physical View

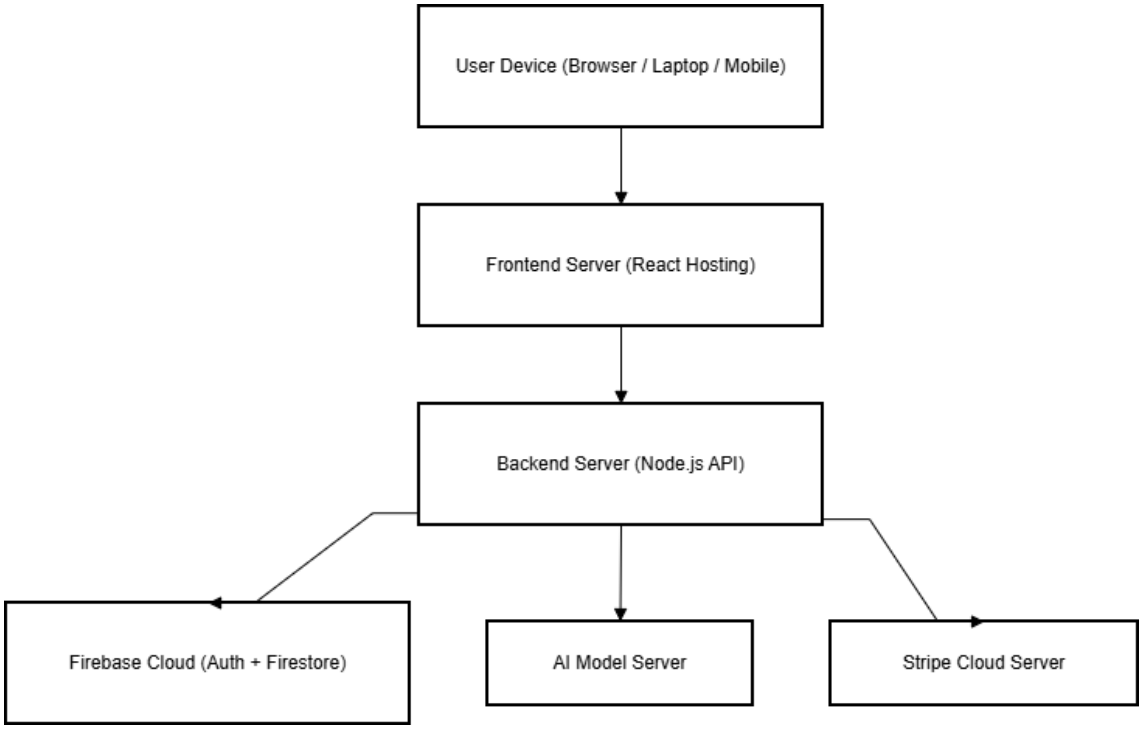

Figure 4.6: Physical View Diagram

4.4.4 Module View

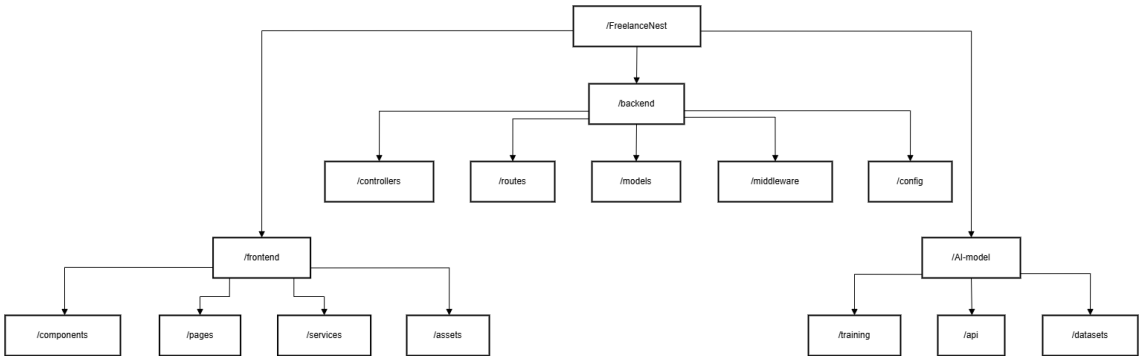

Figure 4.7: Module View Diagram

4.4.5 Security View

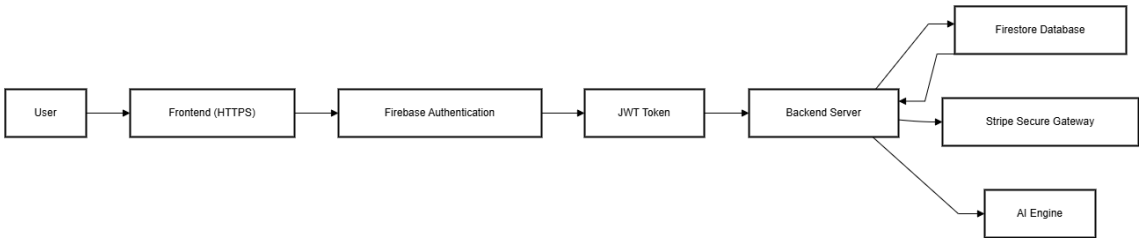

Figure 4.8: Security View Diagram

4.5 Low-Level Design: Major Component Detailed Design

This section is a detailed design of the key parts of the system of FreelanceNest in terms of responsibilities, interfaces, internal architecture, data contracts, error management, and scalability.[17]

1. Frontend (React)

Responsibilities:

- Offer job-related UI (Client / Freelancer / Admin).
- Form validation and rich UX in making projects/gigs.
- Live chats, notifications, and dashboards through Firestore listeners.
- Authenticate auth tokens and session state.

Logical Sub-Modules:

- **Authentication/ Profile Module:** login, register, password reset, profile edit.
- **Dashboard Module:** action cards, analytics, charts, and widgets.
- **Project/Gig Module:** creation forms, listing, and filters.
- **Messaging Module:** chat interface, read receipts, typing indicators.
- **Payments Module:** checkout process, order summary, status pages.

APIs / Interfaces:

- CRUD endpoint(s) on backend (projects, gigs, bids, orders) through REST/GraphQL.
- Firestore listeners for messages, notifications, and order updates.
- Stripe.js for client-side card creation and tokenization.

2. Backend API (Node.js)

Responsibilities:

- Business logic enforcement and RBAC (Client / Freelancer / Admin).
- Authenticate and authorize requests to reduce spam and malicious nodes.
- Coordinate between Firestore, AI microservice, and Stripe.
- Handle input validation, logging, metrics, and rate limiting.

Logical Sub-Modules / Services:

- **Auth Service:** authenticates Firebase tokens, issues internal JWTs, manages refresh tokens.
- **User Service:** CRUD for user profiles and role management.
- **Project Service:** create/read/update/delete projects, indexing for AI recommendations.
- **Bid & Order Service:** manage proposals, milestones, and complete order lifecycle.
- **Payment Service:** integrates with Stripe (payment intents, escrow handling, payouts).
- **Chat Service:** chat metadata, moderation hooks, and message retention.
- **AI Integration Service:** communicates with AI microservice for matching and stores results.

Security & Reliability:

- Use Helmet, strict CORS policies, and request size limits.
- Centralized error-handling middleware with structured logs (request ID tracing).
- Retry mechanism with exponential backoff for external service calls (Stripe, AI service).

Scalability:

- Stateless API servers behind a load balancer; sessionless authentication via JWT.
- Background processes for long-running tasks (AI indexing, report generation, refunds).

3. Database Layer (Firebase Firebase App + Cloud Storage)

Responsibilities:

- Unlimited storage of users, projects, gigs, chats, bids, orders, payments, and logs.
- Real-time chat and notifications using Firestore listeners.

Collections & Data Shapes (excerpt):

- `users/{uid}` profile, role, permissions, timestamps.
- `projects/{projectId}` budget, deadline, clientRef, skills[], status.
- `gigs/{gigId}` packages map, images[], rating.
- `bids/{bidId}` amount, freelancerRef, projectRef, milestones.
- `messages/{conversationId}/messages/{messageId}` sender, receiver, text, attachments.
- `orders/{orderId}` gigRef, buyerRef, sellerRef, status, paymentRef.

Indexes:

- Composite indexes on `status + createdAt` for improved listing performance.
- Full-text search on titles/descriptions via indexing or external search service (e.g., Algolia).

Security Rules:

- RBAC enforced at Firestore rules layer.

- Owners can update only their own projects/gigs.
- Admins have full read access.

Scalability & Cost Strategy:

- Keep hot collections (e.g., messages) sharded by `conversationId` and paginate results.
- Archive old messages to Cloud Storage or BigQuery for analytics and cost reduction.

4. AI Recommendation Microservice (Recommendation Engine)

Responsibilities:

- Compute freelancer–project relevance scores and provide ranked recommendations.
- Periodically re-index profiles and projects as data updates.
- Provide ML models for fraud detection and skill verification.

Design & Pipeline:

- **Ingestion:** Receive project deltas and user profiles from backend (via Pub/Sub or webhook).
- **Preprocessing:** Normalize skills, tokenize descriptions, compute embeddings (TF-IDF, Word2Vec, transformer embeddings).
- **Scoring:** Cosine similarity + heuristics (recent activity, success rate, response time).
- **Serving:** Expose REST endpoints returning top-N ranked freelancers or projects.
- **Offline Training:** Batch retraining and evaluation pipelines (CI for models).

Scalability & Availability:

- Containerized microservice (Kubernetes) with autoscaling rules.
- Redis caching for frequently requested matches to reduce latency.

5. Payments (Stripe)

Responsibilities:

- Create PaymentIntents and maintain escrow for secure transactions.
- Handle refunds, dispute workflows, and payout scheduling.[19, 22]

Flow (Backend Responsibilities):

- Create a PaymentIntent when an order is generated.
- Upon client confirmation, charge the card and update the order with `paymentRef`.
- Funds remain in escrow until milestone completion or final approval, then released to the freelancer.

Security:

- No raw card data stored; use Stripe Elements / tokenization only.
- Strong server-side ownership verification before triggering payout releases.

6. Notification & Messaging (Firestore + FCM)

Responsibilities:

- Real-time messaging between platform users.
- Push notifications for important events (new bid, message, payment update).
- Offline message delivery so users receive messages once reconnected.
- Maintain conversation history for auditing and dispute resolution.

Design Notes:

- Messages saved in `messages` subcollection; clients listen using `onSnapshot`.
- Typing indicators and read receipts stored in lightweight metadata docs.
- FCM used for device push notifications; notifications must be idempotent and include deep-links.
- Message payloads optimized to reduce Firestore read/write costs.
- Rate-limiting applied to prevent message spamming and reduce server load.

7. Admin & Analytics

Responsibilities:

- Platform moderation, user/account management, and operational analytics.
- Track aggregate metrics: revenue, active users, disputes, time-to-hire.
- Generate monthly reports for performance insights and decision-making.

- Monitor fraudulent activities and enforce compliance across the platform.

Design Notes:

- Admin dashboard implemented using protected routes with strict authentication and authorization.
- Analytics data aggregated using scheduled backend jobs or cloud functions to minimize real-time load.
- Use of Firestore queries and indexes optimized for read-heavy analytics operations.
- Role-based access control (RBAC) ensures admins only access permitted modules.
- Logs maintained for all admin actions to ensure accountability and auditability.
- Integration-ready design for future BI tools or advanced analytics platforms.

4.5.1 UML Class Diagram

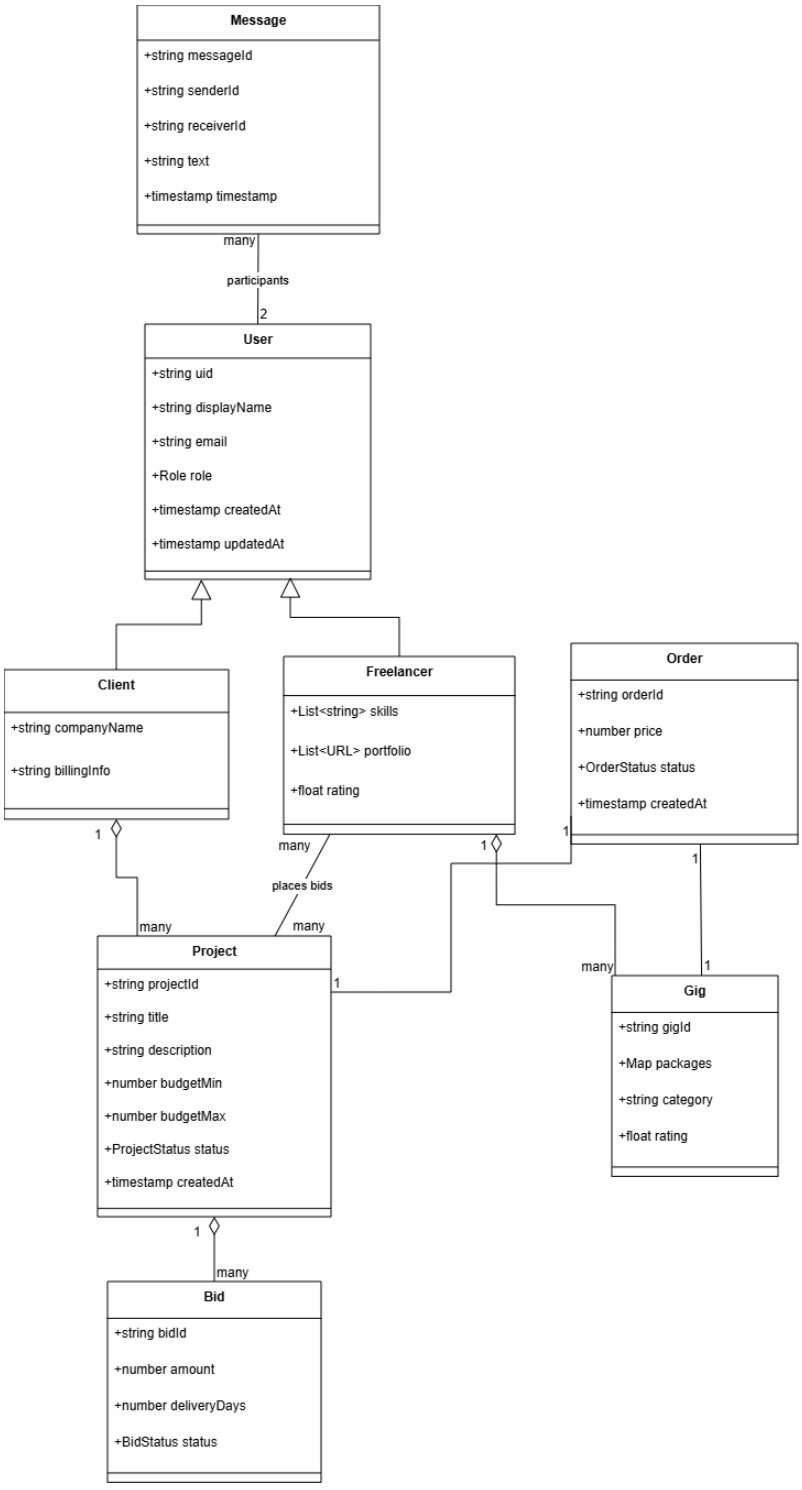

Figure 4.9: UML Class Diagram

4.5.2 Sequence Diagrams

1. User Registration Sequence

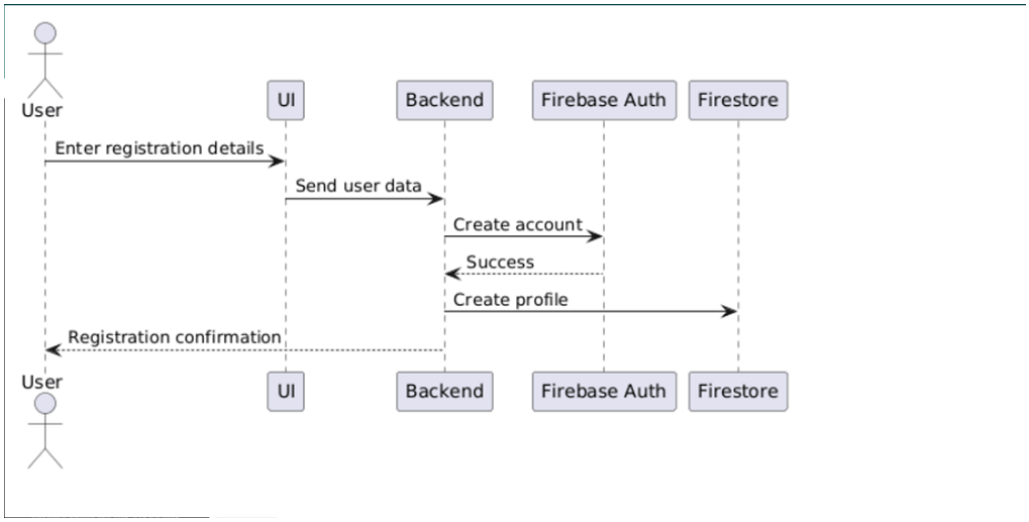

Figure 4.10: User Registration Sequence diagram

2. Login Sequence

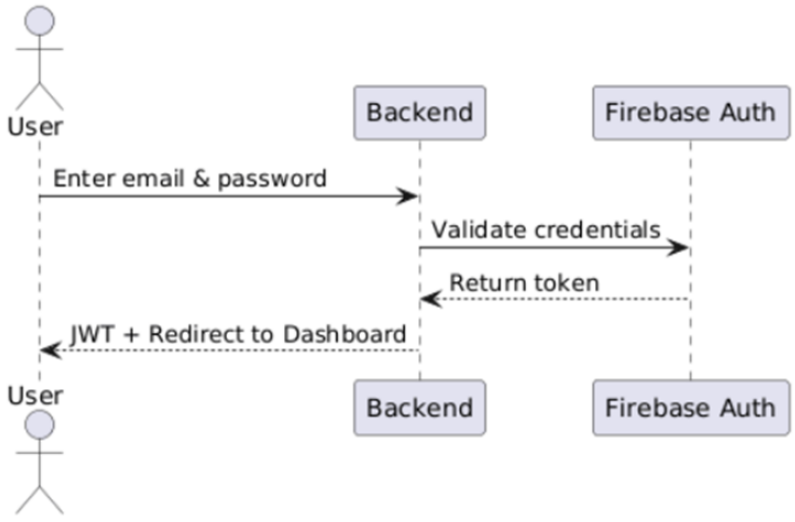

Figure 4.11: Login Sequence diagram

3. Post Project Sequence

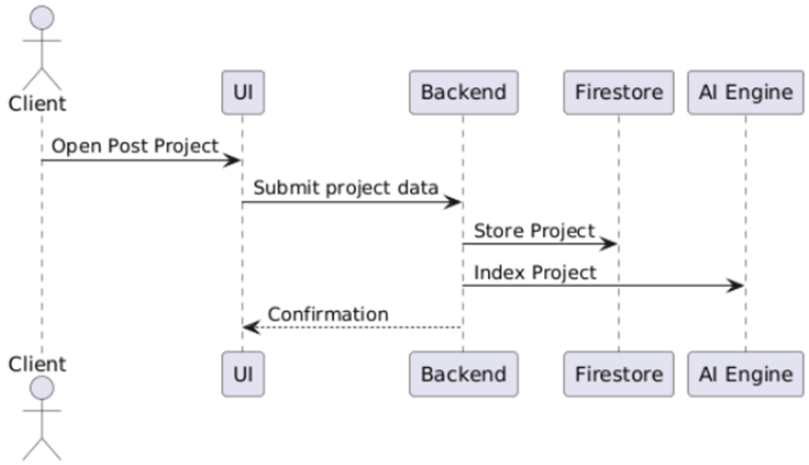

Figure 4.12: Post Project Sequence diagram

4. Messaging Sequence

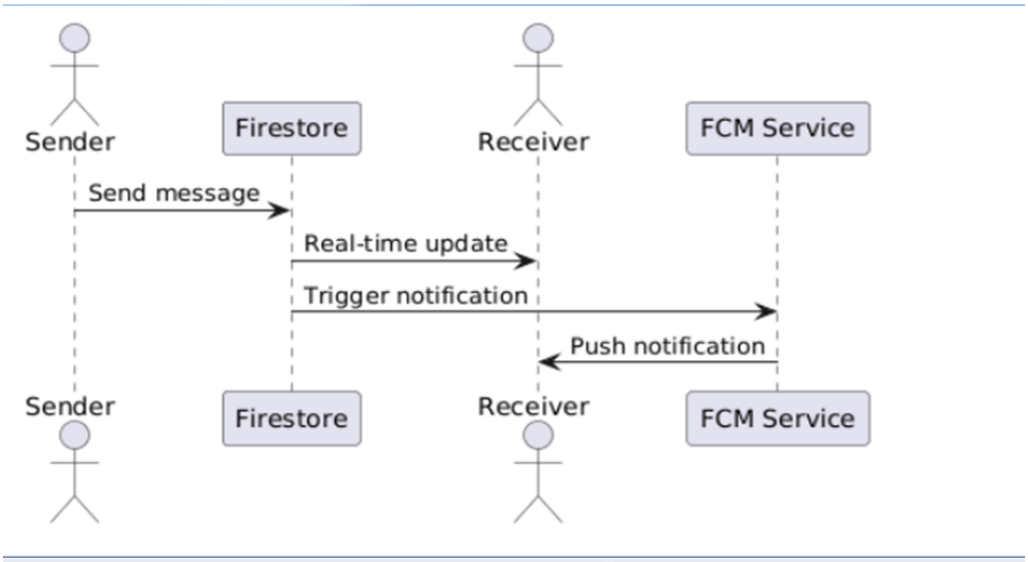

Figure 4.13: Messaging Sequence diagram

5. Secure Payment Sequence

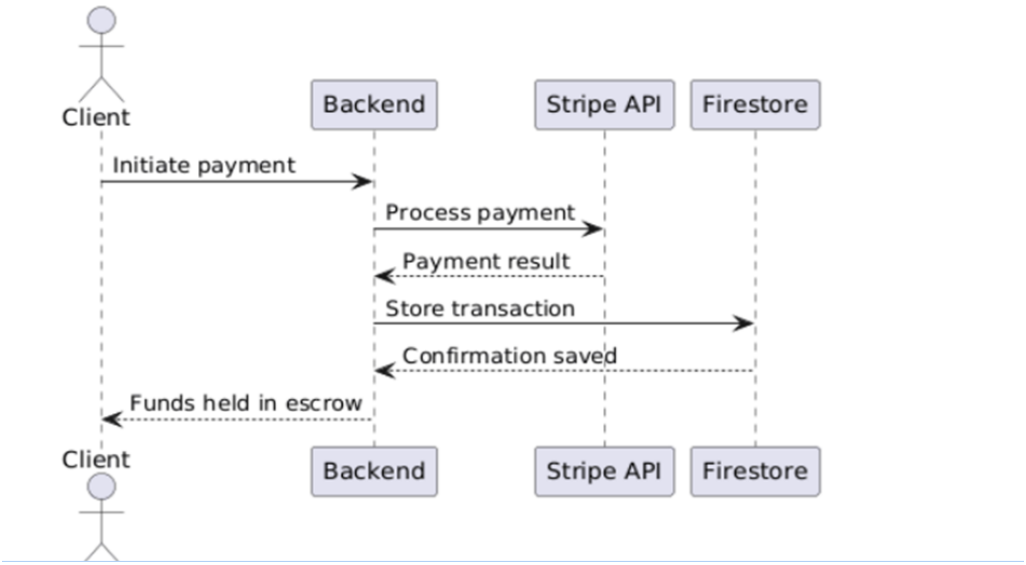

Figure 4.14: Secure Payment Sequence diagram

5. Logout Sequence

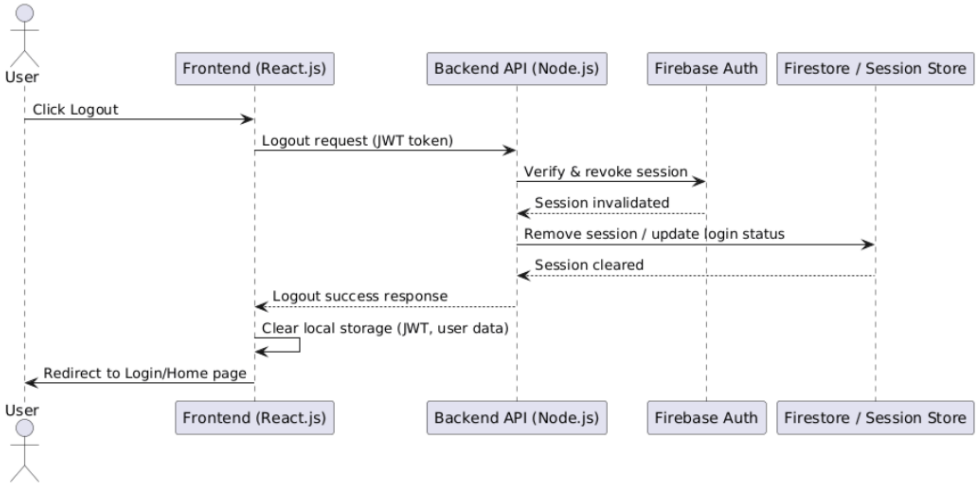

Figure 4.15: Logout Sequence diagram

4.5.3 Flowcharts for Core Functions

1. User Registration flow

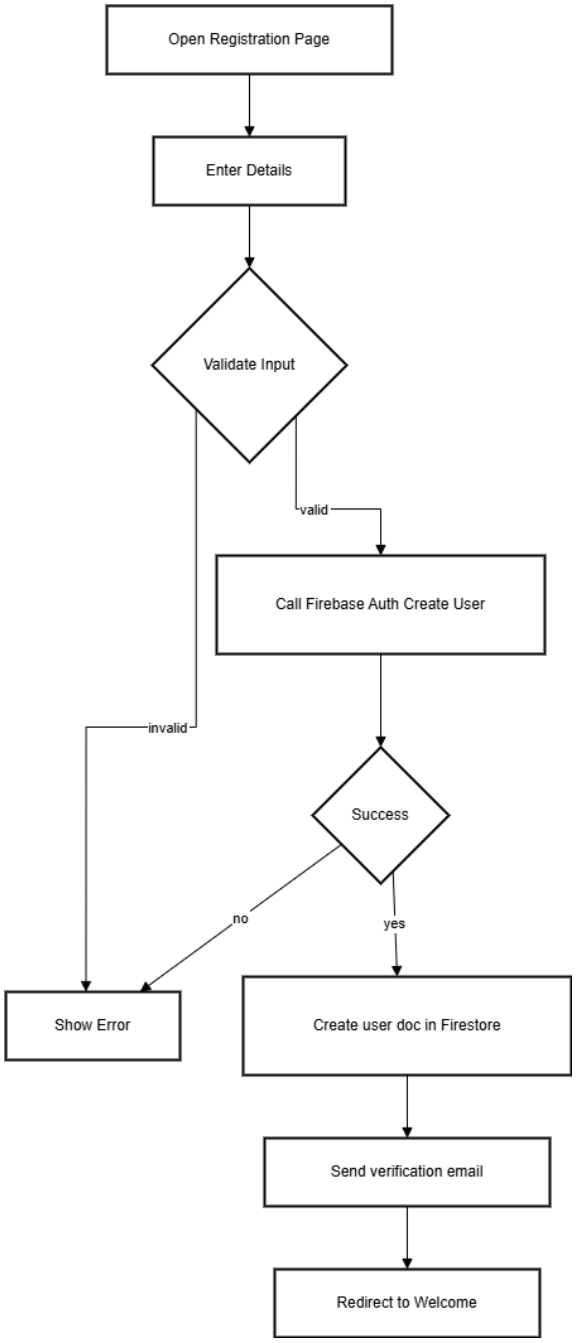

Figure 4.16: User Registration flow diagram

2. Post Project Flow

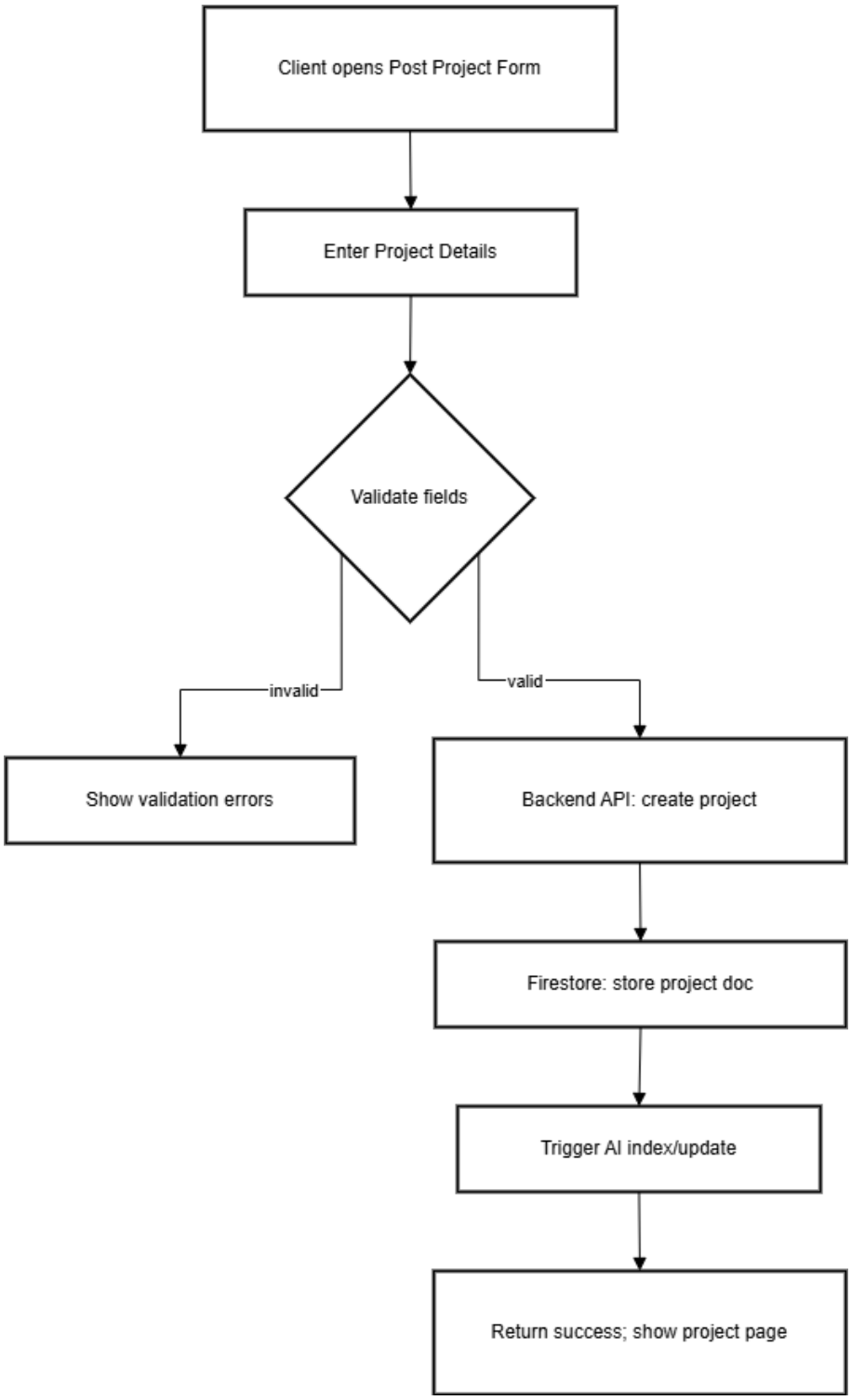

Figure 4.17: Post Project flow diagram

3. AI-Based Matching Flow

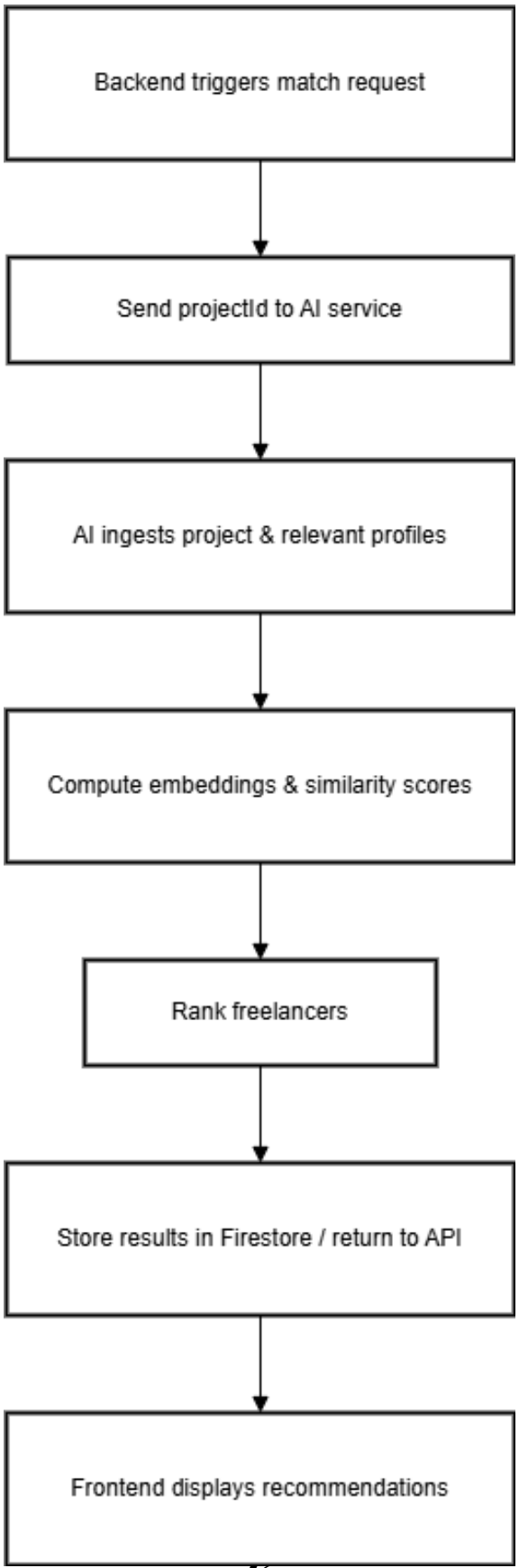

Figure 4.18: AI-Based Matching flow diagram

4. Secure Payment (Order) Flow

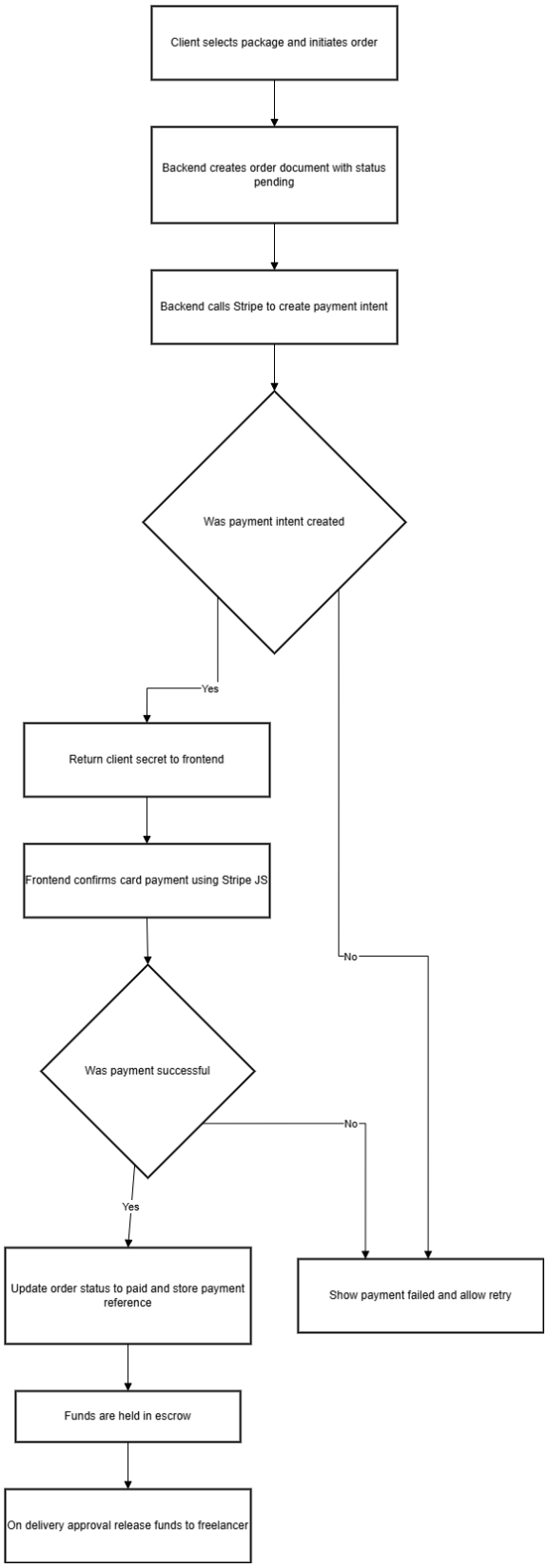

Figure 4.19: Secure Payment (Order) flow diagram

5. Messaging Flow

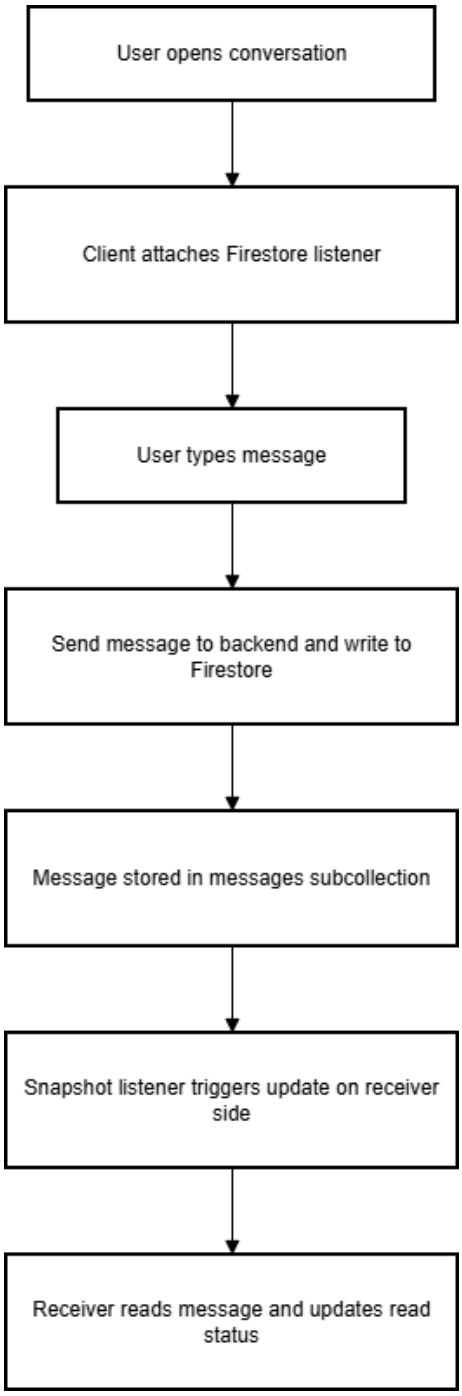

Figure 4.20: Messaging flow diagram

4.6 Database Design

4.6.1 Entity Relationship Diagram

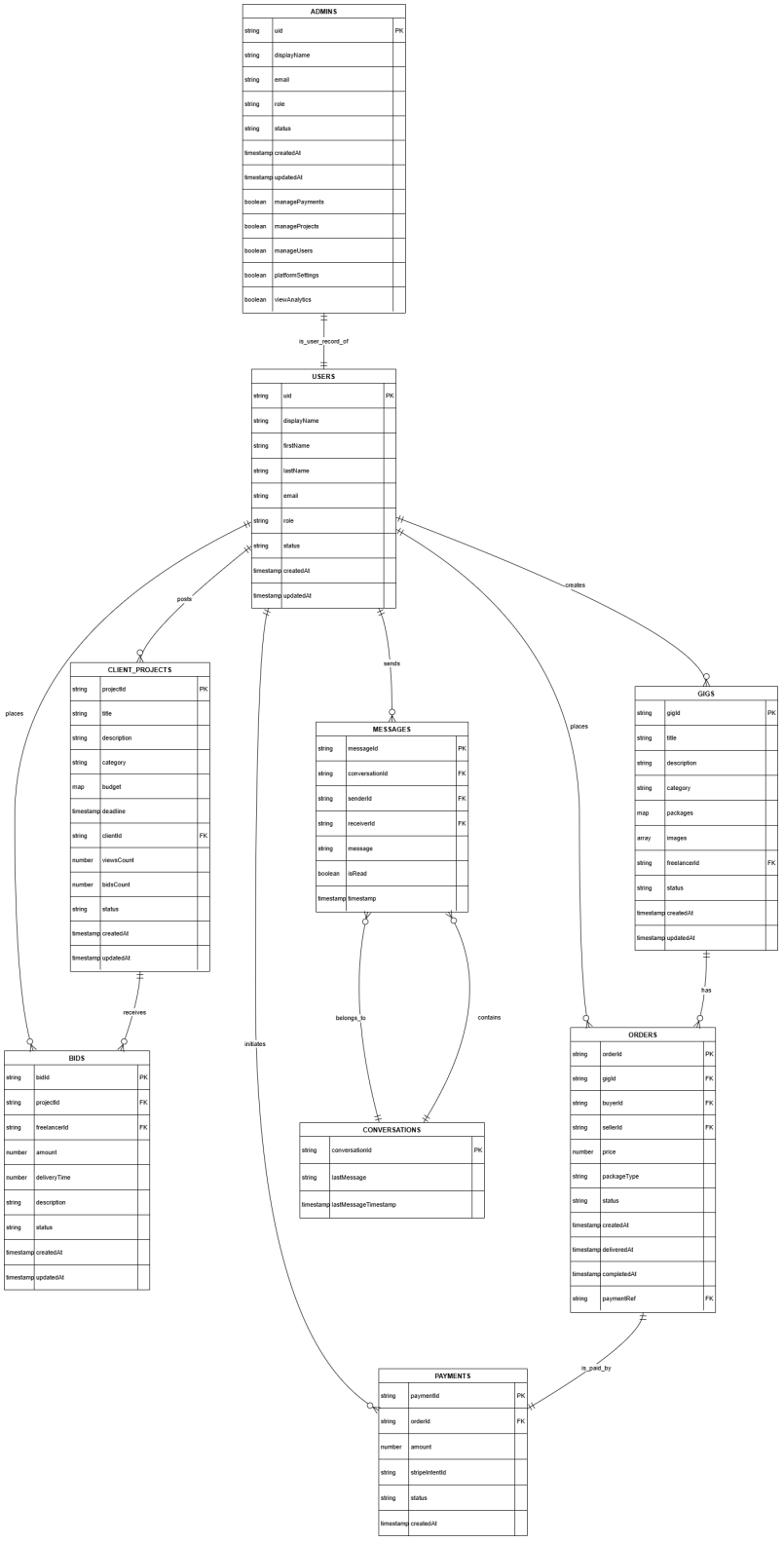

Figure 4.21: Entity Relationship Diagram

4.6.2 DBMS Data (Firestore Observe Data)

All the Firestore collections and their purpose are described below.

Database Collections

1. **Admin Collection**
   Path: admins/{uid}
   Purpose: Stores account permissions and account information.
   Fields include:
   - createdAt, updatedAt, displayName, firstName, lastName, email
   - permissions map: { managePayments, manageProjects, manageUsers, platformSettings, viewAnalytics }
   - status, role, uid
2. **Users Collection**
   Path: users/{uid}
   Purpose: Stores freelancer and client accounts.
   Fields include:
   - createdAt, updatedAt, displayName, firstName, lastName, email
   - role (freelancer/client), permissions, uid, status
3. **ClientProjects Collection**
   Path: clientProjects/{projectId}
   Purpose: Stores posted projects.
   Fields include:
   - title, description, category, skills[]
   - budget map: { min, max, range, type }
   - deadline, clientId, clientEmail, createdAt, updatedAt
   - status, viewsCount, bidsCount
4. **Bids Collection**
   Path: bids/{bidId}
   Purpose: Stores bids by freelancers.
   Fields include:
   - amount, deliveryTime, description
   - milestones[]
   - freelancerId, freelancerEmail, projectId

- status, createdAt, updatedAt

5. **Gigs Collection**
   Path: gigs/{gigId}
   Purpose: Stores freelancer service listings.
   Fields include:

   - title, description, category, rating, tags[], images[]
   - packages map: { basic, standard, premium }
   - freelancerId, freelancerEmail, freelancerLevel
   - status, createdAt, updatedAt

6. **Orders Collection**
   Path: orders/{orderId}
   Purpose: Stores gig purchase orders and lifecycle events.
   Fields include:

   - gigTitle, buyerId, buyerName, gigId
   - packageType, price, canSubmit
   - createdAt, updatedAt, deliveredAt, completedAt
   - paymentRef, status

7. **Conversations / Typing Indicators / Chats**
   Paths:

   - conversations/{conversationId}
   - conversations/{conversationId}/messages/{messageId}
   - typing/{conversationId}/{userId}

   Purpose: Real-time messaging, read receipts, typing indicators.

8. **Payments Collection**
   Path: payments/{paymentId}
   Purpose: Stores Stripe payment metadata.
   Fields include:

   - orderId, amount, stripeIntentId, status, createdAt

4.7 GUI Design

4.7.1 Login Interface

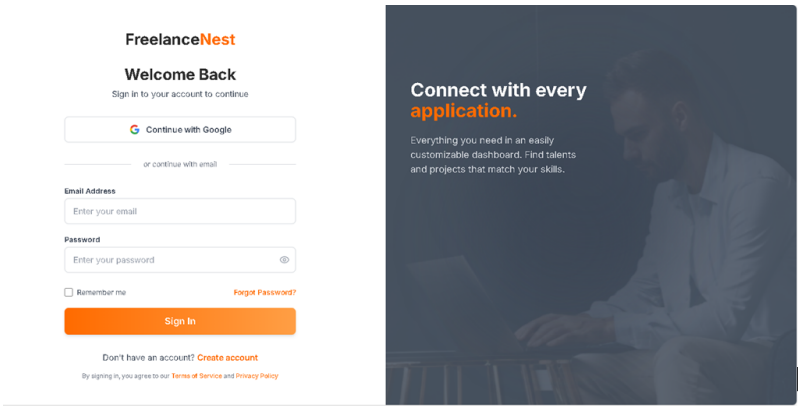

Figure 4.22: Login Screen

4.7.2 Registration Interface

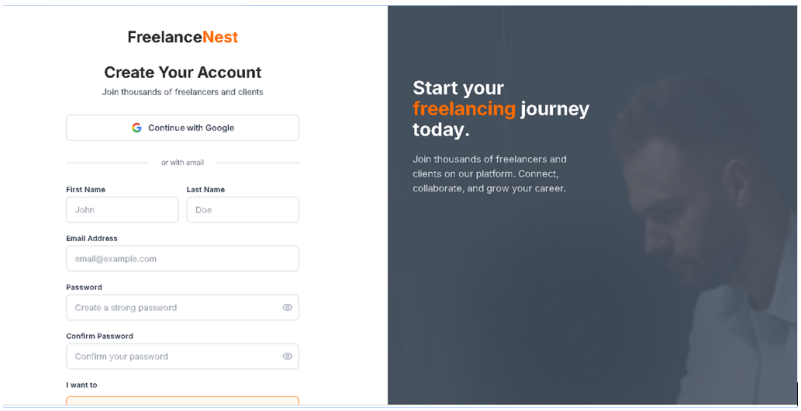

Figure 4.23: Registration Screen (1)

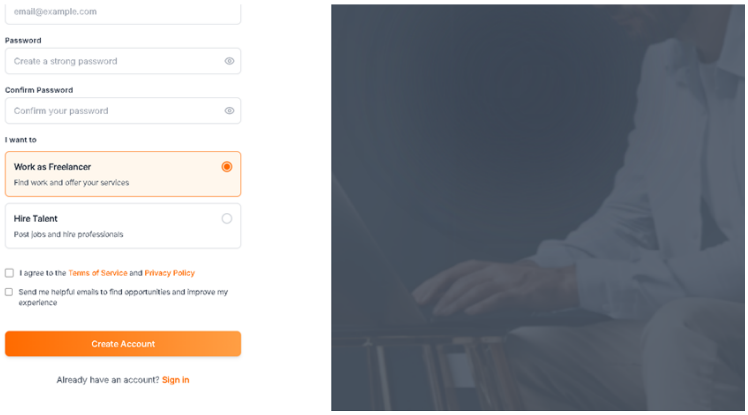

Figure 4.24: Registration Screen (2)

4.7.3 Client Dashboard UI

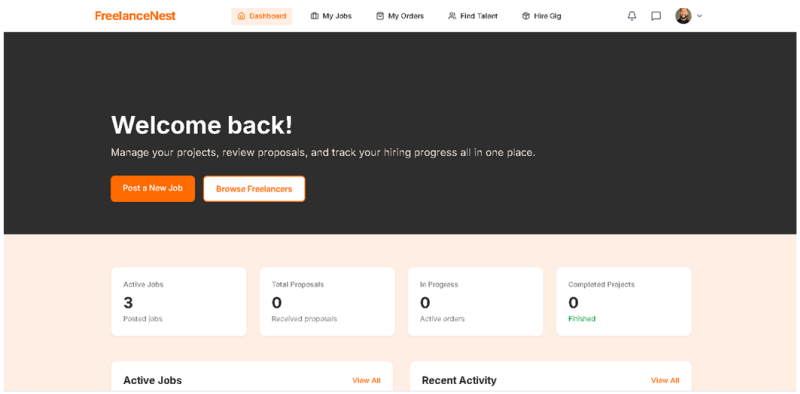

Figure 4.25: Client Dashboard Screen (1)

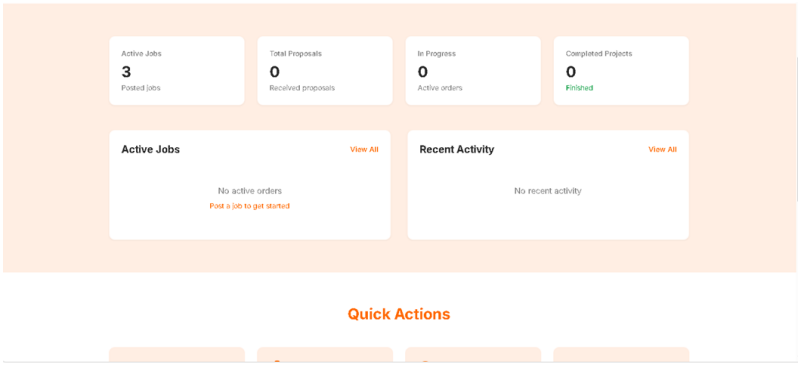

Figure 4.26: Client Dashboard Screen (2)

4.7.4 Freelancer Dashboard UI

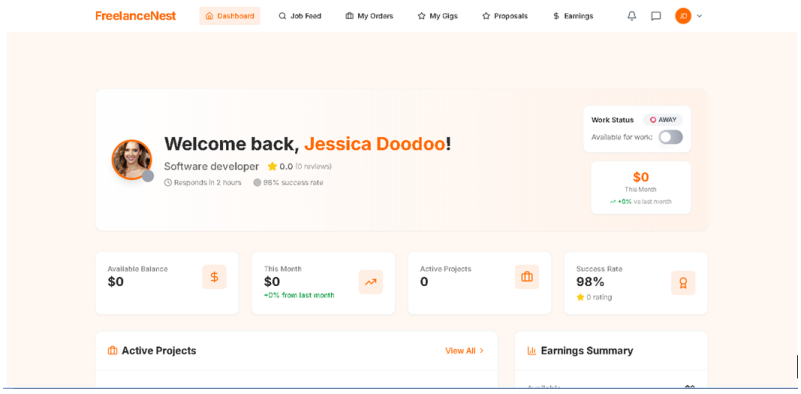

Figure 4.27: Freelancer Dashboard Screen (1)

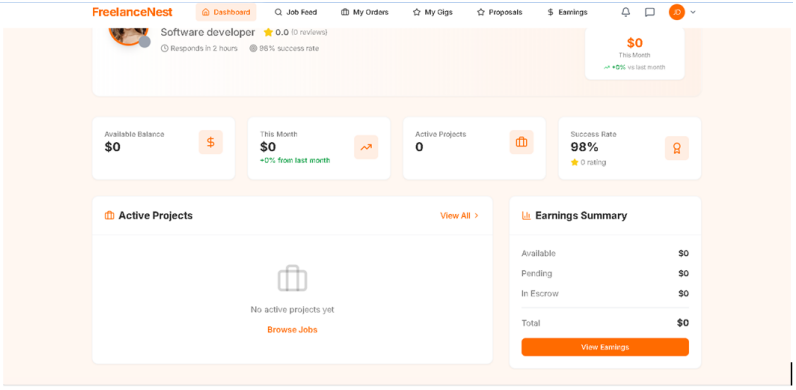

Figure 4.28: Freelancer Dashboard Screen (2)

4.7.5 Messaging Interface

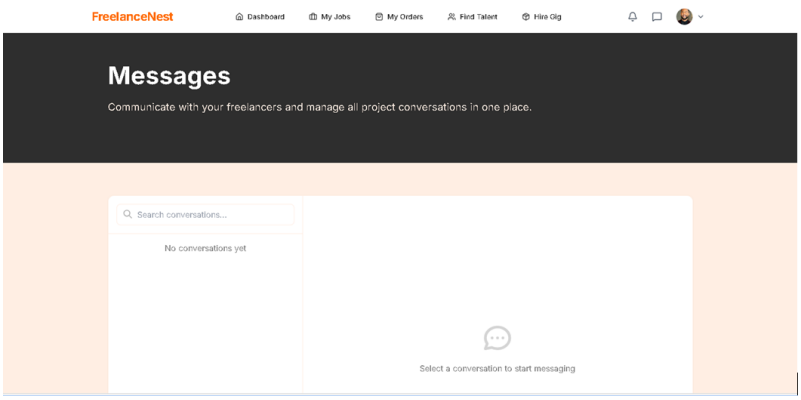

Figure 4.29: Messaging Screen

4.8 External Interfaces

4.8.1 Firebase Authentication

Purpose

Firebase authentication is used to operate safe user login, user registration, password reset, and verification of identity. **FreelanceNest** has chosen Firebase to store user credentials at scale, securely, and without the need to create a more sophisticated system.

Interface Type

- Firebase Authentication SDK (JavaScript / Admin SDK)
- REST API (used internally by Firebase services)

Data Sent to Firebase Auth

- Registering the email and password
- Login credentials
- Requests to generate verification emails
- Token verification requests from the server

Data Received from Firebase Auth

- Unique user identifier (uid)
- Authentication token (ID token / refresh token) allows users to access their profiles and data
- Authentication status and error codes
- Email verification status
- Password reset confirmation

System Integration

- Frontend uses Firebase Web SDK for sign-in/up
- Backend verifies ID tokens via the Firebase Admin SDK for secure API access
- Firestore collections use the `uid` as the primary foreign key

4.8.2 Firebase Cloud Storage & Firestore

Purpose

- Firestore stores structured platform data (users, projects, bids, gigs, orders, conversations)
- Cloud Storage stores all media files (gig images, profile photos, chat attachments)

Interface Type

- Firestore Client SDK
- Firestore REST API (indirect)
- Cloud Storage SDK
- Real-time snapshot listeners (WebSockets under the hood)

Data Sent to Firestore / Storage

- User profile records
- Project postings and updates
- Bid submissions
- Gig details and media files
- Order creation and payment updates
- Chat messages and attachments

Data Received from Firestore / Storage

- Real-time document updates
- Query results for listings
- Media download URLs
- Conversation updates
- Project/gig details

System Integration

- **Frontend:** Uses Firestore client SDK to read/write and subscribe to real-time listeners
- **Backend:** Uses Admin SDK for validated server-side operations
- Firestore acts as the central database system, although externally hosted

4.8.3 Stripe Payment Gateway

Purpose

Stripe handles all payment transactions such as card payments, escrow holding, refunds, and payouts to freelancers. [19, 22]

Interface Type

- Stripe REST API
- Stripe Node.js SDK
- Webhooks (optional)

Data Sent to Stripe

- Amount and currency
- Buyer information
- Gig/order identifiers
- Metadata (orderId, freelancerId, buyerId)
- Request to create PaymentIntent

Data Received from Stripe

- paymentIntentId
- Payment status (requires_action, succeeded, failed)
- Client secret for Stripe.js
- Error codes (insufficient funds, card declined)

System Integration

- Backend manufactures a PaymentIntent using Stripe API.
- Payment is verified by frontend by the use of client secret with Stripe.js.
- Backsends on success update orders and payment in Firestore.
- All card information is to be safely processed by Stripe (PCI compliance).

4.8.4 Recommendation Microservice

Purpose

The AI microservice matches freelancers and client projects using natural language processing and vector-based similarity scoring [21, 20].

Interface Type

- REST API (HTTP/HTTPS)
- JSON request/response format

Data Sent to AI Service

The input structure follows hybrid recommender system practices [2].

- Project description, category, skills
- Freelancer profile features
- Ratings and experience metrics

Information Obtained by AI Service

The outputs resemble traditional collaborative-filtering metadata structures [25].

- Ranked list of recommended freelancers
- Similarity scores and metadata
- Integration and error status codes

System Integration

This workflow aligns with widely used architectures for embedding-based recommendation pipelines [20].

- Backend sends project/freelancer data to AI microservice
- Microservice generates embeddings and calculates similarity rankings
- Backend stores results and delivers them to frontend via Firestore or direct API response

Chapter 5

System Implementation

5.1 Frontend Implementation

FreelanceNest is developed using React.js, which offers fast rendering, modular components, and mobile-friendly performance [26].

5.1.1 Key Features

- Reusable UI components (buttons, forms, modals, cards).
- Dynamic dashboards for Clients, Freelancers, and Admins.
- Navigation using React Router.
- State management using Context API.
- Continuous connection to backend APIs.

5.1.2 Frontend Tools

- **React.js:** Used for building interactive interfaces [26].
- **React Router:** Enables smooth navigation between screens.
- **Firebase SDK:** Used for real-time messaging and notifications [19].

5.2 Backend Implementation

FreelanceNest uses Node.js as a backend technology due to its asynchronous, scalable, and event-driven architecture.

Backend Modules

- **User Module:** Registration, login, roles, logout.
- **Project/Gig Module:** Posting and updating gigs or projects.
- **Messaging Module:** Chat endpoints and message logs.
- **Payment Module:** Integrated with Stripe for secure transactions [22].
- **Automated Intelligence Module:** Handles recommendation requests.
- **Admin Module:** User management and permissions.

Backend Functionalities

- **JWT Authentication:** Secure session handling.
- **Input Validation:** Checks request payloads.
- **Error Handling:** Centralized middleware.
- **Stripe API Integration:** Payment processing [22].
- **AI Microservice Integration:** Communicates with DL recommendation engine [20].

Backend Stack

- Node.js
- Firebase Admin SDK [19]
- JWT
- Stripe SDK [22]

5.3 AI/DL Implementation

The AI module uses convolutional neural networks (CNNs) and embedding-based similarity models for freelancer-to-project matching [20, 21]. These techniques help capture semantic patterns in skills and project descriptions.

AI Features

- **Skill Matching:** Maps skills to project requirements using learned embeddings.
- **Job Recommendations:** Personalized suggestions based on historical patterns.

Fundamental Deep Learning Methodologies

- **Convolutional Layers:** Extract feature patterns from embeddings.
- **Pooling Layers:** Reduces dimensions while preserving key features.
- **Fully Connected Layers:** Predict match scores.
- **Softmax Layer:** Produces ranking probabilities.

AI Pipeline

1. Collect freelancer skills and project descriptions.
2. Preprocess and embed textual inputs.
3. Feed embeddings into CNN model.
4. Calculate similarity/relevancy scores.
5. Return recommended freelancer list via API.

The resulting AI recommendations are stored in Firestore for real-time retrieval [19].

5.4 Database Implementation

FreelanceNest makes use of Firebase Firestore as its main database, as the database can be synchronized in real-time and be scaled.

Major Collections

- Users
- Projects
- Gigs
- Messages
- Chats
- Payments
- Orders
- Bids

Key Firestore Features

- **Document-Based Storage:** Scalable and flexible data structure.
- **Real-Time Updates:** The real-time updates are possible with the help of the listeners to ensure the data are synchronized on the spot.
- **Role-Based Access Rules:** Provides secure and regulated access to data.
- **Cloud Storage:** Files uploading and management.

5.5 Authentication Implementation

The user management is secure and efficient using Firebase Authentication along with backend-based JWT session management [5, 17].

Features

- **Email/Password Login:** Basic and safe user authentication [19].
- **Password Reset:** Helps users securely recover access to their accounts [19].
- **Secure Session Handling:** Provides stateless and secure user sessions using JWT [17].
- **Role-Based Access Control:** Enforces controlled access based on assigned roles [27].
  - Client
  - Freelancer
  - Admin

Security Measures

- **JWT Expiration and Refresh:** Prevents unauthorized session reuse and token abuse [17].
- **Email Verification:** Verifies user identity during the registration process [19].
- **Access Control:** Restricts unauthorized access to protected system resources [8].

5.6 Payment Integration

FreelanceNest uses Stripe as a secure payment gateway to enable smooth and reliable transactions between clients and freelancers [6].

Payment Features

- **Payment Initiation:** Clients can easily initiate payments through secure checkout sessions [22].
- **Escrow Holding:** Funds are securely held until project completion and approval [22].
- **Fund Release:** Payments are released to freelancers after client confirmation [22].
- **Transaction Logging:** Firestore maintains records of all financial transactions [5].
- **Error Handling:** Robust mechanisms handle failed or incomplete transactions [17].

Payment Flow

- **Client Places an Order:** Initiates the payment transaction [22].
- **Stripe Processes the Card:** Ensures secure and compliant payment processing [22].
- **Funds Sitting in Escrow:** Protects both parties until project fulfillment [22].
- **Funds Released:** Payments are transferred after project approval [22].
- **Logs Stored:** All transaction data is persisted in the database [5].

5.7 Messaging Implementation

The communication system is implemented as a real-time chat module using Firestore and Firebase Cloud Messaging (FCM) to ensure instant message delivery [19].

Messaging Features

- **One-on-One Chat:** Direct communication between clients and freelancers [19].
- **Read Receipts:** Tracks message delivery and read status [19].
- **Typing Indicators:** Displays real-time typing status of users [19].
- **Real-Time Sync:** Ensures immediate UI updates as messages change [19].
- **Push Notifications:** Notifies users of new messages using FCM [19].

How It Works

1. **Message Storage:** Messages are stored in Firestore collections [18].
2. **Live Updates:** Firestore snapshot listeners provide real-time UI synchronization [19].
3. **Notifications:** FCM delivers instant message alerts to users [19].

5.8 Admin Panel Implementation

The Admin Panel provides centralized administrative control over the FreelanceNest platform [23].

Admin Features

- **User Administration:** Administrators can view, edit, disable, or delete user accounts [27].
- **Project/Gig Management:** Enables monitoring and control of posted projects and gigs [16].
- **Role Assignment:** Supports assigning and modifying user roles [27].
- **Active User Monitoring:** Tracks platform activity and logged-in users [23].
- **Query/Report Handling:** Manages user complaints and reports efficiently [17].
- **Analytics of Platform:** Provides insights into system performance and user behavior [23].

Admin Tools

- **Firestore Queries:** Enables structured data retrieval and management [18].
- **Role-Based Permissions:** Ensures secure access control across admin functions [8].
- **Server-Side Logs:** Maintains audit trails for accountability and monitoring [17].

5.9 Error Handling

Effective error handling is essential for maintaining system stability, reliability, and user trust in distributed software systems [17, 16].

Error Handling Implemented In

- **Backend API Validation:** Ensures that all incoming requests are properly validated before processing [17].
- **Payment Failures:** Safely processes transaction errors and failed payment scenarios [22].
- **AI Service Failures:** Handles unavailability or failure of recommendation services gracefully [21].
- **Authentication Problems:** Manages login failures and access-related issues [19].
- **Network / Database Failures:** Minimizes the impact of connectivity or data access issues [18].

Error Responses

- **HTTP Status Codes:** Uses standard and meaningful HTTP error codes to indicate failure types [17].
- **Custom Messages:** Provides user-friendly and descriptive error messages [16].
- **Error Logging in Firestore:** Supports debugging and system monitoring through centralized logs [5].
- **Retry Mechanisms:** Automatically retries transient failures to improve system resilience [17].

5.10 Security Implementation

Security was treated as a core requirement throughout the development lifecycle of the system [8, 17].

Security Measures

- **JWT-Based Authentication:** Ensures secure and stateless session management [17].
- **Hashing Passwords:** Securely handled using Firebase Authentication mechanisms [19].
- **Stripe Encryption:** Protects payment data and transactions using industry-standard encryption [22].
- **Role-Based Access Control (RBAC):** Restricts access according to assigned user roles [27].
- **Firebase Security Rules:** Enforces strict read and write permissions on cloud data [19].

- **Malicious File Upload Prevention:** Protects the system from harmful or unauthorized file uploads [8].
- **HTTPS:** Secures all API communication channels [8].
- **Input Sanitization and Validation:** Prevents injection attacks and invalid data submission [8].

Protected Actions

- **Administrative Role Assignments:** Restricted to authorized administrators only [27].
- **Project Updates:** Prevents unauthorized modification of project data [16].
- **Payment Operations:** Secured to ensure transaction authenticity and integrity [22].
- **Messaging Access:** Limited to relevant and authorized participants [19].
- **User Management:** Protected to maintain overall platform security [8].

Chapter 6

System Testing and Evaluation

6.1 Unit Testing

Unit testing was conducted to ensure the reliability of individual system components. Each function was tested for proper input handling and accurate output generation. The key modules tested included authentication, project posting, gig creation, AI integration, and messaging.

6.2 Graphical User Interface Testing

GUI testing was performed to verify that all user interfaces load properly and function as expected. Key aspects tested included screen alignment, field visibility, responsiveness, and user interactions [10].

6.2.1 Register

Table 6.1: Test Case TC_01 – Register

TC_01			
Register			
UC_01			
The system should have an environment ready.			
Steps	**Tasks**	**Expected Output**	**Result (Pass/Fail)**
1	Open register screen.	Pass or Fail	Pass
2	Enter required fields.	Pass or Fail	Pass
3	Press register button.	Pass or Fail	Pass
4	Check button is submitting data.	Pass or Fail	Pass

6.2.2 Login

Table 6.2: Test Case TC_02 – Login

TC_02			
Login			
UC_02			
User must have an account.			
Steps	**Tasks**	**Expected Output**	**Result (Pass/Fail)**
1	Open login screen.	Pass or Fail	Pass
2	Enter credentials.	Pass or Fail	Pass
3	Click login button.	Pass or Fail	Pass
4	Verify redirection.	Pass or Fail	Pass

6.2.3 Forgot Password

Table 6.3: Test Case TC_03 – Forgot Password

TC_03			
Forgot Password			
UC_03			
User must have a registered email.			
Steps	**Tasks**	**Expected Output**	**Result (Pass/Fail)**
1	Open password reset page.	Pass or Fail	Pass
2	Enter registered email.	Pass or Fail	Pass
3	Press reset button.	Pass or Fail	Pass
4	Verify reset email sent.	Pass or Fail	Pass

Table 6.4: Test Case TC_03 – Forgot Password

TC_03			
Forgot Password			
UC_03			
User must have a registered email.			
Steps	**Tasks**	**Expected Output**	**Result (Pass/Fail)**
1	Open password reset page.	Pass or Fail	Pass
2	Enter registered email.	Pass or Fail	Pass
3	Press reset button.	Pass or Fail	Pass
4	Verify reset email sent.	Pass or Fail	Pass

6.3 Functional Testing

Functional testing was carried out to ensure the system operates as intended, meeting all specified requirements. Core functionalities were validated by simulating real-world user interactions.

6.3.1 Post Project

Table 6.5: Test Case TC_04 – Post Project

TC_04			
Post Project			
UC_06			
Client must be logged in.			
Steps	**Tasks**	**Expected Output**	**Result (Pass/Fail)**
1	Open project form.	Pass or Fail	Pass
2	Enter project details.	Pass or Fail	Pass
3	Click submit.	Pass or Fail	Pass
4	Confirm data stored.	Pass or Fail	Pass

6.3.2 Create Gig

Table 6.6: Test Case TC_05 – Create Gig

TC_05			
Create Gig			
UC_05			
Freelancer logged in.			
Steps	**Tasks**	**Expected Output**	**Result (Pass/Fail)**
1	Open gig form.	Pass or Fail	Pass
2	Input gig data.	Pass or Fail	Pass
3	Press create button.	Pass or Fail	Pass
4	Validate gig is saved.	Pass or Fail	Pass

6.3.3 Messaging System

Table 6.7: Test Case TC_06 – Messaging

TC_06			
Messaging			
UC_09			
Two users must be online.			
Steps	**Tasks**	**Expected Output**	**Result (Pass/Fail)**
1	Open chat screen.	Pass or Fail	Pass
2	Send a message.	Pass or Fail	Pass
3	Check real-time update.	Pass or Fail	Pass
4	Confirm message stored.	Pass or Fail	Pass

6.4 Exception Testing

Exception testing validates system stability by handling invalid data inputs and unexpected operations gracefully.

6.4.1 Invalid Login

Table 6.8: Test Case TC_07 – Invalid Login

TC_07			
Invalid Login			
UC_02			
Wrong credentials entered.			
Steps	**Tasks**	**Expected Output**	**Result (Pass/Fail)**
1	Enter incorrect email.	Pass or Fail	Pass
2	Enter incorrect password.	Pass or Fail	Pass
3	Click login.	Pass or Fail	Pass
4	Error message displayed.	Pass or Fail	Pass

6.5 Performance Testing

Performance testing verifies the scalability and efficiency of the system under load.

6.5.1 System Load

Table 6.9: Test Case TC_08 – Load Testing

TC_08			
Load Testing			
System-wide			
Multiple users simulated.			
Steps	**Tasks**	**Expected Output**	**Result (Pass/Fail)**
1	Simulate multiple logins.	Pass or Fail	Pass
2	Trigger concurrent requests.	Pass or Fail	Pass
3	Observe response time.	Pass or Fail	Pass
4	System remains stable.	Pass or Fail	Pass

Chapter 7

Conclusions

7.1 System Evaluation

Performance, reliability, usability, and security were evaluated based on established software quality criteria [16, 17]. Professional tests were completed successfully, and all modules performed well without critical failures [13, 28].

7.2 Discussion

Through the FreelanceNest platform, it provides:

- Secure Authentication
- Artificial Intelligence Job Recommendations
- Real-Time Messaging
- Project and Gig Management
- Secure Payment Integration

The testing indicated strong performance across all modules, and usability feedback highlighted a positive user experience [17].

7.3 Summary

The system is reliable, user-friendly, and conforms to all functional requirements [27]. It maintains stable performance under moderate load and provides real-time responsiveness [16]. Overall, the results confirm that FreelanceNest has been successfully implemented.

References

[1] Paul Resnick and Hal R. Varian. Recommender systems. In *Communications of the ACM*, volume 40, pages 56–58. ACM, 1997. Cited on pp. 1, 2, 8, and 12.

[2] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002. Cited on pp. 1, 2, 5, 8, 9, 10, 11, 12, and 68.

[3] Charu C. Aggarwal. *Recommender Systems: The Textbook*. Springer, 2016. Cited on pp. 1, 2, 3, 5, 6, 8, 9, 11, and 13.

[4] Meta Platforms Inc. React official documentation. https://react.dev, 2024. Cited on pp. 1, 3, 4, 5, and 6.

[5] Google. Firebase documentation. https://firebase.google.com/docs, 2024. Cited on pp. 1, 2, 3, 4, 6, 9, 11, 12, 13, 14, 72, 73, and 75.

[6] Stripe. Stripe api documentation. https://stripe.com/docs/api, 2024. Cited on pp. 1, 2, 3, 4, 6, 7, 9, 11, 13, 14, and 72.

[7] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Recommender Systems Handbook*. Springer, 2015. Cited on pp. 2, 3, 4, 5, 6, 9, 11, 12, and 13.

[8] OWASP Foundation. *OWASP Top 10 Security Risks*, 2024. Cited on pp. 2, 4, 6, 10, 37, 72, 74, 75, and 76.

[9] Ian Sommerville. *Software Engineering*. Pearson, 10 edition, 2015. Cited on pp. 2, 3, 4, and 13.

[10] Roger S. Pressman and Bruce R. Maxim. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 9 edition, 2019. Cited on pp. 2, 4, 13, and 77.

[11] Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, 3 edition, 2003. Cited on p. 3.

[12] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 2 edition, 2005. Cited on p. 3.

[13] Glenford J. Myers. *The Art of Software Testing*. Wiley, 3 edition, 2011. Cited on pp. 3 and 82.

[14] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 43–52, 1998. Cited on p. 9.

[15] Karl E. Wiegers and Joy Beatty. *Software Requirements*. Microsoft Press, 3 edition, 2013. Cited on p. 12.

[16] Ian Sommerville. *Software Engineering*. Pearson, 10 edition, 2015. Cited on pp. 37, 74, 75, 76, and 82.

[17] Roger S. Pressman and Bruce Maxim. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 9 edition, 2019. Cited on pp. 37, 44, 72, 73, 74, 75, and 82.

[18] Thomas Connolly and Carolyn Begg. *Database Systems: A Practical Approach to Design, Implementation, and Management*. Pearson, 6 edition, 2015. Cited on pp. 37, 74, and 75.

[19] Google. *Firebase Documentation*, 2024. Cited on pp. 37, 38, 39, 47, 66, 69, 70, 71, 72, 73, 74, 75, and 76.

[20] Charu C. Aggarwal. *Recommender Systems: The Textbook*. Springer, 2016. Cited on pp. 37, 67, 68, and 70.

[21] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Recommender Systems Handbook*. Springer, 2015. Cited on pp. 37, 67, 70, and 75.

[22] Stripe. *Stripe Developer Documentation*, 2024. Cited on pp. 37, 39, 47, 66, 70, 73, 75, and 76.

[23] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, Boston, MA, 3rd edition, 2012. Cited on pp. 39 and 74.

[24] Jim Highsmith. *Agile Software Development Ecosystems*. Addison-Wesley, 2002. Cited on p. 41.

[25] John Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI)*, 1998. Cited on p. 68.

[26] Meta Open Source. *React Documentation*, 2024. Cited on p. 69.

[27] Karl Wiegers and Joy Beatty. *Software Requirements*. Microsoft Press, 3 edition, 2013. Cited on pp. 72, 74, 75, 76, and 82.

[28] Paul Ammann and Jeff Offutt. *Introduction to Software Testing*. Cambridge University Press, 2 edition, 2016. Cited on p. 82.