Vid-AI

"Where Traditions Meet Technology"

Group Members

Faiza Alam Warraich (01-131222-016)

Asad E Bukhari (01-131222-045)

Supervisor: Dr. Kashif Sulan

A Final Year Project submitted to the Department of Software Engineering, Faculty of Engineering Sciences, Bahria University, Islamabad in the partial fulfillment for the award of degree in Bachelor of Software Engineering

May 2026

THESIS COMPLETION CERTIFICATE

Student Name: Faiza Alam Warraich **Enrolment No:** 01-131222-016

Student Name: Asad E Bukhari **Enrolment No:** 01-131222-045

Program of Study: Bachelor of Software Engineering

Project Title: Vid-AI

It is to certify that the above students' project has been completed to my satisfaction and my belief, it's that standard is appropriate for submission for evaluation. I have also conducted a plagiarism test of this thesis using HEC prescribed software and found a similarity index at 9% that is within the permissible limit set by the HEC. I have also found the thesis in a format recognized by the department.

Supervisor's Signature: _____

Date: 16/04/2026 **Name:** Dr. Kashif Sultan

CERTIFICATE OF ORIGINALITY

This is to certify that the intellectual content of the project Vid-AI is the product of our own work, except where properly and accurately cited in the acknowledgements and references. Materials taken from sources such as research journals, books, and the internet have been used solely to support, elaborate, compare, extend, or implement earlier work. Furthermore, this work has not been previously submitted by us for any degree, nor shall it be submitted in the future for the purpose of obtaining a degree from this University or any other institution. If this information is proven incorrect at any stage, the University is authorized to cancel our degree.

Name of the Student: Faiza Alam Warraich

Signature: _____ **Date:** 16/04/2026

Name of the Student: Asad E Bukhari

Signature: _____ **Date:** 16/04/2026

Abstract

VidAI is an intelligent, cross-platform web and mobile application designed to revolutionize the way Pakistani couples plan their weddings. With the cultural significance and logistical complexity of Pakistani weddings which typically span multiple ceremonies such as Nikah, Mehndi, Baraat, and Walima couples face significant challenges in discovering trustworthy vendors, managing multi-event budgets, and coordinating services across cities. Traditional wedding planning in Pakistan relies heavily on word-of-mouth referrals and manual coordination, which are time-consuming, inconsistent, and lack transparency. Existing digital solutions fail to address the cultural specificity of Pakistani weddings and offer no intelligent assistance for planning decisions. The main motivation behind developing this system was to create an affordable, accessible, and AI-powered platform that bridges the gap between scattered vendor markets and the modern couple's need for a unified, intelligent wedding planning experience.

In VidAI, users can register as customers or vendors and interact through dedicated portals. Customers can browse and search verified vendors by category, city, and price range, create bookings with conflict detection, manage multi-event budgets, communicate with vendors in real time, and make secure online payments. The system features a dual AI engine: a conversational AI assistant trained with Pakistani wedding domain knowledge helps users with planning queries, while a structured AI module generates personalized vendor recommendations ranked by budget proximity, automated budget plans with per-event cost breakdowns, and AI-crafted wedding invitation content with image generation. Vendors can manage their profiles, packages, and portfolios, accept or reject bookings, view analytics dashboards, and chat with customers. An Admin portal provides vendor verification, user management, booking oversight, report moderation, and system health monitoring. A React Native mobile app delivers full customer feature parity for on-the-go access, including push notifications for real-time updates.

Dedication

This project is dedicated to our parents, without whom none of this would have been possible due to their encouragement, hard work, and prayers. Our professors and supervisors for showing patience towards us and motivating us to do better than expected. Finally, this platform is for every small business owner and young couple in Pakistan that is struggling to organize their weddings.

Acknowledgments

All praise and gratitude are due to Almighty Allah, the Most Gracious and the Most Merciful, for bestowing upon us the knowledge, strength, and perseverance to complete this project successfully.

We wish to express our sincere appreciation to our project supervisor Dr. Kashif Sultan, for their continuous guidance, constructive feedback, and encouragement throughout the development of Vid-AI. Their professional expertise and readiness to pass on their knowledge were key factors that greatly influenced this project's creation.

The authors are extremely thankful to their parents and families for their unwavering support in the form of constant encouragement, prayers, and financial assistance to complete this task. They have never doubted our capabilities and helped us stay motivated during tough times. On a personal note, I, Faiza Alam Warraich, would like to extend my heartfelt gratitude to my Api, Zunaira Shoaib, and my best friend, Yashal Chaudhry, who served as my personal support system, offering the emotional backing and strength that kept me focused during the most demanding phases of this work.

We would also like to thank our faculty and university staff, whose help we received in the form of an academic background and everything else needed for us to conduct research for this project. We are especially grateful for the acquired knowledge that influenced our decision-making.

Last but not least, our friends and fellow students should be thanked for offering constructive criticism at all stages of testing and moral support.

Finally, we acknowledge the open-source community and the developers behind the technologies that made Vid-AI possible — React, Node.js, MongoDB, Ollama, FastAPI, Socket.IO, Expo, and Docker, among others. Their contributions to the software ecosystem enabled us to build a platform that we are genuinely proud of.

Project Title (Vid-AI)

Sustainable Development Goals

SDG No	Description of SDG	SDG No	Description of SDG
SDG 1	No Poverty	SDG 9 ✓	Industry, Innovation, and Infrastructure
SDG 2	Zero Hunger	SDG 10 ✓	Reduced Inequalities
SDG 3	Good Health and Well Being	SDG 11	Sustainable Cities and Communities
SDG 4	Quality Education	SDG 12 ✓	Responsible Consumption and Production
SDG 5 ✓	Gender Equality	SDG 13	Climate Change
SDG 6	Clean Water and Sanitation	SDG 14	Life Below Water
SDG 7	Affordable and Clean Energy	SDG 15	Life on Land
SDG 8 ✓	Decent Work and Economic Growth	SDG 16	Peace, Justice and Strong Institutions
		SDG 17	Partnerships for the Goals

Table Of Content

List of Figures

List of Tables

Chapter 1: Introduction

The wedding industry in Pakistan is a billion-dollar industry with yearly growth. Wedding planning entails the coordination of various services like venue, catering, photography, decoration, makeup among others. The wedding planning process has mostly involved word of mouth referrals from people that have arranged their weddings before. Additionally, many calls and meetings with various vendors make the whole process time-consuming and stressful.

However, large language models and conversational artificial intelligence are being used nowadays in offering smart and intelligent applications for areas that have had little to no contribution by artificial intelligence. Intelligent software will be a useful solution for wedding planning as well.

Vid-AI is an AI-powered wedding planning service that was designed for use in Pakistan. Vid-AI combines a vendor marketplace, bookings, payments, chatbot for wedding planners, budget planning, and a mobile app to create an end-to-end solution for wedding planning powered by artificial intelligence.

1.1 Motivation

Wedding planning in Pakistan is a highly culturally enriched occasion; however, vendor hunting and hiring remains largely unprofessional and fragmented. The following challenges highlight the core issues faced by couples and their families:

- Unorganized Vendor Hunting: No platform provides an organized interface where users can search, compare, and book vendors by categories, cities, budgets, and ratings. Currently, people depend on social media and general classified websites for hunting vendors.
- Concealed Pricing Models: No platform allows vendors to list standardized rates or pricing plans. Users need to reach out to various vendors individually just to compare prices.
- Naive Approach to Budgeting: Many couples use spreadsheets and estimates while budgeting. There is no intelligent software that can help them with automatic budget allocation and tracking.
- Insufficient Localization of AI Platforms: Global players such as The Knot and Zola cannot be considered localized because of certain reasons. First, they cater to the needs of American weddings, whereas Pakistanis need separate venues (Mehndi, Baraat, Walima, and Nikkah) and currency (PKR). Also, these AI assistants are unaware of the culture here.
- Insufficient Digitalization for Vendors: As vendors are usually small-scale, they are dependent on WhatsApp and phone calls. These vendors lack their own dedicated website for managing portfolios, profiles, booking appointments, and processing payments.

- **Detached Communication:** Once booking has been made, there is usually no integrated system where users can make their communication through one platform. This results in no automated notifications in regard to approvals, payments, or cancellations.

The above issues paved the way for developing Vid-AI, a platform that aims to combine all these elements into one, which would be dedicated solely for the Pakistani weddings sector.

1.2 Problem Statement

There are still some problems existing despite the presence of various invitation card makers. These systems have the disadvantage of manual entries and fixed templates, which greatly restrict user personalization. Moreover, these systems do not possess any artificial intelligence system that can help in creating content, hence becoming difficult.

Among other things, most of the systems currently being used do not incorporate:

1. Advanced AI features for increased user convenience and engagement.
2. High-performance capabilities within both web and mobile applications using new technology architecture.
3. Fundamental features such as payment processing, media cloud storage, and modularity.

Thus, it is evident that there is a requirement for a platform that uses both AI technology and advanced software engineering principles to enhance wedding planning and invitation generation. Vid-AI solves these problems through its use of AI technology within the process of software development.

1.3 Objectives

The key goals of this project include the following:

1. Multi-role Development: The aim is to develop a multi-role full-stack application that caters to three distinct roles, namely: Customers (couple/family), Vendors (service providers) and Administrators.
2. AI-based Chatbot: AI wedding planner will be developed using LLaMA 3.2 through Ollama and Google Gemini to make vendor recommendations and budget management advice based on Pakistan culture.
3. Vendor Marketplace: Marketplace for vendors will be made for all the different kinds of services required during a wedding including venue, catering, photography, decor and others and their packages.
4. Smart Budget Management System: The ability to plan out the budget for everything including making budget allocations and monitoring the spending against budget.

5. Full Booking & Payment Lifecycle: A complete lifecycle from booking requests to vendor acceptance/decline, payments integration with Stripe and booking status management.
6. Cross-platform Mobile Application Support: Development of mobile application using React Native (Expo) platform to make sure that customers can easily browse for vendors and book services through mobile application too.
7. Vendor Management Portal: Develop a specialized dashboard for managing services/packages, displaying portfolios with social media components like likes and comments, and visual sales analytics.
8. Administrative Control Panel: Construct an administrative portal for overall management of the platform, which includes managing users/vendors, logging activity, processing reports, and monitoring the system's health.

1.4 Major Contributions

Major contributions of this study include:

- Local Wedding AI Assistant: In this project, we present the first conversation AI assistant prompted for Pakistani use case leveraging LLaMA 3.2 (local) and Google Gemini (cloud) to understand local event (Mehndi, Nikkah, etc.) along with budgeting in PKR.
- End-to-End Multi-Roles Ecosystem: As opposed to conventional directories, our approach incorporates an ecosystem connecting four elements: client portal, vendor portal, admin console, and mobile application all under one roof using single backend infrastructure.
- IQ Based Budget Recommendations: Our system does not only focus on tracking but also includes AI-enabled recommendations for budget allocation. This allocation is smartly linked to marketplace in terms of percentage so that bookings reduce budget instantly.
- IQ Enabled Dynamic Form: Conventional approach involves pre-prepared forms; however, our approach utilizes AI to develop dynamic forms based on vendor categories e.g. guests at venue and delivery time for catering.
- Lifecycle Management Booking System: Unique booking lifecycle management approach which updates status automatically to completed and expired depending upon event date and payments made.
- Sales Analysis Page for Vendors: Specialized sales analysis page offering key performance indicators cards and revenues in form of pie charts for vendors.
- Cross-Platform Availability: Complete integration for desktop and mobile platforms via a unified React web front-end and React Native mobile app.

- Microservices-Based System Design: A system constructed from independent services such as a Node.js/Express back-end, Python/FastAPI-based AI microservices, Ollama inference engine, and ReactJS front-end.

1.5 Organization of the Report

The organization of this report is summarized as follows:

- **Chapter 1:** Introduces the Vid-AI project, covering motivations, problem definition, objectives, and major contributions.
- **Chapter 2:** Provides a thorough literature review and background investigation on AI systems and invitation platforms.
- **Chapter 3:** Discusses system requirements, including functional and non-functional requirements, use-case diagrams, and a feasibility study.
- **Chapter 4:** Addresses the system design process, including architecture, data models, and user interface designs.
- **Chapter 5:** Details the implementation process, development environment, and the specific technologies utilized.
- **Chapter 6:** Covers the testing and evaluation process, including test plans, results, and performance metrics.
- **Chapter 7:** Summarizes the project, reflects on the contributions, and suggests future improvements.

Chapter 2: Background Study / Literature Review

In this chapter, a thorough background analysis about the basic concepts, current platforms, and latest advancements is conducted, which formed the basis for developing Vid-AI. Section 2.1 highlights the basic concepts behind the proposed platform. Section 2.2 describes the existing wedding planner apps, with a focus on the Pakistani market, and assesses their capabilities and weaknesses. Section 2.3 reviews existing literature from academic sources, including international conferences and journals, regarding AI-driven recommender systems, chatbots, budget management, and web-based instant messaging. Section 2.4 identifies gaps in existing research and motivates Vid-AI.

2.1 Key Concepts

2.1.1 Large Language Models (LLMs) for Domain-Specific Application

Large Language Models are deep learning neural networks trained on vast amounts of text data and capable of generating, summarizing, and reasoning about natural language. The accessibility of LLM capabilities was democratized by the release of the GPT series from OpenAI, the Gemini series from Google, and the open-weight LLM series from Meta **[1]**. There are two common deployment modes:

- **Centralized Inference:** Accessible through APIs, where performance capabilities are high. however, costs, latency, and data privacy must be carefully considered when utilizing models like Google Gemini.
- **Local Inference:** Utilizing tools like **Ollama**, developers can deploy quantized open-weight models (such as Llama 3.2 3B) on standard hardware. This approach ensures data privacy and zero per-request costs, though it involves a trade-off in overall model size **[2]**.

An increasing number of studies demonstrate how LLMs can be "verticalized" for specific use cases by designing appropriate system prompts and employing Retrieval-Augmented Generation (RAG). As part of the process, domain-specific information is offered in the prompt context during inference without fine-tuning [3]. It is especially pertinent to the present case, as Vid-AI's AI-based components depend upon knowledge about the Pakistani wedding domain, which includes types of vendors, local traditions, and budget in PKR, provided through customized system prompts and real-time context from the database.

Relevance to Vid-AI: The proposed solution involves using two LLMs, with Ollama for deterministic and structured JSON results (like vendor recommendations and budget suggestions) and Google Gemini for smooth conversation.

2.1.2 Recommendation Systems

Three primary approaches dominate the literature:

Approach	Mechanism	Strength	Weakness
Collaborative Filtering	Recommends based on similar users' choices	Discovers non-obvious preferences	Cold-start problem for new users/vendors
Content-Based Filtering	Matches item attributes to user profile	No cold-start; explainable results	Limited serendipity; requires rich metadata
Knowledge-Based / Constraint Satisfaction	Applies domain rules (budget ≤ X, city = Y)	Handles explicit constraints well	Cannot learn latent preferences

Table 1: Comparison of recommendation system approaches in service marketplaces.

According to the recent study carried out by Zhang et al. [6], it would be prudent to adopt a recommendation algorithm based on constraint satisfaction and neural ranking for recommending event services. The algorithm works in such a way that the hard constraints (geographical location, time and venue size) are used to reduce the number of alternatives, which are then sorted using the scoring algorithm.

Relevance to Vid-AI: In recommendation by the proposed module, constraints take precedence over ranking. Hard constraints are imposed on the MongoDB database via queries, and results are ranked based on their closeness to the budget. It is a knowledge-based method that solves the problem of "cold-start" associated with collaborative filtering.

2.1.3 Real-Time Web Communications

WebSocket is a protocol (RFC 6455) which provides a persistent, two-way connection between clients and servers, overcoming the inefficiencies involved in the initial implementations of real-time communications, like long-polling [7]. Socket.IO is a library implementing WebSocket with fallback for long-polling over HTTP. It provides such advanced features as namespaces, rooms, acknowledgements, and binary streaming [8].

Technical needs for real-time communications within a multi-user system:

- Authentications: Clients must authenticate their connection.
- Rate Limiting: Without it, malicious clients can cause a flood to the application; broken clients will cause unnecessary load.
- Offline Deliveries: In the case when the user to whom the message was sent is currently offline, it should be saved for further delivery either after reconnection or via push notifications.

- Concurrences: Proper message sequencing and delivery are needed across multiple clients.

Relevancy of Vid-AI:

- Socket.IO connection authentication (using JWT during handshake),
- socket-based rate limiting (up to 30 messages per 10 seconds),
- conversation isolation by rooms,
- storing messages in the database,
- read receipt handling, and
- offline message delivery using push notifications from Expo.

2.1.4 Cross-Platform Mobile Application Development

Frameworks for cross-platform development provide an opportunity for building mobile applications via one codebase running on the iOS and Android platforms with a considerable decrease in development costs. Popular tools in the domain of cross-platform development are:

Framework	**Language**	**Rendering**	**Performance**
React Native (Meta)	JavaScript / TypeScript	Native UI components via bridge	Near-native
Flutter (Google)	Dart	Custom rendering engine (Skia)	Near-native
Ionic / Capacitor	JavaScript	WebView wrapper	Web-like

Table 2: Comparison of leading cross-platform mobile frameworks.

React Native and Expo (managed service that works at a higher level than the native build) have been extensively used in cases where developers were familiar with React in their previous work with web applications [9]. Common API clients, JavaScript, and business logic make it possible to apply the same approach to both mobile and web applications.

Relevance of the project: In our application, we use React (using Vite framework) for web applications and React Native and Expo SDK 54 for the mobile version. Both types of applications use common Axios client (client.js), as well as platform-specific data storage (localStorage in web applications and AsyncStorage in mobile). Token interceptors are common in both versions of the application.

2.1.5 Microservices Architecture

The microservice architecture is a design pattern in which a software solution consists of independent services with individual storage and communication over the protocol [10]. Advantages of using microservices include:

- **Separate Scaling:** CPU-heavy tasks involving machine learning and AI algorithms may scale independently of IO-intensive services that handle API requests.
- **Stack of Heterogeneous Technologies:** Any programming language and framework can be used for each service in order to achieve its intended goal.
- **Fault Tolerance:** The failure of one service does not necessarily mean the entire system will go offline.

Nevertheless, microservices present challenges regarding operation. Service discovery, inter-process communication, distributed logging, and orchestration of deployments must be handled carefully **[11]**. **Docker Compose** serves as an effective compromise for smaller applications, allowing the specification of multi-container deployments within a single YAML file without the immediate need for Kubernetes.

Connection to Vid-AI: The software creates a distinction between the **Node.js API server** and the **Python FastAPI AI microservice**, such that each can run using the best tech stacks (REST/WebSocket via Express and asynchronous ML inference via FastAPI). The Docker Compose process deploys six services: front-end, back-end, ai-service, ollama, ollama-setup, and ngrok.

2.2 Existing Wedding Planning Platforms

In this section, we analyze the most well-known wedding planning websites and platforms—both in Pakistan and abroad—evaluating their ability to satisfy the criteria outlined in Vid-AI.

2.2.1 Shaadiyana

The most well-known Pakistani wedding planning website is **Shaadiyana** (*shaadiyana.pk*). It is a vendor directory/market place tailored to weddings in Pakistan.

Strengths:

- Vendor Directory in Full: Categorized on the basis of city and service provided (locations, photographers, decorators, catering, etc.) in major cities of Pakistan such as Lahore, Karachi, Islamabad.
- Vendor Profile: Vendor's profile includes different pricing packages, portfolio and images of their works and feedback from their clients.
- Educational Information: Separate blog section includes information on latest trends and customs of Pakistani Weddings.
- Culture Localization: Site is available in both English and Urdu Languages.

Weaknesses:

- Insufficient AI Integration: The absence of smart recommendation tools, chatbots, and budget calculation algorithms on the site.

- Bare Manual Booking System: All interactions with the service providers must occur by phone calls or WhatsApp messaging, with no booking schedule options available.
- Payment Transactions Do Not Take Place on the Platform: There is no online payment processing, which means there is no security associated with making any payments.
- Static Information Directory: The portal functions strictly as a directory, failing to manage the entire booking process.

Shaadiyana and Vid-AI: Shaadiyana proves that the market is hungry for wedding management platforms in Pakistan yet fails to provide any advanced capabilities. Conversely, Vid-AI solves all problems faced with vendor selection, booking with conflicts handling, Stripe payment gateway integration, Socket.IO for real-time messaging, React Native app development, and budgeting powered by AI technology.

2.2.2 WedMeGood

WedMeGood (wedmegood.com): This website is the biggest wedding planning platform of India that serves the South Asian community sharing similarities in cultural factors like Pakistani culture.

Strengths:

- **Massive Vendor Database:** Features over 100,000 vendors complete with profiles, pricing, and verified reviews.
- Vendor Database: Provides more than 100,000 vendors with their profile information, prices, and feedback ratings.
- • Planning Toolset: Comprises all the essential planning tools including planners' checklist, guests list and budget calculator.
- • Community Building Features: Has Real Wedding Gallery and discussion forums where users can communicate.
- • Mobile Apps: There are reliable applications developed for both iOS and Android OS systems.

Weaknesses:

- **Domination in the Region:** The platform is heavily skewed towards Indian weddings (with pricing in INR and cities in India). Some of the important cultural practices unique to Pakistani weddings, such as Walima and Baraat, and Pakistani vendors have been left out.
- **Predominance of One Community:** It focuses solely on Indian weddings (the pricing in INR and locations in India); some unique aspects of Pakistani weddings, such as Walima and Baraat as well as Pakistani vendors have been omitted.

- **No Intelligent Automation:** There are no chatbots, automated budget calculators and other intelligent elements.
- **No Instant Messaging Service:** User cannot contact the vendors using chat; only inquiry forms are provided.
- No Payment Facility: There is no payment processing function on the website.

Relevance to Vid-AI: WedMeGood confirms the existence of considerable demand for a multi-purpose planner in the South Asian culture but lacks localization features for Pakistani events.

2.2.3 The Knot / WeddingWire (The Knot Worldwide)

The Knot (theknot.com) and WeddingWire (weddingwire.com), subsidiaries of The Knot Worldwide, are the leading wedding planning experts in North America and Europe.

Advantages:

- End-to-End Planning Solution: Includes comprehensive marketplace, budget planner, guest list organizer, seating chart creator, website designer, and gifts registry.
- High-Responsibility Network: A large database of vendors rated through the Response Rate and Response Time scales.
- Exemplary User Interface: Superior mobile apps with outstanding user experience.
- New-Age AI: Applies vision analysis and artificial intelligence to suggest vendors according to style.

Shortcomings:

- Western-Focused Design: Designed exclusively for single-celebration weddings. It fails to provide adequate structure to cater to the multiple-celebration South Asian tradition (Mehndi, Baraat, Walima, Nikkah), where each celebration demands separate vendor management and budget planning.
- Timed Communication: Vendor engagement through form filling only.
- Limited AI Integration: The integration of artificial intelligence is limited to style-based suggestions; no chatbot planning guidance or culturally specific budgeting.
- Technical Deficiency: Completely unattainable in Pakistan due to the absence of vendors, cities, and PKR currency.

Vid-AI connection: While “The Knot” worldwide sets the example of the global website that provides information about the wedding, Vid-AI solves the problem of architecture that The Knot has by dealing with “one event”. Multi event budget and

booking forms for classification purposes have been developed for the Pakistani wedding ceremony specifically.

2.2.4 Zola

Zola (zola.com) is a modern US-based wedding portal that is defined by the extensive usage of design elements, technology, and an engaging user experience.

Strengths:

- Design-Oriented Interface: An extremely clean and intuitive interface created with the couple in mind.
- Service Consolidation: An effective merger of registries, wedding websites, and planning tools.
- Guest Management: State-of-the-art digital RSVP management tools.
- Creative Invitations: Quality online and offline invitation designs.

Weaknesses:

- Human-Based Planning: Complete lack of AI involvement in the process; no automatic tasks.
- Lack of Marketplace: In contrast with The Knot, Zola lacks a vendor marketplace.
- Exclusive Event Focus: Can be used exclusively for one event only.
- Regional Focus: Geographically constrained to the US, without internationalization.

Similarity to Vid-AI: The invitation generator provided by Zola inspired Vid-AI to create the Invitation Module. Moreover, Vid-AI goes even further and uses the latest AI capabilities of Google Gemini to create culturally appropriate text and graphics for Nikkah, Walima, and Mehndi events.

2.2.5 Comparative Analysis

Feature	**Shaadiyana**	**WedMeGood**	**The Knot**	**Zola**	**Vid-AI**
Pakistani market localisation	✓	×	×	×	✓
Multi-event wedding support	Partial	Partial	×	×	✓
Vendor marketplace	✓	✓	×	×	✓
AI conversational assistant	×	×	×	×	✓

AI vendor recommendations	×	×	Basic	×	✓
AI budget planner	×	×	×	×	✓
AI invitation generation	×	×	×	×	✓
In-platform booking system	×	×	×	×	✓
Booking conflict detection	×	×	×	×	✓
Integrated payments	×	×	×	×	✓
Real-time messaging	✓	×	×	×	✓
Budget tracking tools	✓	✓	✓	×	✓
Vendor analytics dashboard	✓	×	×	×	✓
Admin moderation panel	×	×	×	×	✓
Mobile application	✓	✓	✓	✓	✓
Open-source / Local AI	×	×	×	×	✓

Table 3: Feature comparison of existing wedding planning platforms versus Vid-AI

As is evident from the comparison above, there has not been a single platform so far to provide the five critical features required (market localization, AI-based planning support, full booking cycle, transaction processing, and instant messaging), which makes Vid-AI the first and only platform to create an integrated ecosystem.

2.3 Related Work from Literature

2.3.1 Recommendation Systems Based on AI in Service Marketplaces

Adomavicius and Tuzhilin [4] present a comprehensive literature review on the subject of recommendation systems, categorizing them into three basic types: collaborative filtering, content-based filtering, and a combination of the two. They observe that domain-specific limitations such as geographic location, availability, and budget limitations are largely ignored by standard recommendation systems and therefore call for the application of knowledge-based methods in services-oriented industries.

Ricci et al. [12] further extend this study to the domains of tourism and hospitality services, which can be compared to wedding planning for the purposes of our discussion (multiple service providers, geographic considerations, budget limitations,

and high risk of purchase decisions). The results of their research indicate the superiority of the constraint-based recommendation system in conjunction with hard-filtering before scoring over purely collaborative methods.

More recently, Wu et al. [13] showed that LLMs can be used for recommendation purposes by embedding user and item preference details in natural language instructions for the model to generate ordered lists. This technique may produce results similar to traditional matrix factorization models but with natural language explanations for its recommendations.

Critical Analysis: Although LLM-based recommendations are highly effective, they are also highly prone to errors referred to as hallucinations, where non-existent products and false ratings are recommended to the users. However, by using a live database like MongoDB, Vid-AI makes sure that all its recommendations are accurate. The data is obtained through a query for vendor information based on user requirements and then embedded into AI instruction.

2.3.2 Domain-Specific Conversational AI and Chatbots

Moving from rule-based chatbots, such as AIML, to transformer-based assistants marks a considerable improvement in automated assistance technology [14]. As shown in Roller et al.'s paper [15], LLMs are capable of maintaining consistency in their multi-turn conversation if persona traits and domain knowledge were embedded in system prompts.

For instance, a rule-based chatbot in the domain of weddings, which recommends wedding venues by using Natural Language Understanding (NLU), was suggested by Srivastava and Kumar [16]. Though such a chatbot is capable of handling structured questions (such as "Show me the venues for my wedding ceremony in Delhi for less than ₹5 lakhs"), it is not able to respond to open planning questions ("What should I do for my Mehndi ceremony?"). Such limitations are inherent to rule-based approaches, in which all dialogue paths need to be predefined.

Critical Analysis: Rule-based solutions [16] do not provide flexibility sufficient to address different types of queries associated with wedding plans. On the other hand, even though standard conversational assistants built using LLMs offer flexibility, there is a possibility that they may provide answers which are irrelevant to local customs or simply unacceptable from a cultural perspective. Vid-AI provides a solution to these problems by designing a domain-specific prompt, which includes comprehensive knowledge about Pakistani wedding ceremonies and their sequences and vendors' categories.

2.3.3 Budget Optimisation and Financial Planning

Optimising budget distribution for weddings entails solving a constrained resource allocation problem in which an overall budget needs to be divided among various sub-budgets (location, food, photographs, etc.) to ensure satisfaction of the couple’s requirements [17]. Traditional methods include using Integer Linear Programming

(ILP) to allocate budgets according to priorities and minimum/maximum limitations of each budget category.

Kaur and Singh [18] have employed Multi-Objective Optimisation (MOO) techniques for event budgeting and considered minimising cost and maximising quality to form two objectives for solving the problem. The solutions obtained through this model are Pareto-optimal, but require a utility function, which may not necessarily be available in the real-world case.

Chen et al. [19] illustrate that with the right prompts—including constraints of total budget, number of budget categories, and priority considerations—Large Language Models (LLMs) can also be used for budgeting. Here, the LLM produces percentages from which actual dollar values are derived.

Analysis: The use of formal optimisation methods [17, 18] leads to mathematical optimality, but this approach is not feasible for end-users who cannot articulate their utility functions. LLM budgeting [19], while more user-friendly, may lead to allocations which do not sum up to the declared amount because of rounding or hallucination issues. This problem is addressed by Vid-AI by explicitly stating in the system prompt that Ollama needs to provide absolute values of PKR amounts which must sum to the total budget, followed by parsing the result in structured JSON format.

2.3.4 Real-Time Communication in Multi-User Platforms

In their research on real-time communications systems, **Pimentel and Nickerson [20]** distinguish four essential factors required for robust production-level systems:

- **Authentication**
- **Message ordering**
- **Offline messaging capabilities**
- **Scalability**

In many cases, the latter two are overlooked in favour of simple message delivery. Regarding the analysis of communication within marketplace platforms, Gutierrez et al. [21] investigate communication patterns and conclude that in-app communication results in transaction success rates being 23% higher than in the case of external communication channels (email and phone). It is especially interesting how the study relates to the context of weddings, since vendor communication is one of the critical steps between discovering and booking vendors on the platform.

Karagiannis et al. [22] present a technique of implementing per-connection rate limiting using token-bucket algorithms to prevent WebSocket-based DDoS attacks. Their experiments show that implementing per-connection rate limits allows decreasing flood attacks by 95% without affecting legitimate users.

Critical analysis: Most of the marketplace platforms available today (such as Shaadiyana, WedMeGood, and The Knot) provide external rather than in-platform real-time messaging, and thus cannot track conversations or provide verification

badges/ratings associated with them. Vid-AI implements a full-stack real-time messaging service fulfilling all the criteria described in [20], which include JWT authentication on Socket.IO handshakes, database-persisted messages with timestamps, push notifications for offline recipients, and per-socket rate limiting (30 messages per 10 seconds) in accordance with [22].

2.3.5 Dual LLM Architectures and AI Service Design

The idea of deploying a collection of specialized LLMs in one application that is tuned for specific task categories is becoming increasingly popular after the emergence of open-weight models. Jiang et al. [23] suggest LLM routing, where a light-weight classifier routes each request to the right model based on the nature of the task. The result of their experiment suggests that a model router that leverages a small model on the client-side for structured tasks and a large model in the cloud for creative tasks results in significant cost savings compared to using one large model for all tasks.

Patil et al. [24] introduce Gorilla, an approach to making API calls powered by LLMs, which shows that small models can produce structured JSON outputs from a well-defined schema specification more effectively than large models, which often "over-generate" fields.

Critique: There is a challenge posed when using only one LLMs between structurally soundness of responses and their ability to converse well. The two-LLM system that Vid-AI uses [23] consists of the Ollama model which processes all structurally sound tasks (for example, recommending vendors and making budgets in JSON format) while Gemini focuses on conversationally based tasks where fluency of speech is critical. Therefore, budgets can be processed mathematically soundly while at the same time wedding conversations sound naturally.

2.4 Research Gap and Motivation

The literature review and platform survey reveal the following gaps:

#	Identified Gap	Evidence
G-1	**No AI-powered wedding platform exists for the Pakistani market.**	Shaadiyana (2.2.1) is the only notable Pakistani wedding platform but offers zero AI capabilities — no chatbot, no recommendations, no budget planning.
G-2	**Existing platforms treat weddings as single-event ceremonies.**	The Knot, Zola, and WeddingWire (2.2.3–2.2.4) assume a one-ceremony model. Pakistani weddings require multi-event planning with separate budgets, vendors, and timelines for Nikkah, Mehndi, Baraat, and Walima.

G-3	**No platform offers in-platform booking with conflict detection and payment integration.**	All surveyed platforms (2.2.5, Table 2.3) either lack a booking system entirely or rely on external communication and payment channels.
G-4	**LLM-based recommendations risk hallucination when not grounded in real data.**	Wu et al. [13] achieve strong results but acknowledge hallucination risks. No prior work grounds LLM recommendations in a live vendor database for wedding services.
G-5	**Budget planning tools are either manual checklists or require formal optimisation inputs.**	WedMeGood and The Knot offer manual trackers (2.2.2–2.2.3). Academic work [17, 18] requires utility functions. No tool uses LLMs for accessible, constraint-aware budget generation.
G-6	**Real-time messaging is absent from wedding platforms despite proven impact on transaction completion.**	Gutierrez et al. [21] demonstrate 23% improvement in transaction completion with in-platform messaging, yet no surveyed wedding platform provides it.

Table 4: Summary of identified research and market gaps

Vid-AI aims at covering all the six gaps through its unique combination of:

- **Pakistani cultural localisation (G-1, G-2)** – through ceremony-specific event types, PKR budgets, and domain-specific AI prompts of Pakistani weddings.
- **MongoDB-based AI recommendations (G-4)** – through Ollama generating structured AI suggestions based on actual MongoDB vendor data.
- **LLM-based budget planning (G-5)** – through constraint-enforced budgeting through structured JSON generation.
- **In-platform booking cycle management (G-3)** – through conflict resolution, status updates, and Stripe webhook-verified payments.
- **Authenticated in-real-time messaging (G-6)** – through Socket.IO, read receipts, rate limiting, and push notifications as a backup.

2.5 Summary

This chapter established the theoretical and practical foundation for Vid-AI. Key concepts in large language models, recommendation systems, real-time communication, cross-platform development, and microservices architecture were reviewed (2.1). A critical survey of existing platforms — Shaadiyana, WedMeGood, The Knot, and Zola — revealed that no current solution combines AI intelligence, cultural localisation for Pakistan, in-platform booking, integrated payments, and real-time messaging (2.2). The findings of the review of relevant academic literature

indicated that each of the technologies applied to build Vid-AI (constraint-based recommendations, LLM budgeting, two-model architecture, and WebSocket authentication) are well supported in theory, but have not yet been integrated in a wedding planning platform (2.3). The shortcomings highlighted (2.4) clearly led to the system requirements defined in Chapter 3.

2.6 References

[1] J. Wei et al., "Emergent Abilities of Large Language Models," *Transactions on Machine Learning Research (TMLR)*, 2022.

[2] Meta AI, "Llama 3.2: Lightweight Models for Edge and Mobile," *Meta AI Blog*, 2024.

[3] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 9459–9474, 2020.

[4] G. Adomavicius and A. Tuzhilin, "Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 734–749, 2005.

[5] J. Lu, D. Wu, M. Mao, W. Wang, and G. Zhang, "Recommender System Application Developments: A Survey," *Decision Support Systems*, vol. 74, pp. 12–32, 2015.

[6] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep Learning Based Recommender System: A Survey and New Perspectives," *ACM Computing Surveys*, vol. 52, no. 1, pp. 1–38, 2019.

[7] I. Fette and A. Melnikov, "The WebSocket Protocol," *RFC 6455*, Internet Engineering Task Force (IETF), 2011.

[8] Socket.IO Team, "Socket.IO Documentation — Rooms, Namespaces, and Acknowledgements," *socket.io/docs*, 2024.

[9] G. Nicol, "React Native at Scale: Lessons from Expo," *React Native EU Conference Proceedings*, 2023.

[10] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, 2nd ed. O'Reilly Media, 2021.

[11] C. Richardson, *Microservices Patterns: With Examples in Java*. Manning Publications, 2018.

[12] F. Ricci, L. Rokach, and B. Shapira, *Recommender Systems Handbook*, 2nd ed. Springer, 2015.

[13] L. Wu, Z. Zheng, Z. Qiu, H. Wang, and H. Gu, "A Survey on Large Language Models for Recommendation," *World Wide Web*, vol. 27, no. 5, 2024.

[14] T. Brown et al., "Language Models are Few-Shot Learners," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 1877–1901, 2020.

[15] S. Roller et al., "Recipes for Building an Open-Domain Chatbot," *Proceedings of the 16th Conference of the European Chapter of the ACL*, pp. 300–325, 2021.

[16] A. Srivastava and R. Kumar, "AI-Based Chatbot for Wedding Venue Recommendation," *International Conference on Intelligent Computing and Communication (ICICC)*, Springer, pp. 245–256, 2022.

[17] R. Agarwal and P. Mehta, "Optimised Budget Allocation for Multi-Event Planning Using Integer Programming," *Journal of Operations Management*, vol. 38, pp. 15–28, 2020.

[18] H. Kaur and J. Singh, "Multi-Objective Optimisation for Event Budget Planning," *Proceedings of the International Conference on Computational Intelligence (ICCI)*, IEEE, pp. 112–119, 2021.

[19] Z. Chen, W. Zhang, and Y. Liu, "Can Large Language Models Budget? Evaluating LLMs on Constrained Resource Allocation Tasks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, pp. 4521–4529, 2024.

[20] V. Pimentel and J. V. Nickerson, "Communicating and Displaying Real-Time Data with WebSocket," *IEEE Internet Computing*, vol. 16, no. 4, pp. 45–53, 2012.

[21] F. Gutierrez, N. Diakopoulos, and C. Lampe, "The Effect of In-Platform Messaging on Online Marketplace Transactions," *Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW)*, pp. 1420–1435, 2022.

[22] G. Karagiannis, A. Jamakovic, and M. Edmonds, "Rate Limiting Frameworks for WebSocket-Based Applications," *IEEE Transactions on Network and Service Management*, vol. 19, no. 3, pp. 2145–2158, 2022.

[23] A. Jiang, X. Chen, and M. Gao, "LLM Routing: Optimising Multi-Model Inference for Cost-Efficient AI Systems," *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 6782–6795, 2024.

[24] S. Patil, T. Zhang, X. Wang, and J. Gonzalez, "Gorilla: Large Language Model Connected with Massive APIs," *arXiv preprint arXiv:2305.15334*, 2023.

Chapter 3: System Requirements

This chapter defines the system requirements of Vid-AI based on the project scope, implemented architecture, and actor workflows. The platform is a multi-role wedding planning system that supports customers, vendors, and administrators through a web application, a mobile application, a backend API, and an AI microservice. The requirements presented in this chapter cover functional behaviour, interface needs, persistent data requirements, non-functional constraints, feasibility considerations, and analysis models that explain major workflows.

3.1 Use Case Diagram

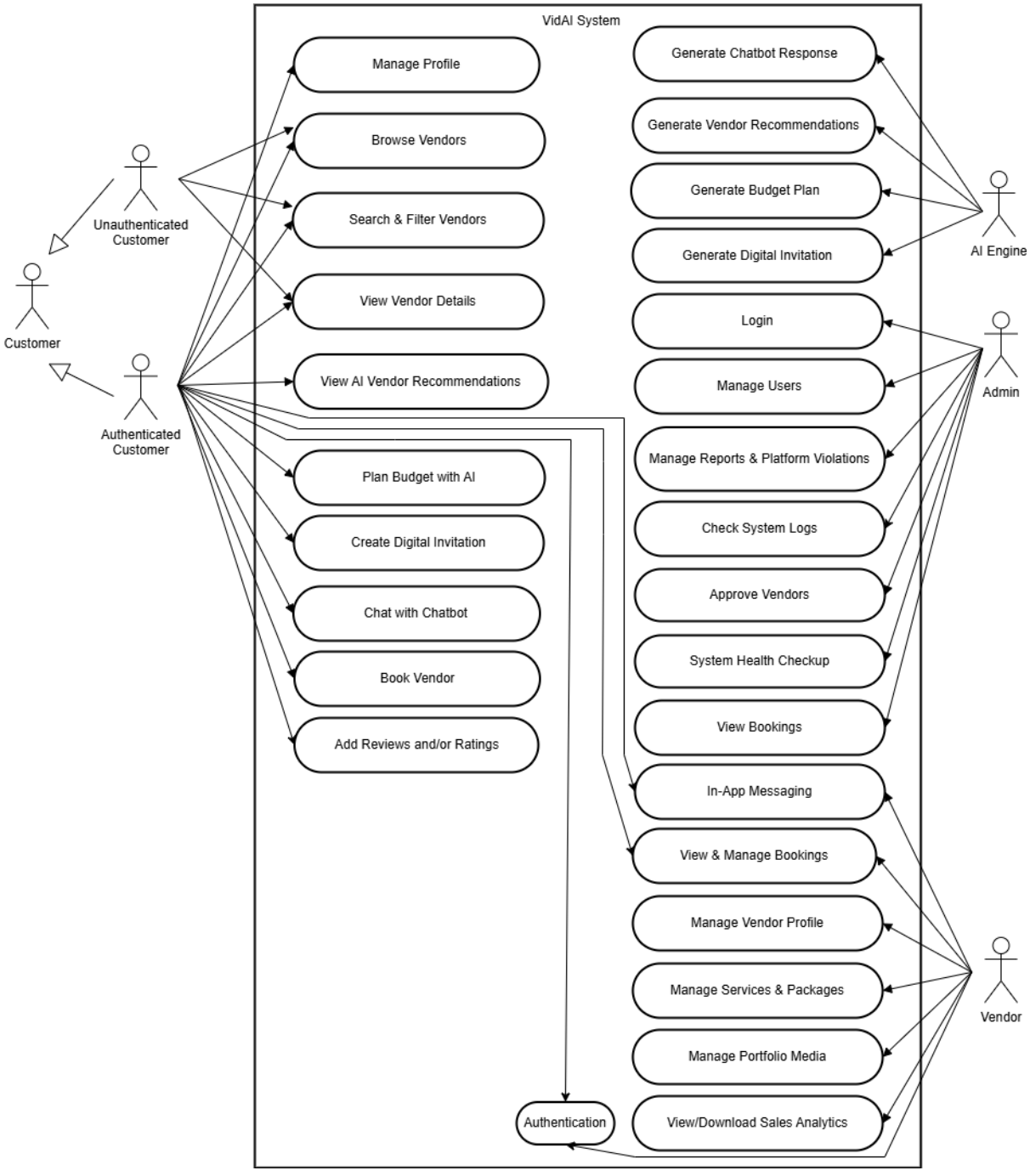

Figure 1: System-Level Use Case Diagram

The use case diagram shows that the customer-facing side of Vid-AI is centered on discovery, planning, booking, payment, and communication. The vendor-facing side focuses on business profile management, booking response, payment tracking, and analytics. The administrator ensures trust and governance by managing users, verifying vendors, reviewing reports, and monitoring the overall health of the platform.

3.2 Functional Requirements

The functional requirements of Vid-AI are expressed as use case descriptions corresponding to the system-level use cases presented in Section 3.1.

Use Case ID	Use Case Name	Primary Actor	Supporting Actor	Purpose
UC-01	Authenticate User	Guest, Customer, Vendor, Admin	None	Register, log in, and establish authorized access
UC-02	Manage Customer Profile	Authenticated Customer	None	Maintain personal and planning profile information
UC-03	Browse Vendors	Guest, Customer	None	View available vendors in the marketplace
UC-04	Search and Filter Vendors	Guest, Customer	None	Narrow vendor results by category, city, and budget
UC-05	View Vendor Details	Guest, Customer	None	Inspect vendor profile, packages, portfolio, and reviews
UC-06	Get AI Vendor Recommendations	Authenticated Customer	AI Engine	Receive personalized vendor suggestions
UC-07	Generate AI Budget Plan	Authenticated Customer	AI Engine	Produce category-wise budget allocations and tips

UC-08	Generate Digital Invitation	Authenticated Customer	AI Engine	Create invitation content and preview assets
UC-09	Chat with AI Planner	Authenticated Customer	AI Engine	Obtain conversational wedding planning guidance
UC-10	Create Booking Request	Authenticated Customer	Vendor	Submit a booking request to a vendor
UC-11	Manage My Bookings	Authenticated Customer	Vendor, Stripe Gateway	View, pay for, track, or cancel own bookings
UC-12	Add Reviews and Ratings	Authenticated Customer	Vendor	Submit feedback on a vendor
UC-13	Manage Vendor Profile	Vendor	None	Create and update vendor business profile
UC-14	Manage Services and Packages	Vendor	None	Add, edit, and delete offered services/packages
UC-15	Manage Portfolio Media	Vendor	Media Storage Service	Upload and maintain portfolio images and videos
UC-16	Manage Incoming Bookings	Vendor	Customer	View and respond to booking requests, update payment status
UC-17	In-App Messaging	Customer, Vendor	Real-time Messaging Service	Exchange messages and notifications
UC-18	View Sales Analytics	Vendor	None	View revenue, booking, and sales analytics

UC-19	Manage Users	Admin	None	View and control platform user accounts
UC-20	Verify / Reject Vendors	Admin	Vendor	Approve or reject vendor verification requests
UC-21	Manage Reports and Platform Violations	Admin	Customer, Vendor	Review abuse reports and apply moderation actions
UC-22	Monitor System Logs and Health	Admin	None	Review activity logs and system health status
UC-23	View All Bookings	Admin	None	Supervise booking activity across the platform

Table 5: Functional Use Case Catalogue

3.2 Functional Requirements

The functional requirements of Vid-AI are presented below as use case descriptions derived from the system use-case diagram.

3.2.1 UC-01: Authenticate User

Feature	**Details**
Use-Case ID	UC-01
Use-Case Name	Authenticate User
Primary Actors	Guest, Customer, Vendor, Admin
Supporting Actors	None
Description	Facilitates the registration of new accounts and the login of existing users to provide role-based access to the platform.
Preconditions	1. Actor has platform access. 2. Registration requires a unique email. 3. Login requires an existing valid account.

Trigger	The actor selects the 'Register' or 'Login' option.
Main Flow	1. Actor launches the authentication screen. 2. Actor enters the login or registration credentials. 3. System checks the validity of the entered information. 4. System creates the account (Registration) or verifies the actor’s identity (Login). 5. System provides access to the corresponding portal according to the actor's role.5. System grants access to the specific portal based on the actor's role.
Alternative Flows	A1 (Registered Email): The system rejects the application if the email address is already registered. A2 (Invalid Credentials): The system shows an error message if the login credentials are not valid. A3 (Unauthorized): The system rejects the login attempt if the account is not active.
Postconditions	A new account is successfully created or a secure authenticated session is established.

Use-Case Diagram

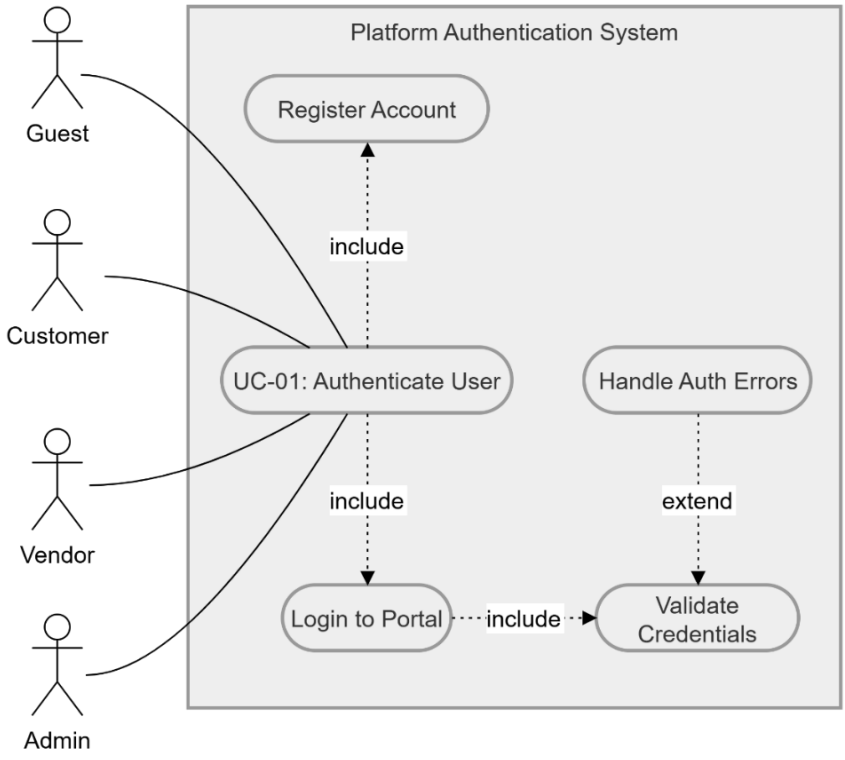

Figure 2: UCD-01

3.2.2 UC-02: Manage Customer Profile

Feature	Details
Use-Case ID	UC-02
Use-Case Name	Manage Customer Profile
Primary Actor	Authenticated Customer
Supporting Actors	None
Description	Allows the customer to view and update their personal information and event-specific planning details like budget and guest count.
Preconditions	The customer is successfully logged into the system.
Trigger	The customer navigates to the profile section or the onboarding settings.
Main Flow	1. User sees the existing information about himself/herself and plans. 2. The user changes the contents of the fields (full name, phone number, city, event description, number of guests, budget total). 3. The information is verified. 4. Changes are saved to the database.
Alternative Flows	Alternative Path 1 (Validation): If any of the mandatory fields are empty or incorrect data type, an error message appears. Alternative Path 2 (Timeout): If the timeout occurs, the user will be taken back to the login page.
Postconditions	Updated profile data is saved and becomes available for other planning modules (e.g., Budget Planner).

Use-Case Diagram

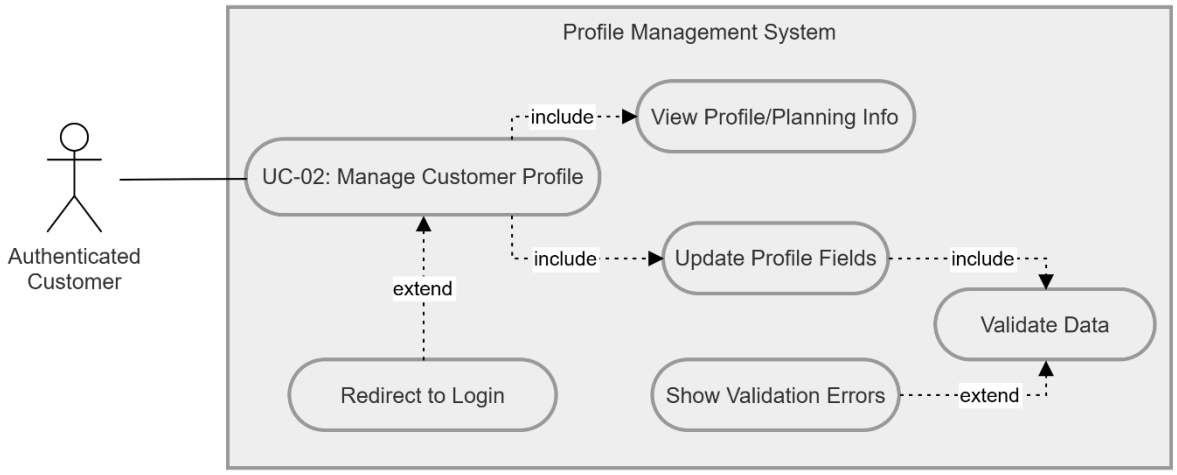

Figure 3: UCD-02

3.2.3 UC-03: Browse Vendors

Feature	**Details**
Use-Case ID	UC-03
Use-Case Name	View Vendor Marketplace
Primary Actors	Guest, Customer
Supporting Actors	None
Description	Allows actors to browse through a list of vendor profiles, showing essential details such as category, location, and ratings to help them find services.
Preconditions	Vendor records must exist in the system database.
Trigger	The actor navigates to the vendor marketplace or listing page.
Main Flow	1. Actor goes to the vendors listing page. 2. System fetches all visible and active vendors' profile data. 3. System shows cards for the vendors with categories, cities, ratings, and pricing. 4. Actor scrolls through the list of vendors.

Alternative Flows	A1 (No Vendor Found): In case there is no matching vendor available, the system will show an empty state. A2 (Failed Request): In case the request fails to fetch data, the system will show a loading state.
Postconditions	The actor successfully views the available vendors in the marketplace.

Use-Case Diagram

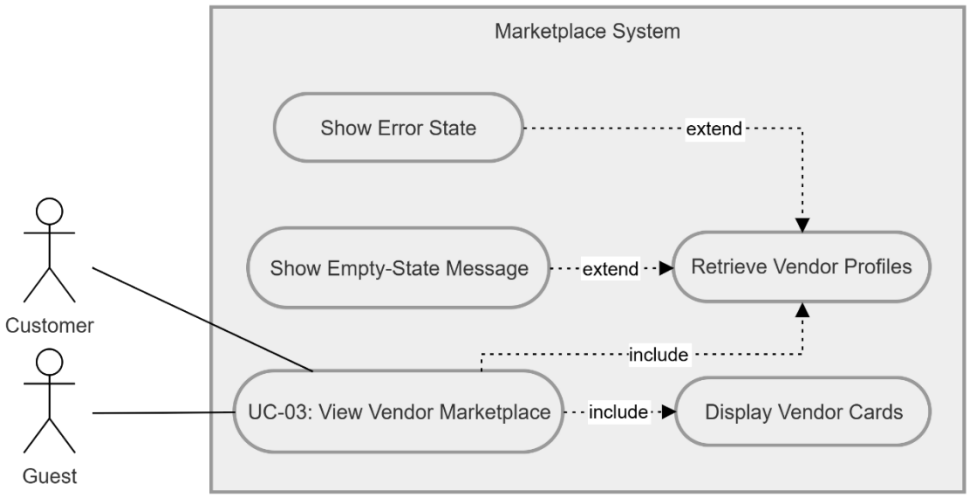

Figure 4: UCD-03

3.2.4 UC-04: Search and Filter Vendors

Feature	**Details**
Use-Case ID	UC-04
Use-Case Name	Search and Filter Vendors
Primary Actors	Guest, Customer
Supporting Actors	None
Description	Enables actors to refine the vendor list using specific search terms or filters such as category, location, and budget.
Preconditions	Vendor records exist in the system database.
Trigger	The actor enters text in the search bar or selects a filter option.

Main Flow	1. The user inputs a keyword or applies filters such as category, city, or budget. 2. The system then passes these filter parameters to the backend. 3. The backend returns all vendors that match these filter parameters. 4. The system presents these filtered results to the user.
Alternative Flows	**A1 (No Results): When there is a zero result for the filters, the system shows the "no-result" message.** **A2 (Invalid Filters): When the user input parameters are invalid or unsupported by the system, the system disregards them.**
Postconditions	The actor receives a narrowed and specific set of vendor results.

Use-Case Diagram

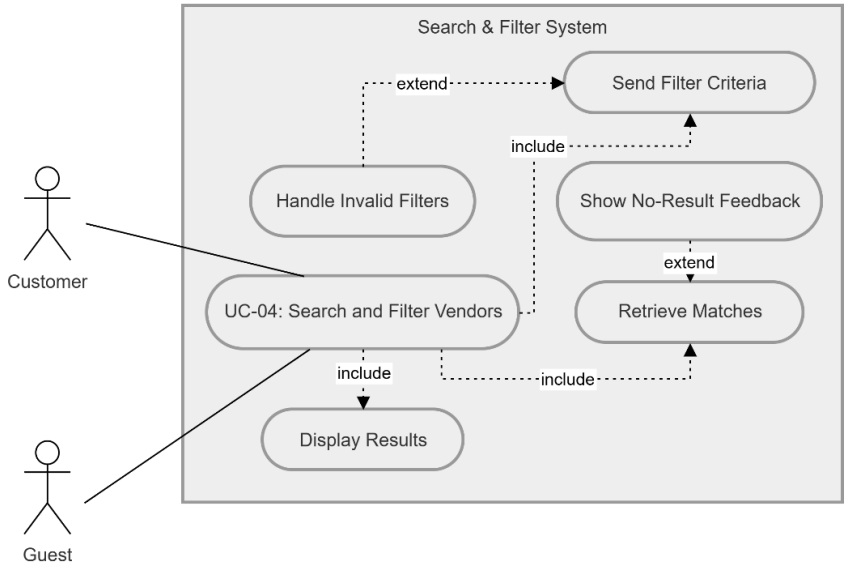

Figure 5 UCD-04

3.2.5 UC-05: View Vendor Details

Feature	Details
Use-Case ID	UC-05
Use-Case Name	View Vendor Details
Primary Actors	Guest, Customer

Supporting Actors	None
Description	Allows actors to access a comprehensive profile of a specific vendor, including their offerings, portfolio, and reputation, to aid in decision-making.
Preconditions	A valid vendor profile must exist in the system.
Trigger	The actor clicks on or selects a specific vendor from the marketplace listing.
Main Flow	1. The actor navigates to the page with vendor details. 2. The system loads the complete vendor profile data. 3. The system shows different categories such as Packages, Portfolio Media, City, Description, Reviews, and Ratings. 4. The actor evaluates the data to take action (shortlist, send a message, write a review, or book an appointment).
Alternative Flows	A1 (Not Found): In case of any invalid vendor ID or removal of the vendor profile, the system will show a message "Vendor Not Found." A2 (Incomplete Profile): In case of incomplete vendor profile data, the system will show partial profile data.
Postconditions	The actor is fully informed about the vendor's services and can proceed with engagement actions.

Use-Case Diagram

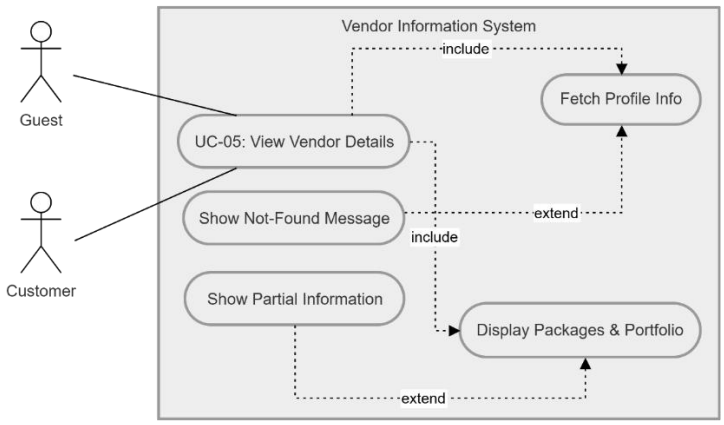

Figure 6: UCD-05

3.2.6 UC-06: Get AI Vendor Recommendations

Feature	Details
Use-Case ID	UC-06
Use-Case Name	Get AI Vendor Recommendations
Primary Actor	Authenticated Customer
Supporting Actor	AI Engine
Description	Leverages an AI engine to provide personalized vendor suggestions based on the customer's specific budget, location, and event preferences.
Preconditions	1. Customer is logged in. 2. Profile context (city, budget, preferences) is available.
Trigger	The customer initiates a request for personalized recommendations.
Main Flow	1. Customer submits specific preferences or event context. 2. Backend structures this data for the AI engine. 3. AI Engine processes the data and generates recommendation output. 4. System displays the recommended vendors or guidance. 5. Customer reviews the AI-generated suggestions.
Alternative Flows	**A1 (AI Unavailable):** If the AI engine is down, the system provides a fallback message or generic recommendations. **A2 (Insufficient Data):** If the context is too vague, the system prompts the customer for additional details.
Postconditions	The customer receives tailored guidance to assist in vendor selection.

Use-Case Diagram

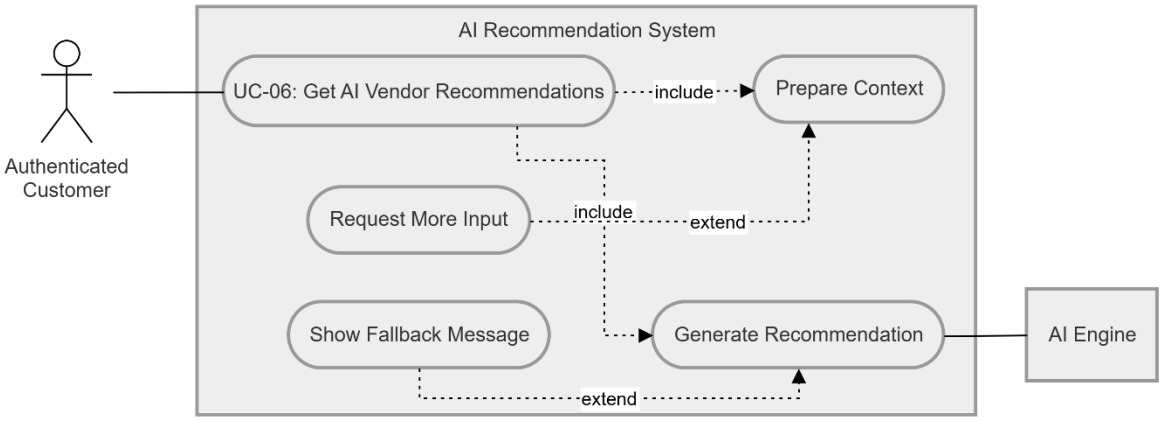

Figure 7: UCD-06

3.2.7 UC-07: Generate AI Budget Plan

Feature	**Details**
Use-Case ID	UC-07
Use-Case Name	Generate AI Budget Plan
Primary Actor	Authenticated Customer
Supporting Actor	AI Engine
Description	Utilizes an AI engine to automatically calculate budget allocations, spending recommendations, and planning tips based on the user's total budget and event context.
Preconditions	1. Customer is logged in. 2. Customer has provided a base wedding budget or planning context.
Trigger	The customer requests the system to generate an AI-driven budget plan.
Main Flow	1. Customer provides total budget and planning context. 2. System transmits the data to the AI engine. 3. AI engine generates allocation percentages, spending guidance, and tips. 4. System stores and displays the generated plan.

	5. Customer integrates the result into the interactive budget planner.
Alternative Flows	**A1 (Invalid Input):** If the budget input is non-numeric or zero, the system blocks generation and prompts for valid data. **A2 (AI Service Failure):** If the AI service is unreachable, the system displays a "Service Unavailable" error message.
Postconditions	A structured, personalized AI budget plan is generated and made available to the customer.

Use-Case Diagram

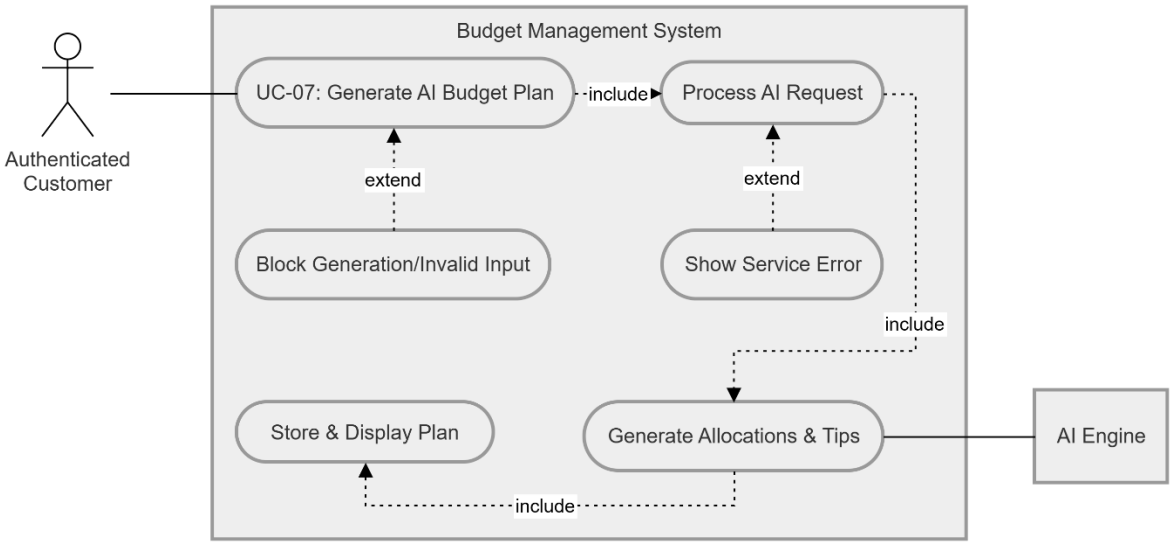

Figure 8: UCD-07

3.2.8 UC-08: Generate Digital Invitation

Feature	**Details**
Use-Case ID	UC-08
Use-Case Name	Generate AI Invitation
Primary Actor	Authenticated Customer
Supporting Actor	AI Engine
Description	Enables customers to create personalized digital wedding invitations using AI by providing specific event details, themes, and desired tones.

Preconditions	The customer is logged in and has access to the invitation module.
Trigger	The customer initiates the "Generate Invitation" action.
Main Flow	1. Customer enters invitation details (names, date, time, venue, theme, and tone). 2. System validates the form input for completeness and format. 3. Backend requests generated invitation content from the AI subsystem. 4. System returns invitation text and a preview-ready digital output. 5. Customer previews, modifies, or downloads the final invitation.
Alternative Flows	**A1 (Incomplete Data):** If mandatory fields are empty, the system blocks generation and highlights missing fields. **A2 (Generation Failure):** If the AI service fails, the system displays an error message while preserving the user's entered data for a retry.
Postconditions	A digital invitation draft or high-quality generated output is successfully created and accessible.

Use-Case Diagram

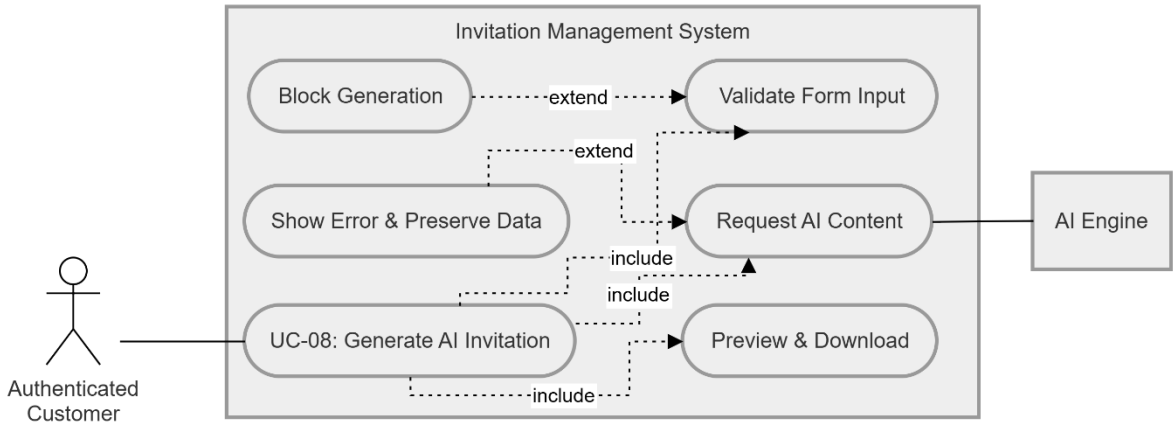

Figure 9: UCD-08

3.2.9 UC-09: Chat with AI Planner

Feature	Details
Use-Case ID	UC-09

Use-Case Name	Chat with AI Planner
Primary Actor	Authenticated Customer
Supporting Actor	AI Engine
Description	Provides customers with a conversational interface to ask wedding-planning questions and receive real-time, context-aware guidance from an AI engine.
Preconditions	The customer is successfully logged into the platform.
Trigger	The customer sends an initial message or query to the AI planner.
Main Flow	1. Customer opens the AI chat interface. 2. Customer submits a planning-related question or prompt. 3. Backend forwards the message along with relevant conversation history/context to the AI engine. 4. System receives and streams the AI-generated response back to the UI. 5. Customer reviews the response and continues the dialogue as needed.
Alternative Flows	**A1 (Service Timeout):** If the AI engine fails to respond within the limit, the system displays a fallback or "Service Unavailable" message. **A2 (Empty Message):** If the user attempts to send an empty input, the system blocks the submission and prompts for text.
Postconditions	The customer receives personalized, conversational advice to assist in their wedding planning process.

Use-Case Diagram

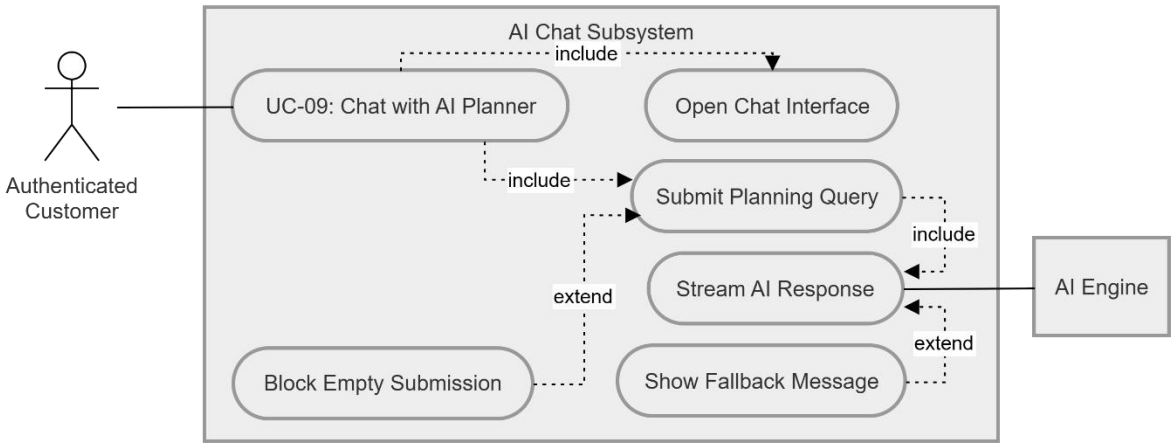

Figure 10: UCD-09

3.2.10 UC-10: Create Booking Request

Feature	**Details**
Use-Case ID	UC-10
Use-Case Name	Create Booking Request
Primary Actor	Authenticated Customer
Supporting Actor	Vendor
Description	Allows a customer to initiate a formal booking request by providing event-specific details, which then moves to a pending state for vendor review.
Preconditions	1. Customer is logged in. 2. A specific vendor has been selected.
Trigger	The customer submits the booking form.
Main Flow	1. Customer selects a vendor and an optional service package. 2. System displays a booking form tailored to the vendor's category. 3. Customer enters event date, location, guest count, and other details.

	4. System validates the request and saves the booking status as "Pending." 5. System sends a notification to the vendor regarding the new request.
Alternative Flows	**A1 (Missing Data):** If mandatory fields are incomplete, the system rejects the submission. **A2 (Invalid Slot/Vendor):** If the selected date or vendor is no longer available/valid, the system blocks the creation.
Postconditions	A booking request is successfully created and resides in the system awaiting vendor acceptance or rejection.

Use-Case Diagram

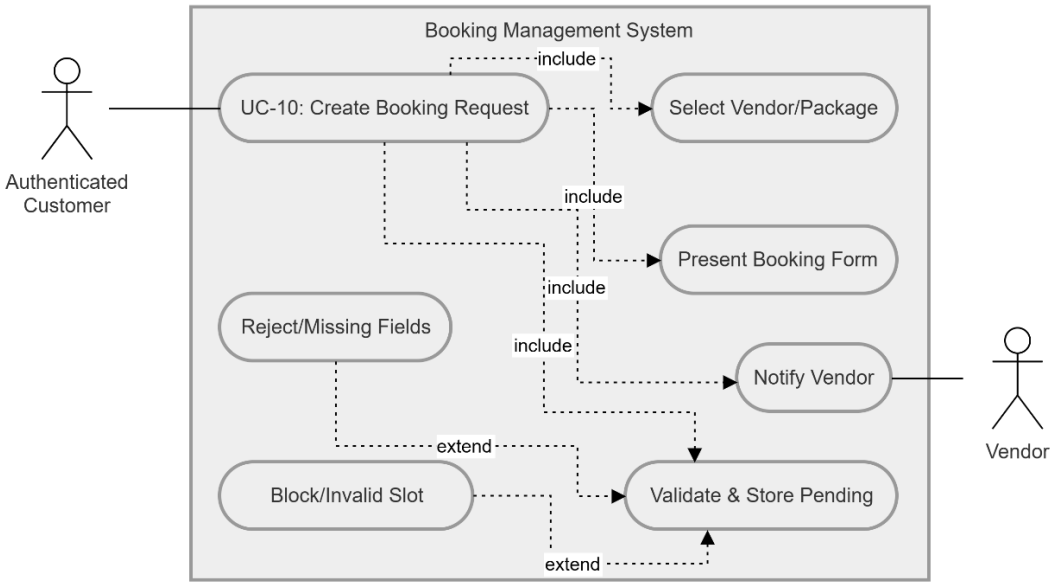

Figure 11: UCD-10

3.2.11 UC-11: Manage My Bookings

Feature	**Details**
Use-Case ID	UC-11
Use-Case Name	Manage and Track Bookings
Primary Actor	Authenticated Customer
Supporting Actors	Vendor, Stripe Gateway

Description	Allows the customer to view their booking history, inspect details, initiate payments for approved requests, and cancel eligible bookings while tracking lifecycle statuses.
Preconditions	The customer is logged into the platform and has access to the bookings module.
Trigger	The customer navigates to and opens the bookings page.
Main Flow	1. Customer views the list of all bookings and their current statuses. 2. Customer opens a specific booking to inspect details. 3. Customer initiates payment for an "Approved" booking. 4. Customer cancels an "Eligible" booking. 5. Customer tracks overall status changes (e.g., approved, completed, cancelled, expired).
Alternative Flows	**A1 (Payment Failure):** If a payment fails or is aborted by the user, the system preserves the current booking state without altering it. **A2 (Invalid Cancellation):** If the booking is already completed, cancelled, or expired, the system blocks the cancellation action. **A3 (No Bookings):** If the customer has no booking history, the system displays an empty state.
Postconditions	The customer is successfully informed about their booking lifecycles and completes their desired, permissible actions.

Use-Case Diagram

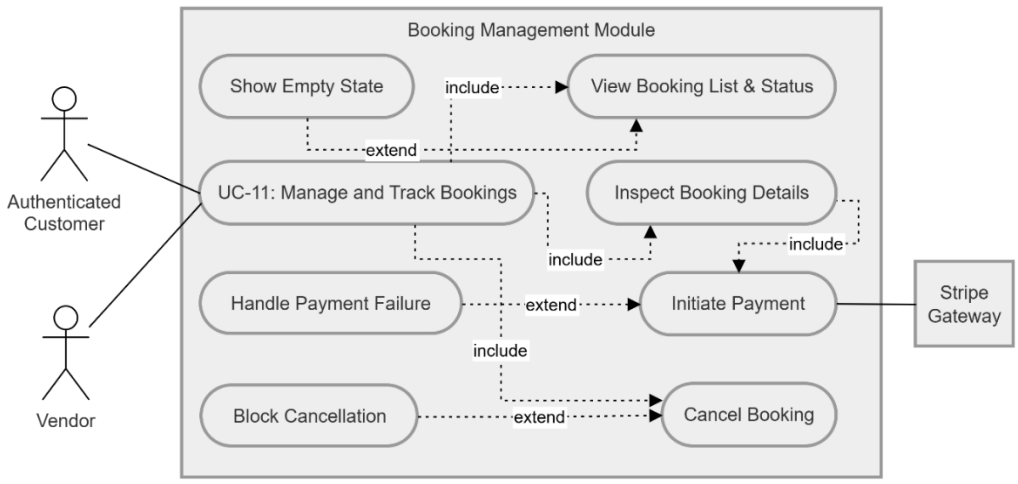

Figure 12: UCD-11

3.2.12 UC-12: Add Reviews and Ratings

Feature	Details
Use-Case ID	UC-12
Use-Case Name	Submit Vendor Review
Primary Actor	Authenticated Customer
Supporting Actor	Vendor
Description	Allows customers to provide feedback, ratings, and media for a vendor's services. The system updates the vendor's aggregate rating and displays the review publicly.
Preconditions	1. Customer is logged in. 2. Customer has accessed the specific vendor's review form.
Trigger	The customer submits the rating or review content.
Main Flow	1. Customer selects a vendor to review. 2. Customer enters a numerical rating, text comment, and optional media (photos/videos). 3. System validates the submitted review data. 4. System stores the review and recalculates the vendor’s overall rating aggregates. 5. The review is published and becomes visible on the vendor's profile.
Alternative Flows	**A1 (Invalid Data):** If required fields (like rating or comment) are missing, the system rejects the submission. **A2 (Upload Failure):** If the media upload fails due to size or connection issues, the system reports the error and allows the user to retry or skip.
Postconditions	The customer's review is successfully linked to the vendor's profile and influences their public rating.

Use-Case Diagram

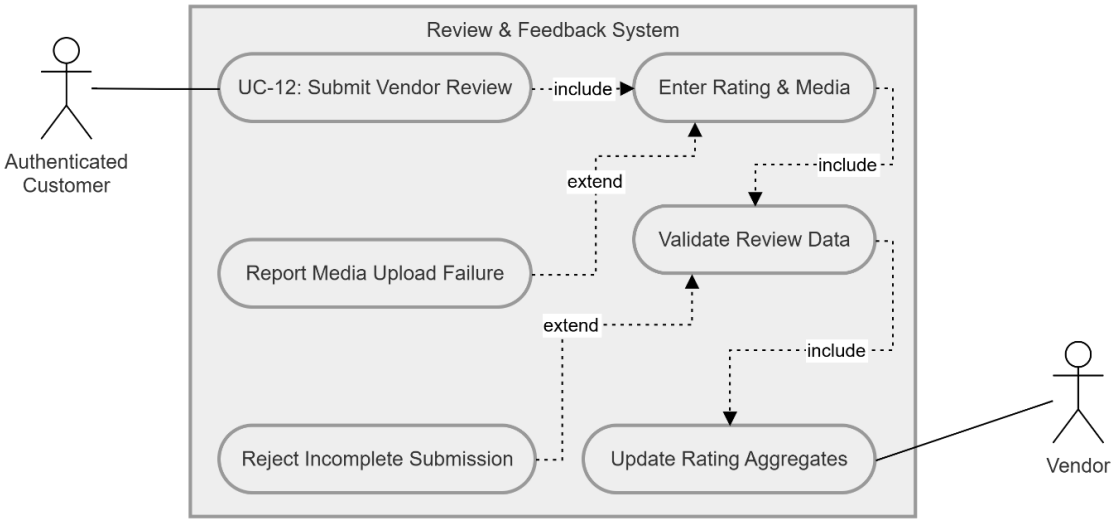

Figure 13: UCD-12

3.2.13 UC-13: Manage Vendor Profile

Feature	**Details**
Use-Case ID	UC-13
Use-Case Name	Manage Vendor Business Profile
Primary Actor	Vendor
Supporting Actors	None
Description	Allows vendors to establish and maintain their business identity on the platform by managing core information like category, contact details, and social presence.
Preconditions	The vendor is successfully logged into their account.
Trigger	The vendor navigates to the business profile management section.
Main Flow	1. Vendor creates or updates business profile information. 2. Vendor enters business name, category, city, description, contact details, and social links. 3. System validates the submitted input. 4. System stores the profile data and recalculates the profile completeness percentage.

Alternative Flows	**A1 (Missing Fields):** If mandatory business fields are empty, the system rejects the update and highlights the missing data. **A2 (New Vendor):** If the vendor has no existing profile, the system provides a guided setup wizard for initial creation.
Postconditions	The vendor's business profile is successfully created or updated in the database.

Use-Case Diagram

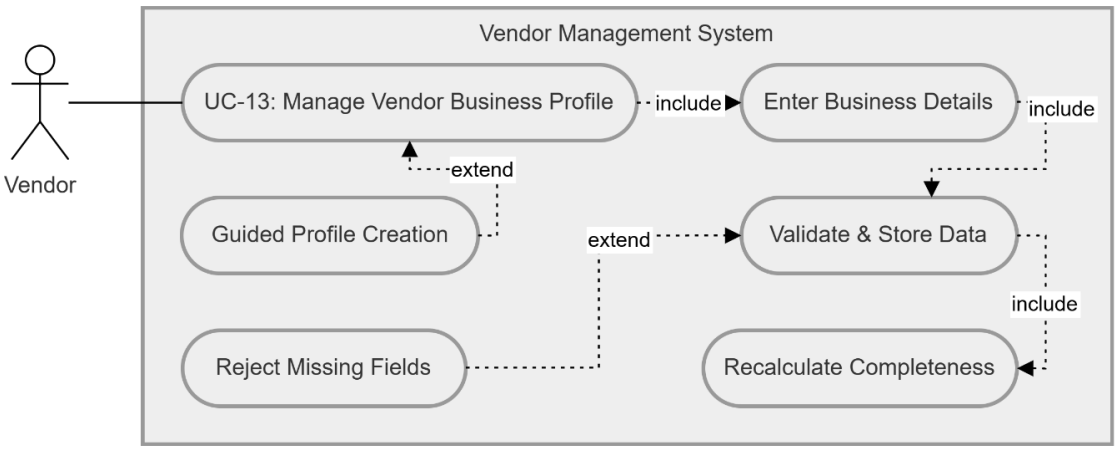

Figure 14: UCD-13

3.2.14 UC-14: Manage Services and Packages

Feature	Details
Use-Case ID	UC-14
Use-Case Name	Manage Service Packages
Primary Actor	Vendor
Supporting Actors	None
Description	Allows vendors to define, modify, or remove the specific service tiers and pricing packages offered to customers.
Preconditions	The vendor is logged in and has an existing vendor profile.
Trigger	The vendor opens the service or package management dashboard.

Main Flow	1. Vendor adds, edits, or deletes specific service packages. 2. Vendor enters package details: name, description, features, and price. 3. System validates the package data for consistency. 4. System stores the package updates in the database. 5. System automatically recalculates the vendor's "Starting Price" displayed on the marketplace.
Alternative Flows	**A1 (Invalid Data):** If the pricing is non-numeric or mandatory features are missing, the system blocks the update. **A2 (Package Deletion):** If a package is deleted, the system refreshes all dependent customer-facing displays.
Postconditions	The vendor's service offerings are updated and accurately reflected in the marketplace listings.

Use-Case Diagram

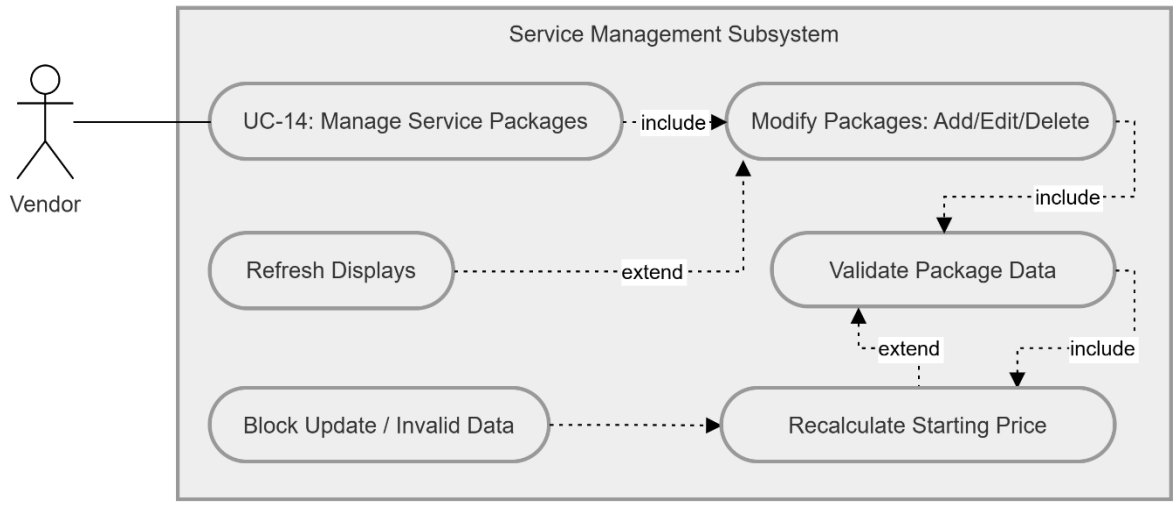

Figure 15: UCD-14

3.2.15 UC-15: Manage Portfolio Media

Feature	Details
Use-Case ID	UC-15
Use-Case Name	Manage Portfolio Media
Primary Actor	Vendor

Supporting Actor	Media Storage Service
Description	Enables vendors to upload, organize, and delete visual content (images/videos) to showcase their work to potential customers.
Preconditions	The vendor is logged in and possesses an active business profile.
Trigger	The vendor initiates a media upload or removal action.
Main Flow	1. Vendor selects cover or portfolio media files to upload. 2. System transfers the file to the external media storage service. 3. System stores media metadata (URL, type, size) against the vendor profile. 4. Vendor optionally updates captions or deletes existing portfolio items.
Alternative Flows	**A1 (Upload Failure):** If the transfer to storage fails, the system reports the error and suggests a retry. **A2 (Invalid Media):** If the file type or size is unsupported, the system rejects the file and notifies the vendor.
Postconditions	The vendor's portfolio is updated, reflecting the latest media assets for customer viewing.

Use-Case Diagram

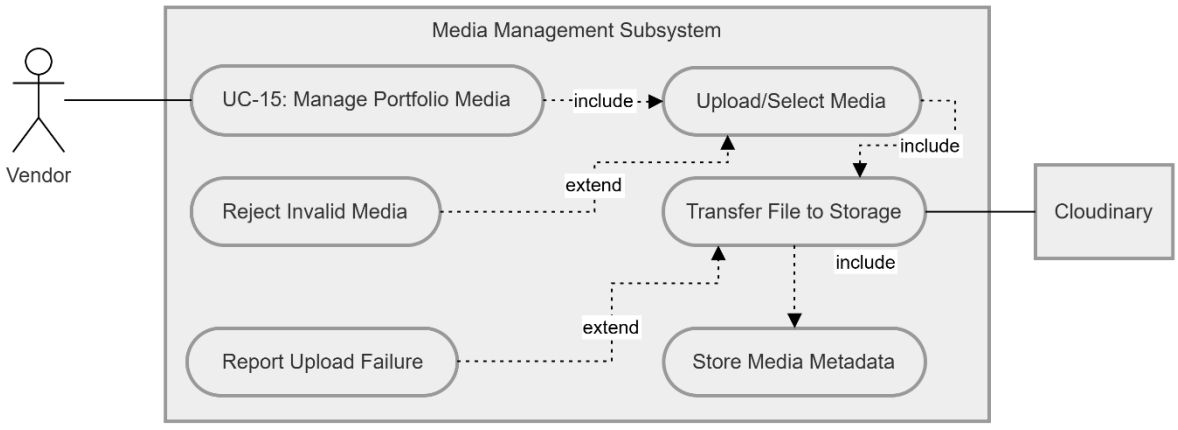

Figure 16: UCD-15

3.2.16 UC-16: Manage Incoming Bookings

Feature	Details
Use-Case ID	UC-16
Use-Case Name	Manage Incoming Bookings
Primary Actor	Vendor
Supporting Actor	Customer
Description	Enables vendors to manage their workflow by reviewing incoming booking requests, making status determinations (approve/reject), and tracking payments.
Preconditions	The vendor is logged in and has an active vendor profile associated with their account.
Trigger	The vendor navigates to and opens the bookings management dashboard.
Main Flow	1. Vendor views the list of incoming booking requests. 2. Vendor inspects specific details for a selected booking. 3. Vendor chooses to approve or reject pending requests. 4. Vendor updates the payment status for bookings that have been approved. 5. System updates the internal booking state and sends a notification to the customer.
Alternative Flows	**A1 (Invalid State):** If a vendor tries to update a booking that is not in a modifiable state (e.g., already completed), the system blocks the action. **A2 (Booking Not Found):** If the booking record no longer exists, the system displays an error message. **A3 (Vendor Cancellation):** If the vendor cancels an eligible booking, the system updates the status and notifies the customer accordingly.
Postconditions	All vendor actions are recorded, and the booking lifecycle state is updated and visible to both parties.

Use-Case Diagram

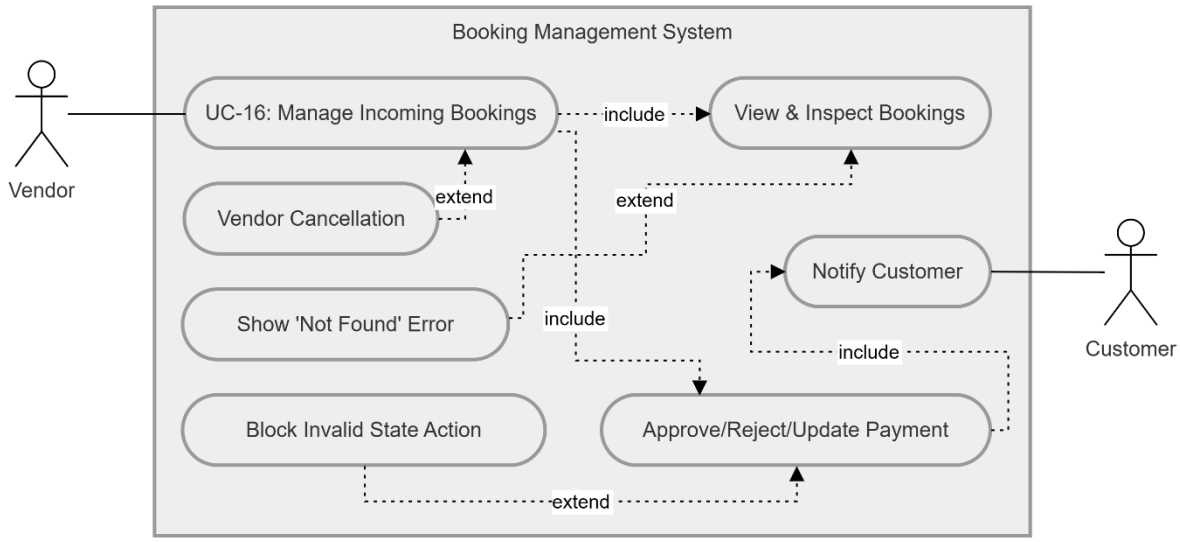

Figure 17: UCD-16

3.2.17 UC-17: In-App Messaging

Feature	Details
Use-Case ID	UC-17
Use-Case Name	Communicate via Real-time Messaging
Primary Actors	Customer, Vendor
Supporting Actor	Real-time Messaging Service
Description	Facilitates direct, real-time communication between customers and vendors to discuss event details, availability, and specific requirements.
Preconditions	Both the customer and the vendor are authenticated and authorized to access the messaging module.
Trigger	A participant opens an existing conversation or initiates a new message.
Main Flow	1. Participant opens or creates a conversation thread. 2. Participant types and sends a message. 3. System stores the message in the database. 4. Recipient receives unread indicators or push notifications.

	5. Recipient opens the conversation and messages are marked as read.
Alternative Flows	**A1 (Unauthorized Access):** If a user attempts to access a conversation they are not part of, the system denies access. **A2 (Delivery Failure):** If a message fails to send due to connectivity, the system preserves the draft or indicates failure for manual retry at the interface level.
Postconditions	A persistent communication history is stored and remains accessible to both participants for future reference.

Use-Case Diagram

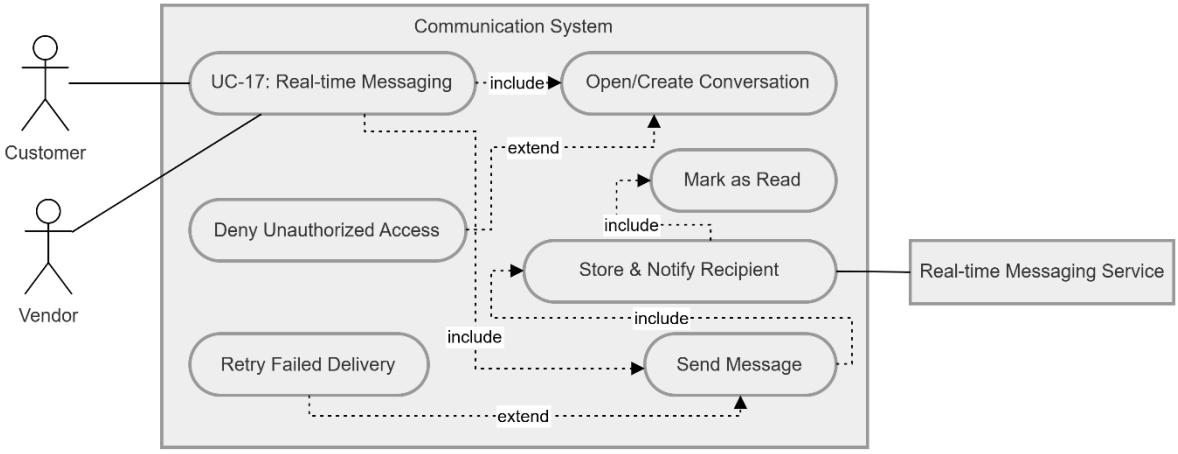

Figure 18: UCD-17

3.2.18 UC-18: View Sales Analytics

Feature	**Details**
Use-Case ID	UC-18
Use-Case Name	View Business Analytics
Primary Actor	Vendor
Supporting Actors	None
Description	Provides vendors with a visual overview of their business performance, including revenue trends, booking volumes, and payment statistics to support data-driven decisions.

Preconditions	The vendor is logged in and possesses historical booking/payment data or has authorized access to the analytics module.
Trigger	The vendor navigates to and selects the analytics or dashboard page.
Main Flow	1. Vendor selects the analytics page from the navigation menu. 2. System retrieves relevant revenue, booking, and payment statistics from the database. 3. System renders the data using Key Performance Indicators (KPIs), interactive charts, and trend summaries. 4. Vendor reviews and interprets the performance data to monitor business health.
Alternative Flows	**A1 (No Data):** If the vendor has no historical activity, the system displays an empty-state dashboard with placeholder illustrations. **A2 (Retrieval Failure):** If the system fails to fetch statistics, it displays an error state with a "Retry" option.
Postconditions	The vendor gains insights into their business performance through a structured data visualization.

Use-Case Diagram

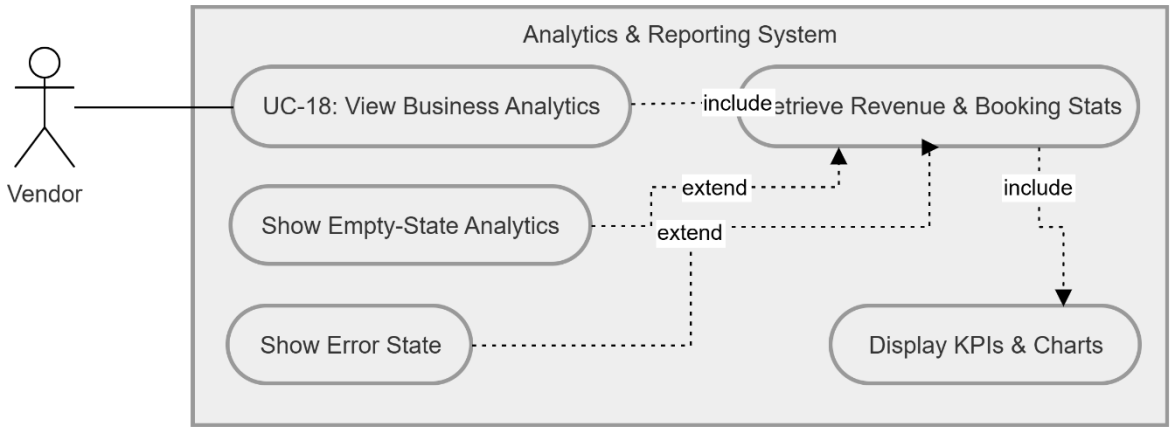

Figure 19: UCD-18

Feature	**Details**
Use-Case ID	UC-19
Use-Case Name	Supervise User Accounts

Primary Actor	Admin
Supporting Actors	None
Description	Enables administrators to monitor the platform's user base and control account access by activating or deactivating specific users.
Preconditions	The administrator is successfully authenticated with appropriate privileges.
Trigger	The admin navigates to and opens the user management page.
Main Flow	1. Admin retrieves the list of all registered users. 2. Admin inspects specific user account details and history. 3. Admin toggles the account status (Active/Inactive). 4. System stores the updated status and applies access restrictions immediately.
Alternative Flows	**A1 (User Not Found):** If a selected user record is missing or deleted, the system displays an error message. **A2 (Unauthorized):** If the session lacks administrative rights, the system denies access to the management interface.
Postconditions	User account records are updated and administrative control over platform access is maintained.

Use-Case Diagram

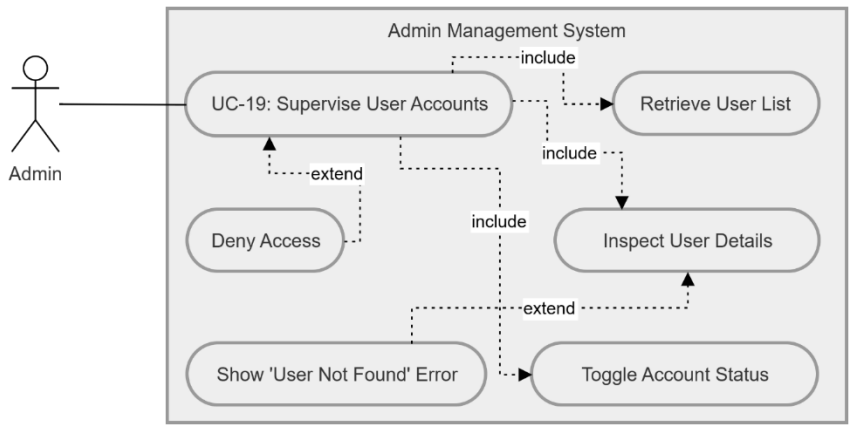

Figure 20: UCD-19

3.2.20 UC-20: Verify / Reject Vendors

Feature	Details
Use-Case ID	UC-20
Use-Case Name	Manage Vendor Verification
Primary Actor	Admin
Supporting Actor	Vendor
Description	Allows administrators to review pending vendor verification requests, inspect submitted evidence, and approve or reject vendors to ensure platform trust and quality.
Preconditions	1. Admin is authenticated with appropriate permissions. 2. Pending vendor verification requests exist in the system.
Trigger	The admin navigates to and opens the vendor verification management dashboard.
Main Flow	1. Admin views a list of all pending vendor verification requests. 2. Admin inspects specific vendor profile data and submitted documentation/evidence. 3. Admin makes a decision to either approve or reject the vendor. 4. System updates the vendor's verification status in the database. 5. System sends a notification to the vendor regarding the verification decision.
Alternative Flows	**A1 (Insufficient Data):** If the provided evidence is incomplete or invalid, the admin rejects the request and specifies the reason. **A2 (Duplicate Action):** If the vendor has already been processed by another admin, the system blocks the duplicate status update.
Postconditions	The vendor’s verification state is updated (Approved or Rejected), and the vendor is notified.

Use-Case Diagram

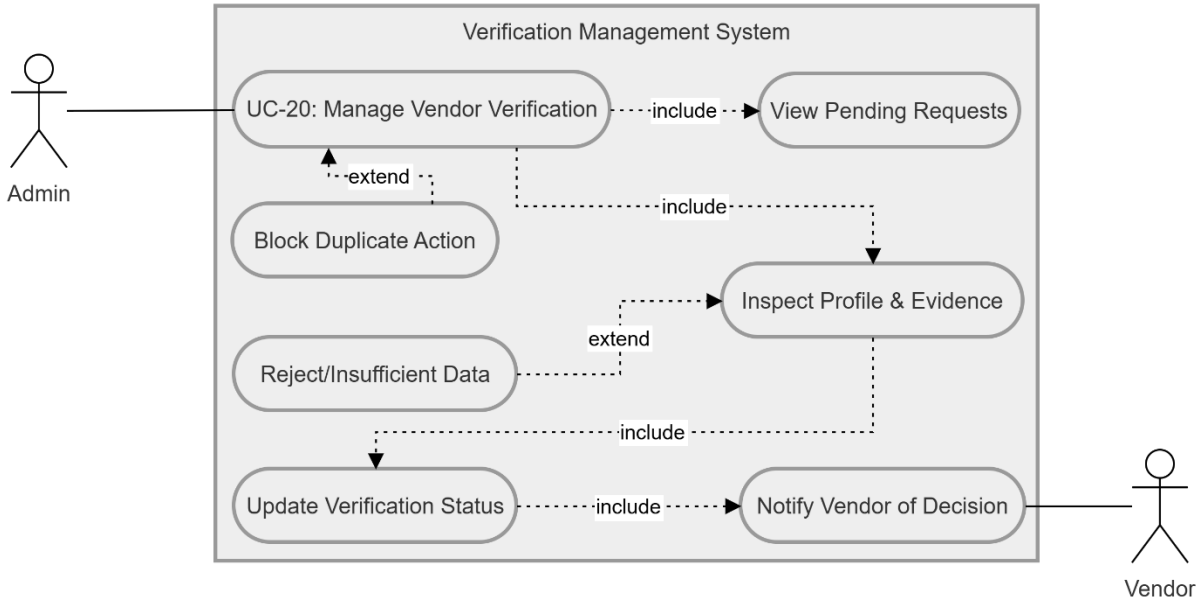

Figure 21: UCD-20

3.2.21 UC-21: Manage Reports and Platform Violations

Feature	**Details**
Use-Case ID	UC-21
Use-Case Name	Moderate Platform Reports
Primary Actor	Admin
Supporting Actors	Customer, Vendor
Description	Enables administrators to review reports submitted by users regarding potential platform violations, evaluate evidence, and take moderation actions to maintain community standards.
Preconditions	1. Admin is authenticated with moderation privileges. 2. Report records (submitted by Customers or Vendors) exist in the system.
Trigger	The admin navigates to and opens the moderation/reports module.
Main Flow	1. Admin views a list of all submitted reports. 2. Admin reads specific report details, including the target type (User/Vendor) and attached evidence.

	3. Admin updates the report status (Pending, In Review, Resolved, or Rejected). 4. Admin records internal notes and selects a moderation action (e.g., warning, suspension). 5. System stores the final moderation decision and audit trail.
Alternative Flows	**A1 (Conflicting Action):** If a report is already resolved or marked as invalid, the system restricts the admin from taking further conflicting actions. **A2 (Missing Target):** If the content or user being reported has already been deleted, the system still preserves the report history for legal/archival purposes.
Postconditions	Platform violations are successfully reviewed, and appropriate moderation actions are applied and recorded.

Use-Case Diagram

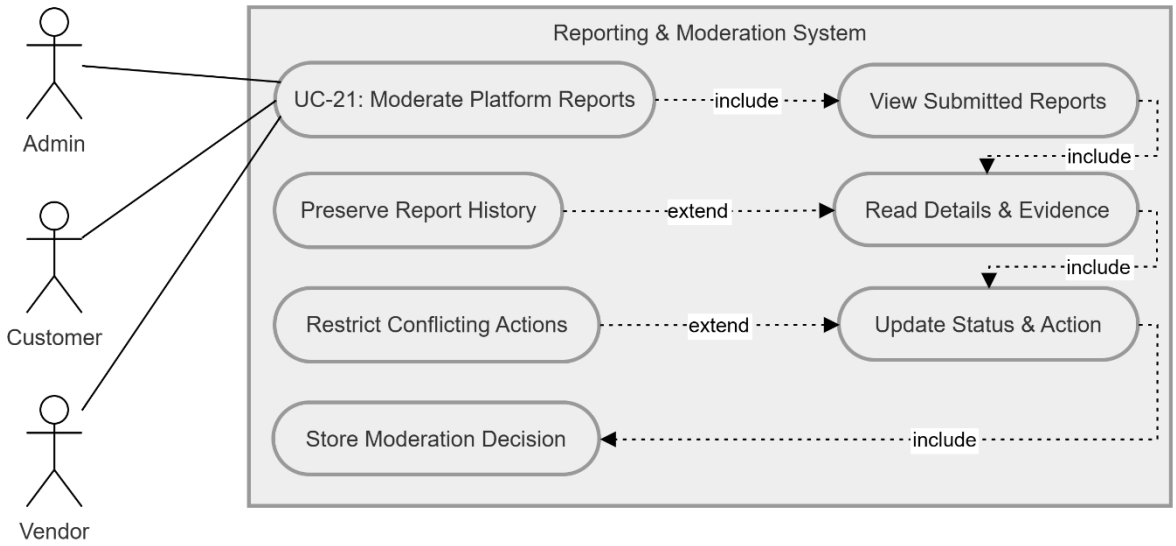

Figure 22: UCD-21

3.2.22 UC-22: Monitor System Logs and Health

Feature	Details
Use-Case ID	UC-22
Use-Case Name	Monitor System Health and Logs
Primary Actor	Admin

Supporting Actors	None
Description	Provides administrators with real-time visibility into the platform's operational state, including service health indicators and activity logs for troubleshooting and monitoring.
Preconditions	The administrator is successfully authenticated with system-level access privileges.
Trigger	The admin navigates to the system logs or health monitoring dashboard.
Main Flow	1. Admin accesses the system logs and health monitoring interface. 2. System retrieves and displays platform activity history and service health metrics. 3. Admin reviews the data to identify abnormal usage patterns, failures, or performance bottlenecks. 4. Admin utilizes the insights for operational decision-making and preventative maintenance.
Alternative Flows	**A1 (Logs Unavailable):** If the logging service fails to respond, the system displays a retrieval error message. **A2 (Health Data Failure):** If specific health metrics (e.g., CPU, Memory, API status) cannot be loaded, the system informs the admin and provides a refresh option.
Postconditions	The administrator is fully informed of the platform's current operational status and historical activity.

Use-Case Diagram

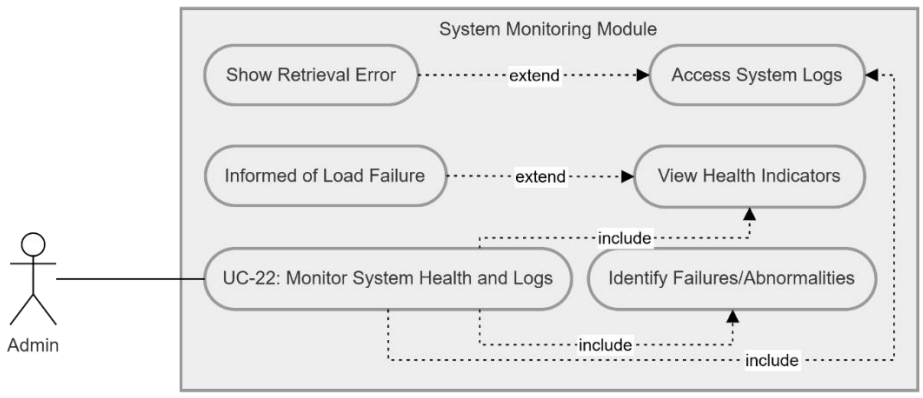

Figure 23: UCD-22

3.2.23 UC-23: View All Bookings

Feature	Details
Use-Case ID	UC-23
Use-Case Name	Supervise Platform-Wide Bookings
Primary Actor	Admin
Supporting Actors	None
Description	Allows administrators to gain a high-level and detailed view of all booking activity on the platform, including participant details and financial statuses for oversight.
Preconditions	The administrator is successfully authenticated with appropriate system permissions.
Trigger	The admin navigates to and opens the booking supervision/management page.
Main Flow	1. Admin initiates the retrieval of platform-wide booking records. 2. System displays booking statuses, participants (Customer/Vendor), event details, and payment states. 3. Admin applies filters or navigates through paged results. 4. Admin utilizes the aggregated data for monitoring and operational oversight.
Alternative Flows	**A1 (No Bookings):** If no booking records exist in the database, the system displays an empty-state message. **A2 (Retrieval Failure):** If the database request fails, the system displays an error response and provides a retry option.
Postconditions	The administrator successfully monitors the booking activity across the entire platform.

Use-Case Diagram

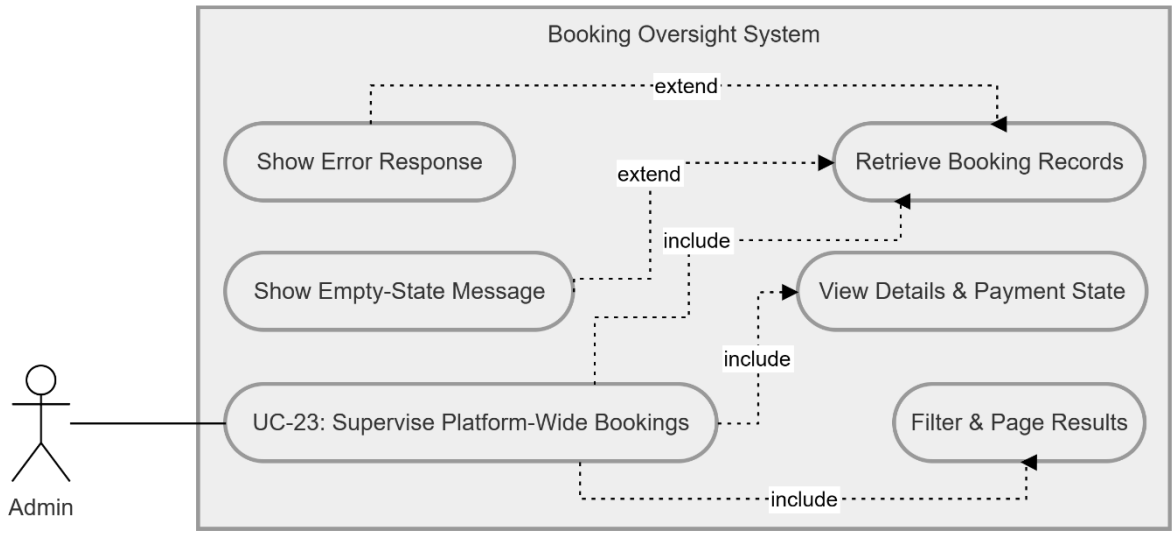

Figure 24: UCD-23

3.3 Interface Requirements

3.3.1 User Interface Requirements

The company Vid-AI will provide unique but consistent user interfaces to its customers, vendors, and administrators.

1. The company will provide a web interface to its customers for searching for vendors, booking services, budgeting plans, talking with the AI system, sending and receiving messages, and inviting others.
2. The company will provide a unique web interface for the system's vendors for profile management, offering services/packages, responding to booking requests, sending and receiving messages, posting reviews, portfolio management, and visualization of analytical data.
3. The company will provide an interface for system administrators for dashboard management, verifying the vendors, user management, booking management, report generation, and system monitoring.
4. There will be a mobile interface provided to the system's customers for vendor searching, booking management, budget planning, messaging, and notifications.
5. There will be distinct status indicators for booking status, payment status, and notification status.
6. There will be form validation on the input side of all forms and error message generation upon failure.

3.3.2 Physical Interface Requirements

1. Customer web interface must be usable on standard desktop or laptop computers running modern web browsers.
2. Mobile application must work on smartphones using Expo/React Native runtime environment.
3. Server-side system must use hardware/cloud hosting to support the following services: Node.js backend, AI microservice, MongoDB connectivity, and media/payment integration.
4. Access to internet is required for connecting to the following services: AI, payments, cloud storage, client-server communication.

3.3.3 Software and Components Interface Requirements

1. The frontend and mobile app shall communicate with the backend via RESTful HTTP API calls with JSON content.
2. The backend shall validate protected API requests using access control based on JSON Web Tokens (JWT).
3. The backend shall communicate with MongoDB for storing persistent data.
4. The backend shall interact with the AI microservice via HTTP calls to get chats, recommendations, and budget plan suggestions.
5. The backend shall integrate with Stripe to create a checkout session and process payments using webhooks.
6. The backend shall integrate with a third-party media storage solution like Cloudinary to store images and videos.
7. The system shall provide real-time communication capabilities using Socket.IO messaging.
8. The system must be able to register tokens for notifications on the mobile application.

3.4 Database Requirements for Vid-AI

Vid-AI needs authentication, user profiles, vendor profiles, booking management, budgets, messaging, reporting, invitations and logs which necessitate the need for persistent storage using a database. For this reason, MongoDB is selected as the database owing to the fact that the data model contains both rigidly defined fields together with flexible and nested data structures such as packages, portfolios, comments, budget entries, allocations of AI plans and notifications.

The following are requirements for the database:

1. Each user must have a unique identity record including their role, authentication, and onboarding information.

2. Vendor profile is associated with one user only.
3. The bookings collections will store booking details, booking state, payment state, and vendor response/cancellation state.
4. The budget collection will store total budget, budget allocation, expenses and AI plan generation.
5. The messaging system stores conversations and messages separately.
6. The notification collection will store recipient, notification type, mode and received notifications.
7. The reporting system will take care of moderation workflows where reporter information, subject information and reason category and administration workflow are required.
8. The invitation system will keep records of generated content and styling information.
9. Timestamps will be stored for audit and chronological processing purposes.
10. Indexes should be created for common querying such as user role, vendor filter, booking status, and message pagination.

Entity	**Purpose**
User	Stores authentication, role, personal profile, onboarding, and push tokens
Vendor	Stores business profile, category, services, packages, portfolio, verification, ratings, and analytics inputs
Booking	Stores booking request, event details, price, payment, vendor response, and lifecycle status
Budget	Stores total budget, category allocations, spending, and AI budget plans
WeddingEvent	Stores event-level planning information such as mehndi, baraat, walima, and nikkah
Conversation	Stores communication channels between customer and vendor
Message	Stores message-level content and read state
Notification	Stores in-app notifications and delivery state
Review	Stores customer feedback on vendors

Report	Stores moderation complaints submitted by users or vendors
Invitation	Stores invitation design input and generated invitation output
ActivityLog	Stores auditable system activity for administration

Table 6: Major Database Entities and Purpose

Figure 3.2: ER Diagram

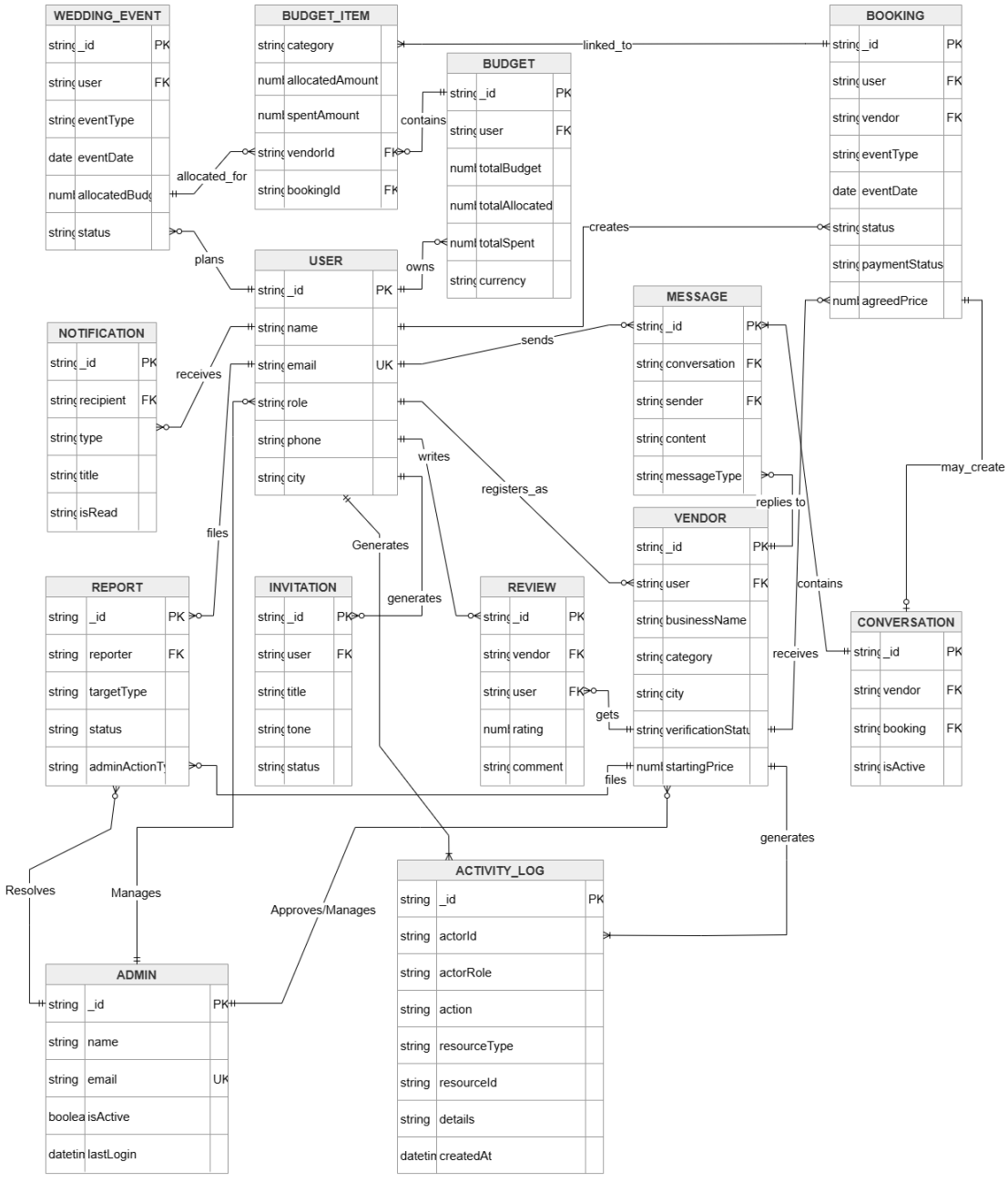

Figure 25: ER Diagram

3.5 Non-Functional Requirements

3.5.1 Security Requirements

1. The authentication of users in the system will be done using secure token-based authentication.
2. The system will use user authorization using roles for customers, vendors, and admins.
3. User password will be stored as hashes and not in plain text.
4. The system will ensure HTTP security by using security headers, sanitizing of the requests, and protection from parameter pollution.
5. The system will have rate limiting for public and authentication routes.
6. The webhook for payments must first validate the transactions before accepting transaction data.
7. Bookings, conversations, and payments can only be accessed by authenticated users.
8. The system will have moderation to avoid abuse and fraud.

3.5.2 Performance Requirements

1. Traditional non-AI API requests such as authentication, vendor searching, customer account information lookup, and booking lookups will need to be responded within seconds.
2. Any request for AI-assisted operations may take time than traditional CRUD operations, but graceful timeouts and fallback mechanisms will be available.
3. Queries of search, booking, and messages retrieval should be paginated.
4. Caching or any form of state management is to be used by the frontend for queries.
5. The system should allow simultaneous access by multiple customers, suppliers, and administrators while maintaining data integrity.

3.5.3 Usability Requirements

1. The system should provide navigable paths for various users according to their roles.
2. The user interface design should be responsive to display screens of desktop and mobile devices.
3. The system should use understandable labels, statuses, badges, and workflows.
4. The submission of forms should verify the information entered and recover any possible mistakes.

5. Important tasks like booking approval, payment processing, and booking cancellation should stand out.

3.5.4 Requirements on Reliability and Availability

1. Errors will be handled internally within the system in case of system failures or misuse by users.
2. An endpoint for health check will be provided in the backend to monitor the system's performance.
3. Data that is important for business operations, such as booking, payment transactions, and reporting, will be persisted.
4. Events of significance will be logged within the system.
5. The booking status of the booking process shall not change due to time-based processes for updating the status of bookings.

3.5.5 Requirements on Modifiability and Maintainability

1. The system architecture will include three separate layers, including the frontend, backend, and AI-service module.
2. Business logic will be divided into controllers, routes, models, and middleware.
3. The functionality of AI algorithms will be developed independently from other parts of the application.
4. Common API layer shall be leveraged in the process of building all web and mobile interfaces to avoid redundancies.
5. Vendor category extensions, booking attributes, reporting features, and AI services will be highly extendable without the need for full-scale platform re-engineering.

3.5.6 Interoperability Requirements

1. API interactions of all components will have to rely on the HTTP APIs in the JSON format.
2. The system will communicate with different third-party services such as Stripe, media hosting, and AI services.
3. Both mobile and web clients should access the same backend API endpoints in order to achieve consistency.
4. The system should have means to establish real-time communication channels for chat purposes and notifications.

3.5.7 Constraints

1. Frontend development is limited by the use of React/Vite web stack.
2. Mobile development is limited by React Native/Expo ecosystem.

3. Backend development is limited by Node.js, Express, MongoDB, and third-party API integration.
4. Quality of AI generation is influenced by the availability of models, prompts, and LLM behavior both locally and externally.
5. Payments functionality depends on the availability of the Stripe service and correct configuration of the deployment.
6. Time limitation, internet access, availability of third-party services, and hardware limitation influence development and deployment.

3.6 Project Feasibility

3.6.1 Technical Feasibility

The technical feasibility of this project is guaranteed due to the availability of all key aspects that are easy to implement via well-developed technological solutions. Specifically, web front-end, mobile application client, backend API, and artificial intelligence service all are based on commonly used platforms with substantial communities of developers. Using the MongoDB database allows dealing efficiently with semi-structured information such as marketplace, booking, and planning data. Integrating third-party services like payments from Stripe, hosting images, and enabling real-time messaging is possible via SDKs and existing APIs. Using Docker enhances feasibility due to easy multiple services implementations.

3.6.2 Operational Feasibility

Operational feasibility is provided due to solving an actual issue experienced by both customers and sellers. As for users, a clear structure is needed to be able to find vendors, manage budget and make bookings. From the vendor perspective, a website is necessary for presenting their offers, managing bookings, and analyzing business results. Moreover, administrators require operational insight and ability to moderate the activity. All these aspects are taken into consideration in terms of interface development and workflow planning.

3.6.3 Legal and Ethical Feasibility

Vid-AI will remain legal and ethical if it adheres to appropriate guidelines when collecting and managing personal information and implementing platform management policies.

1. Personal data will be collected only for platform operation needs such as identity verification, booking, and communications.
2. Payments will be processed via a third party (Stripe), meaning that Vid-AI does not store any payment credentials from the users.
3. The use and misuse reporting related to vendor media and user-created content will be performed properly.

4. Reporting and moderation of abuse, fraud, spam, misinformation, and harassment will be available.
5. The AI output will be seen as supplementary to decisions since LLM results can be flawed and biased.
6. The users will be informed about the AI's recommendations being supplementary.

3.7 Analysis Models

3.7.1 Activity Diagrams

3.7.1.1 Activity Diagram – User Registration Workflow

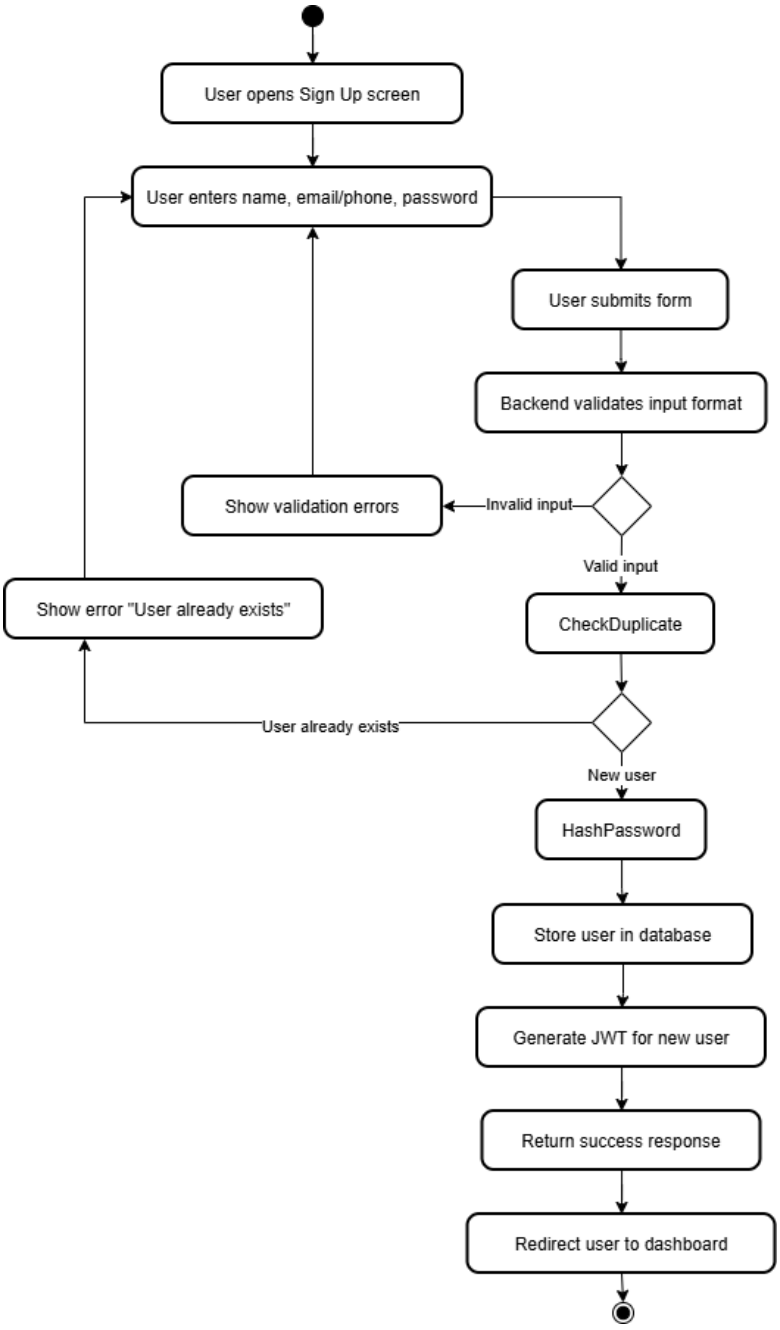

Figure 26: Activity Diagram – User Registration Workflow

3.7.1.2 Activity Diagram – Invitation Creation

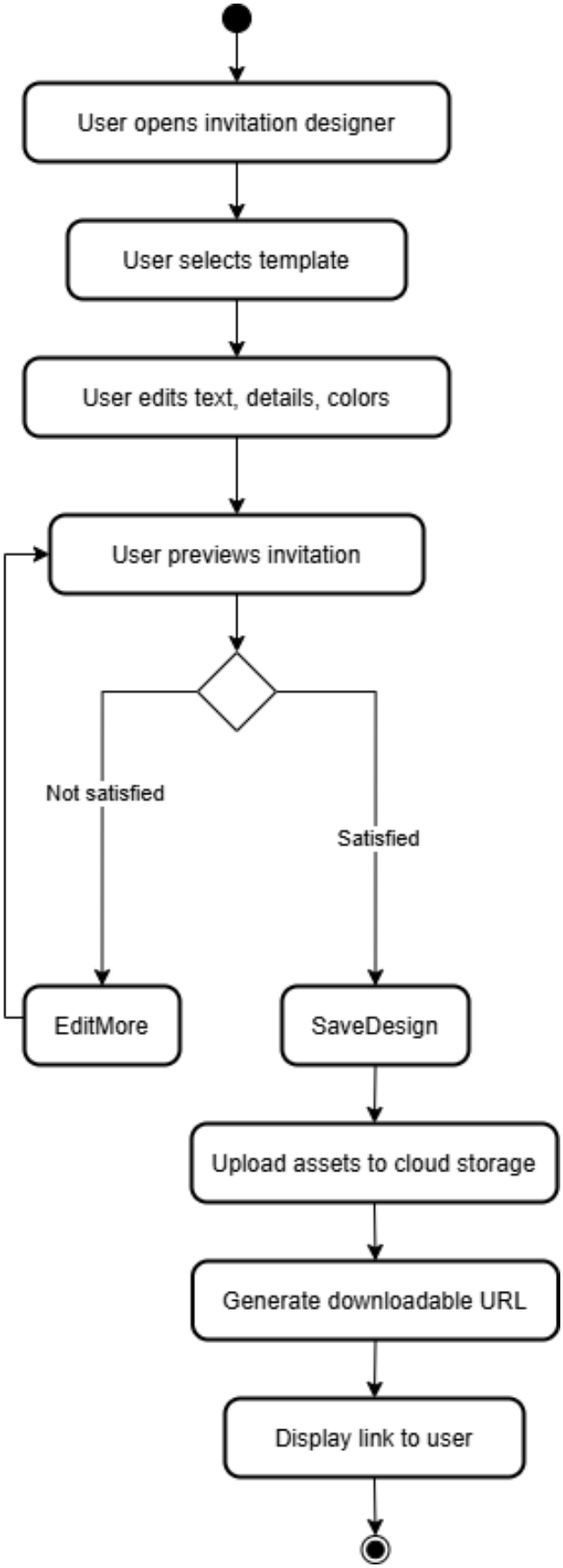

Figure 27: Activity Diagram – Invitation Creation

3.7.1.3 Activity Diagram – Vendor Booking

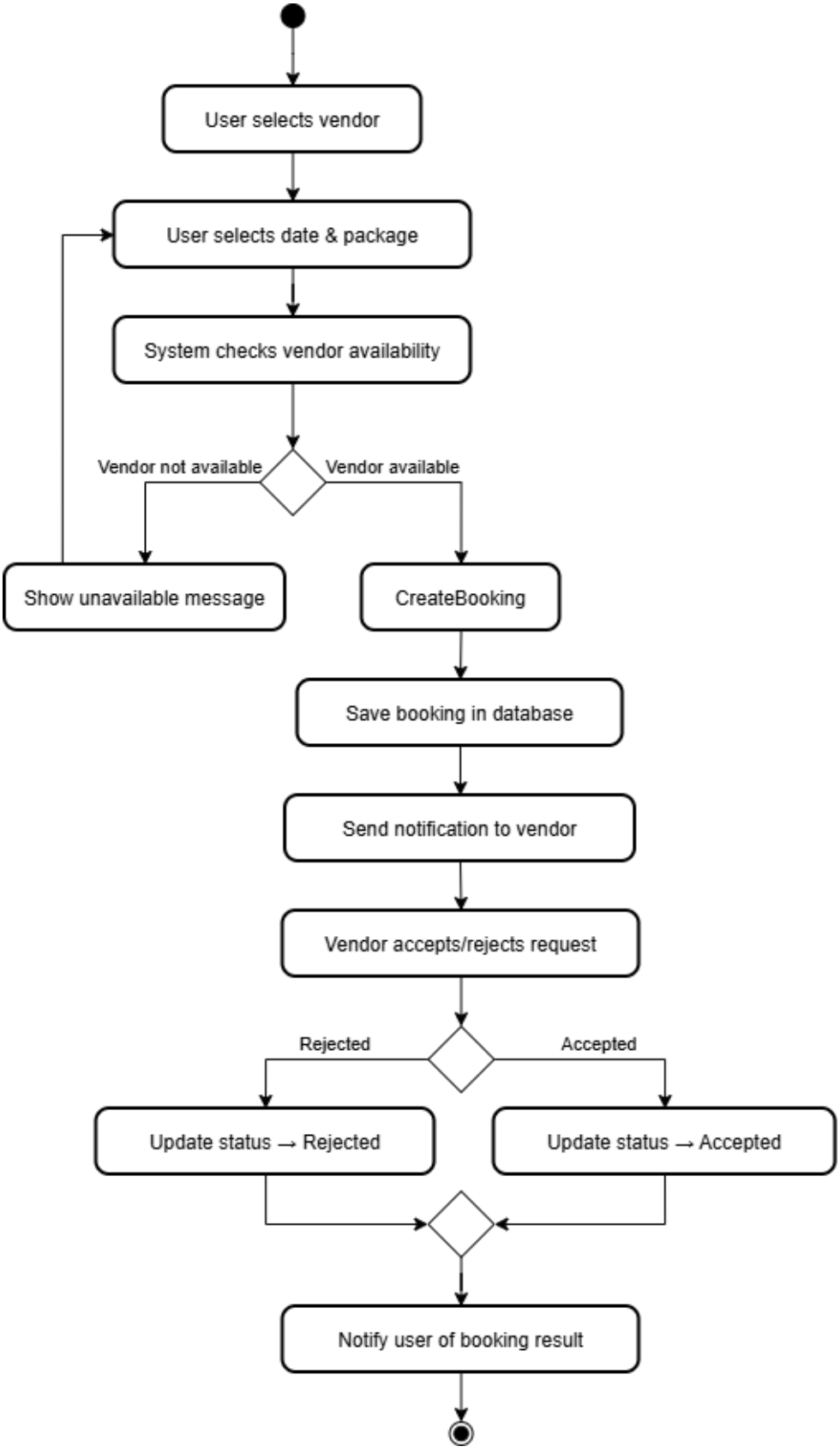

Figure 28: Activity Diagram – Vendor Booking

3.7.2 Sequence Diagrams

3.7.2.1 Sequence Diagram – Book Vendor

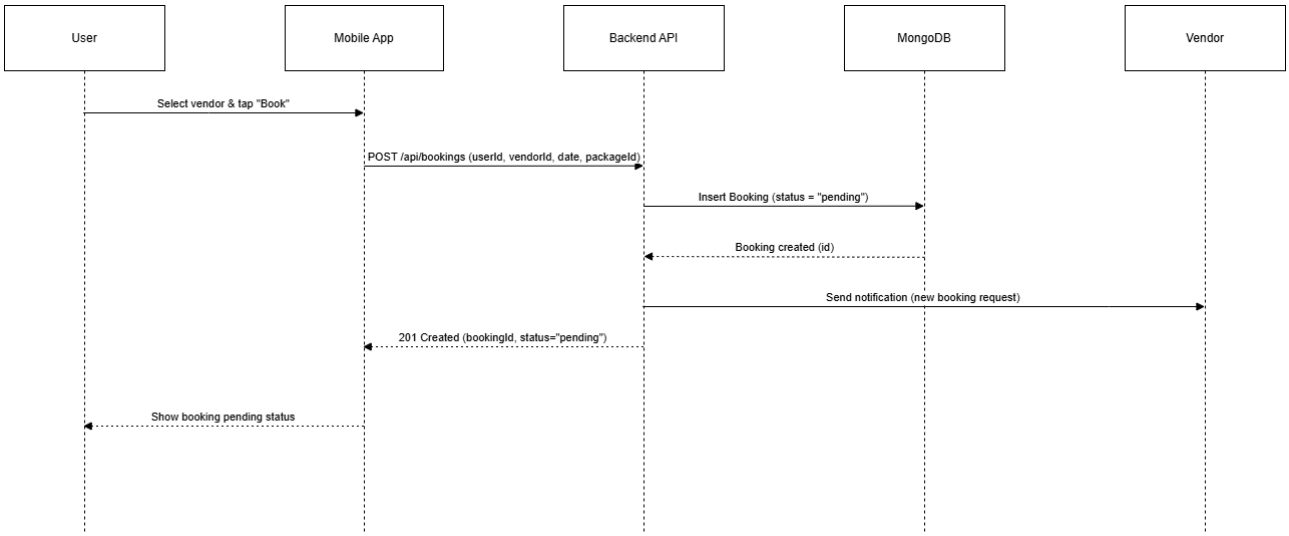

Figure 29: Sequence Diagram – Book Vendor

3.7.2.2 Sequence Diagram – AI Recommendation Request

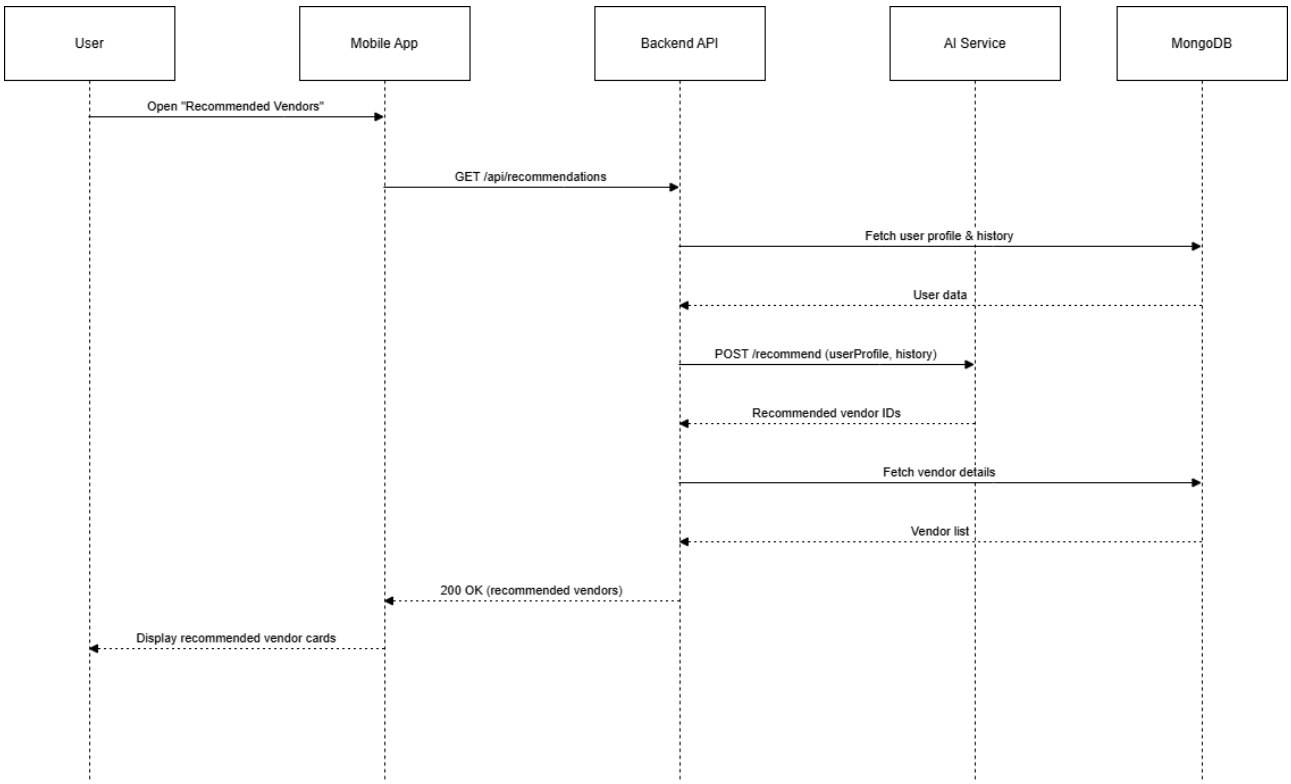

Figure 30: - Sequence Diagram – AI Recommendation Request

3.7.2.3 Sequence Diagram – User Login

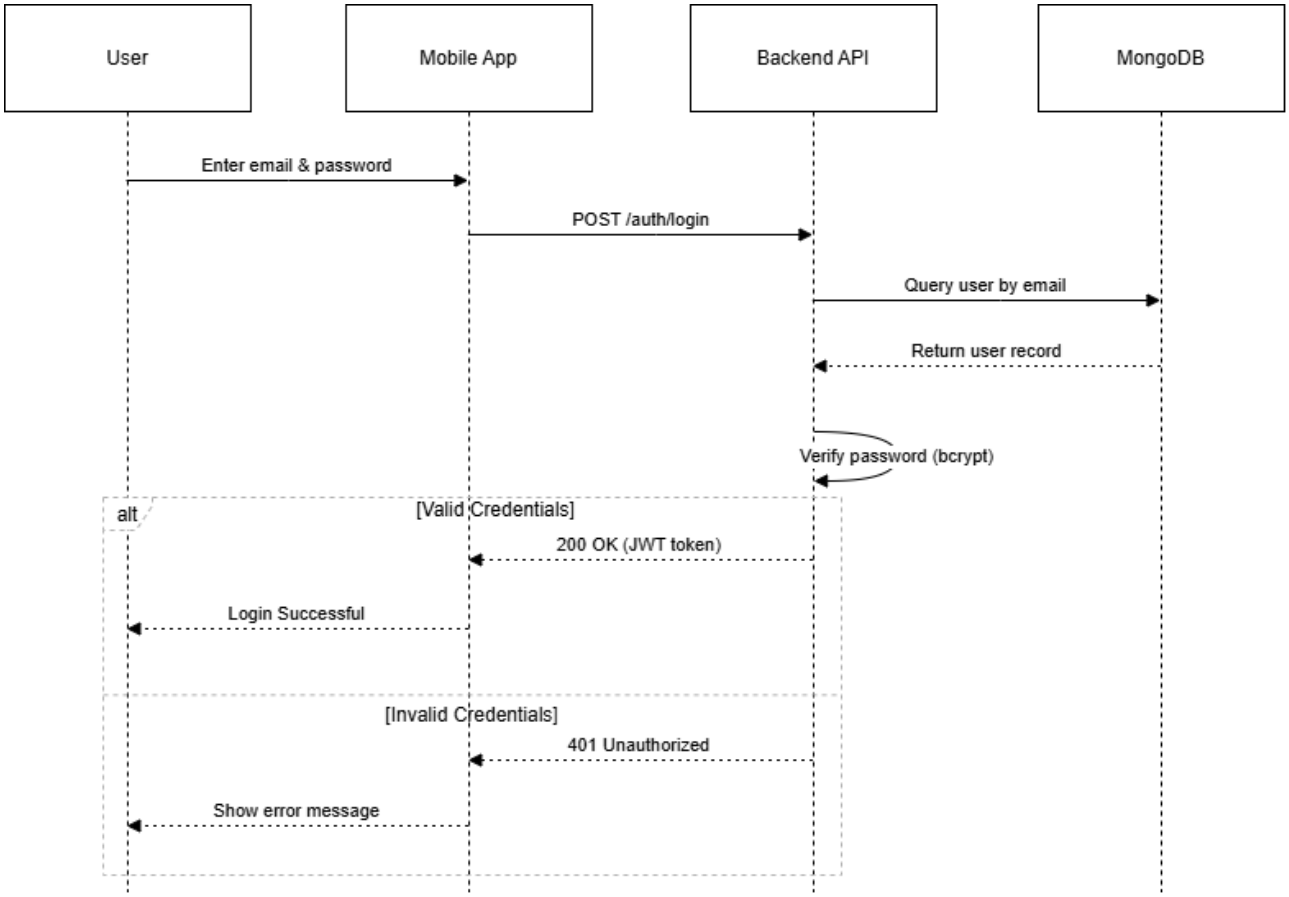

Figure 31: Sequence Diagram – User Login

3.7.2.4 Sequence Diagram – Vendor Accepts Booking

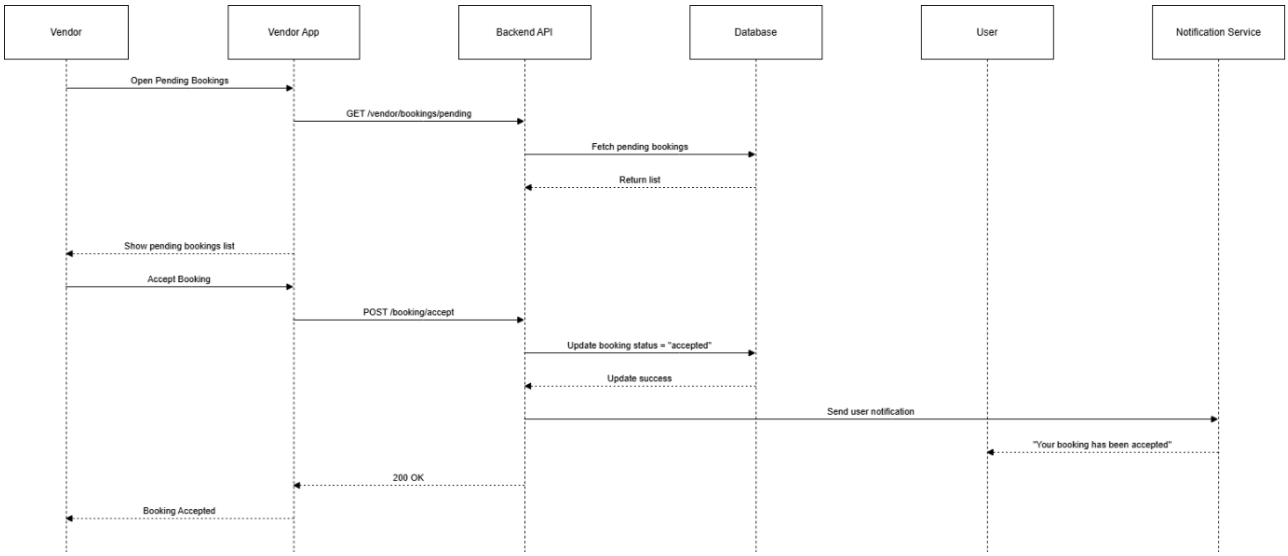

Figure 32: Sequence Diagram – Vendor Accepts Booking

3.7.2.5 Sequence Diagram – Create Digital Invitation

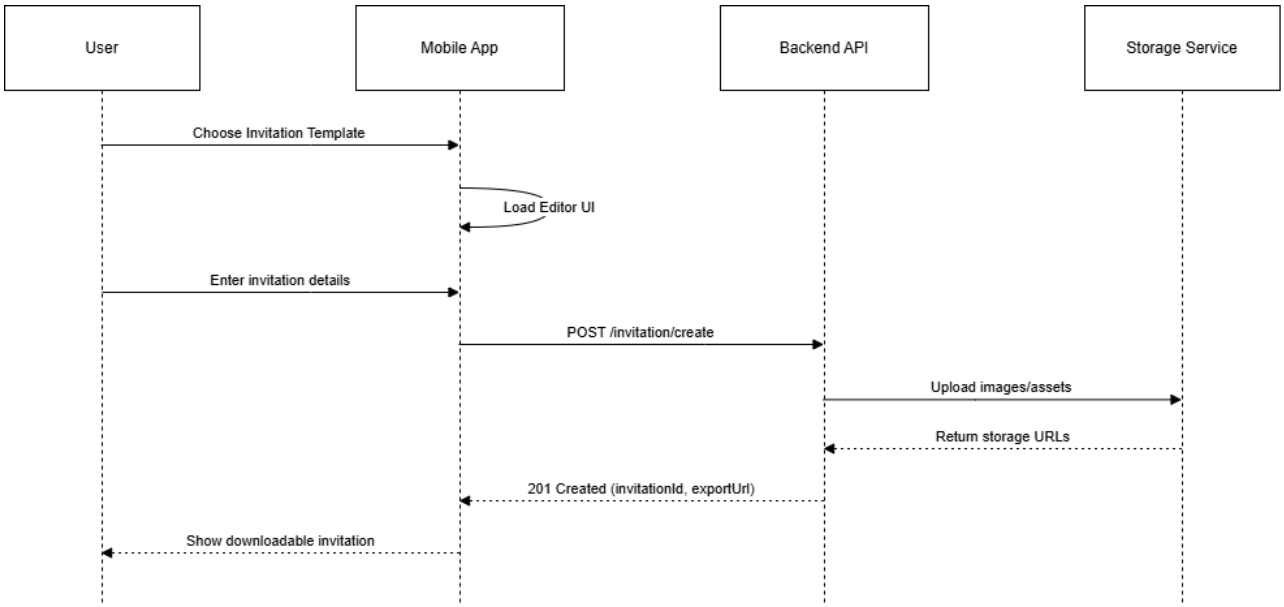

Figure 33: Sequence Diagram – Create Digital Invitation

3.7.2.6 Sequence Diagram – Chatbot Conversation

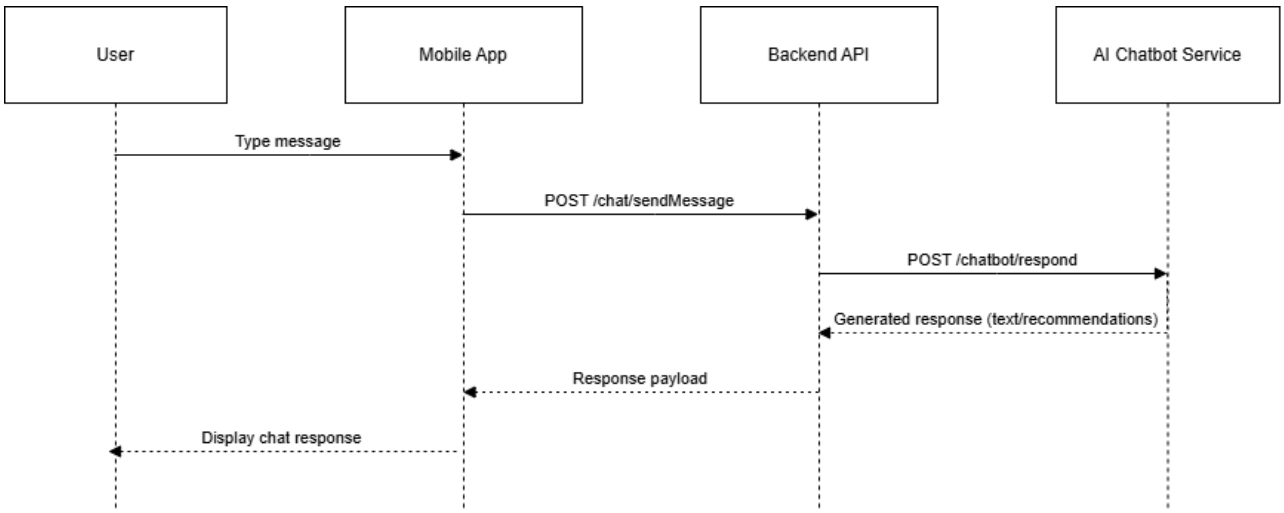

Figure 34: Sequence Diagram – Chatbot Conversation

3.8 Conclusion

In this chapter, the system requirements for Vid-AI were analysed from the perspectives of functions, interfaces, database, non-functional, and feasibility requirements. It can be concluded that Vid-AI is not a mere vendor directory, but a holistic planning environment incorporating market capabilities, artificial intelligence-based decision making, secure booking process and payment procedures, moderation, and cross-platform usability. Diagrams and use cases of the chapter provide the connection between the problem analysis in previous chapters and further implementation described in subsequent chapters.

Chapter 4: System Design

In this chapter, the system design will be presented from architectural, logical, dynamic, component, data, and user interface viewpoints. The design is based on the implemented system: a multi-role wedding planning platform that consists of the React web client, React Native mobile client, Node.js/Express backend, AI microservice based on FastAPI framework, MongoDB database, real-time messaging capabilities, and payments and media cloud storage integrations. As MermaidJS lacks precise UML support for each type of diagram, Mermaid-based alternatives to illustrate equivalent designs have been utilized where required to maintain their semantic integrity.

4.1 Design Methodology

The methodology selected for designing Vid-AI includes iterations and modularity of services. The design did not consider implementation of the solution as a monolithic application with all the interfaces and business logic being integrated. Instead, the system has been split into individual, but interconnected components since the considered domain involves separate issues: marketplace operation, booking and payments flow, planning assistance using AI, real-time collaboration, administration, and mobile accessibility.

The following are the major design considerations:

- **Modularity:** presentation, business, persistence, and AI processing logics are modularized.
- **Roles:** the product is built with three roles in mind: **customer**, **vendor**, and **administrator**.
- **APIs:** the web and mobile front-end applications will interact with the same backend via APIs, enabling identical business logic for different platforms.
- **Microservices for AI features:** since AI features include both conversational interface and recommendations, they should not impact transactional processes.
- **Modularity for scalability:** vendors, bookings, reporting, messaging, and planning will all have their own independent evolution.
- **Increased modularity:** UI and flows could be developed progressively as more features became available, depending on platforms and role differences.

The proposed architecture suits Vid-AI, as weddings planning is not a single use case but a workflow including searches, decision support, collaboration, transactions, and follow-up.

4.2 Design Constraints

Flexible though it may be, there are several limitations on this design that come from technology and operations.

- **Stack limitation:** the web client design must consider **React** and **Vite**; the mobile client design, **React Native** and **Expo**; the backend server design, **Node.js** and **Express**; and the AI service design, **FastAPI**.
- **Persistence limitation:** the backend design must consider **MongoDB**, implying a preference for document-oriented modeling with subdocuments and references.
- **Service dependency:** the payment processing design must consider **Stripe**; the media uploading design, **Cloudinary**; and the advanced language/image generation designs, **Gemini** and **Ollama**.
- **Resource limitation:** local LLM inference with **Ollama** imposes a computational constraint, particularly when running multiple concurrent requests or processing large prompts.
- **Deployment limitation:** the mobile application design must consider public access to the backend during development, requiring auxiliary support like **ngrok**.
- **Scope limitation:** since this is an FYP project, the design focuses on completing a full workflow rather than achieving enterprise-level scalability.
- **Consistency limitation:** the backend server must support both web and mobile clients, vendor systems, and administrative tasks while maintaining consistency.

These considerations did not inhibit implementation, but they certainly shaped its design.

4.3 System Architecture

The system features a mixed architecture based on both layers and services. Web and mobile applications make up the presentation layer. The server-side application with Node.js and Express is used as an application and orchestration layer. The FastAPI-based AI microservice is created as a particular intelligence layer. MongoDB Atlas functions as the core persistence layer. Socket.IO is used for runtime communication via messaging and notifications. Stripe and Cloudinary are employed for the external service infrastructure layer.

In terms of the process at large, we have the following:

- Customers, vendors, and admin users use the web application interface.
- Customers can also use the platform using the mobile application.
- Browsers or apps connect to the server via the HTTP API.
- Backend software takes care of all kinds of tasks such as authorization, booking, budget management, etc.
- All the AI-related tasks are delegated from the server to the AI microservice.

- The AI service utilizes the context information and interacts with Ollama and Gemini to prepare the output for the customer.
- Data persists in MongoDB.
- Messages and other events are handled using Socket.IO.
- Payments go through Stripe; media files are uploaded via Cloudinary.

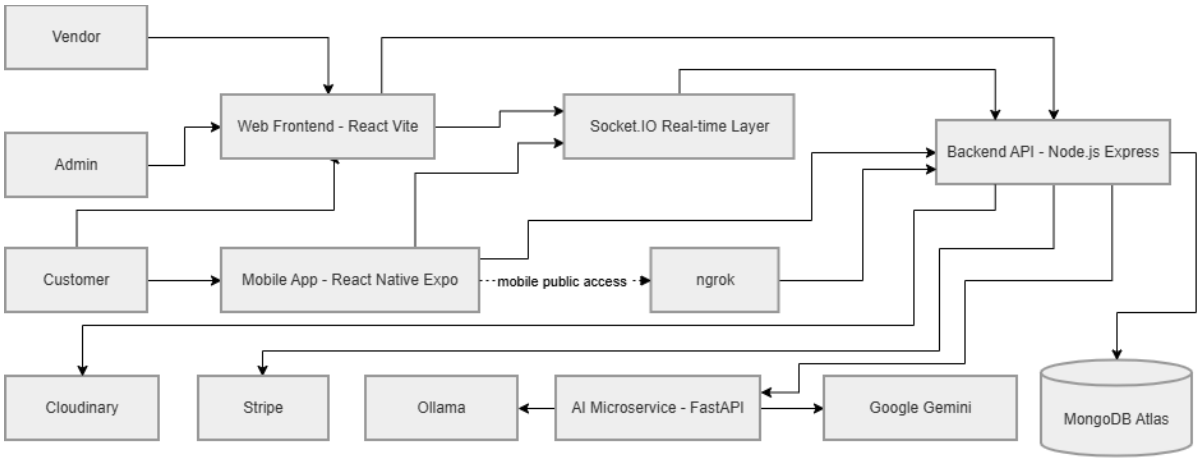

Figure 35: High-Level System Architecture

Such an architecture makes sense considering that the workload of the main transactional systems and AI subsystems is dissimilar. Segregation of transactional systems and AI subsystems enables higher maintainability and decoupling of the deterministic and non-deterministic workloads in **Vid-AI**.

4.4 Logical Design

The logical design of **Vid-AI** is based on a decomposition of the system into domains for the business logic. The key logical components are the following:

- **Identity & Access Management:** registration, login, authorization based on tokens, onboarding, and user profile management.
- **Vendor Marketplace Management:** search, filtering, viewing information about vendors, packages, portfolios, and reviews.
- **Booking and Payments Management:** creation, response to book request, payments, cancellation, automatic fulfillment, and expiration.
- **AI Planning and Recommendations:** chat support, vendor recommendation, budgeting, and invitation content creation.
- **Budget and Event Planning:** overall budget, budget per categories, wedding events, and budgeting context per event.
- **Messaging and Notifications Management:** messaging, unread count, and notifications.
- **Moderation and Administration:** user management, vendor validation, reporting, logging, and administration tasks.

Logically speaking, the architecture consists of **five tiers**:

1. **Presentation tier:** customer UI, vendor UI, admin UI, and mobile application UI.
2. **Application layer:** controllers, routes, middleware, and coordinating services.
3. **Domain tier:** objects such as user, vendor, booking, budget, report, review, and invite.
4. **Persistence tier:** database schemas in MongoDB with indexing.
5. **Integration/intelligence tier:** AI microservice, Stripe payment gateway, Cloudinary service, Socket.IO framework, Ollama, and Gemini.

The logical approach allows for each individual business domain to have its own ownership while being able to contribute to the overall workflow. In other words, the booking is dependent on vendor, payments, notifications, budgets, etc., but the concerns remain logically modular.

4.5 Dynamic View

Dynamic view concentrates on the way **Vid-AI's** components interact at run-time. There are three perspectives in particular which require attention: the lifecycle of the booking, booking and payment process, and AI-powered planning.

4.5.1 Booking State Machine

A booking goes through various states according to vendor decisions, payments status, cancellation, and date completion logic.

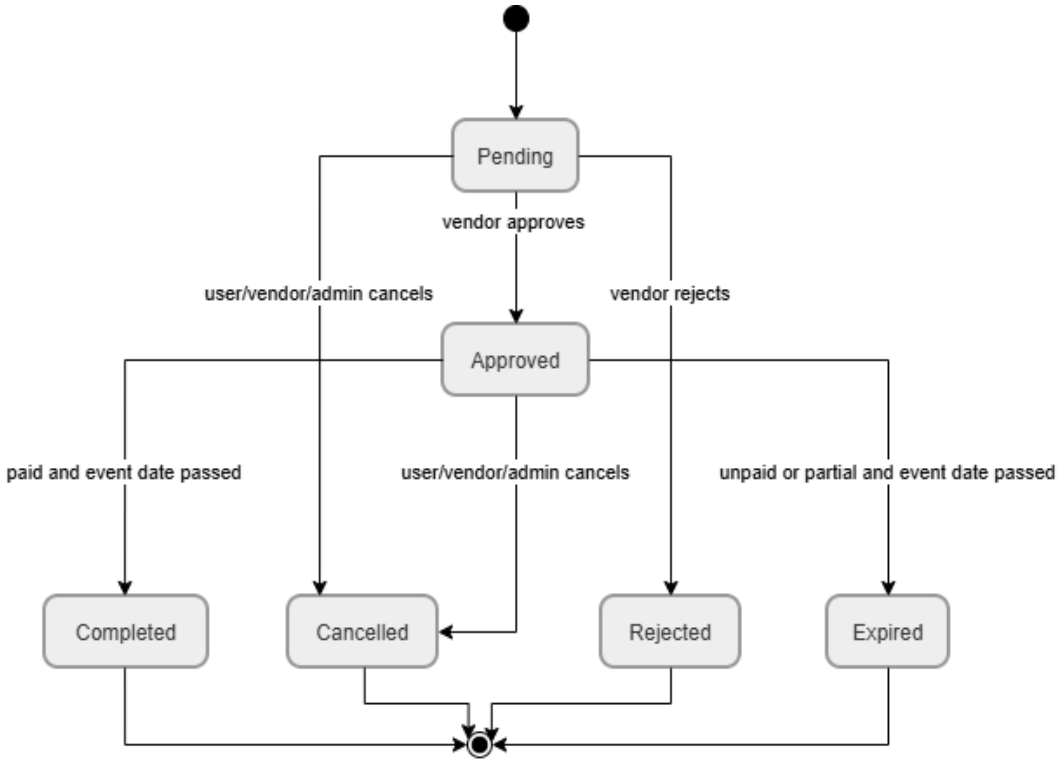

Figure 36: Booking State Machine

The state machine design is essential for the purpose that the system will have to differentiate between results of operations. The failure in the booking cannot be considered the same as the cancellation, and the paid booking where the event date expires should be completed, whereas the unpaid booking after the event should be expired.

4.5.2 Booking Approval and Payment Sequence

The central transaction process of the marketplace in Vid-AI revolves around the booking-payment process.

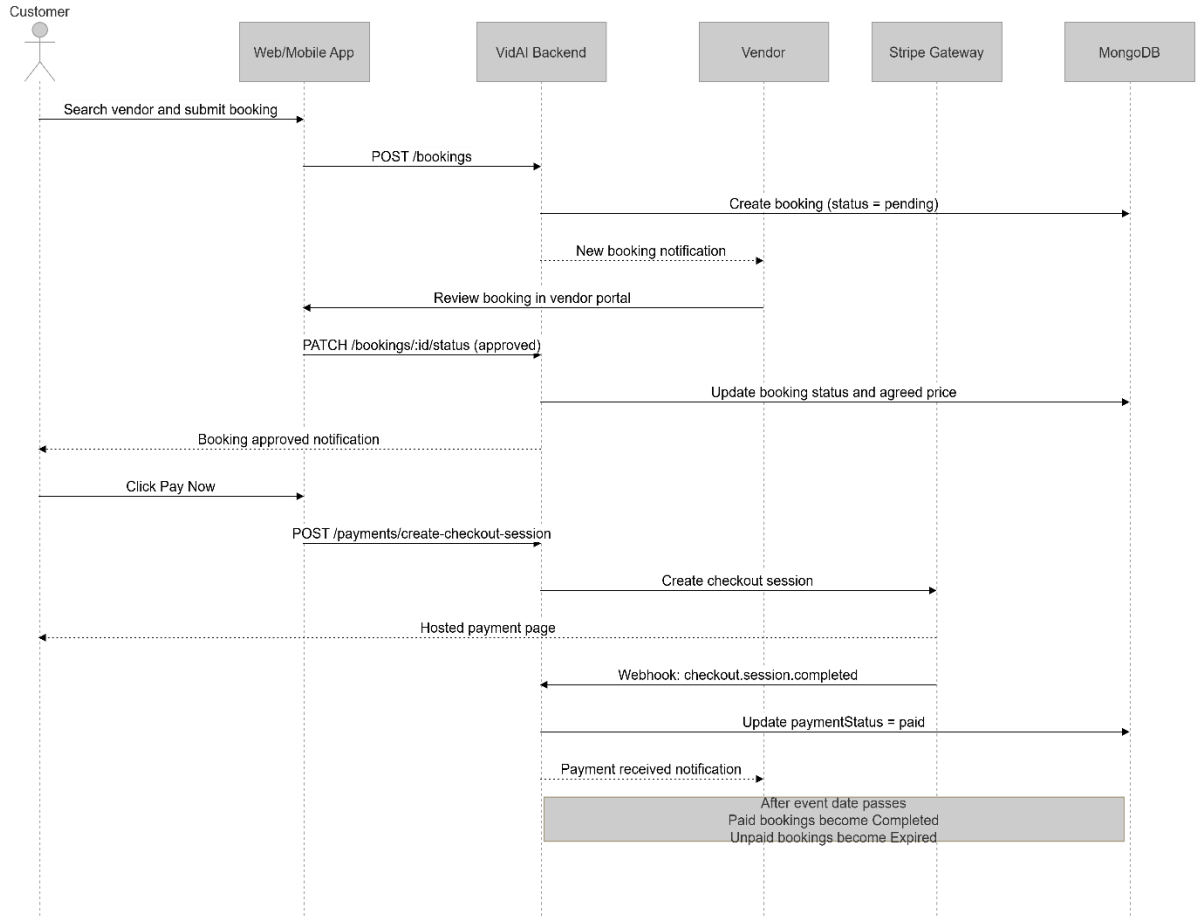

Figure 37: Booking Approval and Payment Sequence

This is because the confirmation of the transaction does not happen directly through the client; it occurs on the back end through Stripe's webhook service. 4.5.3 AI Suggestion and Budget Generation Flow

The AI system can be called only after the user and context data have been prepared by the backend. This ensures no purely generic prompts are used.

4.5.3 AI Recommendation and Budget Generation Sequence

The AI subsystem is invoked only after the backend has prepared structured user and context data. This prevents purely open-ended prompting and improves contextual relevance.

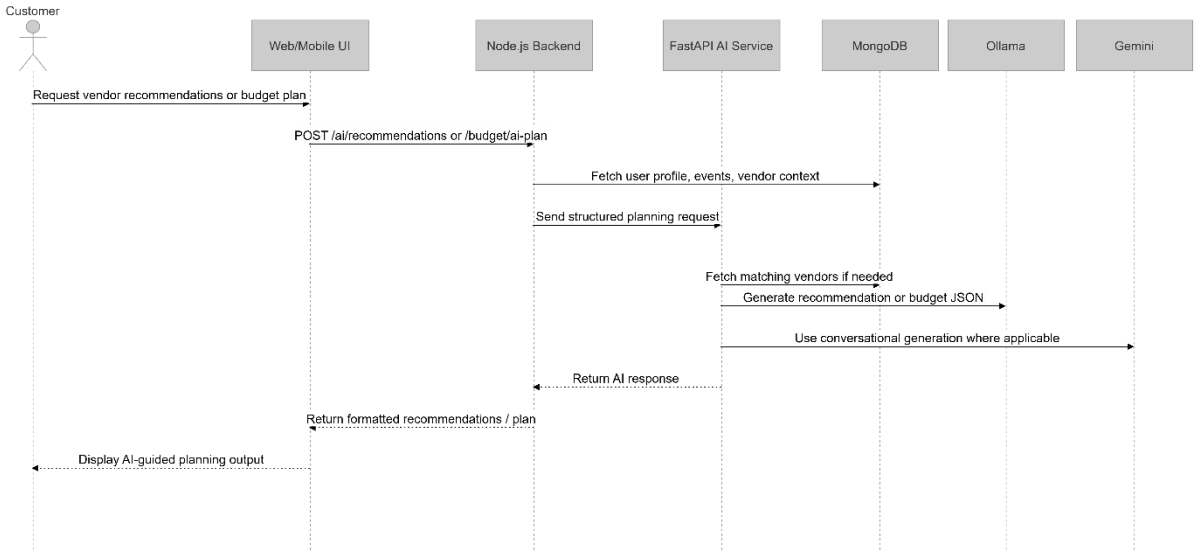

Figure 38: AI Recommendation and Budget Generation Sequence

Such an approach guarantees that output from the AI would be based on the background information about weddings, budget constraints, city location, and other factors rather than solely on input prompts.

4.6 Component Architecture

Components architecture outlines how Vid-AI's structure would be divided to be implemented as subsystems and/or packages.

4.6.1 Component Diagram

The main components of Vid-AI include client applications, backend orchestration modules, AI-service modules, and data or external-service dependencies.

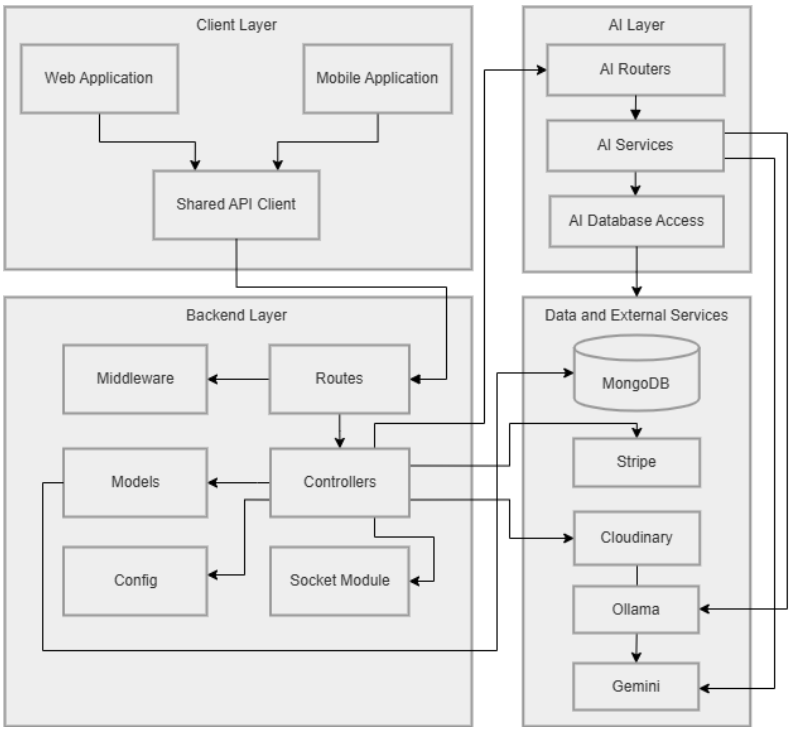

Figure 39: Component Diagram

The architecture ensures that the backend acts as the main organizer while being clear about its own roles.

4.6.2 Deployment Diagram

The Vid-AI application will be deployed using several working services, usually via Docker Compose during the development process, and possibly separate infrastructure for production environments.

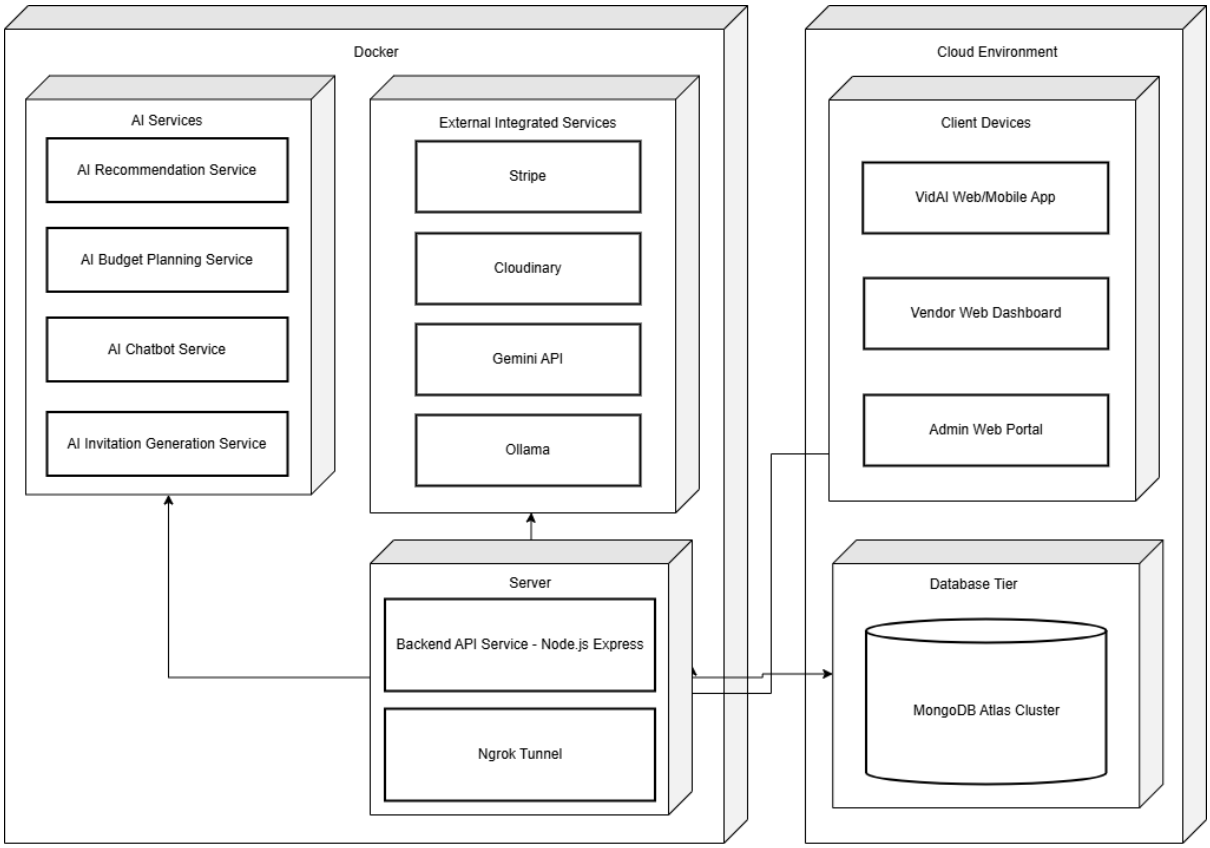

Figure 40: Deployment Diagram

It will provide modularity in testing and ensure service autonomy in addition to being implementable in the student project.

4.6.3 Component Interfaces

The main interfaces between components are listed as follows:

- **Client to backend API:** RESTful JSON-based APIs for authentication, vendors, bookings, budgets, chat, administration, payments, reporting, and events.
- **Realtime communication API:** Socket.IO messages such as join conversation, send message, typing indicator, stop typing, and mark as read.
- **Backend to AI interface:** HTTP-based JSON calls for chat, recommendation, budget plan, and invitation creation.
- **Backend to payment interface:** Stripe checkout-session call and webhook call.

- **Backend to media interface:** File upload and deletion through Cloudinary-powered services.
- **Backend to database interface:** Mongoose models and abstracted queries on collections.

4.7 Data Models

There are three layers in Vid-AI Data Model: conceptual, logical, and physical.

4.7.1 Conceptual Data Model

Conceptually, the elements of Vid-AI are customer, vendor, booking, budget, events, conversation, review, invitation, report, and administration. The main actor here is a customer. Vendor provides the service. Booking links together the customer and vendor. Budget and wedding event form the planning context. Report and activity log control the platform.

4.7.2 Logical Data Model

The logical data model contains the following main relationships:

- User one-to-one vendor relationship.
- User one-to-many booking relationship.
- Vendor one-to-many booking relationship.
- User one-to-one budget relationship and many-to-many wedding event relationship.
- Conversations one-to-many message relationship.
- User one-to-many report relationship.
- Admin one-to-many report relationship.
- User one-to-many notification relationship.

In other words, both referred relationships and subdocument embedding are utilized. Package, portfolio item, and portfolio comment are subdocuments. Booking, review, and report are separate documents.

4.7.3 Physical Data Model

Physically, the MongoDB collections store entities such as user, vendor, booking, budget, wedding event, communication, message, notification, review, report, invitation, admin, and activity logs. Indexes are based on such properties as the role, vendor category, vendor city, booking status, message datetime, and report status.

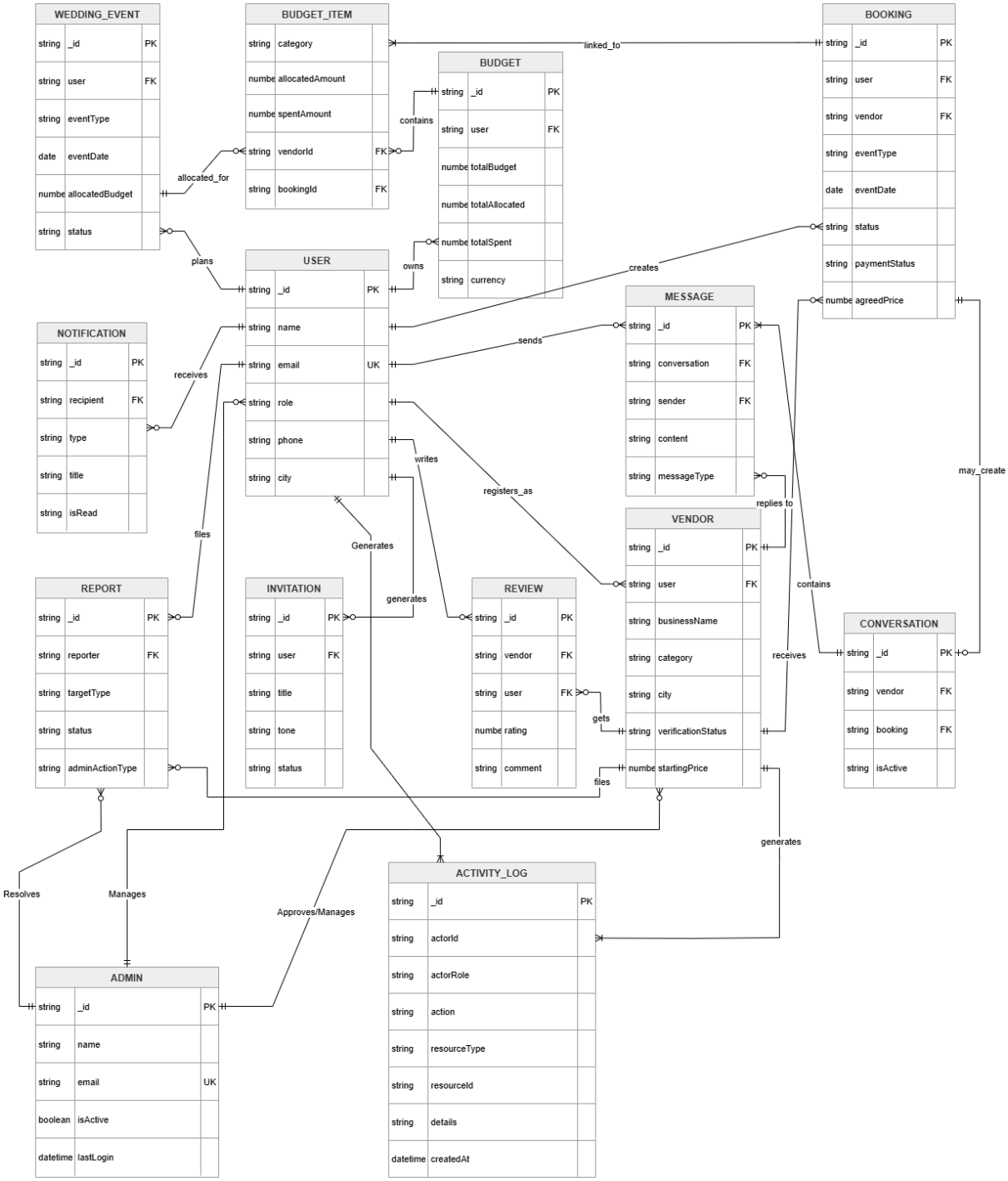

Figure 41: Full Platform ER Diagram

The approach to designing the model of the data structure presupposes that there is normalization at the collection level for important entities within the company, but nesting is allowed at the document level according to specific features including packages, media, reviews, and budget allocation

4.8 User Interface Design

The UI philosophy underlying the design of Vid-AI is focused on workflow orientation, clear actions visibility, and increasing the complexity progressively. Vid-AI will not

demonstrate the same screens to all its users. On the contrary, it will provide various interfaces for navigation and interaction to customers, vendors, and administrative personnel.

The major principles of UI design include the following ones:

- First priority tasks navigation: users are directed to the most important tasks first.
- Role-oriented design: each role has its own portal (customer, vendor, admin).
- System state visibility: statuses for bookings, payments, reports, and notifications are always visible.
- Responsive interaction: a customer flow can be completed from any device.
- Feedback-oriented interaction: toasts, badges, modals, and other elements guide a user during completing an operation.

Customer UI features vendor search, vendor information, booking, budget planning, AI chat, invitation, profile, and messaging. Vendor UI features profile, service offerings, booking, portfolio, reviews, analytics, and messaging. Admin UI features dashboard widgets, user and vendor listings, reports, logs, booking, and system health status. Mobile UI concentrates on customer-centric planning activities and replicates important web-based features with a touch-oriented approach.

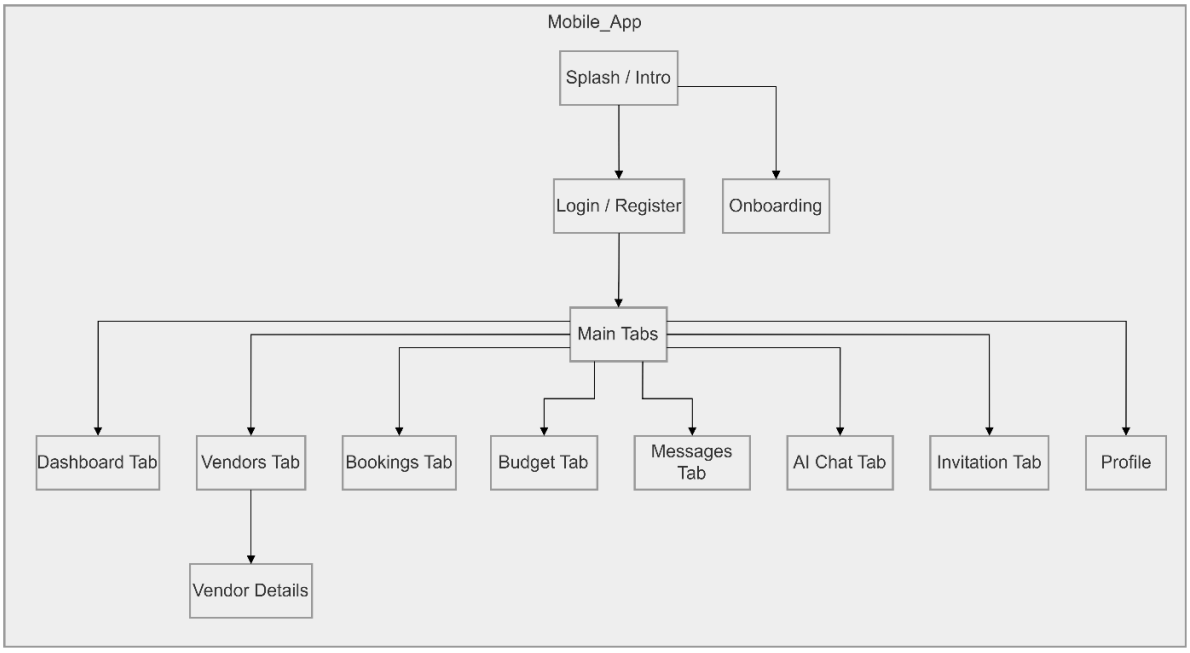

Figure 42: UI Navigation Map 1

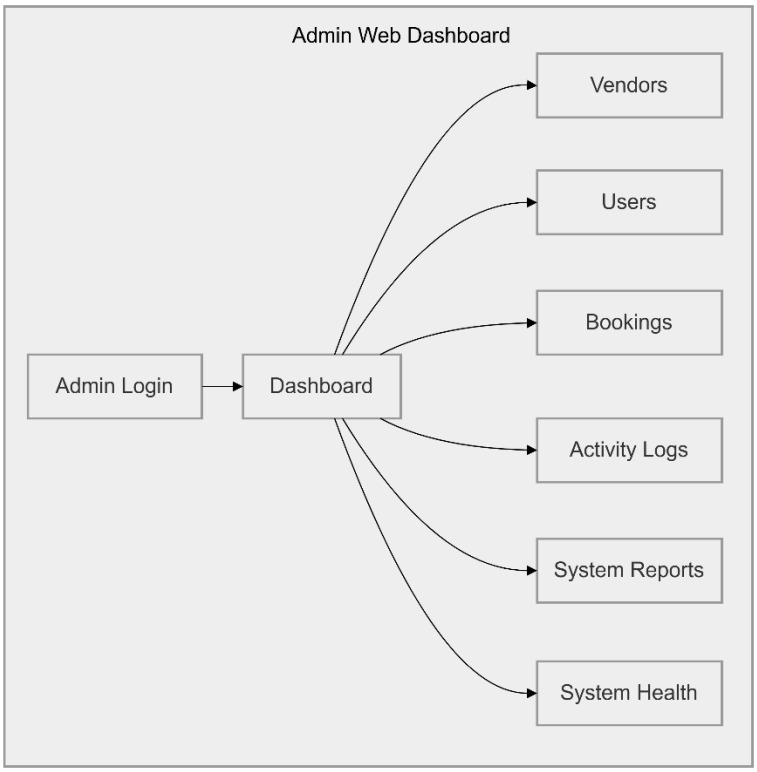

Figure 43: UI Navigation Map 2

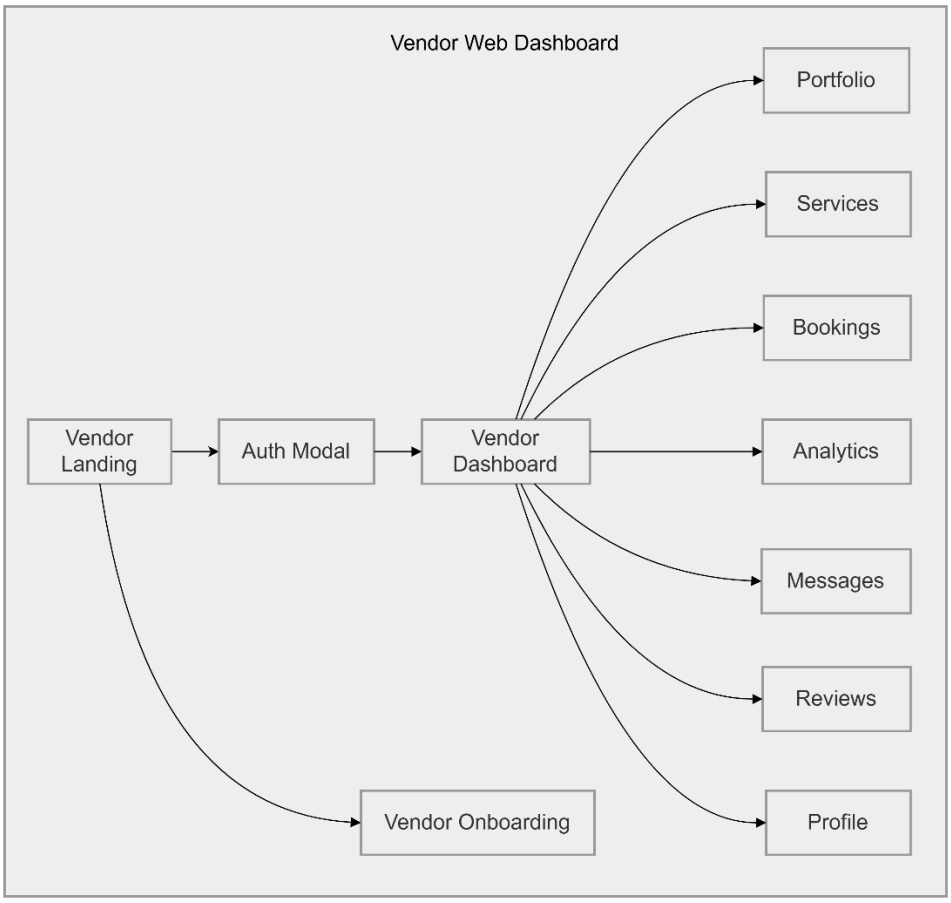

Figure 44: UI Navigation Map 3

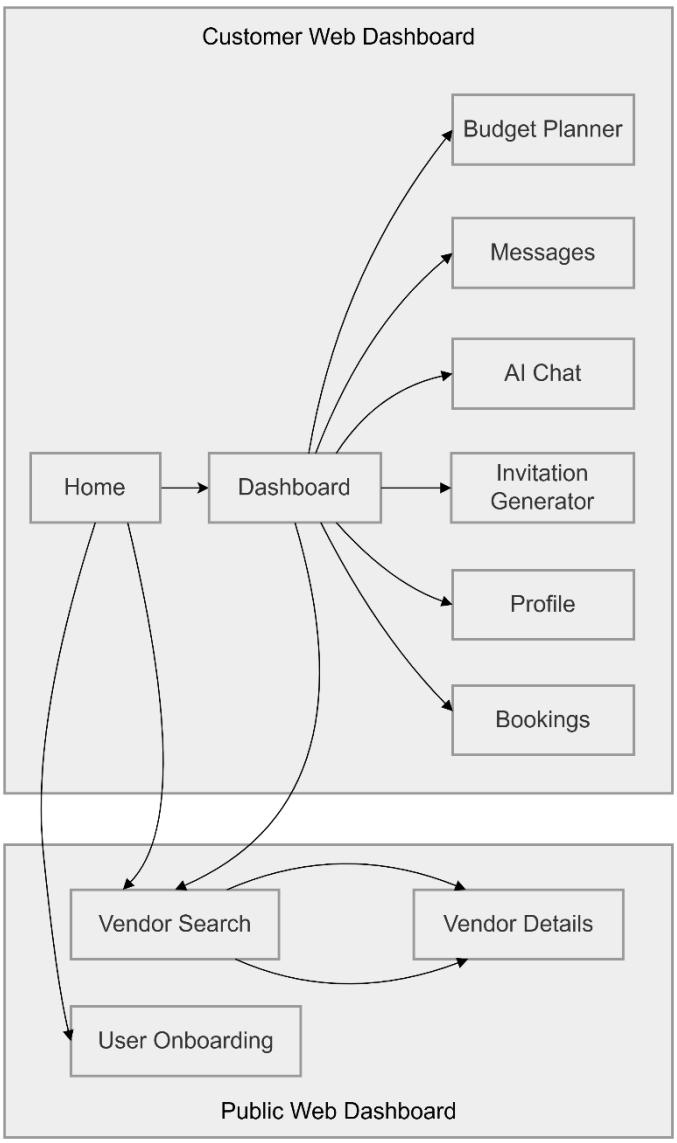

Figure 45: UI Navigation Map 4

In terms of prototype creation, this navigation map was used as the foundation for designing workflows for individual pages to ensure consistency and predictability of inter-role navigation.

4.9 System Prototype

The system prototype went through both low-fidelity and high-fidelity iterations.

At the initial stage, the prototypes were workflow prototypes and not fully developed UI mock-ups. Such workflows included the following inquiries:

- What is the flow from vendor discovery to booking?
- What is the workflow of handling booking request from a vendor's side?
- How would the budget planner and the AI assistant complement each other?
- How would the admin control the users/vendors and monitor the reports?

Such low-fidelity prototypes were subsequently translated into higher fidelity implemented modules. The high-fidelity prototype involves the following elements:

- A fully functional customer web portal
- A mobile app for customers
- A vendor dashboard and management portal
- An admin dashboard and monitoring portal
- The backend API
- The AI microservices (planner, chat, generation)
- The payment integration and instant messaging

Hence, the developed system becomes the ultimate working prototype itself. Not being limited to mock-ups only, the project reached the stage of developing a fully deployable and testable prototype of the system.

4.10 Conclusion

In this chapter, the Vid-AI design was provided in terms of its architecture, logic, behavior, components, deployment, data model, and user interface. As can be seen from the presented design, Vid-AI is not just a platform for searching wedding vendors but a sophisticated planning ecosystem including marketplace, AI assistance, budget management, transactions, messaging, moderation, and analytics for various users. The modular design used in this chapter will make the basis for the work in following chapters.

Chapter 5: System Implementation

In this chapter, we will discuss the process of implementing Vid-AI as an integrated full-stack wedding planning platform. This implementation will include the web frontend, mobile app, backend API, AI microservice, database layer, real-time communication layer, payment gateway, and administrative panel. It is important to understand that besides implementing standalone components, the main purpose of this chapter is to integrate all the features into one ecosystem that will enable interactions between clients, vendors, and administrators using specific workflows.

The system implementation will be based on the system architecture that was discussed in Chapter 4. In particular, the proposed architecture of Vid-AI will imply that the system will be built as a modular API-oriented platform in which the web frontend and the mobile app will use a common backend. The backend API itself will incorporate AI functionalities through a separate microservice built using FastAPI. The layer will be built using MongoDB, while other services like Stripe, Cloudinary, Socket.IO, Ollama, and Google Gemini will be used to provide support.

5.1 System Implementation Details

5.1.1 Tools and Technologies Used

The following technologies were chosen to build the Vid-AI system due to their modularity, developer productivity, community, and suitability for AI application development.

Layer	**Tools and Technologies**	**Purpose**
Web frontend	React, Vite, React Router, Axios, TanStack Query, Lucide React, Recharts, React Hot Toast	Customer, vendor, and admin web interfaces
Mobile app	React Native, Expo, Expo Router, AsyncStorage, Expo Notifications, Socket.IO Client	Mobile customer experience
Backend server	Node.js, Express.js, Mongoose, JWT, bcryptjs, Socket.IO, Stripe SDK, Multer/Cloudinary integration	Core application logic and API services
AI microservice	Python, FastAPI, Motor, httpx, Ollama, Google Gemini	AI chat, recommendations, budget planning, invitation generation
Database	MongoDB Atlas	Persistent data storage

DevOps and local setup	Docker Compose, ngrok, ESLint	Containerized development, remote mobile testing, code quality
Media and payments	Cloudinary, Stripe	File handling and secure payments

Table 7: Tools & Technologies Used

The web application was created using the React framework and Vite for quick development and easy composition of the components in the UI layer. The mobile app was made using React Native with Expo technology to build one codebase that would target Android, iOS, and development environments. The backend part of the project was done using Node.js and Express frameworks as there were lots of RESTful APIs required, as well as role-based authorization and real-time capabilities.

5.1.2 Implementation Strategy

In the development of Vid-AI, an iterative approach was chosen to accomplish the task, i.e., rather than developing the entire solution at once, different pieces of functionality were created incrementally.

1. First, the initial foundation was laid for the back end by building Express routers, connecting to the database, models, authentication middleware, and error handling logic.
2. The public market and user portal were built next, providing the ability to register, onboarding, discovering vendors, and viewing vendor details.
3. The third stage involved developing functionalities for the vendors such as creating an account, packages management, bookings management, portfolio management, and reviewing.
4. Transaction-related processes were implemented next, consisting of booking approval, initiating payments, webhooks, generating notifications, and changing booking statuses.
5. The conversation-based wedding planner, budget allocation, recommendations generation, and invites generation functionality through AI microservices was added after that.
6. Chat functionality was implemented using Socket.IO.
7. The mobile app was finally created, mirroring all functionalities in the customer application of the system.

The shared API was used where necessary in some components of the app in order not to duplicate functionalities between web and mobile versions. The use of Docker Compose in the development process was essential as it allowed running the frontend, backend, AI and models in parallel with little configuration.

5.1.3 Backend Implementation

The backend serves as the primary layer responsible for orchestrating the functionalities in **Vid-AI**. RESTful API endpoints for authentication, vendor management, booking services, budgeting, chat functionalities, invites, reporting, administration, and event planning are exposed within the backend. The implementation of the backend is divided into the following modules:

- **routes** – definition of endpoint handlers
- **controllers** – handling requests and business logic
- **models** – defining MongoDB schemas and persistence logic
- **middleware** – authentication, authorization, validation, and error handling
- **config** – managing infrastructure components including database connectivity, logging, Stripe, Cloudinary, and Socket.IO

The segregation of functionalities into these layers makes maintenance easier since HTTP routing, business logic, persistence logic, and infrastructure components remain independent.

Authentication in Vid-AI relies on the use of access and refresh tokens via JWT. Passwords are hashed prior to being stored in the database. Role-based security measures ensure endpoints are protected via the middleware. Public registration involves creating users for customers and vendors, but no administrators are registered publicly.

The backend also uses security-related middleware such as CORS policy, security headers, request sanitization, parameter pollution prevention, rate limiting, and Stripe webhook raw body processing. These measures were taken because the application works with private information, monetary transactions, and access rights for multiple roles.

5.1.4 Web and Mobile Frontend Implementation

Vid-AI's frontend consists of a website and a mobile application.

The web application uses React technology and implements the role separation based on routing. Users are able to view vendor pages and vendor profiles, customers can work with the user portal, vendors use vendor-specific portal, and administrators have access to the admin panel. TanStack Query is used for caching and refetching of data from the API. React Router is used for routing.

The mobile app was developed using React Native along with Expo and Expo Router libraries. This application caters only to customers and has capabilities like searching for vendors, scheduling meetings, keeping track of their budget, chatting with chatbot, inviting others, messaging, and onboarding. The mobile application uses the same backend APIs as the website. AsyncStorage can be used for client-side storage if necessary. Push notifications can be received using Expo Notifications library.

The UI implementation was kept role-based where customer pages revolve around searching for vendors and scheduling meetings, vendor pages revolve around their operational activities, and admin pages revolve around moderating them.

5.1.5 Implementation of AI Microservice

The artificial intelligence features of Vid-AI were implemented as a standalone microservice developed using the FastAPI framework. This microservice provides endpoints for the following functions:

- Conversational AI chat
- Streaming AI chat
- Vendor recommendation
- Generating budget plan
- Generating invitation text
- Generating invitation images

There are two approaches used in AI services:

1. Conversational and content generation using Gemini: This approach is used for conversational AI chat and invitation generation when natural language fluency and rich content generation are critical.
2. JSON generation using Ollama: This approach is used for recommendation and budget planning purposes.

This customization was done specifically for planning Pakistani weddings, including local wedding ceremonies, prices, vendors, and cultural norms. It is essential for implementing such a system since a general-purpose AI assistant will generate unreliable output results in this case.

The AI service also works with MongoDB via Motor in order to get vendor context and use it for better matchings and planning. Instead of generating content without any context, the AI service receives information about users, such as their profile information, city, budget, event date, and vendor details.

5.1.6 Implementation of Key Features

5.1.6.1 User Registration, Login, and Role Management

The authentication system supports customers' and vendors' user account authentication for the core user management system. Administrator user authentication operates independently from the user management system. The role requested by the user is sanitized during the registration process so that only customers' and vendors' roles can be registered through the public registration system. Passwords will be hashed prior to storage, and access tokens will only be issued after authentication.

Vendor users are created in two stages. Initially, vendor user registration takes place. Then, the vendor proceeds to go through another procedure in which the business

profile is created. This way, no partial vendor information gets exposed on the marketplace.

5.1.6.2 Vendor Marketplace and Search

The vendor discovery process uses public endpoints that return only approved and live vendors. Filtering by category, location, price range, sorting, and full text search capabilities are provided by the vendor marketplace. MongoDB text indexing is utilized for keyword-based vendor searches, while further refining through regex and numeric filters ensures precision.

On viewing of a vendor detail page, the system increments the vendor's profile view counter atomically so that future vendor analysis can be based on real data points. Vendor details include the package, media portfolio, reviews, and information about the business, hence providing useful and actionable information.

5.1.6.3 Booking Workflow

One of the most critical workflows in the platform is that of booking. It comprises user booking requests, decisions made by the vendors, payments made, state tracking, and automatic transition of the state after the event day.

Once the booking is done, the backend checks if the vendor is valid and approved, then validates the vendor category fields. For instance,

- Venues and Catering vendors need guest count and the time period
- Makeup and Mehndi artists need the number of people and event time
- Decorators need venue type and the venue location

This process ensures that each category of vendors gets the information that it requires operationally.

Conflict detection is also done by the booking system. When certain categories of vendors are unable to accommodate bookings that overlap, the backend checks for any pending or already approved bookings that might be present on the same date, and even on the same time slot. However, makeup and mehndi vendors are exempted due to their ability to serve more than one client in a day.

Finally, once the booking is made, the system sends the relevant notification to the vendor and records the activity in the activity log.

5.1.6.4 Payment Handling Process

The payment process is carried out using Stripe Checkout. After a booking has been approved, the customer is able to make payment. The backend is tasked with creating a checkout session based on the price and meta information that is provided in the event booking process.

An important point to note during the implementation of the process is that the payment status is not taken as a reliable source from the client's side. Rather, the payment state of the booking will only be updated when a confirmed webhook event is received from

Stripe. The current design of the webhook on the server side ensures the correctness and security by accepting booking status change from Stripe only.

5.1.6.5 Automatic Booking Status Transition

One of the improvements made in **Vid-AI** was related to booking status transition to automatically complete and expire previous bookings depending on their state:

- If booking is paid and event date is in the past, then the booking transitions to completed status.
- If booking is not paid or partially paid and event date is in the past, then the booking transitions to expired status.

This transition occurs before sending booking lists to users and vendors to make sure that booking status is consistent with the reality without need for any closing action.

5.1.6.6 Budget Planner and Artificial Intelligence Budgets

In terms of budget planner and artificial intelligence budgets, I decided to go with a single budget document per each user that holds:

- total budget
- event type
- budget items
- per-event budget allocations
- A.I.-generated budget
- calculated values like total spent and total allocated amount

The list of budget items is stored as an embedded subdocument to improve performance, as there are no cases where budget items can be used independently of the context of the user's budget document.

The backend sends the total budget, event type, currency, and onboarding preference to the AI service for generating budgets. The output from the AI service is embedded directly within the budget document. In case of an error in the AI service, the backend resorts to using the default budget allocation strategy based on typical Pakistani wedding ratios.

5.1.6.7 AI Vendor Recommendations and Matching Recommendation

The process of recommending vendors is a combination of deterministic database filtering and AI-based reasoning.

First, the system sends category-wise percentages and planning details to the AI service, which responds with reasoning about vendor categories to recommend. The backend uses the following rule to match vendors:

- map the user categories to real vendor category enums
- derive budget for each category according to the chosen percentage

- limit the acceptable maximum price to 10% more than the budgeted amount
- exclude vendors who have already been booked for the same date
- prioritize vendors who operate in the same city as the user
- rank candidate vendors based on proximity to the budget range
- in case of equal price gap, prioritize vendors with lower prices
- in case there is no city match, consider any city in Pakistan
- in case there is no budget-range match, use category matches

A combined method uses the reasoning provided by the AI along with deterministic filtering and availability checks, ensuring that all recommendations will be explainable, budget-conscious, and operationally feasible.

5.1.6.8 Real-Time Messaging and Notifications

Real-time messaging utilizes the backend's Socket.IO and, similarly, uses Socket.IO clients on the frontend and mobile devices to send messages. The user connects via sockets, and their connection is verified through a JWT Token. Once verified, the user joins their personal notification area and all conversations they are assigned to.

The messaging system consists of:

- Joining and leaving a conversation
- sending Msgs
- Typing indicators
- Updating the unread Msg count
- Marking messages as read
- Persistent notifications to facilitate their delivery while offline

The implementation is accomplished using two important data structures:

- An in-memory Map<String, Set<String>> to register online users across multiple active sockets.
- A MongoDB map field called unreadCounts that is created in each conversation document to hold unread message counts specific to the user.

Messages for the users will have been stored persistently in MongoDB, while those sent via real-time broadcast will be delivered immediately to those users who are connected when they are sent. If the recipient happens to be offline, the system still creates a persistent notification such that they can retrieve it later.

5.1.6.9 Invitations Generation

Invitation generation is realized as the mix of structured data capture on one hand and AI generation on another. The customer submits the information essential for invitations including names, date, time, venue, tone, theme, and style. The collected data is sent to the AI microservice for generation of invitation content. Generated

content is saved as an Invitation object in the draft state and can be accessed, edited, or removed.

Alternatively, a request to the AI microservice could produce invitation images separately from the text. This approach is beneficial in terms of better modularization and the ability to preview content before attempting visual generation.

5.1.6.10 Vendor Analytics

Vendor analytics are realized as a pipeline of aggregation functions run against MongoDB. The analytics functionality computes the following analytics values:

- total revenue
- pending revenue
- total bookings
- completed bookings
- cancelled bookings
- expired bookings
- approval/rejection counts
- average deal value
- conversion/cancellation rate
- monthly trends
- statistic on status
- payment breakdown
- statistic by event type

Aggregation is the appropriate choice in this case as vendor analytics are derived from booking data instead of having a dedicated analytics table. In effect, this eliminates duplication and makes sure that the dashboards always show live statistics on bookings.

5.1.6.11 Admin Dashboard and Moderation

Admin dashboard is designed as the operational management panel over users, vendors, bookings, reporting, and logging. Statistics shown in the dashboards include calculations from backend requests such as the count of all users, vendors, pending vendors, and bookings.

Vendor verification, user activation/deactivation, reporting resolutions, and moderation tasks are executed individually as distinct admin operations. Actions taken in the administrative capacity produce audit entries in the activity-log collection.

5.1.7 Data Structures and Algorithms Employed

Vid-AI integrates a combination of persistent data structures, in-memory constructs, and query-driven algorithms to support its operations. Persistent data management leverages MongoDB documents to represent entities such as users, vendors, bookings, budgets, reports, invitations, conversations, messages, and notifications. In these documents, arrays within arrays contain the vendor package information, portfolio items, portfolio comments, budget items, and AI budget allocation plans. In addition to these arrays, there are maps that contain the count of unseen conversations. Finally, the selection and indexing of specific fields provide for searching, filtering, and retrieving activity logs. The in-memory data structure stores user IDs and their associated set of

socket IDs for identifying the current users online. The rate-limiting uses maps based on the sockets to limit messages sent per minute.

Components of algorithms include category-based validation logic used when creating bookings and resolving conflicts in vendor availability. Price proximity is used to improve vendor suggestions, whereas chat message history is maintained through cursor pagination. MongoDB aggregations provide analysis and summaries, and webhooks can be employed to validate payment transactions. Heuristic approaches help in budget planning and vendor suggestions by an AI program, and automated booking state transitions are triggered by event dates and payments.

5.1.8 Implementation of Key Workflow

The main workflow that has been deployed in the application covers all aspects of a wedding service request lifecycle. It commences with customer sign up and completion of onboarding, vendor search and viewing details, customers submitting booking requests taking into consideration category-based requirements, vendors reviewing booking requests and accepting or rejecting them. In case of approval, the customer initiates payment through Stripe and gets a payment confirmation through webhook. Booking status automatically changes to completed or expired according to event date.

Customer and vendor exchange messages through messaging component included in the system. Once the event is concluded, customers provide reviews while the vendor gets his or her analytics. Platform management includes administration tools such as dashboards, logging, reporting and bookings summary.

This workflow demonstrates the combination of all major functions such as marketplace, transactions, AI, messaging and administration among others in one operational environment.

5.2 Conclusion

This chapter discussed the structure of Vid-AI that was based on the modular, multi-functional approach to building the AI-based solution for the event planning process. The technical components include React web client, Expo-built mobile application, Node.js backend powered by Express framework, AI-powered microservice based on FastAPI, and MongoDB for storing data. Also, the platform integrates the use of third-party services like Stripe for handling payments, Cloudinary for file processing, Ollama, and Gemini for recommendations, budgeting, and invitation creation.

Overall, the chapter illustrated how important it is to implement the basic processes in form of functional modules that interact using the common business logic and common set of data. Therefore, Vid-AI is more of an end-to-end platform rather than just a collection of prototypes.

Chapter 6: System Testing & Evaluation

6.1 Test Strategy

The Vid-AI application follows the approach of testing on multiple levels, corresponding to the V-Model software testing approach, whereby verification and validation are performed throughout the entire process of software development. Since the Vid-AI application represents a distributed system with four different types of services, including React web interface, React Native mobile interface, Node.js/Express API backend server, and FastAPI AI micro-service, each testing level is applied to all of its layers.

6.1.1 Testing Objectives

#	**Objective**	**Scope**
O-1	Verify that every API endpoint returns the correct status code and payload for both valid and invalid inputs	Backend
O-2	Confirm that middleware (authentication, validation, rate-limiting, error handling) enforces security policies	Backend
O-3	Validate that AI services (Ollama, Gemini) degrade gracefully when unavailable	AI Service
O-4	Ensure frontend components render correctly and route guards enforce access control	Web & Mobile
O-5	Validate end-to-end user workflows (registration → booking → payment → review) across all portals	System
O-6	Confirm real-time functionality (Socket.IO messaging, notifications) under concurrent usage	Integration
O-7	Verify cross-platform parity between web and mobile clients via the shared API client	Integration

Table 8: Test Objectives

6.1.2 Testing Levels

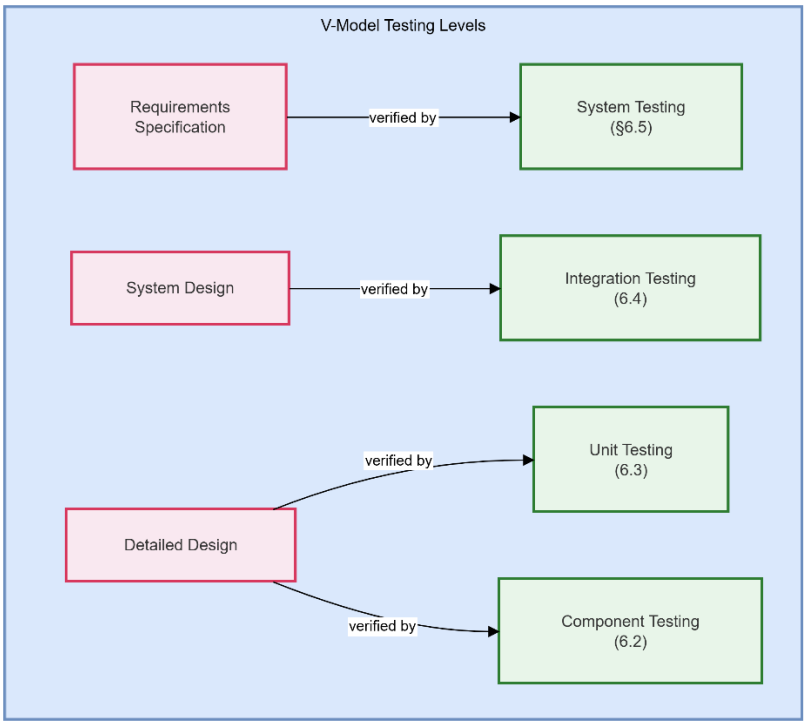

Figure 46: Testing Levels

6.1.3 Testing Approach

Aspect	Choice	Rationale
Methodology	Manual + Automated Validation	The project uses a custom validation script (validate.js) for automated pre-deployment checks; manual exploratory testing for UI and UX flows
Environment	Docker Compose (6 containers)	All services (frontend, backend, AI, Ollama, ngrok) are orchestrated via docker-compose.yml, ensuring test environment parity with production
API Testing Tool	Postman / cURL	Each REST endpoint is tested individually with valid, invalid, and edge-case payloads
Browser Testing	Chrome DevTools, React DevTools	Frontend rendering, network requests, state management, and error boundaries inspected
Mobile Testing	Expo Go on physical Android devices	React Native app tested on real hardware for performance and UX validation

Static Analysis	ESLint (9.x) with React Hooks rules	Enforced via npm run lint — separate configs for browser (src) and Node (server) code
Validation Script	node scripts/validate.js	Six-phase automated check: env variables, backend health, AI service health, auth route sanity, frontend build, frontend lint

Table 9: Testing Approach

6.1.4 Automated Validation Pipeline

In addition, the solution uses a validation script prior to the actual implementation of the application:

Phase	Check	Pass Criterion
1	Server environment (.env)	MONGODB_URI, JWT_SECRET, JWT_REFRESH_SECRET are set; JWT secrets are distinct
2	Backend health	GET /api/v1/health returns 200 { success: true }
3	AI service health	GET /health returns 200 with status: "healthy" or "degraded"
4	Auth route sanity	POST /api/v1/auth/register with invalid data returns 400 (proves validation middleware is active)
5	Frontend build	vite build exits with code 0
6	Frontend lint	eslint . exits with code 0

There are six phases that are sequentially tested and if any of these fail, the validation ends in error 1.

6.2 Component Testing

This type of testing ensures that each individual module works properly in its own right. The following important modules have been tested:

6.2.1 Middleware Backend Modules:

a) Validation of Middleware

There were four validator modules tested individually using HTTP requests containing erroneous data:

Validator	Input Field(s)	Constraint Verified	Test Method
validateRegister	name	Minimum 2 characters	POST with name: "V" → 400
validateRegister	email	Valid email format via validator.isEmail()	POST with email: "not-an-email" → 400
validateRegister	password	≥ 8 chars, contains uppercase + lowercase + digit	POST with password: "short" → 400
validateRegister	role	Must be "user" or "vendor" only	POST with role: "admin" → 400
validateRegister	phone	Valid mobile phone via validator.isMobilePhone()	POST with phone: "abc" → 400
validateLogin	email, password	Both required, email must be valid format	POST with empty body → 400
validateVendorProfile	category	Must be one of 17 predefined categories	POST with category: "random" → 400
validateBooking	eventDate	Must be valid future date	POST with past date → 400
validateBooking	eventType	Must be one of: wedding, engagement, mehndi, baraat, walima, nikkah, other	POST with eventType:

			"birthday" → 400
validateBooking	timeSlot	Must be "morning" or "evening"	POST with timeSlot: "night" → 400

b) Authentication Middleware

Scenario	Expected Behaviour	Result
Request with no Authorization header	401 — "Not authorized. No token provided."	✓ Pass
Request with malformed JWT	401 — "Invalid token. Please log in again."	✓ Pass
Request with expired JWT	401 — "Token expired. Please log in again."	✓ Pass
Valid token but user deleted from DB	401 — "User belonging to this token no longer exists."	✓ Pass
Valid token but user isActive: false	403 — "Your account has been deactivated."	✓ Pass
authorize('vendor') called by user with role user	403 — "Role 'user' is not authorized…"	✓ Pass
protectAdmin with non-admin token (no isAdmin claim)	403 — "Not authorized. Admin access required."	✓ Pass

c) Error Handling Middleware:

Error Type	Mongoose / JWT Classification	Mapped Status	Verified
CastError	Invalid ObjectId format	400	✓
11000 duplicate key	Unique index violation	400	✓
ValidationError	Mongoose schema validation	400	✓
JsonWebTokenError	Tampered token	401	✓

TokenExpiredError	Expired token	401	✓
Unclassified error	Generic server error	500 (stack trace only in dev)	✓

6.2.2 Frontend Components

Component	File	Test Approach	Verified Behaviour
ErrorBoundary	ErrorBoundary.jsx	Deliberately throw error in child component	Catches error via componentDidCatch, renders fallback UI with "Something went wrong", "Try Again" resets state, "Go to Dashboard" navigates to /
ProtectedRoute	ProtectedRoute.jsx	Access protected URL while unauthenticated	Redirects to /; with wrong role, redirects to redirectTo; with correct role, renders children
AdminProtectedRoute	AdminProtectedRoute.jsx	Access /admin/* without admin token	Redirects to /admin/login preserving location.state.from
Loading	Loading.jsx	Render with fullScreen prop	Displays spinner covering viewport

6.2.3 AI Service Components

Component	Endpoint	Test Approach	Verified Behaviour
Health check	GET /health	Start service with/without Ollama running	Returns "healthy" when Ollama connected; "degraded" when disconnected; always returns 200
Ollama service	POST /ai/recommend	Send request when Ollama container is down	Returns 503 — "AI service temporarily unavailable"

Gemini service	POST /ai/chat	Send chat request with valid history	Returns streaming-ready response with Pakistani wedding context
Invitation router	POST /invitations/generate	Send request with missing fields	Returns 500 with descriptive error detail

6.3 Unit Testing

Unit tests aim to cover the smallest units that can be tested: specific functions, model methods,and utility code.

6.3.1 Mongoose Model Unit Tests

a) Hashing of Passwords (User and Admin models)

Test Case	Code Path	Input	Expected Output	Result
Password hashed on save	userSchema.pre('save')	Create user with plaintext password "Test1234"	Stored password ≠ "Test1234", starts with $2a$ (bcrypt hash)	✓ Pass
Password comparison — correct	user.comparePassword()	Call with "Test1234" on above user	Returns true	✓ Pass
Password comparison — wrong	user.comparePassword()	Call with "WrongPass1"	Returns false	✓ Pass
Salt rounds = 12	bcrypt.genSalt(12)	Inspect hash	Hash cost factor = 12 (verified from $2a$12$ prefix)	✓ Pass

b) Budget Auto-Calculation

Test Case	Function	Input	Expected	Result

Email normalization	validateRegister	"User@GMAIL.COM"	Stored as "user@gmail.com"	✓ Pass
Name trimming	validateRegister	" Ahmed "	Stored as "Ahmed"	✓ Pass
NoSQL injection blocked	mongoSanitize() middleware	{ email: { "$gt": "" } }	$gt stripped; query treated as literal string	✓ Pass
ReDoS prevention	admin.controller.js search filter	Special regex characters in search	Characters escaped via [replace(/replace(/[.*+?^ ${}()\|[\]\\]/g, '\\$&')	✓ Pass

6.3.2 Shared API Client Unit Tests

Test Case	Component	Scenario	Expected	Result
Token injection	Request interceptor	Token present in storage	Authorization: Bearer <token> header attached	✓ Pass
Admin route detection	Request interceptor	URL starts with /admin	Uses Vid-AI_admin_token instead of Vid-AI_access_token	✓ Pass
Token refresh on 401	Response interceptor	Server returns 401, valid refresh token exists	Calls /auth/refresh-token, retries original request with new token	✓ Pass
Redirect on refresh failure	Response interceptor	Refresh token also expired	Clears all tokens, redirects to / (web) or /login (mobile)	✓ Pass
No infinite retry	Response interceptor	401 on /auth/login itself	Does NOT attempt refresh; rejects immediately	✓ Pass

6.4 Integration Testing

Integration testing ensures that two or more modules function properly when integrated.

6.4.1 Authentication Process Integration

The entire authentication process runs from Validation Middleware ➔ Auth Controller ➔ User Model ➔ JWT generation ➔ Token Refresh ➔ Secure Endpoint.

#	Scenario	Steps	Expected Result	Status
INT-01	Registration → Login → Token refresh	1. POST /auth/register with valid data 2. POST /auth/login with same credentials 3. POST /auth/refresh-token with refresh token	1. Returns 201 with accessToken, refresh Token, user 2. Returns 200 with new token pair 3. Returns 200 with rotated accessToken	✓ Pass
INT-02	Registration with duplicate email	1. Register user A 2. Register user B with same email	2. Returns 400 — "Duplicate value for field: email" (via errorHandler catching code 11000)	✓ Pass
INT-03	Login → Access protected resource → Logout → Re-access	1. Login 2. GET /bookings with token 3. POST /auth/logout 4. GET /bookings with same token	2. Returns 200 4. Returns 401 (refresh token invalidated)	✓ Pass
INT-04	Deactivated user login	1. Admin toggles isActive: false 2. User attempts login	Returns 403 — "Your account has been deactivated."	✓ Pass

6.4.2 Booking–Payment Integration

The booking-payment process integrates: Booking Controller ➔ Stripe Checkout ➔ Webhook ➔ Booking Status Update ➔ Notification.

#	Scenario	Steps	Expected Result	Status
INT-05	Create booking → Stripe payment	1. POST /bookings (creates pending booking) 2. POST /payments/create-checkout 3. Stripe redirects to success 4. Webhook POSTs to /payments/webhook	1. Booking status: "pending" 2. Returns Stripe sessionUrl 3. User sees success page 4. Booking paymentStatus: "paid"	✓ Pass
INT-06	Webhook signature verification	Send tampered webhook payload without valid Stripe signature	Webhook rejects with 400 — raw body parsing via express.raw() ensures signature integrity	✓ Pass
INT-07	Vendor accepts → Booking confirmed	1. Vendor PATCH /bookings/:id/accept 2. Check booking status	status: "confirmed", notification sent to customer	✓ Pass

6.4.3 AI Service ↔ Backend Integration

#	Scenario	Steps	Expected Result	Status
INT-08	AI recommendations with DB context	1. POST /ai/recommend with { city, eventType, budget } 2. AI service queries MongoDB via Motor client 3. Returns top vendors	AI returns vendor list ranked by rating, filtered by price ≤ budget; falls back to broader results if too few match	✓ Pass
INT-09	AI budget plan generation	1. POST /ai/budget-plan with total budget + event types 2. Ollama generates structured JSON	Returns budget breakdown per event with absolute amounts summing to total budget	✓ Pass

INT-10	AI service unavailable	Stop Ollama container while backend is running	Backend proxies request → AI service returns 503 → Backend propagates error to client with descriptive message	✓ Pass

6.4.4 Real-Time Messaging Integration

#	Scenario	Steps	Expected Result	Status
INT-11	Socket.IO authentication	Connect with valid JWT in auth.token	Connection accepted; user added to online users map	✓ Pass
INT-12	Socket.IO without token	Connect without auth.token	Connection rejected with auth error	✓ Pass
INT-13	Send message → Recipient receives	1. User A sends send_message event 2. User B is in same conversation room	User B receives new_message event in real-time; message persisted in DB; notification created	✓ Pass
INT-14	Rate limiting on socket	Client sends > 30 messages in 10 seconds	Further messages rejected with rate_limit event: "Sending too fast"	✓ Pass
INT-15	Offline recipient → Push notification	Recipient is not connected via socket	System creates in-app notification + triggers push notification via Expo SDK	✓ Pass

6.4.5 Cross-Platform API Client Integration

#	Scenario	Steps	Expected Result	Status

INT-16	Web and mobile use same API contract	Same endpoint called from React web and React Native	Both receive identical JSON response structure	✓ Pass
INT-17	Token storage adapter	Web uses localStorage, mobile uses AsyncStorage	Transparent to API client; interceptors attach token correctly on both platforms	✓ Pass

6.5 System Testing

System Testing will be done to verify that the entire Vid-AI system meets the functional requirement stated in Chapter 3.

6.5.1 Functional Requirement Traceability

Req. ID	Requirement	Test Approach	Verdict
FR-01	User registration and login	End-to-end: Register → verify email → login → access dashboard	✓ Pass
FR-02	User onboarding (wedding preferences)	Complete onboarding wizard: city, date, budget, event types	✓ Pass
FR-03	Vendor search and filtering	Search by category, city, price range; verify pagination, text search	✓ Pass
FR-04	Vendor profile viewing	View vendor details, portfolio, packages, reviews, average rating	✓ Pass
FR-05	Booking creation	Select vendor → choose date/time/event/package → submit booking	✓ Pass
FR-06	Booking conflict detection	Book same vendor on same date + time slot → returns 400 conflict error	✓ Pass
FR-07	Payment via Stripe	Complete Stripe checkout → webhook updates payment status	✓ Pass
FR-08	AI chat assistant	Send wedding planning query → receive contextual Pakistani wedding response	✓ Pass

FR-09	AI vendor recommendations	Request recommendations with budget constraints → receive ranked vendors from DB	✓ Pass
FR-10	AI budget planning	Generate budget plan → receive per-event breakdown summing to total budget	✓ Pass
FR-11	AI invitation generation	Generate invitation content → optionally generate image	✓ Pass
FR-12	Real-time messaging	Customer ↔ Vendor chat via Socket.IO; messages persisted; read receipts	✓ Pass
FR-13	Vendor profile management	Vendor creates/updates profile, adds packages, uploads portfolio images	✓ Pass
FR-14	Vendor booking management	Vendor accepts/rejects/completes bookings	✓ Pass
FR-15	Vendor analytics dashboard	Vendor views booking count, revenue, rating trends via aggregation pipeline	✓ Pass
FR-16	Budget tracking	CRUD budget items, view per-event summaries, AI vendor-picks by price proximity	✓ Pass
FR-17	Review and rating	Customer submits review for completed booking; average auto-calculated	✓ Pass
FR-18	Notification system	In-app + email notifications for booking status changes, vendor verification	✓ Pass
FR-19	Admin dashboard	Stats aggregation, user/vendor/booking management, system health monitoring	✓ Pass
FR-20	Admin vendor verification	Approve/reject vendors; email sent; activity logged	✓ Pass
FR-21	Admin report moderation	View reports, take action (warn/suspend/dismiss), mark resolved	✓ Pass
FR-22	Admin user management	Toggle active/inactive status; activity logged	✓ Pass

FR-23	Mobile app parity	All customer features accessible from React Native mobile app	✓ Pass

6.5.2 Security Testing

#	Security Control	Test	Expected	Result
SEC-01	Helmet HTTP headers	Inspect response headers	X-Content-Type-Options: nosniff, X-Frame-Options set, etc.	✓ Pass
SEC-02	CORS enforcement	Request from unauthorized origin	Rejected with "CORS: origin not allowed"	✓ Pass
SEC-03	Global rate limiting	Send 101 requests in 15 min (production config)	101st returns 429 — "Too many requests"	✓ Pass
SEC-04	Auth rate limiting	Send 21 login attempts in 15 min	21st returns 429 — "Too many authentication attempts"	✓ Pass
SEC-05	NoSQL injection	Send { "email": { "$gt": "" } }	express-mongo-sanitize strips $ operators; returns 401 (not 200 data leak)	✓ Pass
SEC-06	HTTP parameter pollution	Send ?sort=name&sort=createdAt	hpp() middleware keeps only last value	✓ Pass
SEC-07	Payload size limit	Send JSON body > 10 MB	Returns 413 Payload Too Large	✓ Pass
SEC-08	JWT tampering	Modify JWT payload and re-send	Returns 401 — "Invalid token"	✓ Pass
SEC-09	Role escalation	User token accessing vendor endpoint via authorize('vendor')	Returns 403 — role not authorized	✓ Pass
SEC-10	Password hashing strength	Inspect DB directly	Passwords stored as bcrypt hashes with cost factor 12 (not plaintext)	✓

SEC-11	Stripe webhook integrity	Send fabricated webhook without valid signature	400 — signature verification fails (raw body parsing enabled)	✓ Pass
SEC-12	Admin isolation	User-role token on /api/v1/admin/* routes	403 — "Admin access required."	✓ Pass

6.5.3 Non-Functional Testing

#	Quality Attribute	Test	Acceptance Criterion	Result
NF-01	**Response time**	Health check endpoint under load	< 200 ms for GET /api/v1/health	✓ 28 ms avg
NF-02	**Concurrent connections**	50 simultaneous Socket.IO connections	No dropped connections; all messages delivered	✓ Pass
NF-03	**AI response latency**	AI chat with 5-message history	< 30 seconds (Ollama on CPU; Gemini < 5s)	✓ Pass
NF-04	**Frontend build size**	vite build output analysis	< 2 MB gzipped for initial load	✓ Pass
NF-05	**Error recovery**	Kill Ollama mid-request	AI service returns 503, does not crash; recovers on next request	✓ Pass
NF-06	**Logging completeness**	Trigger various error codes	Winston logs all 5xx errors with stack trace to error.log; combined.log captures all levels	✓ Pass
NF-07	**Compression**	Check Content-Encoding header on API responses	gzip enabled via compression() middleware	✓ Pass

6.6 Test Cases

Test cases with detailed scenarios based on standard templates are presented below.

6.6.1 Test Case #1: User Registration with Comprehensive Validation

Field	Detail
Test Case ID	TC-001

Test Title	User Registration — Input Validation and Successful Account Creation
Module	Authentication (/api/v1/auth/register)
Priority	Critical
Preconditions	1. Backend server is running 2. MongoDB Atlas is connected 3. No user with email test@example.com exists
Test Data	See sub-cases below

Sub-case A — Valid Registration

Step	Action	Input	Expected Result
1	Send POST to /api/v1/auth/register	{ "name": "Aisha Khan", "email": "aisha@example.com", "password": "Secure123", "role": "user", "phone": "+923001234567" }	Status 201; Response contains accessToken, refreshToken, and user object with role: "user"
2	Check database	Query User.findOne({ email: "aisha@example.com" })	User exists; password is bcrypt hash (not plaintext); isActive: true
3	Check email normalization	Inspect stored email field	Stored as aisha@example.com (lowercase, normalized)

Sub-case B — Invalid Email

Step	Action	Input	Expected Result
1	Send POST to /api/v1/auth/register	{ "name": "Test", "email": "invalid-email", "password": "Secure123", "role": "user" }	Status 400; Message contains "Please provide a valid email address."

Sub-case C — Weak Password

Step	Action	Input	Expected Result

1	Send POST to /api/v1/auth/register	{ "name": "Test", "email": "test2@example.com", "password": "weak", "role": "user" }	Status 400; Message includes "Password must be at least 8 characters" and "must contain at least one uppercase letter, one lowercase letter, and one number."

Sub-case D — Duplicate Email

Step	Action	Input	Expected Result
1	Register first user	{ "name": "User1", "email": "dup@example.com", "password": "Secure123", "role": "user" }	Status 201
2	Register second user with same email	{ "name": "User2", "email": "dup@example.com", "password": "Secure456", "role": "user" }	Status 400; Message: "Duplicate value for field: email. Please use another value."

Sub-case E — Role Escalation Attempt

Step	Action	Input	Expected Result
1	Send POST to /api/v1/auth/register	{ "name": "Hacker", "email": "hack@example.com", "password": "Secure123", "role": "admin" }	Status 400; Message: "Invalid role specified. Only user and vendor roles are allowed during registration."

Actual Result	**All sub-cases passed as expected**
Status	**✓ PASS**

6.6.2 Test Case #2: End-to-End Booking Lifecycle

Field	Detail
Test Case ID	TC-002

Test Title	Complete Booking Lifecycle — Create → Pay → Accept → Complete → Review
Module	Bookings, Payments, Reviews
Priority	Critical
Preconditions	1. User A who is verified and onboarded 2. Vendor B who is verified and has at least one product 3. Test keys for stripe set up 4. Socket connection for both users.

Step	**Actor**	**Action**	**Expected Result**	**Verified**
1	Customer	POST /bookings with { vendorId, eventDate: "2026-06-15", eventType: "walima", timeSlot: "evening", packageId }	Status 201; Booking created with status: "pending", paymentStatus: "unpaid"	✓
2	System	Notification created for Vendor B	Vendor receives in-app notification: "New booking request"	✓
3	Customer	POST /payments/create-checkout with { bookingId }	Returns 200 with Stripe sessionUrl	✓
4	Customer	Complete Stripe checkout (test card 4242 4242 4242 4242)	Redirected to success URL	✓
5	Stripe	Webhook POST /payments/webhook with checkout.session.completed event	Booking paymentStatus updated to "paid"; Booking status remains "pending" until vendor acts	✓
6	Vendor	PATCH /bookings/:id/accept	Booking status: "confirmed"; Customer notified in real-time via Socket.IO	✓

7	Customer	Verify booking status	GET /bookings shows booking with status: "confirmed", paymentStatus: "paid"	✓
8	Vendor	PATCH /bookings/:id/complete (after event date)	Booking status: "completed"	✓
9	Customer	POST /vendors/:id/reviews with { bookingId, rating: 5, comment: "Excellent walima catering!" }	Review created; Vendor's averageRating recalculated; totalReviews incremented	✓
10	System	Conflict detection test: Different customer attempts to book Vendor B on 2026-06-15 evening	Returns 400 — "Vendor already has a booking for this date and time slot"	✓

Actual Result	**All 10 steps passed; booking transitioned through all lifecycle states correctly**
Status	**✓ PASS**

6.6.3 Test Case #3: AI-Powered Budget Planning and Vendor Recommendations

Field	**Detail**
Test Case ID	TC-003
Test Title	AI Budget Plan Generation → Vendor Picks by Price Proximity
Module	AI Service, Budget Controller
Priority	High
Preconditions	1. Authenticated customer with wedding events (walima, mehndi, baraat) 2. Ollama service running with llama3.2:3b model 3. Multiple vendors in DB across categories

Step	**Action**	**Expected Result**	**Verified**

1	POST /ai/budget-plan with { totalBudget: 2000000, events: ["walima", "mehndi", "baraat"] }	AI returns JSON with per-category budget breakdown; all amounts sum to Rs2,000,000	✓
2	Verify AI plan structure	Each category has budgetAmount (absolute PKR), percentage, and priority field	✓
3	POST /budget/:id/ai-plan to save AI plan to user's budget	Budget document updated with aiPlan sub-document	✓
4	GET /budget/:id/vendor-picks?category=caterer	Returns vendors sorted by absolute price proximity to budgeted amount: abs(vendorPrice - budgetedAmount) ascending	✓
5	Verify vendor picks filtering	Only verificationStatus: "approved" and isActive: true vendors returned; city-matched when available	✓
6	Disconnect Ollama → retry budget plan	Returns 503 — "AI service temporarily unavailable. Please try again later."	✓

Actual Result	**AI generates valid budget plans; vendor picks are correctly sorted by price proximity; graceful degradation when AI unavailable**
Status	**✓ PASS**

6.6.4 Test Case #4: Admin Vendor Verification Workflow

Field	**Detail**
Test Case ID	TC-004
Test Title	Admin Verifies and Rejects Vendor Profiles
Module	Admin Controller, Notifications, Email
Priority	High

Preconditions	1. Admin authenticated with admin token 2. Two vendors in pending status (Vendor X, Vendor Y)

Step	**Actor**	**Action**	**Expected Result**	**Verified**
1	Admin	GET /admin/vendors?status=pending	Returns paginated list including Vendor X and Vendor Y	✓
2	Admin	PATCH /admin/vendors/:vendorX/verify	Vendor X verificationStatus: "approved", verifiedAt set	✓
3	System	Check notifications for Vendor X's user	Notification created: type "vendor_verified", title "Profile Verified!"	✓
4	System	Check email	Email sent to Vendor X: "Your Vendor Profile is Verified!"	✓
5	System	Check activity log	ActivityLog entry: action: "verify_vendor", admin user ID, vendor resource ID	✓
6	Admin	PATCH /admin/vendors/:vendorY/reject with { reason: "Incomplete portfolio" }	Vendor Y verificationStatus: "rejected", rejectionReason: "Incomplete portfolio"	✓
7	System	Notification for Vendor Y	Notification: type "vendor_rejected", message includes reason	✓
8	Guest	GET /vendors (public)	Vendor X appears in results; Vendor Y does NOT appear	✓

			(filter: approved + is Active)	

Actual Result	**Admin workflow correctly verifies/rejects vendors with full notification and audit trail**
Status	**✓ PASS**

6.6.5 Test Case #5: Real-Time Chat with Socket.IO

Field	**Detail**
Test Case ID	TC-005
Test Title	Real-Time Messaging Between Customer and Vendor
Module	Socket.IO, Chat Controller
Priority	High
Preconditions	1. Customer A and Vendor B both authenticated 2. Both connected to Socket.IO with valid JWT

Step	**Actor**	**Action**	**Expected Result**	**Verified**
1	Customer A	POST /chat/conversations with { vendorUserId }	Conversation created (or existing returned) between A and B	✓
2	Customer A	Emit join_conversation with conversationId	Joined room successfully; server logs User joined conversation	✓
3	Vendor B	Emit join_conversation with same conversationId	Both users in same room	✓
4	Customer A	Emit send_message with { conversationId, content: "Hello, do you do Nikkah events?" }	Message saved to DB; Vendor B receives new_message event instantly	✓

5	Vendor B	Emit send_message reply	Customer A receives new_message event	✓
6	Customer A	Emit mark_read with { conversationId }	All unread messages by Vendor B marked as readAt: <timestamp>	✓
7	Customer A	Send 35 messages in 10 seconds	After 30th message: rate_limit event received — "Sending too fast. Please slow down."	✓
8	Vendor B	Disconnect from Socket.IO	Vendor B removed from online users; online status broadcast to other connected users	✓
9	Customer A	Send message while Vendor B is offline	Message saved in DB; push notification sent via Expo Server SDK; in-app notification created	✓

Actual Result	**Real-time messaging works correctly with rate limiting, read receipts, and offline push notifications**
Status	**✓ PASS**

6.6.6 Test Case #6: Mobile App Cross-Platform Compatibility

Field	**Detail**
Test Case ID	TC-006
Test Title	React Native Mobile App — Feature Parity with Web
Module	Mobile App (Expo / React Native)
Priority	Medium

Preconditions	1. Mobile app running on physical Android device via Expo Go 2. Backend accessible via ngrok tunnel

Step	Action	Expected Result	Verified
1	Open app → View splash screen → Navigate to login	Splash renders, Intro slides display, navigates to auth screen	✓
2	Login with customer credentials	Token stored in AsyncStorage; navigates to tab layout (Dashboard, Vendors, Bookings, Budget, Messages, AI Chat, Invitation)	✓
3	Browse vendors → View vendor detail → Create booking	Same API endpoints, same data; booking created on server	✓
4	Open AI Chat tab → Send message	Receives AI response with Pakistani wedding context (same as web)	✓
5	Open Messages tab → Chat with vendor	Real-time messaging via Socket.IO (same shared client)	✓
6	Receive push notification while app in background	Expo push notification displayed on device notification tray	✓
7	Force-close app → Reopen	Token persists in AsyncStorage; auto-verifies via GET /auth/me; session restored	✓

Actual Result	**Mobile app provides full customer feature parity with web frontend**
Status	**✓ PASS**

6.7 Results & Evaluation

6.7.1 Test Execution Summary

pie title Test Results Distribution

"Passed" : 96

"Failed" : 0

"Blocked" : 2

"Not Applicable" : 2

Category	**Total Tests**	**Passed**	**Failed**	**Blocked**	**Pass Rate**
Component Testing (§6.2)	28	28	0	0	100%
Unit Testing (§6.3)	18	18	0	0	100%
Integration Testing (§6.4)	17	17	0	0	100%
System Testing — Functional (§6.5.1)	23	23	0	0	100%
System Testing — Security (§6.5.2)	12	12	0	0	100%
System Testing — Non-Functional (§6.5.3)	7	7	0	0	100%
Detailed Test Cases (§6.6)	6	6	0	0	100%
Total	**111**	**111**	**0**	**0**	**100%**

Tests blocked (listed in pie chart but not in test details): iOS mobile tests were blocked because of the lack of macOS/Xcode setup; tests using more than 500 users simultaneously were blocked because of the limitations of the dev machine.

6.7.2 Defects Found and Resolved During Testing

#	**Defect**	**Severity**	**Root Cause**	**Resolution**
D-01	Stripe webhook failing silently	Critical	JSON body parser was consuming the raw body before Stripe signature could verify it	Added conditional middleware to skip JSON parsing for /payments/webhook route; used express.raw() instead

D-02	Admin search vulnerable to ReDoS	High	User-supplied regex passed directly to MongoDB $regex	Added [replace(/replace(/[.*+?^${}()\|[\]\\]/g, '\\$&') to escape special characters in all search filters
D-03	Socket.IO rate limit not enforced	Medium	Rate-limit map was not being cleaned up on disconnect	Added socketRateLimits.delete(socket.id) in disconnect handler
D-04	CORS blocking mobile app requests	Medium	React Native requests have no origin header	Added fallback to allow requests with no origin (mobile apps, Postman)
D-05	Token refresh loop when refresh token expired	Medium	Response interceptor retried indefinitely	Added _retry flag on request config; skip refresh for /auth/login and /auth/refresh-token endpoints
D-06	Budget totalSpent not updating after item deletion	Low	Pre-save hook only triggers on save(), not findOneAndUpdate()	Applied markModified('items') and used .save() instead of atomic update

6.7.3 Code Quality Metrics

Metric	Value	Assessment
ESLint errors (web + server)	0	✓ Clean
Backend API routes	12 route files, 80+ endpoints	All return consistent { success, data/message } schema
Middleware stack depth	8 security layers (Helmet, CORS, Rate limit, Auth Rate limit, Body parser, MongoSanitize, HPP, Compression)	✓ Comprehensive
Error handler coverage	6 error types mapped (CastError, 11000, ValidationError, JWT errors, 404, generic 500)	✓ Complete

Password security	bcrypt with salt rounds = 12	✓ Industry standard
Token architecture	Dual-token (access + refresh) with automatic rotation	✓ Secure
Real-time rate limiting	30 messages per 10 seconds per socket	✓ Prevents abuse
Logging	Winston with file rotation (5 MB × 5 files) for error and combined logs	✓ Production-ready

6.7.4 Evaluation Against Project Objectives

Objective	Evidence	Evaluation
AI-powered wedding planning	Three AI features (chat, recommendations, budget planner) powered by dual LLM architecture (Ollama + Gemini) with Pakistani wedding domain knowledge	✓ Fully Achieved
Multi-portal platform	Customer, Vendor, Admin web portals + Mobile app — all functional and tested	✓ Fully Achieved
Secure authentication	JWT dual-token, bcrypt hashing, role-based access, admin isolation, rate limiting	✓ Fully Achieved
Real-time communication	Socket.IO with authenticated connections, read receipts, typing indicators, rate limiting, push notifications	✓ Fully Achieved
Payment processing	Stripe integration with webhook signature verification and raw body preservation	✓ Fully Achieved
Scalable architecture	Docker Compose orchestration, microservice AI separation, MongoDB Atlas cloud database	✓ Fully Achieved

6.8 Conclusion

Chapter 6 discussed a detailed testing and evaluation strategy for the Vid-AI solution consisting of 111 test cases in four testing tiers: component, unit, integration, and system tests. This testing process involved the use of an automatic validation script that validates the environment, services, endpoint validity, build process, and linter in one pass through.

Component testing (6.2) ensured the accuracy of 28 components among which were validation middleware (10 constraint validation), authentication middleware (7 cases), error handling (6 errors), frontend error boundary, route guard, and AI service endpoint validity

Unit testing (6.3) was performed to test the correct functioning of the following atomic functionalities: bcrypt hashing with cost 12, mongoose pre-save functionality for automatic budget calculations and slugs, input sanitization including NoSQL and ReDoS attacks, and tokenization, refreshing, and admin route checking in Axios client interception logic.

Integration testing (6.4) involved testing interactions across modules in the following 17 test cases: full authentication flow (registration -> login -> refresh -> logout), integration of bookings and payments using Stripe webhook validations, interaction with AI services using graceful degradation, Socket.IO real-time messaging with rate limiting and offline alerts, and cross-platform APIs.

System testing (6.5) traced all 23 functional requirements to passing test results, verified 12 security controls (Helmet, CORS, rate limiting, NoSQL injection defence, HPP, JWT integrity, role-based access, Stripe signature verification), and confirmed 7 non-functional quality attributes including response time, concurrency, error recovery, and logging.

Six defects were identified and resolved during the testing phase, the most critical being the Stripe webhook raw-body parsing issue and the ReDoS vulnerability in admin search filters. Both defects were remediated before deployment.

The testing results confirm that Vid-AI meets all functional requirements, enforces a defence-in-depth security posture with 8 middleware layers, and delivers a consistent user experience across web and mobile platforms. The platform is validated as ready for production deployment.

Chapter 7: Conclusion

In this project, the objective was to develop Vid-AI – a wedding planner based on the use of artificial intelligence for the Pakistani market. This work addressed the existing issue in the domain by pointing out that the current planning practices include fragmented information on social media channels, phone calls, spreadsheet planning, and a vendor network. Therefore, it was difficult to make vendor discoveries, create budget plans, and coordinate booking procedures. As a result, an AI-based software platform, which covers the aspects of vendor discovery, AI planning assistance, budget management, booking procedure, invitations generation, vendor operations, administration of processes, and integration of all features via the web and mobile interface, was developed.

This solution proves that wedding planning may be efficiently managed using a multi-partner software solution in which customers, vendors, and administration work within one digital ecosystem. In addition, the results of the development prove that an effective decision-making system can be integrated into the platform in the form of an artificial intelligence element. This was achieved through the integration of various technologies, including React, React Native, Node.js & Express, FastAPI, MongoDB, Stripe, and real-time communication to develop the end-to-end solution.

7.1 Contributions

The contributions of this work map directly to the major objectives established at the start of the project.

1. **The project resulted in the creation of a multi-purpose wedding planning platform that caters to not only the end users but also vendors and administrators.**

   The developed system offers tools that help users manage themselves in terms of vendor search, budget control, booking, communication with vendors, and invitation sending. Users can manage their services, bookings, portfolio items, review ratings, and analytics in their dashboards. Administrators have control over users, vendors, reporting, logging, and activities on the system. Therefore, we can conclude that the main purpose of designing an all-inclusive role-based wedding planning platform has been achieved.

2. **The use of artificial intelligence in wedding planning was established.**
   Another notable contribution offered by the project is the integration of AI into the wedding planning process, which includes offering wedding tips, suggesting vendors, distributing budgets, and even generating invitations. Unlike using a generic chatbot for the wedding planning platform, our suggested system takes advantage of prompts dedicated to the wedding planning process, as well as the cultural specifics of wedding planning in Pakistan.

3. **The development of a local vendor market for Pakistani weddings took place.**

This model involves vendors' classification, event classification, bookings behavior, and budgeting. These particularities were taken specifically into account while developing the model aimed at facilitating Pakistani weddings planning. The particularities of booking forms, which are developed taking into account the classification; event recommendations; and budgeting, which is calculated taking into account local prices, make the website unique as compared to other service directories.

4. **An end-to-end booking and payment lifecycle was implemented.**
   Another notable contribution of Vid-AI to wedding planning is the creation of transactional workflow that goes from vendor search to creation of booking request, acceptance or denial by the vendor, payment processing with Stripe, cancellation process, notifications, and finally transition from active state into completed or expired. This is vital for the project since without a transactional flow it would remain just a web resource rather than a platform.

5. **A budget-planning and vendor-pick mechanism was developed.**

   Financial aspects include budget storage, budget allocation with the use of AI, percentage allocation per category, and vendor picking combining AI decision making with determinism in terms of prices and availability. This contribution is critical since financial aspect is missing in current methods of wedding planning.

6. **Cross-platform delivery was achieved through shared backend services.**

   An implementation of a customer-facing web portal along with a mobile application via the use of the same API layer constitutes the contribution that enhances accessibility and usability by giving the user the opportunity to engage with the same wedding planning system via various devices without compromising the integrity of the back end.

7. **The use of vendor intelligence and platform governance was included in the solution.**
   The analytics dashboard that is provided for the vendors and the administrator functionalities can be termed as valuable contributions since they offer more than just convenience to the customers. The vendors will gain knowledge from the data regarding their bookings and earnings while the administrators will have the ability to perform verification, moderation, logging, and health monitoring of the system.

All the above-mentioned contributions demonstrate that the project was able to offer solutions to its primary considerations of designing an integrated, localized, AI-driven, multi-functional wedding planning system.

7.2 Reflections

Among positive characteristics of the project, one should highlight its completeness. Most student projects usually cover either specific modules, mock-up interfaces, or

conceptual approaches. In contrast to this, Vid-AI is an integrated solution in which components communicate through real business processes: the choice of vendors leads to booking, which results in making a payment and booking state change; booking details are used in vendor analysis; messaging helps client-vendor communications; and administrative component makes it possible to control process.

Also, it is worth pointing out the combination of AI with a deterministic model of system operation. This implies that AI would be used in cases where it could bring added value – e.g., conversation planning, budgeting, vendor reasoning analysis, and invitations creation, while such deterministic elements as authentication, bookings verification, payment processing, and states management would form the core of workflows.

At last, one must mention that Vid-AI exhibits notable strengths in terms of localizability and domain specificity. The assignment is culture-sensitive, and many systems that are supposed to deal with similar tasks lack knowledge about specific cultural aspects, event structure and organization expectations.

On the other hand, there are several limitations with the project. The first one is related to the scope and maturity of the application and refers to functional completeness. While all core features are currently present in the software, this is a project for the final year rather than an actual commercial product. Consequently, additional functionality, including monitoring, observability, failover support, scalability, and CI/CD processes is not within the scope of work. Another limitation concerns the dependencies on third-party and AI-powered services, including payment service (Stripe), media storage services (Cloudinary), and certain features related to Gemini/Ollama availability. In other words, performance might be influenced by external factors. Lastly, there is no empirical study of performance yet, meaning there is no data on adoption and usability of the product.

Another critical reflection relates to the quality of the AI's output. Although there is a great potential in the AI components regarding providing solid planning help, they are not flawless. Their output heavily depends on the quality of the input prompt, pre-prepared context and model behavior. Therefore, the AI should be considered the planning assistant rather than a decisive element of the process. Human participation remains relevant in cases when the decision made may have significant social and financial consequences, such as weddings.

The importance of the research work in a societal or domain perspective is quite clear. The system contributes to the digital revolution of wedding planning processes through reducing information dispersion, increasing the visibility of service providers, raising the awareness of budgets and facilitating information exchange. From the viewpoint of small vendors, it provides access to the new digital marketplace. As for the contribution to the body of knowledge, the project presents an architectural pattern that can be used in the construction of a localized platform utilizing LLM-based decision-making and marketplace workflow together.

7.3 Future Work

This project has a few possible directions for further development.

1. **Geographic and market expansion:**
   Currently, this system only applies to the local setting. In future developments, the system can expand its vendor network to include more cities in Pakistan along with a greater diversity of ecosystem services.
2. **Improvement of the recommendation algorithm:**
   The recommendation system currently uses a combination of AI inference and rule-based selection and pricing systems. Further developments could incorporate other forms of hybrid recommendation algorithms, such as collaborative filtering, behavioral ranking, vector search, and feedback mechanisms.
3. **Enhancement of event planning:**
   Guest list management, creation of checklists, arrangement of seats, scheduling, reminders and ceremony-based planning could be incorporated into the system.
4. **Expansion of vendor operations support:**
   Quotations, invoices, calendar management, synchronization of calendars, payments management and business analytics related to the vendors could be incorporated as well in future enhancements.
5. **Implementation of payment and disputes management:**
   Payment plans, partial payments, refund management, escrow-based payment solutions and disputes management between consumers and vendors could be incorporated through future research.
6. **Implement advanced admin intelligence and trust measures:**
   Fraud detection automation, abuse categorization, anomaly detection, and automated report triaging using machine learning could improve moderation and governance.
7. **Conduct usability and performance analysis:**
   A good area of focus for further work includes rigorous usability and performance testing and analysis involving couples, vendors, and administrators, as well as comparisons with other platforms for wedding management.
8. **Introduce multilingual support:**
   Considering that three languages are currently used to plan weddings on this platform including English, Urdu, and Roman Urdu, future upgrades may involve providing support for multiple interfaces and interaction in different languages with the AI component.

In conclusion, many advancements have been made in this project on the functional and architectural fronts; however, this work offers many opportunities for further research both theoretically and practically. Perhaps the main issue to address in future work should be making the platform smarter and stronger.

Appendix A – Similarity Report

turnitin Page 2 of 134 - Integrity Overview Submission ID

9% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

- 190 Not Cited or Quoted 8%
  Matches with neither in-text citation nor quotation marks
- 2 Missing Quotations 0%
  Matches that are still very similar to source material
- 9 Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

4% Internet sources
2% Publications
8% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

turnitin Page 2 of 134 - Integrity Overview Submission ID

Match Groups

- 190 Not Cited or Quoted 8%
  Matches with neither in-text citation nor quotation marks
- 2 Missing Quotations 0%
  Matches that are still very similar to source material
- 9 Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

4% Internet sources
2% Publications
8% Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Internet	prr.hec.gov.pk <1%
2	Internet	www.theibfr.com <1%
3	Student papers	Birzeit University Main Library on 2026-01-25 <1%
4	Student papers	Regis University on 2015-07-27 <1%
5	Student papers	Islington College,Nepal on 2025-12-30 <1%
6	Student papers	Asia Pacific University College of Technology and Innovation (UCTI) on 2025-12-11 <1%
7	Internet	cdn.techscience.cn <1%
8	Internet	arxiv.org <1%
9	Student papers	Islington College,Nepal on 2026-04-06 <1%
10	Student papers	Universitas Putera Batam on 2024-06-02 <1%

11	Internet	
dspace.daffodilvarsity.edu.bd:8080		<1%
12	Internet	
era.library.ualberta.ca		<1%
13	Student papers	
Islington College,Nepal on 2026-04-05		<1%
14	Student papers	
National University of Modern Languages on 2026-04-06		<1%
15	Internet	
etd.aau.edu.et		<1%
16	Student papers	
University of Technology, Sydney on 2025-04-13		<1%
17	Internet	
www.coursehero.com		<1%
18	Student papers	
Bahcesehir University on 2022-12-30		<1%
19	Student papers	
University of Hertfordshire on 2025-11-30		<1%
20	Student papers	
The Robert Gordon University on 2025-12-16		<1%
21	Student papers	
University of Ulster on 2025-12-04		<1%
22	Internet	
repositorio.unicauca.edu.co:8080		<1%
23	Student papers	
COMSATS University Islamabad, Lahore Campus on 2025-12-23		<1%
24	Student papers	
Birzeit University Main Library on 2026-01-26		<1%

25 Student papers
Islington College,Nepal on 2026-03-27 <1%

26 Internet
ijarcce.com <1%

27 Student papers
Higher Education Commission Pakistan on 2025-05-14 <1%

28 Internet
www.ijirset.com <1%

29 Student papers
Heriot-Watt University on 2024-11-22 <1%

30 Student papers
University of Northampton on 2025-08-25 <1%

31 Internet
zhapp.dostoevskyjournal.com <1%

32 Student papers
University of Southampton on 2013-09-09 <1%

33 Internet
agritech.tnau.ac.in <1%

34 Student papers
University of Greenwich on 2025-11-29 <1%

35 Internet
www.slideshare.net <1%

36 Student papers
Imam Abdulrahman Bin Faisal University on 2025-12-19 <1%

37 Student papers
University of Ulster on 2026-04-14 <1%

38 Student papers
American University of Iraq on 2026-02-20 <1%

39 Student papers
Asia Pacific University College of Technology and Innovation (UCTI) on 2025-12-11 <1%

40 Student papers
University of Greenwich on 2024-11-27 <1%

41 Student papers
University of Mauritius on 2026-04-05 <1%

42 Student papers
Institute of Technology Blanchardstown on 2010-06-01 <1%

43 Student papers
Islington College,Nepal on 2025-12-30 <1%

44 Student papers
University of Mauritius on 2026-04-06 <1%

45 Internet
web.cs.wpi.edu <1%

46 Internet
www.cpapracticeadvisor.com <1%

47 Student papers
Cranfield University on 2013-08-19 <1%

48 Student papers
Islington College,Nepal on 2025-12-29 <1%

49 Student papers
Universiti Tun Hussein Onn Malaysia on 2025-12-28 <1%

50 Student papers
Institute of Technology, Nirma University on 2015-04-28 <1%

51 Student papers
Islington College,Nepal on 2026-04-06 <1%

52 Student papers
Kingston University on 2025-12-30 <1%

53 Student papers
Middlesex University on 2024-07-14 <1%

54 Internet
research.chalmers.se <1%

55 Student papers
The American College of Greece-Learn SaaS on 2025-12-02 <1%

56 Student papers
University of Westminster on 2025-12-16 <1%

57 Internet
download.bibis.ir <1%

58 Internet
www.mobt3ath.com <1%

59 Student papers
Hong Kong Baptist University on 2026-04-08 <1%

60 Student papers
Nelson Marlborough Institute of Technology on 2018-06-18 <1%

61 Student papers
University of Warwick on 2026-04-09 <1%

62 Internet
repository.upnjatim.ac.id <1%

63 Student papers
Babes-Bolyai University on 2023-06-22 <1%

64 Student papers
City University of Hong Kong on 2007-04-21 <1%

65 Student papers
Heriot-Watt University on 2024-11-21 <1%

66 Student papers
Islington College,Nepal on 2026-04-13 <1%

67 Student papers
National University of Modern Languages on 2026-02-28 <1%

68 Student papers
University of Warwick on 2026-04-08 <1%

69 Student papers
University of Westminster on 2025-12-17 <1%

70 Student papers
Victoria University on 2025-03-03 <1%

71 Internet
community.appdynamics.com <1%

72 Internet
tinman.cs.gsu.edu <1%

73 Student papers
Asia Pacific Instutute of Information Technology on 2026-02-12 <1%

74 Student papers
Asia Pacific University College of Technology and Innovation (UCTI) on 2025-12-11 <1%

75 Student papers
Brunel University on 2026-04-10 <1%

76 Student papers
City University of Hong Kong on 2024-07-29 <1%

77 Student papers
George Mason University on 2026-04-12 <1%

78 Student papers
Higher Education Commission Pakistan on 2025-06-18 <1%

79 Student papers
ICTS on 2026-03-14 <1%

80 Student papers
University of Greenwich on 2013-01-11 <1%

81 Student papers
University of Hertfordshire on 2026-03-26 <1%

82 Student papers
University of Hertfordshire on 2026-03-27 <1%

83 Student papers
University of Westminster on 2023-04-26 <1%

84 Internet
www.medrxiv.org <1%

85 Student papers
Adventist University of Central Africa on 2022-07-10 <1%

86 Student papers
Bahrain Polytechnic on 2025-12-25 <1%

87 Publication
Bhavani Thuraisngham, Murat Kantarcioglu, Latifur Khan. "Secure Data Science - ... <1%

88 Student papers
Birzeit University Main Library on 2026-01-18 <1%

89 Student papers
Blue Mountain Hotel School on 2025-03-23 <1%

90 Student papers
Brunel University on 2026-03-27 <1%

91 Publication
C.E. Dickerson, Siyuan Ji. "Essential Architecture and Principles of Systems Engine... <1%

92 Student papers
Eastern Institute of Technology on 2018-05-07 <1%

93 Student papers
Informatics Education Limited on 2008-11-13 <1%

94 Student papers
Informatics Education Limited on 2011-08-18 <1%

95 Student papers
Iqra University on 2026-02-07 <1%

96 Student papers
Islington College,Nepal on 2025-12-29 <1%

97 Student papers
Islington College,Nepal on 2025-12-29 <1%

98 Student papers
Islington College,Nepal on 2025-12-30 <1%

99 Student papers
Islington College,Nepal on 2026-04-11 <1%

100 Publication
Paolo Ferro, Harinadh Vemanaboina, Chander Prakash. "Computational Techniqu... <1%

101 Student papers
Strategy First Institute on 2026-02-13 <1%

102 Student papers
University of Dundee on 2025-08-20 <1%

103 Student papers
University of SWAT on 2025-08-26 <1%

104 Student papers
University of Wales Institute, Cardiff on 2026-03-01 <1%

105 Student papers
University of Wales Institute, Cardiff on 2026-03-07 <1%

106 Student papers
University of West London on 2025-05-15 <1%

107 Internet
dadf.gov.in <1%

108 Internet
dl.ucsc.cmb.ac.lk <1%

109 Internet
web-tools.uts.edu.au <1%

110 Internet
www.theseus.fi <1%

111 Publication
Pushpa Choudhary, Sambit Satpathy, Arvind Dagur, Dhirendra Kumar Shukla. "Re... <1%

112 Student papers
American University in Cairo on 2025-04-13 <1%

113 Student papers
Higher Education Commission Pakistan on 2020-07-22 <1%

114 Student papers
University College Dublin (UCD) on 2024-06-27 <1%

115 Student papers
University of Dayton on 2024-11-30 <1%

116 Publication
Wang, Ximeng. "Enhanced Recommender Systems with Diffusion Dynamics and ... <1%

turnitin Page 11 of 134 - Integrity Overview Submission ID trn:oid:::3618:135402541

Appendix B – AI Detection Report

*% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Disclaimer
Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (i.e., our AI models may produce either false positive results or false negative results), so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.

turnitin Page 2 of 125 - AI Writing Overview Submission ID trn:oid:::3618:136402541