**P-10 Odysseum**

**SPROJ Report**



**Muhammad Affan Naved (25100283)**

**Mohammad Haroon Khawaja (25100225)**

**Shahrez Aezad (25100212)**

**Pir M. Shahraiz Chishty (25100097)**

**Luqman Aadil (25100023)**

**Advisor: Waqar Ahmed  
School of Science and Engineering  
Lahore University of Management Sciences  
Submission Date**

**Acknowledgement and Dedication**

**Certificate**

I certify that the senior project titled “**Add project title here**” was completed under my supervision by the following students:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

and the project deliverables meet the requirements of the program.

------------------------------------- Date:

**Advisor (Signature)**

------------------------------------- Date:

**Co-advisor (if any)**

# Table of Contents

[Table of Contents](#)

[List of Figures](#)

[1. Introduction](#)

[a. Introduction](#)

[b. Objective and Scope](#)

[c. Development Methodology](#)

[d. Contributions](#)

[2. System Requirements](#)

[a. System Actors](#)

[b. Functional Requirements](#)

[c. Non-functional Requirements](#)

[3. System Architecture](#)

[a. Architecture Diagram](#_3osu65v3tiph)

[b. Architecture Description](#_98dnk7u1nu7)

[c. Justification of the Architecture](#_baxqz77atix1)

[d. Tools and Technologies](#)

[4. Requirements Specifications](#)

[a. Use Cases](#)

[b. Class Diagram](#_eifrbxyb240h)

[c. Users](#_gobvxw65ja4)

[c.1. Tourists](#_ykfgvr73ojhd)

[c.2. Admin](#_nauqpb5738id)

[c.3. Business Owners](#_htkxq86q4wds)

[d. Itinerary](#_jqmu20lvi399)

[e. Bookmark](#_rusmdn42lwiw)

[f. Chat](#_j1rbtqbo0p8q)

[f.1. Direct](#_yizo96kmdi61)

[f.2. Group](#_2mvjy4sjoiu)

[g. Posts](#_v3hrlr1w5wvn)

[h. Comments](#_n7hsp422v4q6)

[i. Location](#_t2i116jvb49m)

[j. Review](#_vdyni6jd00rw)

[k. Business](#_8k86mugmq7yi)

[l. Sequence Diagrams](#_uala3a2rko2b)

[1. Login](#_5vtydo3bthdb)

[2. Register](#_9dwpd4dntwj)

[3. Make Itinerary](#_pkpidndcq7zf)

[4. Post Moderation](#_sqrd2ytn6lhy)

[5. Create Posts](#_s03gamp41zes)

[6. Posting a Review](#_qsg18w9qmlng)

[7. Report Content](#_sstzfpk5j3uf)

[8. Bookmark Locations](#_axbyf1nisv)

[9. AI-Based Travel Suggestions + Reviews](#_kzqbecnh1afj)

[10. Follow Users](#_reirdiw0cwcb)

[5. Software Development Methodology and Plan](#)

[a. Software Process Selection](#_q4ypwdksq0f2)

[b. Gantt Chart](#_8iiq09cajm5e)

[6. Database Design and Web Services](#)

[a. Database Design](#)

[b. API Specification](#)

[7. System User Interface](#)

[8. Project Security](#)

[a. Project Threats](#)

[b. Potential Losses](#)

[c. Security Controls](#)

[d. Static and Dynamic Security Scanning Tools](#)

[9. Risk Management](#)

[Potential Risks and Mitigation Strategies](#_7zggx2gqw1ca)

[10. Testing and Evaluation](#)

[11. Deployment Guidelines](#)

[12. Conclusion](#)

[a. Summary](#)

[b. Challenges](#)

[c. Future](#)

[13. Review checklist](#_5ipa08ni553w)

[14. References](#)

# List of Figures

# Introduction

## Introduction

Odysseum is a mobile-first travel companion designed to revolutionize the way travelers explore Pakistan. Positioned at the intersection of tourism, social networking, and artificial intelligence, Odysseum offers users a streamlined, engaging platform to plan, discover, and experience journeys more meaningfully. Whether it’s uncovering hidden gems, organizing travel itineraries, or connecting with fellow adventurers and local businesses, Odysseum empowers users to turn every trip into a curated, memorable experience.

The domain of Odysseum spans tourism and local business promotion, bringing digital innovation to Pakistan’s rapidly growing travel industry. Our target users are both domestic and international tourists looking for an intuitive tool to plan their travels, as well as local businesses seeking greater visibility and customer engagement. Additionally, platform administrators and service providers form an important user segment for moderation, management, and service facilitation.

The primary deliverable is a **cross-platform mobile application** developed in React Native, coupled with a robust backend infrastructure utilizing Node.js, MongoDB, Firebase, and Python Flask microservices. With features such as AI-generated itineraries, real-time social interactions, review summaries, and curated location discovery, Odysseum transforms conventional travel apps into an intelligent, user-centric travel ecosystem.

## Objective and Scope

The objective of Odysseum is to bridge the gap between travelers and authentic local experiences through a smart, integrated platform. We aim to offer users personalized itineraries, real-time social engagement, trusted reviews, and seamless interaction with businesses, all within a single mobile app.

We chose to develop Odysseum to address key pain points observed in traditional travel planning — fragmented information, lack of personalization, and limited local engagement. By integrating artificial intelligence with user-driven content, we sought to empower travelers with smarter choices and deeper connections to the places they visit.

The scope of Odysseum extends beyond mere itinerary planning. Users can explore curated destinations, review and bookmark businesses, chat with fellow travelers, and receive dynamic, AI-based suggestions — making the platform a holistic digital travel companion. For businesses, it provides a direct channel to market their services to travelers actively planning or experiencing a trip.

Ultimately, Odysseum enhances business operations in the tourism domain by increasing visibility for local businesses, fostering community-driven content, and improving the overall quality of tourist experiences through personalized digital solutions.

## Development Methodology

To ensure a flexible and user-centric development process, we adopted the **Agile (Scrum)** methodology. Recognizing that user needs and feature priorities could evolve throughout the project lifecycle, Agile allowed us to maintain an iterative workflow, deliver components in manageable sprints, and integrate continuous feedback into development.

Each sprint focused on delivering key functionalities — starting with core features like user registration, location exploration, and itinerary creation, and gradually moving towards advanced capabilities like AI-generated travel suggestions, real-time chat, and dynamic business listings.

Regular sprint reviews, and backlog refinements enabled us to identify bottlenecks early, adapt to changing requirements, and maintain momentum without sacrificing code quality or project objectives. This iterative, feedback-driven approach was crucial in balancing innovation with stability, particularly when integrating AI services and real-time features.

## Contributions

**AI-Powered Personalization:** Unlike static itinerary builders, Odysseum uses a lightweight LLM-based backend to generate tailored travel plans based on user preferences, ensuring highly customized and relevant experiences.

**Integrated Social Features:** Users can not only create and share itineraries but also engage socially through posts, comments, chats, and bookmarks — building a vibrant traveler community.

**Business Empowerment:** Local businesses have the tools to register, promote services, interact with users, and manage their reputation directly through the app, helping boost local tourism economies. Moreover, users can explore these services and make bookings and reservations within the app.

**Seamless Media Management:** By leveraging both MongoDB and Firebase, Odysseum handles structured data and rich media assets efficiently without compromising speed or scalability.

**Real-Time Notifications and Updates:** Socket-based server components enable dynamic communication, allowing users to stay updated on friend activities, bookings, or location trends in real-time

# System Requirements

This chapter highlights system requirements, its primary users and their requirements to use the app.

## System Actors

| **Actor Name** | **Description** |
| --- | --- |
| Tourist | The primary user of the application who can create itineraries , explore places, and connect with other travelers and local service providers. They can view recommendations, rate locations, leave reviews, create itineraries and share their travel experiences with the community. |
| Business Owner | Businesses such as hotels, restaurants, shopping centers etc. They use the app to promote their services and engage with travelers by providing information, answering inquiries, and receiving feedback. |
| Administrator | Responsible for managing and maintaining the platform. This actor oversees user management, content moderation, and system performance to ensure smooth operation. |

## Functional Requirements

| **Requirements** | |
| --- | --- |
| **Sr#** | **Requirement** |
|  | **Tourist** |
| 1 | As an tourist, I want to be able to create and login to an account using my email |
| 2 | As a tourist,I should be able to edit my profile and add information such as a bio, profile picture and traveling interests. |
| 3 | As a tourist, I should be able to search for different destinations and local businesses, pin them, view their ratings and/or add my own review. |
| 4 | As a tourist, I want to create my own travel itinerary and want to be able to share them with others. |
| 5 | As a tourist, I should be able to receive personalized recommendations based on my travel interests related to possible travel destinations and local businesses to visit. |
| 6 | As a tourist, I should be able to add other users as a friend in order to communicate with them. |
| 7 | As a tourist, I should be able to search for different businesses and their services and make bookings and payment through the app. |
|  |  |
|  | **Business Owners** |
| 1 | As a business owner, I should be able to create a business account for my restaurant, hotel or any local business at a particular location to attract tourists. |
| 2 | As a business owner, I should be able to update my business profile. |
| 3 | As a business owner, I want to be able to create service offerings for a business. |
| 4 | As a business owner, I want to receive bookings and approve or reject them accordingly. |
| 5 | As a business owner, I want my business to be viewable on a map so its easy for users to locate it. |
|  |  |
|  | **Administrator** |
| 1 | As an admin, I want to monitor/manage all other actor accounts which include account recovery, suspending or deleting accounts if indeed. |
| 2 | As an admin, I should be able to moderate content posted by actors (post, comments, reviews) and take necessary actions against inappropriate or illegal content. |
| 3 | As an admin, I want to be able to send new app features and updates to all users. |
| 4 | As an admin, I should be able to approve a business or tourist guide account before they are allowed on the app to ensure authenticity. |
| 5 | As an admin, I should be able to access the app’s analytics including traffic amount and any system generated logs. |

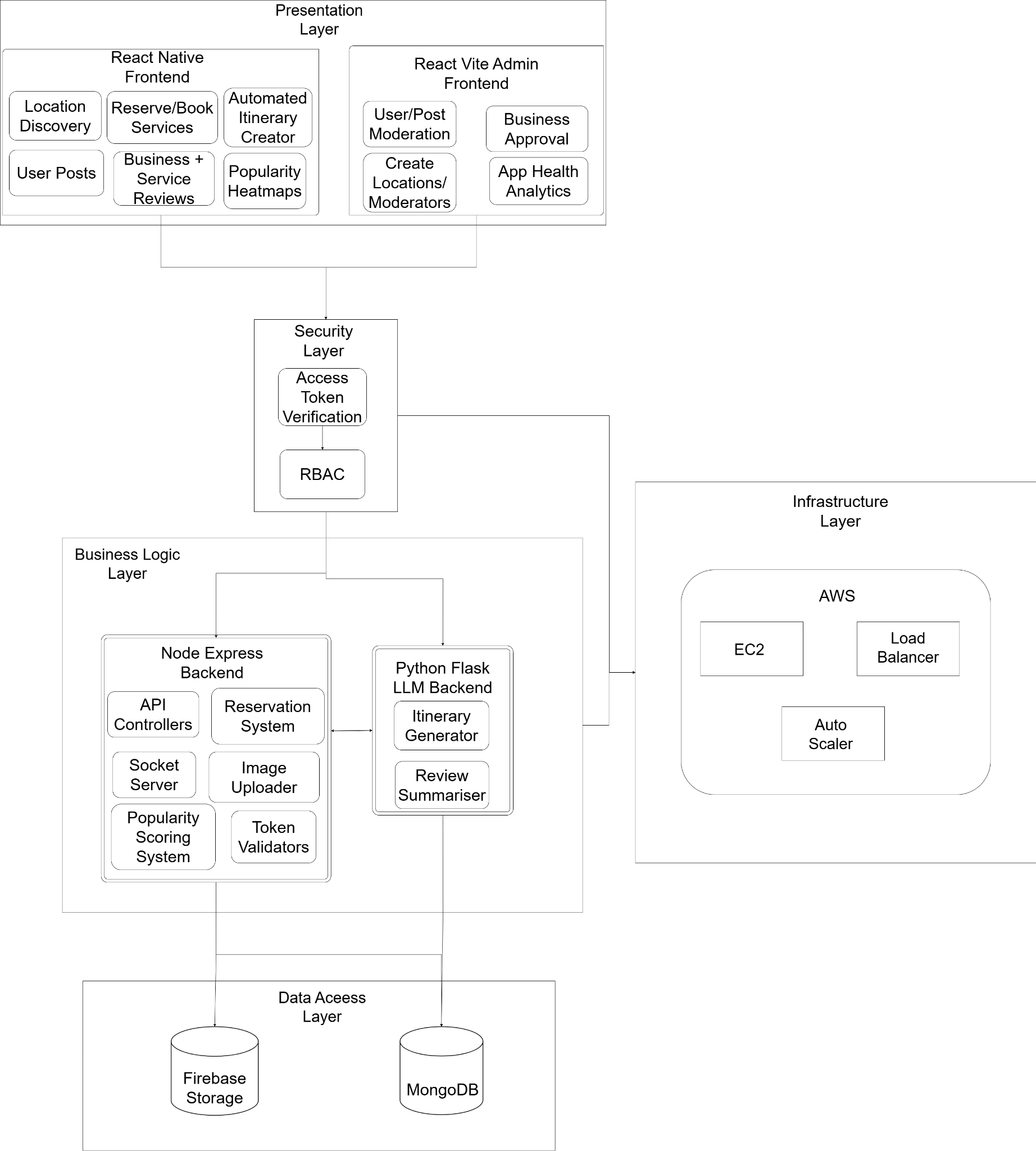
## Non-functional Requirements

| **Sr#** | **Requirements** |
| --- | --- |
| 1 | At any time during its execution, the system must not utilize more than 250MB of memory to ensure efficient resource usage and memory-related issues. |
| 2 | In the event of a failure, the system must restore to normal operations within 2-3 minutes of a failure to minimize downtime and maintain user satisfaction. |
| 3 | The system must handle at least 10,000 users without performance degradation. |
| 4 | During peak usage, the system must respond to user requests within 7-8 seconds. |
| 5 | The system must be highly available, maintaining an uptime of 99.9%, especially during peak travel seasons. |
| 6 | The system must be able to run on both IOS and Android devices to cater to a wide range of users across different platforms. |
| 7 | The system must be easily usable by all demographics, maintaining accessibility for a wide user base. |
| 8 | The system must be scalable to accommodate the increase or decrease in traffic. |
| 9 | The system must be able to handle up to 400-500 requests at peak usage. |
| 10 | The system must be designed to be easily maintainable to facilitate testing, ongoing updates, and support. |
| 11 | The system must ensure the data is well protected, ensuring data integrity through backups and redundancy measures. |

# System Architecture

This section gives an overview of the system architecture and descriptions of its components.

## Architecture Diagram



## Architecture Description

1. **Presentation Layer**

The presentation layer is responsible for the user interface (UI) and user experience (UX) of the application. It allows users and administrators to interact with the platform through visually intuitive components and workflows.

* 1. React Native Frontend: A mobile app built with React Native, allowing tourists and users to:
     1. Discover locations
     2. Make reservations and bookings for services
     3. Post content (e.g., reviews, images)
     4. Write business and service reviews
     5. Generate automated itineraries
     6. View popularity heat maps based on user activity
  2. React Vite Admin Frontend: A web-based admin dashboard built with Vite, enabling platform administrators to:
     1. Moderate user posts and activities
     2. Approve or reject business registration requests
     3. Create and manage location entries
     4. Assign moderators
     5. Analyze app health and usage statistics
  3. Interactions: Users and admins interact with the platform through these interfaces. User actions (e.g., posting a review, booking a service) are sent to the backend for processing through secure API calls, while admins manage content and businesses similarly via authenticated portals.

1. **Security Layer**

The security layer ensures that only authorized and authenticated users can access specific features of the platform, enforcing proper access control and token validation.

* 1. Access Token Verification: Validates JWT access tokens attached to API requests to ensure they are authentic and not expired.
  2. Role-Based Access Control (RBAC): Grants different permissions to users, businesses, moderators, and admins based on their roles (e.g., normal users cannot approve businesses).
  3. Interactions: Every request from the presentation layer passes through the security layer to verify tokens and permissions before reaching business logic.

1. **Business Logic Layer**

This layer handles the core functionality and operations of the platform. It contains the backend servers that process requests, apply business rules, and interact with the data layer.

* 1. Node Express Backend:
     1. API Controllers: Handle CRUD operations for users, posts, locations, reservations, etc.
     2. Reservation System: Manages booking and reservation flows for businesses and users.
     3. Socket Server: Facilitates real-time features such as notifications and chats.
     4. Image Uploader: Handles user/business uploads (e.g., profile photos, place images) and stores them securely on Firebase.
     5. Popularity Scoring System: Calculates metrics such as trending locations or popular services based on user activity.
     6. Token Validators: Provide additional security checks on access tokens beyond initial verification.
  2. Python Flask LLM Backend:
     1. Itinerary Generator: Uses AI (likely LLMs) to create personalized travel itineraries for users based on preferences.
     2. Review Summariser: Uses AI to summarize long reviews or generate insights from multiple reviews for easy consumption.
  3. Interactions: Business logic layer servers receive authenticated requests from the presentation layer, process them, perform necessary actions, and respond with results or errors.

1. **Data Access Layer**

The data access layer manages persistent storage and retrieval of the application’s data assets, including text data, images, and user-generated content.

* 1. MongoDB: A NoSQL database used to store structured application data such as user profiles, business details, bookings, reviews, posts, and location metadata.
  2. Firebase Storage: A cloud storage service used to handle large unstructured data such as images and media files uploaded by users and businesses.
  3. Interactions: The business logic layer reads from and writes to these data stores while abstracting storage details from higher layers.

1. **Infrastructure Layer**

The infrastructure layer ensures the application is deployed, scalable, highly available, and resilient.

1. AWS EC2: Virtual servers running the Node.js and Python Flask backends.
2. Load Balancer: Distributes incoming traffic across multiple EC2 instances to ensure high availability and performance.
3. Auto-Scaler: Dynamically adjusts the number of EC2 instances based on real-time traffic and resource utilization to optimize costs and maintain performance.
4. Interactions: The infrastructure layer hosts and manages the underlying servers for the business logic layer, ensuring efficient delivery of services to the presentation layer through secure, scalable operations.

## Justification of the Architecture

**Pros:**

1. **Separate Layers:**

Each layer has a distinct, individual role, such as the presentation layer that handles user interaction and the business logic layer that handles managing core functionality such as user requests.

1. **Server Scalability:**

The architecture will support horizontal scaling due to the infrastructure layer where all the backend components are deployed. This will ensure that the system can efficiently handle large amounts of traffic.

1. **Security:**

The security layer, along with its components that include JWT tokens, encryption, role-based access, etc, ensures that the sensitive data within the databases are protected and access is controlled. Potential security breaches will also be minimized due to the security layer.

1. **Extensibility and Integration:**

The integration layer enables easy interaction with third-party APIs and external services. This makes it easier to add new features, such as social media logins without affecting the rest of the system.

1. **Fault Tolerance and Availability:**

The use of load balancers in our infrastructure ensures that the backend servers distribute services across multiple layers. If one server goes down, the others can contribute by handling its requests and improving server availability.

**Cons:**

1. **Complexity:**

Having multiple layers makes our system more complex in terms of developing and managing. The interactions between layers can add a layer of overhead, especially when scaling the system.

1. **Server Latency:**

Third-party app extensibility along with multiple layers of abstraction can introduce latency in data processing and handling, leading to slower response times.

1. **Security Overhead:**

Implementing security components, such as JWT tokens, can add computational overhead, which might affect server performance, especially when dealing with high amounts of traffic.

**Justification for architecture**

This architecture is well suited for our app because:

1. **It is Modular**

Each layer operates independently, allowing easy modifications, updates, and scaling of components.

1. **It is Scalable**

The app can handle varying traffic through horizontal scaling of resources. This will dynamically adjust due to the server demand and amount of user traffic, making the server adaptable for a growing user base.

1. **it is Secure**

The security layer ensures data protection and user privacy, complying with standard security requirements.

**Implementation of Non-Functional Requirements by the Architecture**

1. **Scalability**

Our infrastructure layer supports horizontal scaling to accommodate an increase or decrease in user traffic. The load balancers ensure the even distribution of user requests.

1. **Security**

The non-functional security requirements are addressed by the security layer components. Encryption, JWT tokens, and other security measures ensure that our non-functional security requirements are met.

1. **Reliability**

The architecture provides fault tolerance with load balancing and auto-scaling features, ensuring the system remains available even during high traffic or server failures.

1. **Maintainability**

The distinct layers separate the responsibilities of the system, making the system more maintainable. This would allow our team to work on different layers without affecting the other components.

## Tools and Technologies

**Frontend**:

* React Native (Version: [0.75])

**Backend**:

* Node.js (Version: [v22.9.0])

**Database**:

* MongoDB (Version: [8.0])
* Firebase (Version: [10.14.1])

**Deployment/Cloud Hosting**:

* AWS or Azure (AWS EC2, S3, or Azure App Service)
* Render

**APIs**:

* Integration with various third-party APIs (e.g., Google Maps API, weather services)

**Version Control**:

* GitHub for code repository and collaboration

**Code Editor/IDE**:

* Visual Studio Code (Version: [1.94.2])
* Postman (API testing) (Version: [v11.16])

**Package Management:**

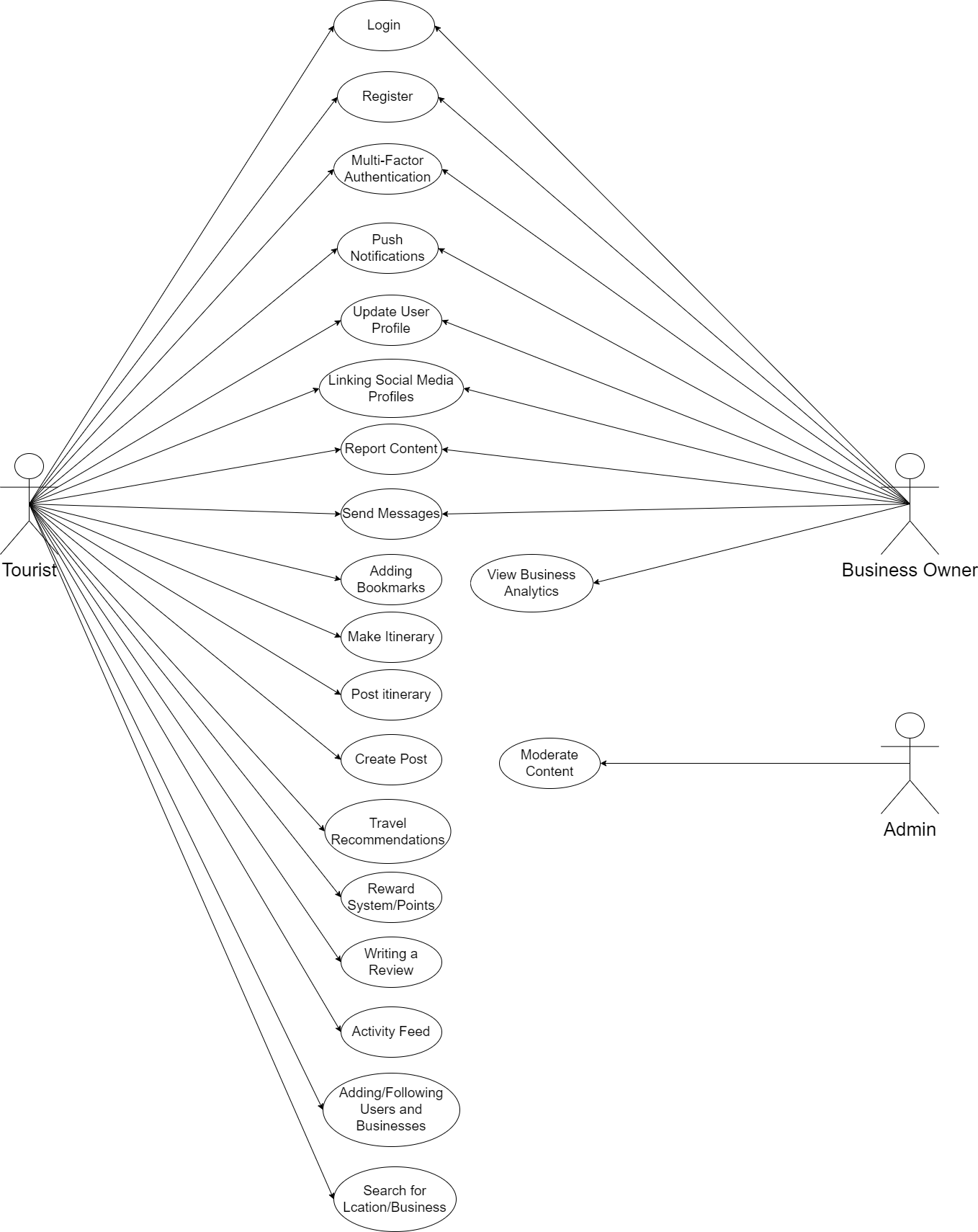
* NPM(node package manager) (Version: [10.9.0])
* PIP (Python package Manager) (Version: [24.3.1])

# 

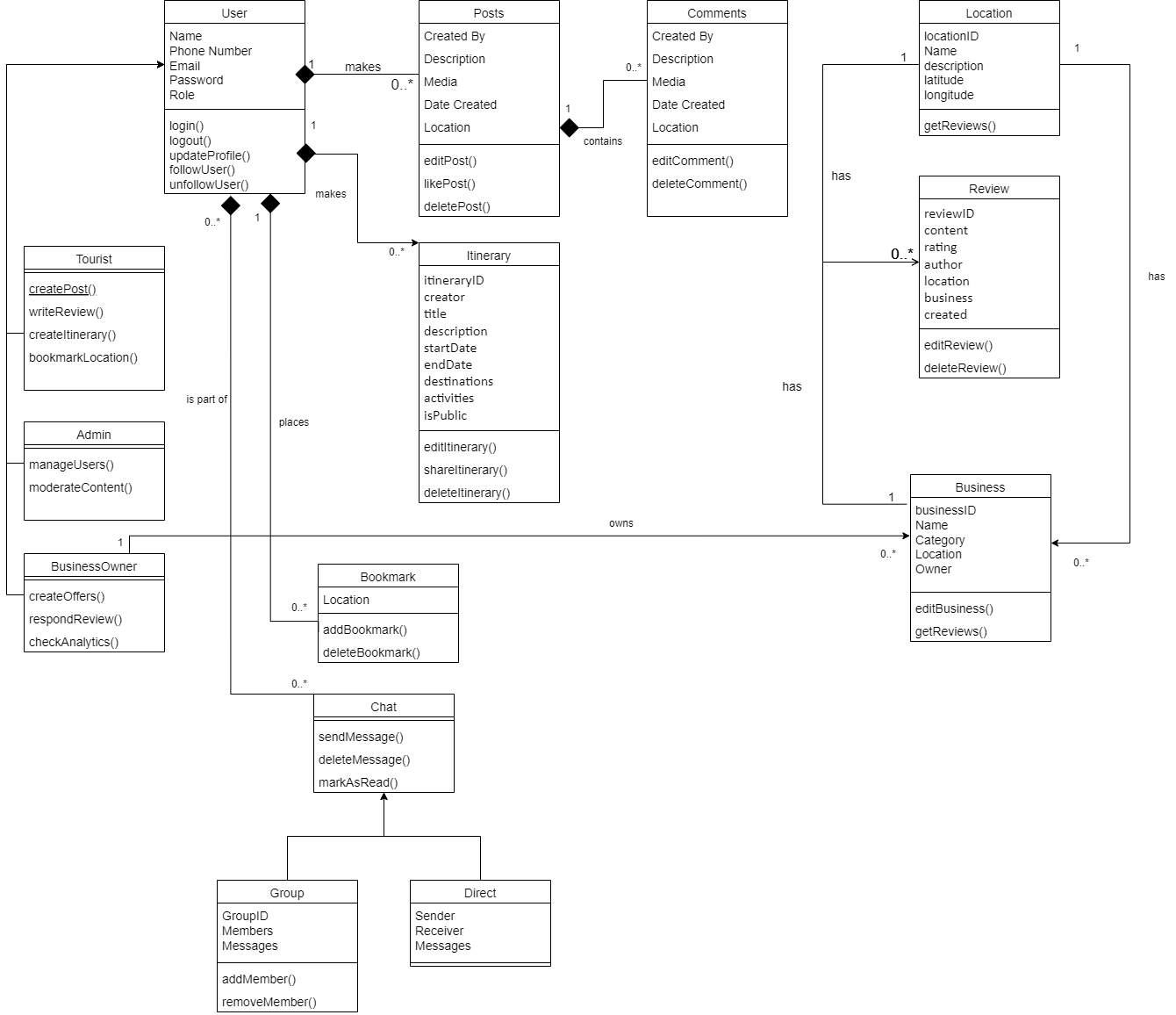
# Requirements Specifications

This section highlights the important use cases, classes and use case diagrams of the system.

## Use Cases



## Class Diagram



### Users

This represents the base class for all users (Tourist, Admin, Business Owner). Contains common attributes and methods shared by all users. Each user can login and register (Admin does not need to register).

#### *Tourists*

These are the primary users of the app. They can create posts and itineraries, comment on other posts, search for locations, leave reviews and bookmark them. They can chat with other users and businesses and follow them.

#### *Admin*

These users are responsible for monitoring the app health and making sure that its users follow the guidelines of the app. Their main responsibilities are checking app status, moderating posts and managing users.

#### *Business Owners*

These users are the secondary users of the app. They can create an account and their business pages and advertise their services to incoming tourists. They can chat with others, check user reviews and monitor overall business analytics.

### Itinerary

Itinerary allows tourists to create and manage their travel plans by adding locations, activities, and dates. Users can choose to make them public or private.

### Bookmark

Bookmark allows users to save certain locations they are interested in visiting if they plan to visit sometime in the future or want to recommend it to others.

### Chat

Represents the chat functionality for direct and group communication between users.

#### *Direct*

This subclass is to represent direct one on one communication between two users.

#### *Group*

This subclass represents communication between multiple members.

### Posts

Posts class will allow users to create posts about their experiences so that other viewers can check them out.

### Comments

Comments class represents a list of comments left by users on a certain post.

### Location

This represents tourist destinations for which users can make posts about, create itineraries, leave reviews and bookmark.

### Review

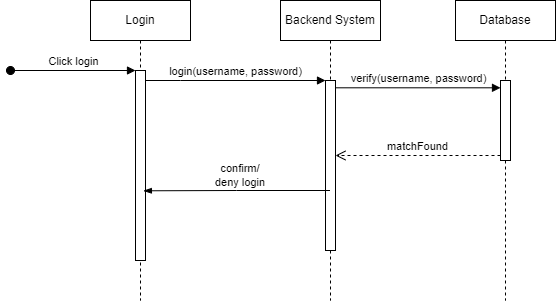
This class represents reviews left by users for a certain location or a business in that location for others to view.

### Business

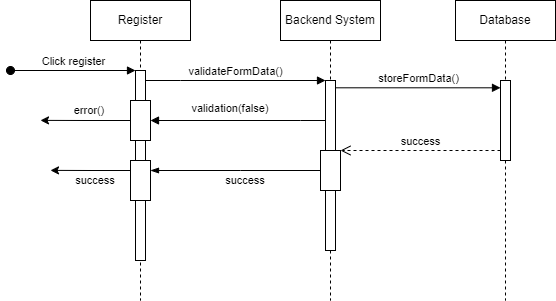
Business class represents a business in a certain location owned by a certain business owner. Users can view businesses, chat and write a review about them. Meanwhile business owners can advertise their services and check overall performance.

## Sequence Diagrams

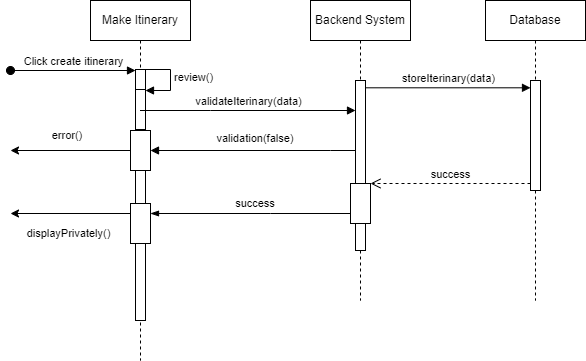
### Login



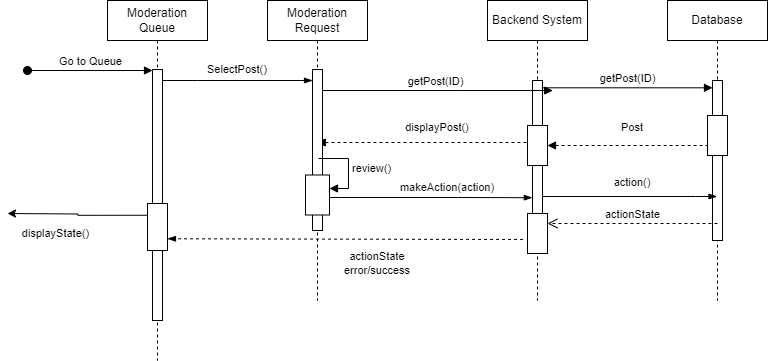
### Register



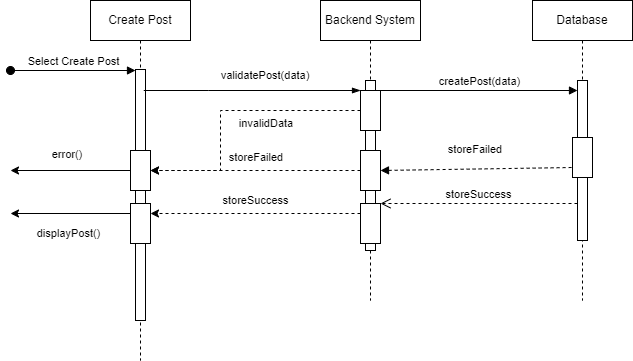
### Make Itinerary



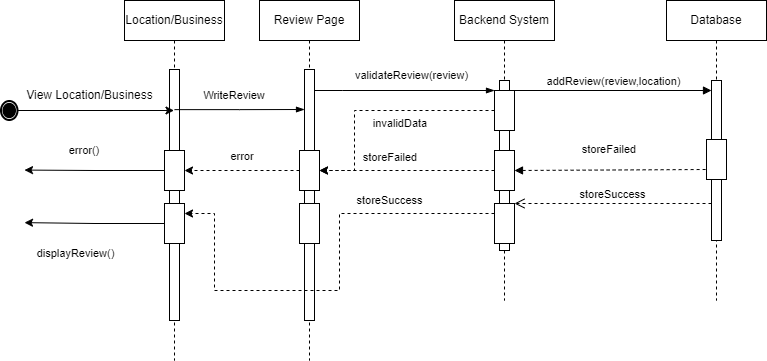
### Post Moderation



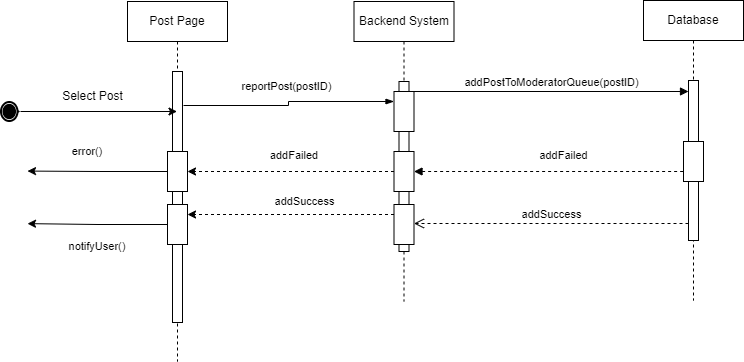
### Create Posts



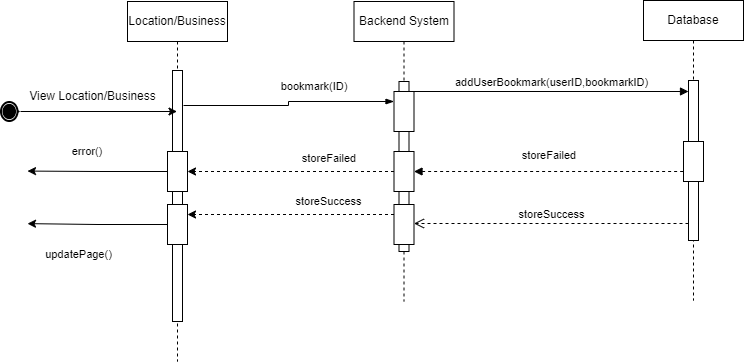
### Posting a Review



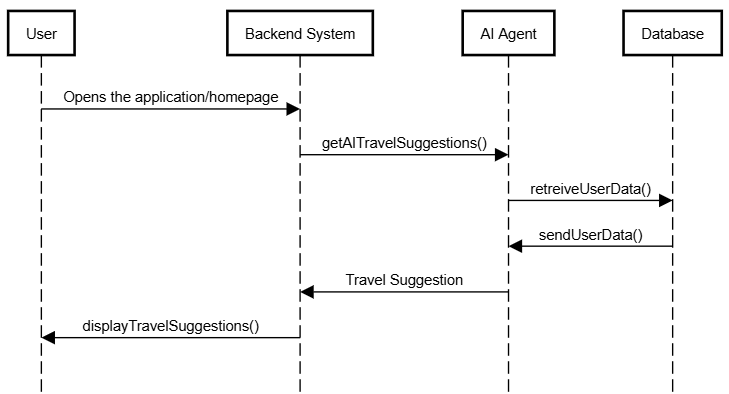
### Report Content



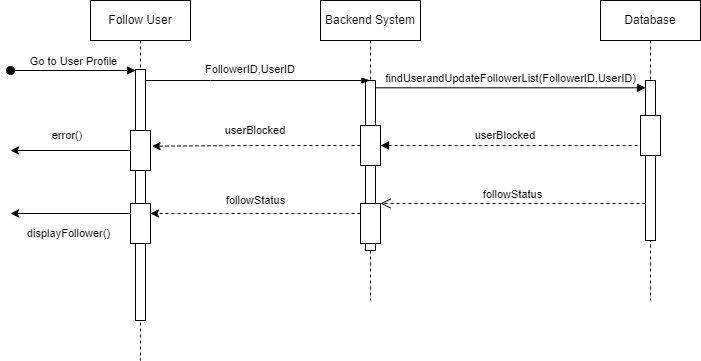
### Bookmark Locations



### AI-Based Travel Suggestions + Reviews



### Follow Users



# Software Development Methodology and Plan

This section highlights the software process chosen and development timeline for the project.

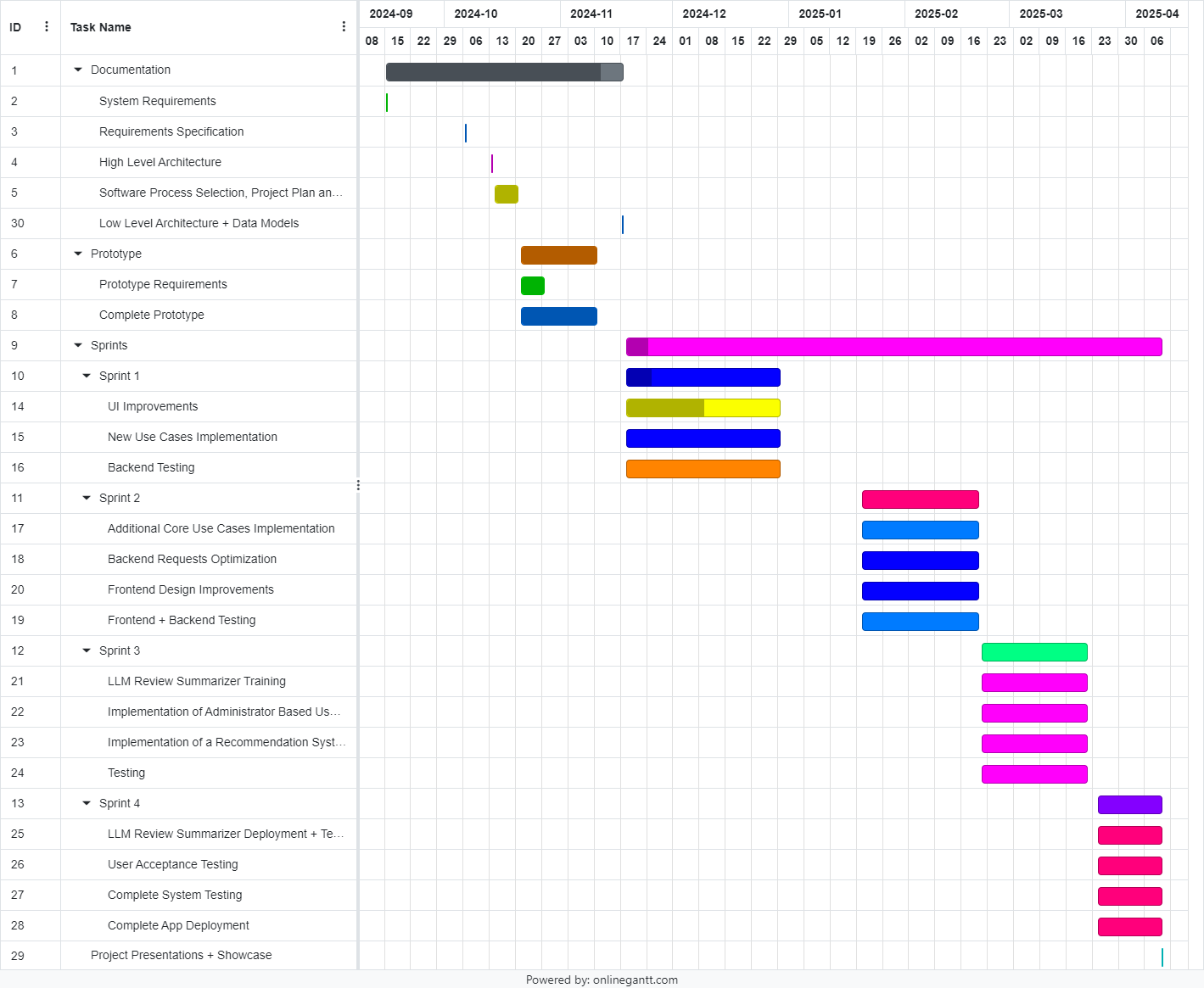
## Software Process Selection

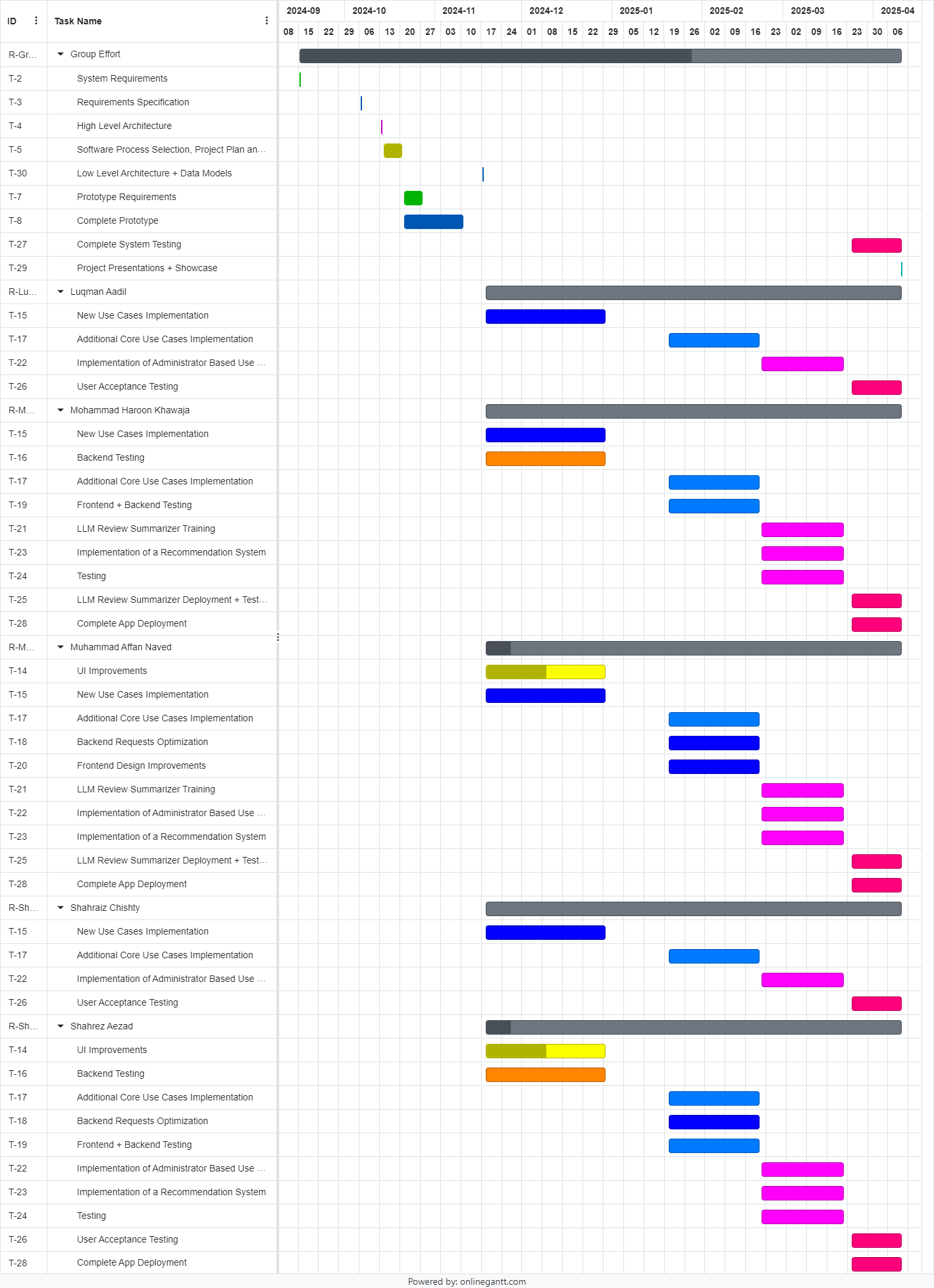
1. **Waterfall**
   1. Pros
      1. Clear Structure: Each phase in development has well defined deliverables to be met by a given deadline.
      2. Thorough Documentation: Each phase requires the documentation of all deliverables to be well defined and comprehensive before going to the next phase.
      3. For projects that have well defined requirements and are not subject to major changes in the functionality, this process is ideal.
   2. Cons
      1. Late Feedback: Users usually only see the final project after the development is entirely completed which can create problems in case the developed software does not meet expectations.
      2. Inflexibility to Change: If a requirement has been changed by the client it can be difficult and costly if the project has moved on to later phases.
      3. Delays: In case of any delays during development, this can lead to a delay in delivery and can also be outdated upon delivery or release.
2. **Agile (Scrum)**
   1. Pros
      1. Flexibility: Changes in requirements or needs of the client can be incorporated into the project at any stage.
      2. Continuous Feedback: Users or clients are involved during all stages of development and their feedback ensures the product meets required standards.
      3. Faster Delivery: With the aid of sprints, the development team is able to deliver main components of the product quickly to the clients.
   2. Cons
      1. Inadequate Documentation: Since the main goal of this process is to produce a proper working software as fast as possible, documentation can get left behind, which can cause issues in the future in case of changes or if a new member joins the team.
      2. Predictability: Harder to predict exact deadlines and costs of development due to constant changes in product requirements.
3. **Chosen Strategy - Agile Scrum**
   1. **Justification**

We have chosen to use the Agile (Scrum) process for the following reasons.

* Since our project is a social media app for travellers and tourists, user feedback is critical for development in order for us to make necessary changes to improve user experience or add new features. Agile will allow us to keep these clients in the development cycle.
* Secondly, the sprint based approach of agile process allows us to provide primary features of the app faster to the user so that we can get their feedback in order to improve those features.
* Furthermore, we believe it is easier to identify any logical or integration issues during development and they can be addressed on the go in each sprint.
* Agile also provides flexibility for evolving requirements, so in the case of new requirements or changes in existing ones to keep up with current trends, they can be easily implemented during development.

## Gantt Chart

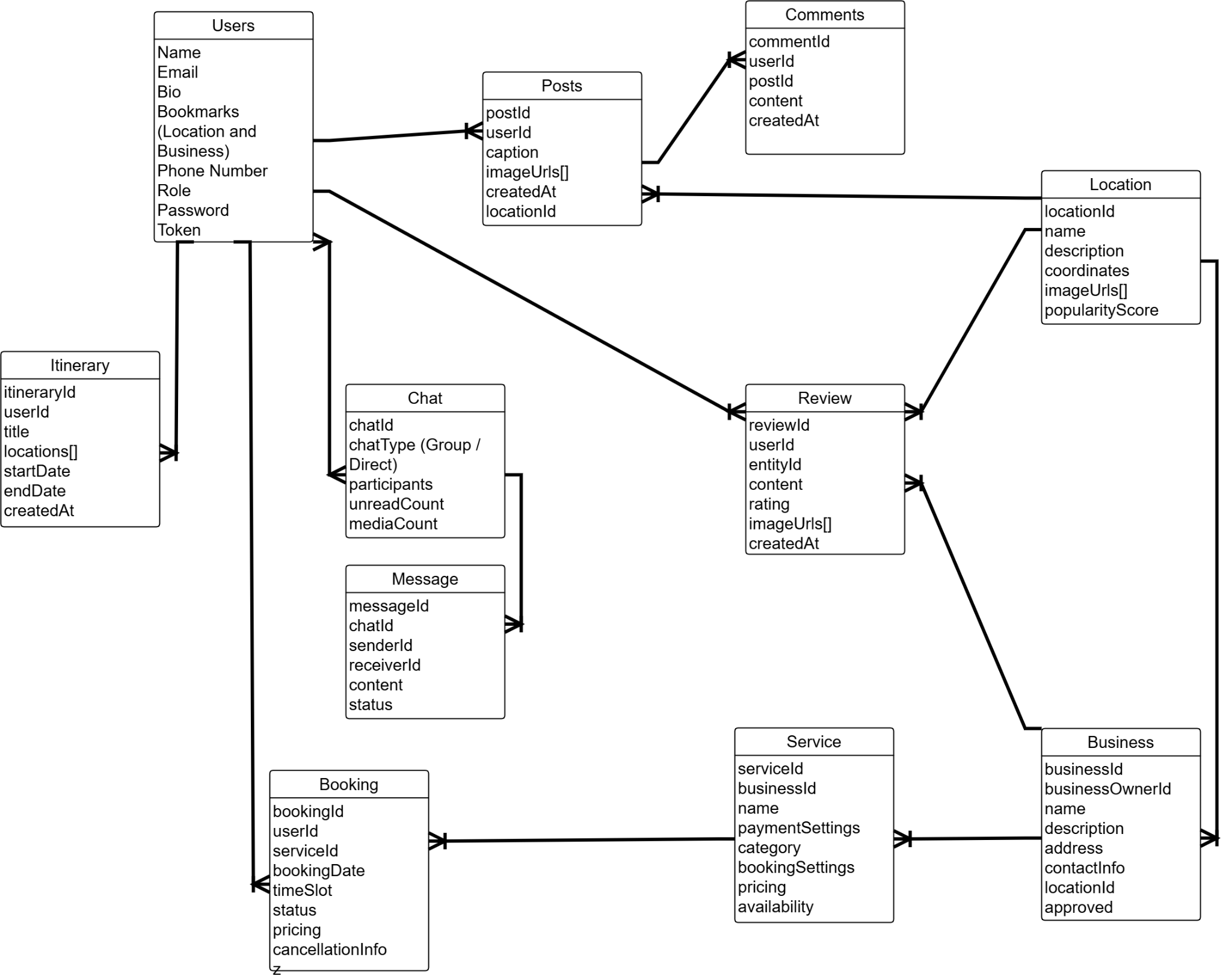




# Database Design and Web Services

This section gives an overview of how our database and its schemas are defined and the relationships schemas have with each other.

## Database Design



1. User

This model holds all user data including email, name, bio, profile pictures and bookmarks.

1. Posts

This model contains the data for a post which a tourist or a business owner can make. It can contain images or videos too.

1. Comment

This model contains all the comments made by different users under a post.

1. Location

This model contains all popular tourist locations for different people to view and add to their itineraries.

1. Review

This model will store data containing reviews for a location or a business so that other users can view and make well informed decisions on whether a certain place is worth visiting.

1. Business

This model contains data of a specific business including its locations, owner, description, reviews etc.

1. Chat

This model will hold all chat interactions between different users who are communicating directly or in a group.There are two different types of Chats:

1. Direct: Two people can chat with one another directly.
2. Group: 3 or more people can chat together.
3. Service

This model contains data related to a service that is offered by a specific business which tourists may choose to book. A single business can offer multiple services of different categories.

1. Booking

This model contains data related to any booking made by a specific user for a specific service. It contains the booking dates, payment methods and price paid.

1. Itinerary

This model stores data related to a user created travel itinerary so that they may share it with other users.

## API Specification

There were 2 external APIs which were used using the help of predefined NPM packages.

1. Google Maps: We used the Google Maps API to provide business locations and their popularity heatmaps to users. Package used: [react-native-maps](https://www.npmjs.com/package/react-native-maps) Version 1.20.1.
2. Firebase Admin SDK: We used this NPM package to allow us to communicate with our Firebase storage which stores user uploaded media for others to see. We use URLs generated by Firebase to display them on our app. Package used: [firebase-admin](https://www.npmjs.com/package/firebase-admin) Version 12.7.0.

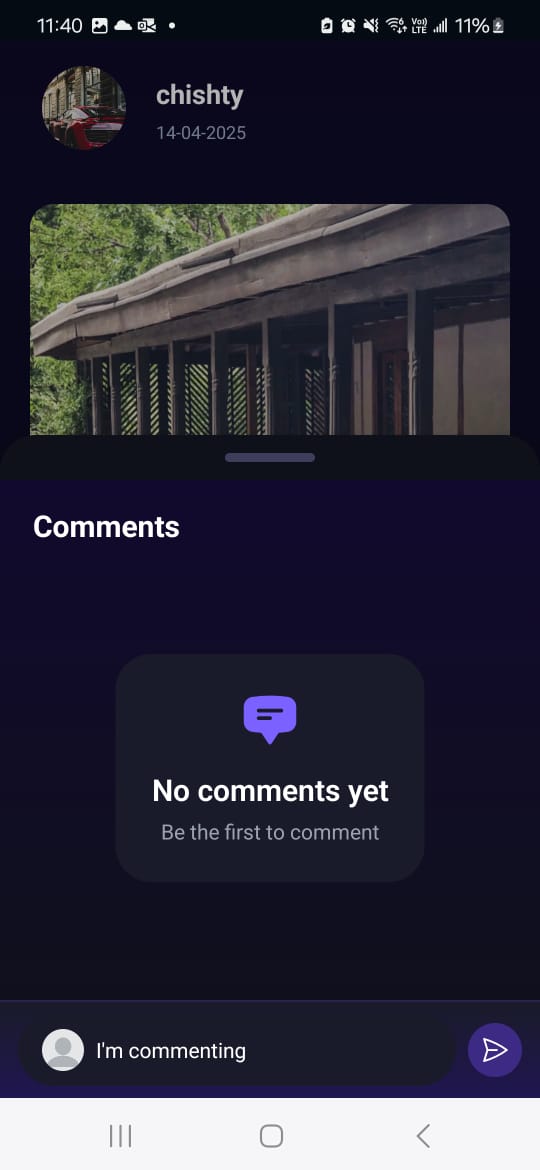
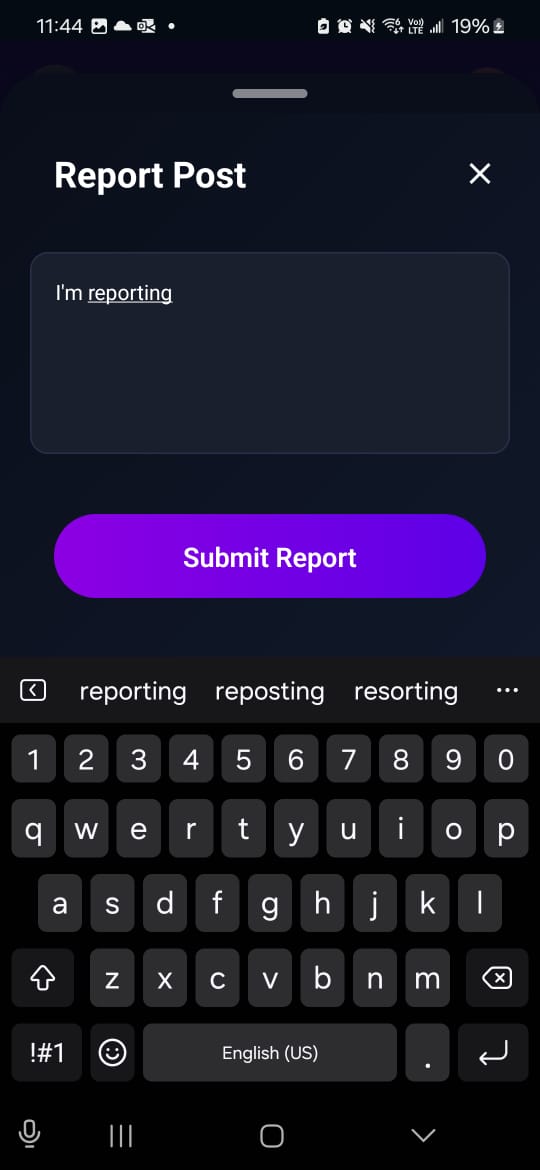
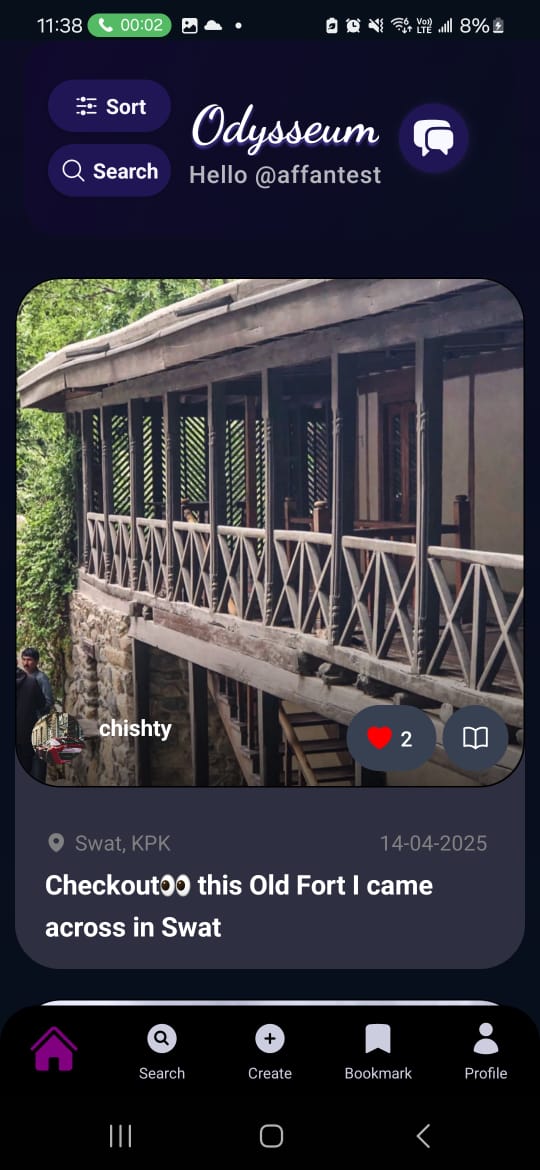
# System User Interface

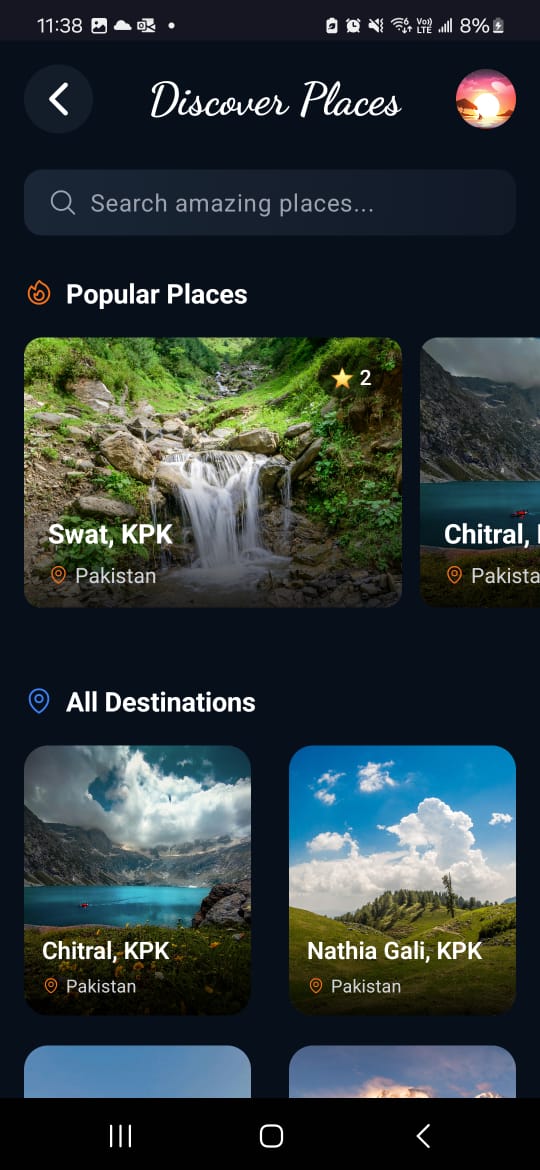
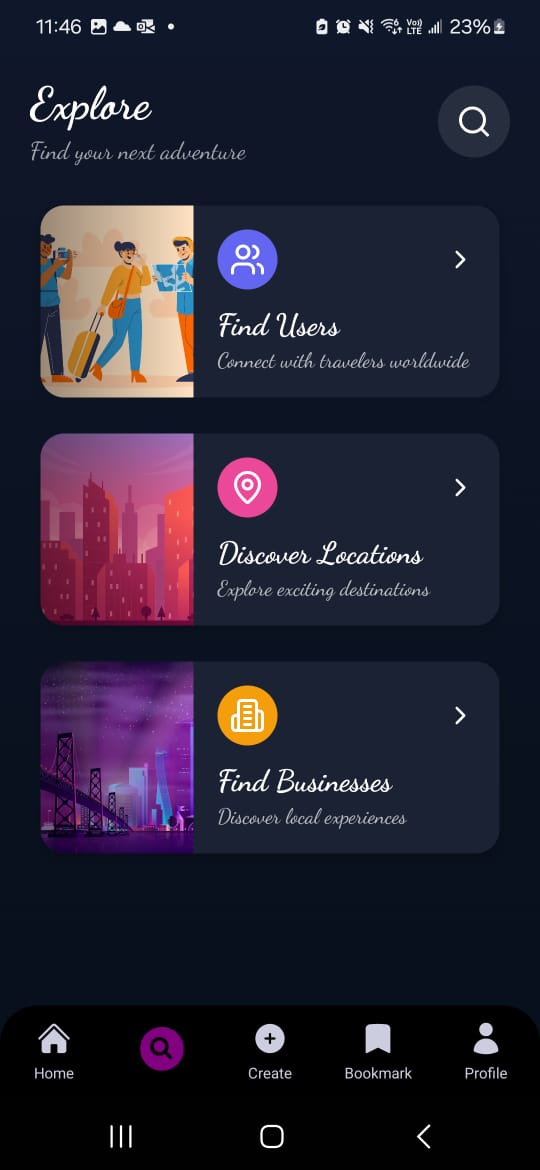
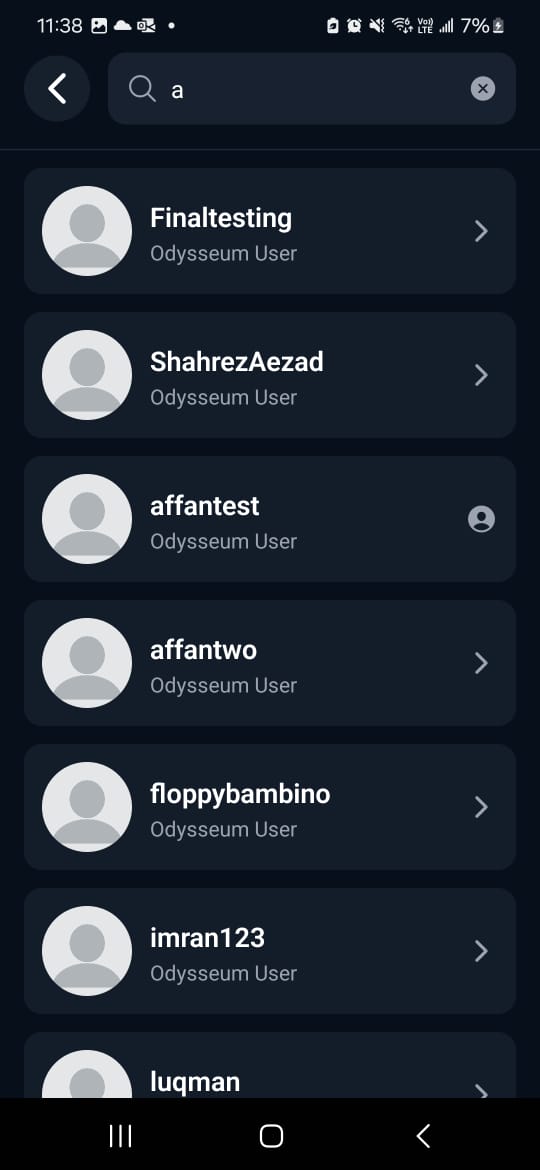
**This section gives a brief overview of all important UI screens of the application.**

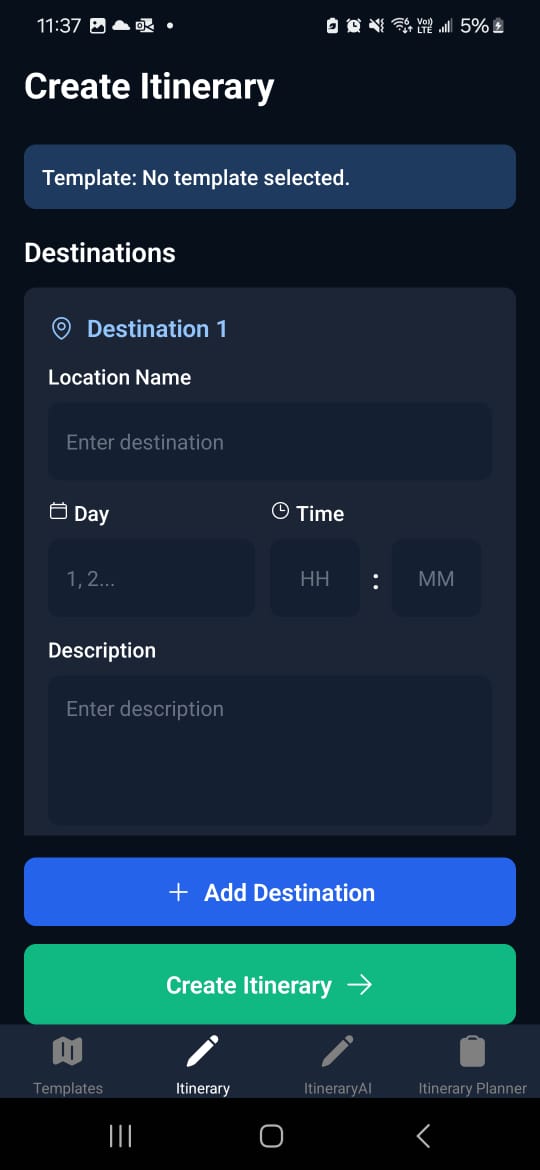
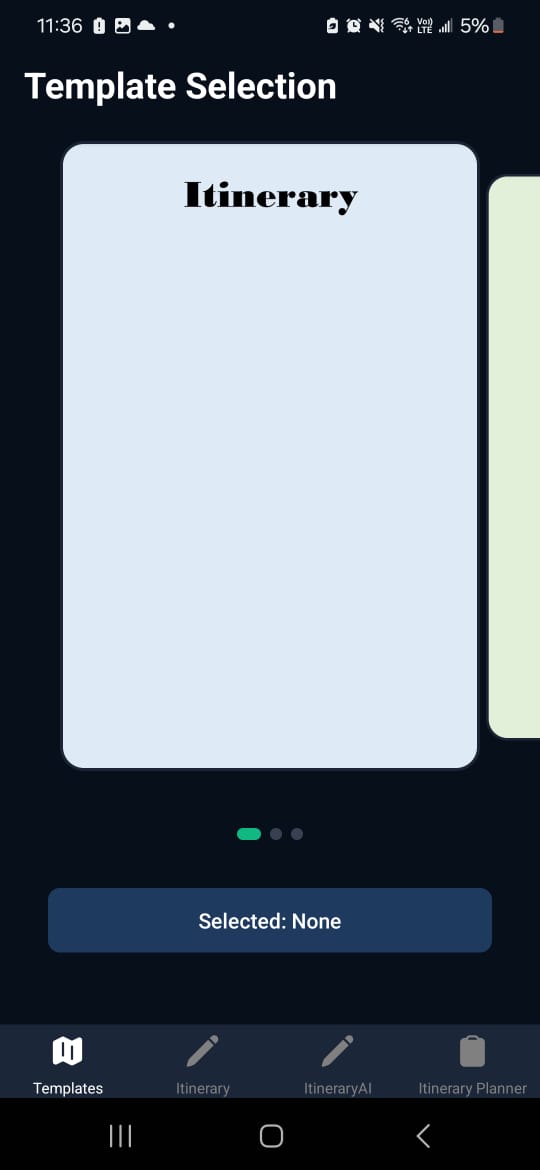
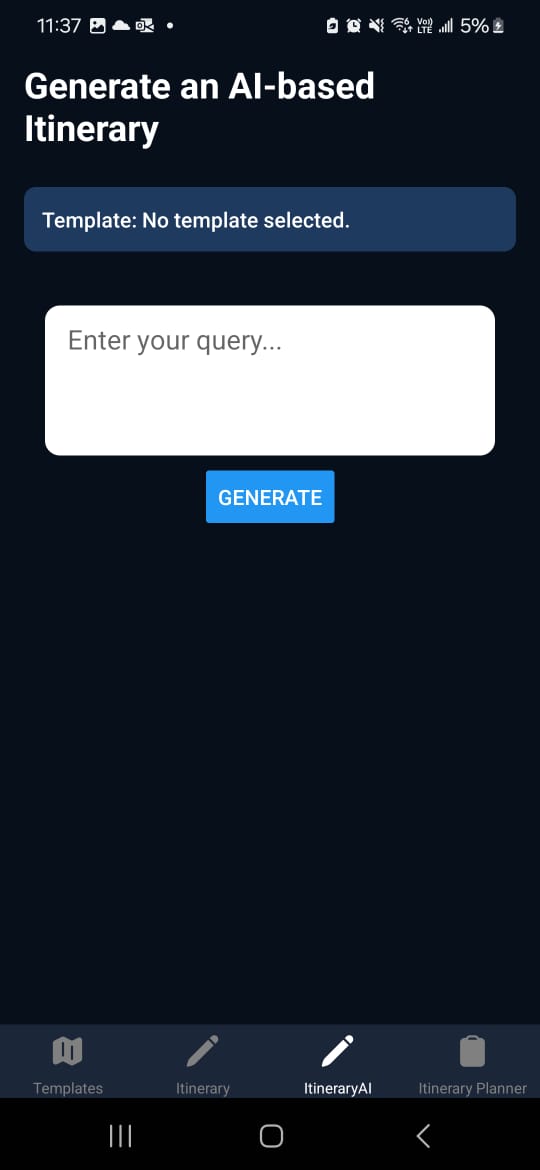
Important UI Screens:

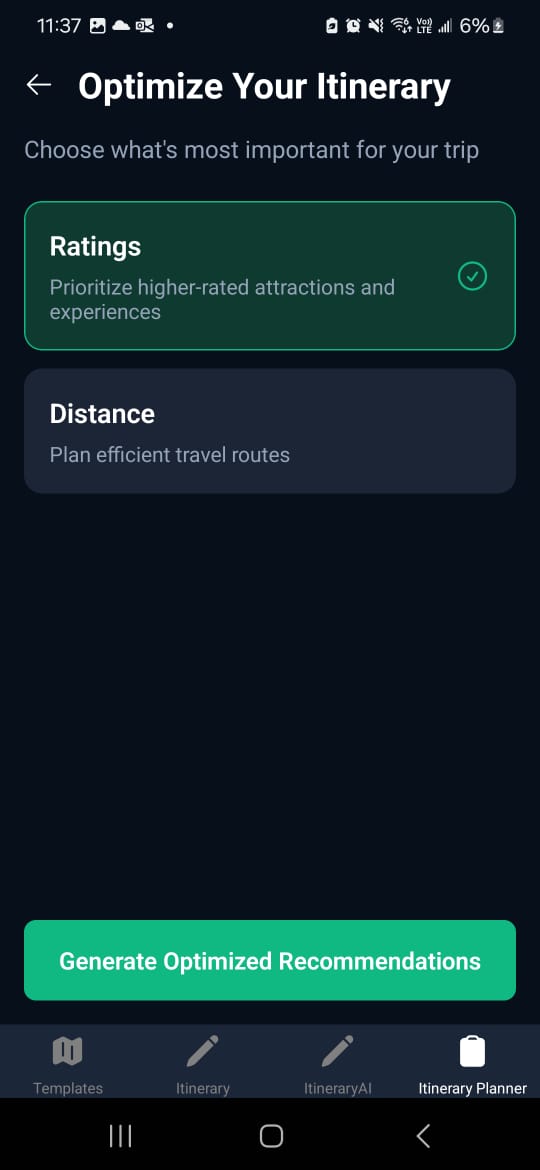
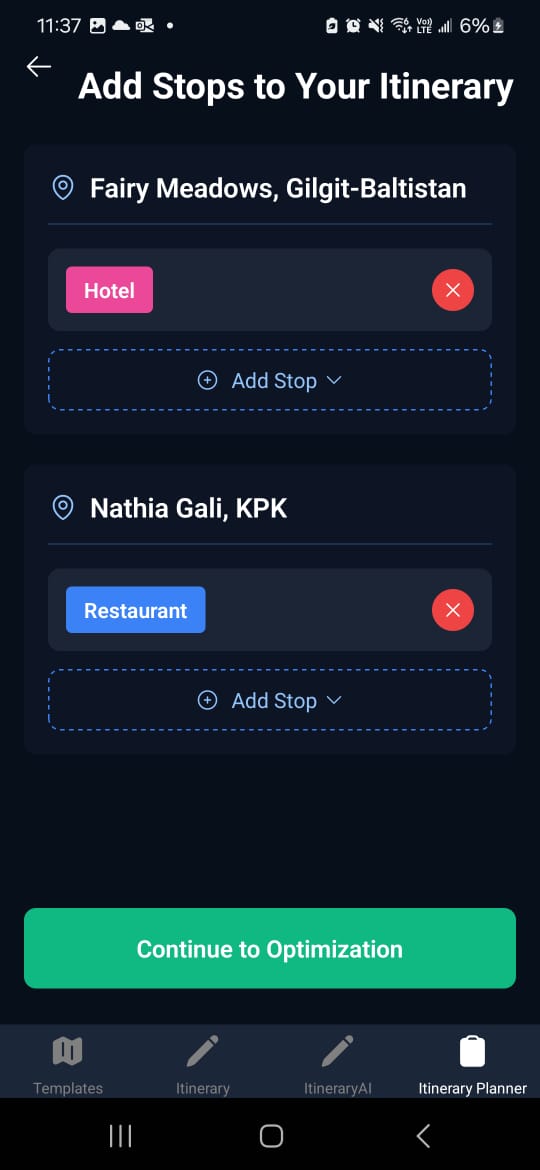
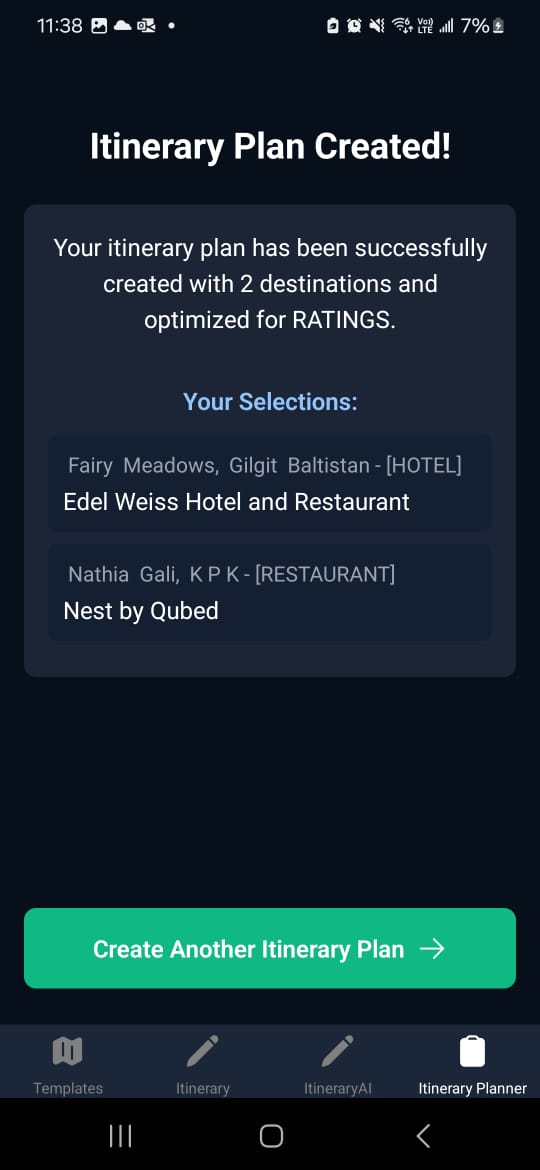
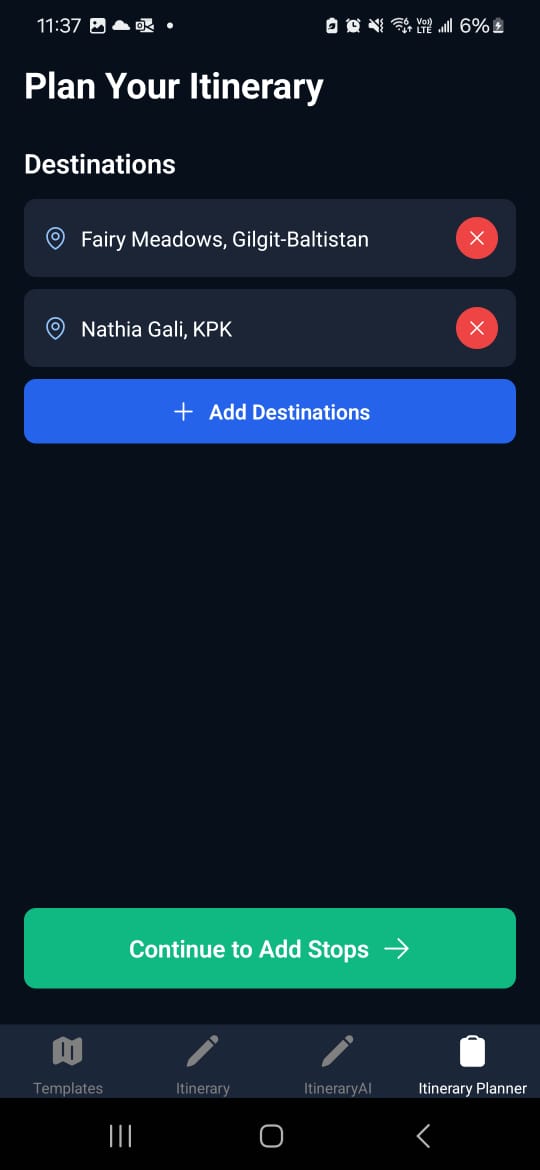
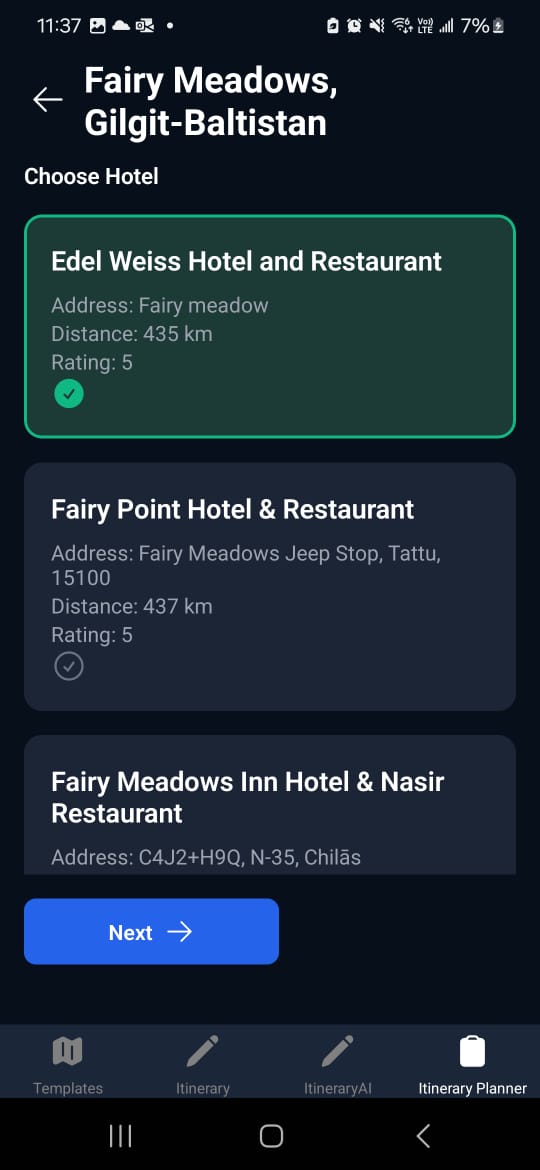
1. Home Screen: The homescreen is the default screen for users to swipe through and interact with fellow users posts. They can like, bookmark, and report posts
2. Search (Explore) Screen: Users can find other users, discover locations, and find businesses
   1. Find Users Screen: Users can search for other existing users.
   2. Discover Locations Screen: Users can search locations. They can view popular places and all possible destinations.
   3. Find Businesses Screen: Users can search for locations. Upon choosing a location, Users can interact in various ways with the business, such as calling, emailing, sharing, reviewing, and visiting their website and viewing their services.
3. Create Screen: Users can create posts to share and create itineraries to plan and create their travel plans
   1. Create Post Screen: Users can add photos, write captions, and post their photos.
   2. Create Itinerary Screen: Users can choose from a list of templates and can then either:
      1. Create a manual itinerary based on their needs
      2. Create an AI generated Itinerary based on their preferences and destinations.
      3. Plan an itinerary by choosing locations, stops, preferences, and locations catered for the user.
4. Bookmarks Screen: Uses can search abd view their bookmarked locations and businesses.
   1. Bookmarked Locations Screen: users can find all related businesses and reviews related to the location. Users can also view all popular spots related to the location as well.
   2. Bookmarked Business Screen: Users can interact with businesses and their services. They can view all possible info related to the business.
5. Profile Screen: Users can interact with their own profile where they can see their followers, followings, and posts. Users can also view their posts, as well as their saved and liked ones. Users can also view their booking. They can also change their settings and log out.

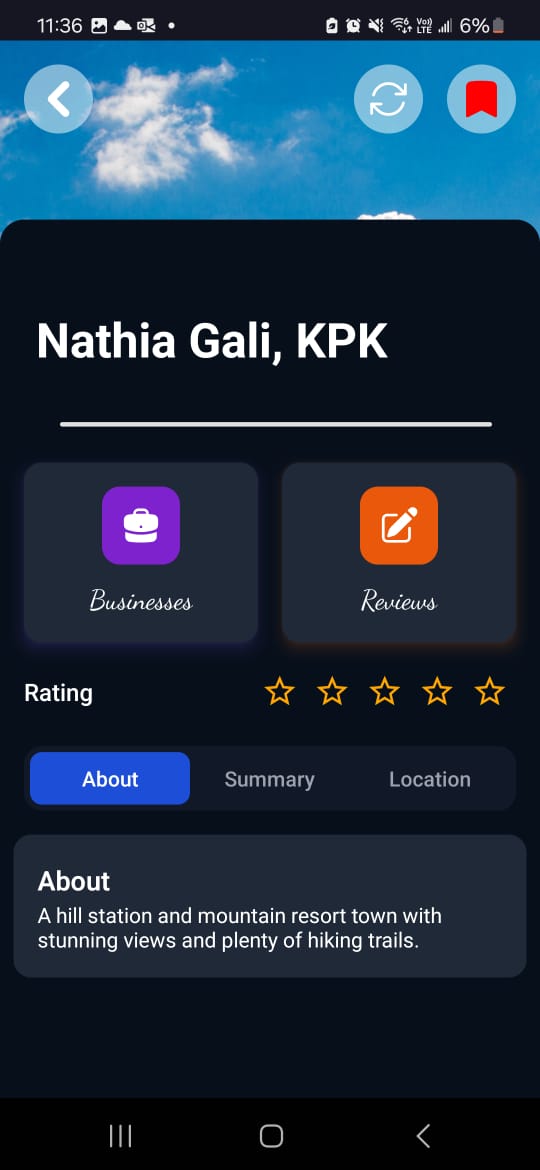
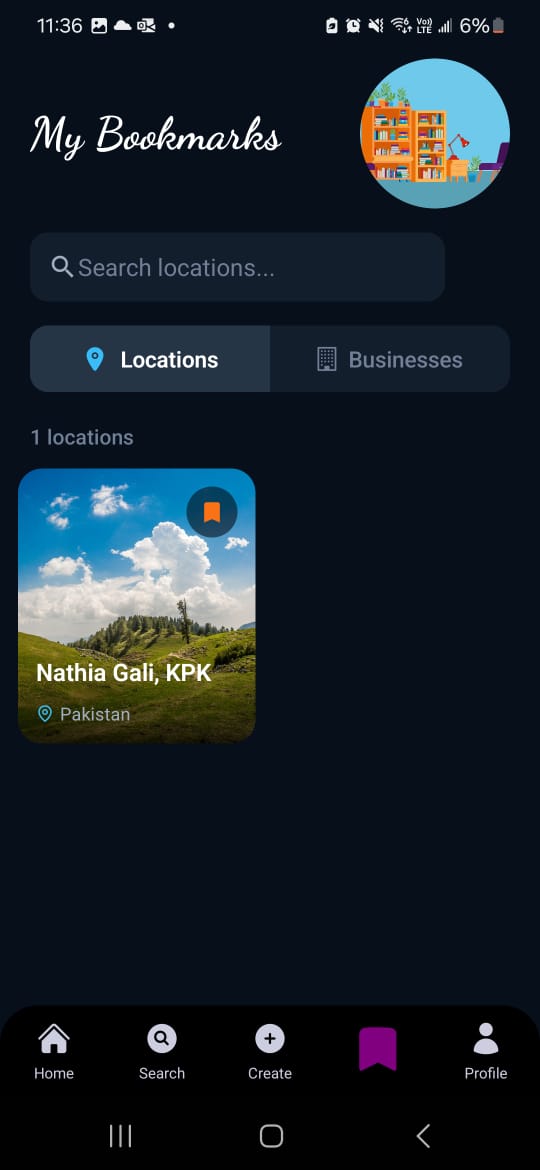
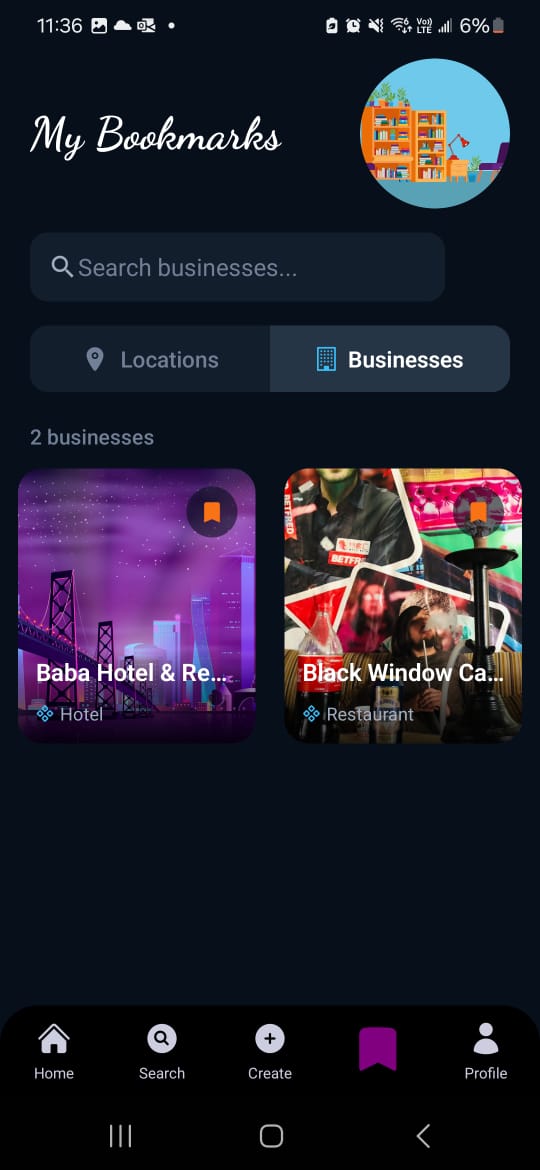
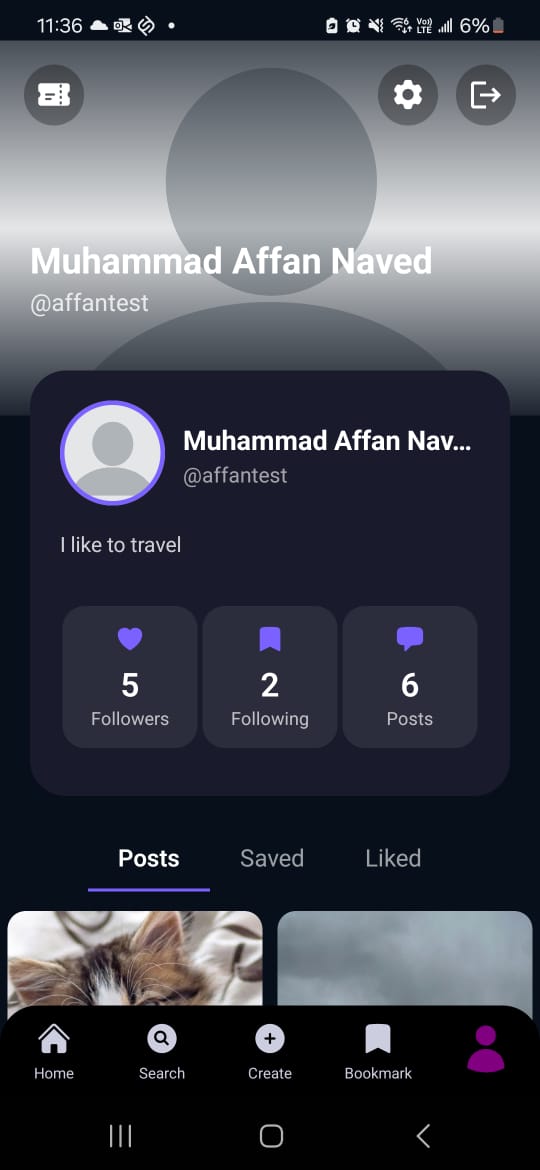
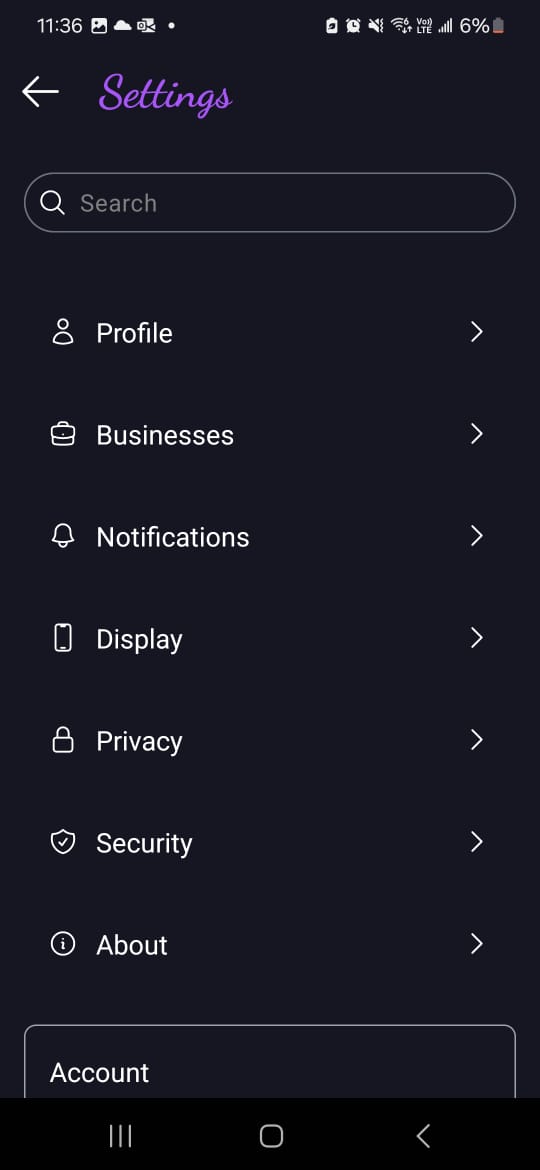
**This sub-section should explain the functionality of your application to the end user with supporting screenshots.**

1. Home Screen: The Home Screen serves as the default landing page where users can:
   1. Scroll through posts from other users
   2. Like posts by tapping the heart button
   3. Bookmark posts for later reference
   4. Report inappropriate content
2. Search (Explore) Screen: Users can find other users, discover locations, and find businesse.
   1. Find Users Screen: Search for existing users by their usernames
   2. Discover Locations Screen: Explore popular destinations and search for specific locations
   3. Find Businesses Screen: Search for businesses by location and interact with them through various options:
      1. Call the business
      2. Send an email
      3. Share with others
      4. Read and write reviews
      5. Visit their website
      6. View available services



1. Create Screen: The Create Screen offers options for sharing experiences and planning trips:
   1. Create Post Screen: Users can post their experiences by uploading photos and adding captions
   2. Itinerary Create Screen: Users can choose a template and also plan their travels using three methods:
      1. Manual Itinerary Create Screen: Users can build a custom itinerary based on their specific preferences
      2. AI-Generated Itinerary: Users can use our AI-agent to created an itinerary based on their preferences and destinations
      3. Preference Guided Itinerary Planner: Users can select destinations, stops, preferences for a personalized and optimized itinerary plan.



1. Bookmarks Screen: Access saves content through two main Screens:
   1. Bookmarked Locations Screen: View locations you have saved along with related business, reviews, and popular sites.
   2. Bookmarked Business Screen: Access your saved businesses and their associated services and information.
2. Profile Screen: The Profile Screen provides user access to personal content and account settings:
   1. View followers and following lists
   2. Browse your posts
   3. Access saved and liked content
   4. Manage your bookings
   5. Adjust account settings
   6. Log 

# Project Security

This section gives an overview of potential security risks and their mitigation strategies.

## Project Threats

## Potential Losses

## Security Controls

All the above details have been written in table below:

| **Sr#** | **Security Risks** | **Potential Losses** | **Controls** |
| --- | --- | --- | --- |
| 1 | Broken Access Control | Unauthorized information disclosure, modification, or destruction of data. | Server-side; default deny, enforce record ownership, API rate limits, session validity control, etc. |
| 2 | Injections | Unauthorized disclosure, modification or destruction of data. | Positive input validation, safe APIs, and prepared statements. |
| 3 | Sensitive Data Exposure | Unauthorized exposure of sensitive user data such as emails, credit card numbers, itineraries, phone numbers, etc. | Ensure strong cryptography - both in storage and in-transit, usage of secure protocols, e.g. TLS, HTTPS, for transfer of data, disabling of caching for responses containing sensitive data. |
| 4 | Insecure Design | Potential unauthorized access to sensitive system components. | Strict adherence to the secure development lifecycle. |
| 5 | Identification & Authentication Failures | Financial loss, litigation, user-information access, unauthorized and unintended actions on behalf of users. | MFA, secure credentials for both admins and end-users, limiting login attempts, server-side secure session manager using randomized session IDs with high entropy. |

# Risk Management

## Potential Risks and Mitigation Strategies

| **Sr.** | **Risk Description** | **Mitigation Strategy** |
| --- | --- | --- |
|  | Compromise of user data, loss of trust, legal implications. | Implement role-based access control to only allow admins to modify data. |
|  | Use of Unauthorized Developer APIs which can allow users access to sensitive operations. | Implement token based authentication to prevent app users from accessing these APIs and regularly update API keys and monitor access logs for suspicious activities. |
|  | Backend failure or maintenance downtime. | Implement auto scaling with load balancing to provision more backend servers on the cloud. |
|  | Loss or corruption of data due to a disaster. | Use database replication and implement regular data backup routines, create a disaster recovery plan. |
|  | Inappropriate or Malicious User Content | Allow users to report content and implement content moderation using AI tools and manual reviews. |
|  | Poor User Retention Engagement due to loss in interest or a competitor entering the market. | Regularly analyze user behaviour to stay up to date on latest trends.  Introduce a rewards system to enhance engagement. |
|  | Cultural and Language Barriers for tourists. | Include multi -language support along with local language translation features. |
|  | User Safety Concerns in Locations | Provide safety tips, verified reviews for tourist locations and contacts for local authorities. |
|  | High cost of Deployed Backend | Make use of cost optimized cloud services and serverless services (e.g. ECS Fargate on AWS) |
|  | Issues with scalability as platform and number of users grow. | Make use of horizontal scaling on cloud to provision more servers to keep up with growing demand. |

# Testing and Evaluation

Our frontend testing strategy revolves around comprehensive end-to-end testing of critical user flows to ensure the application functions correctly from the user's perspective. We adopted a behavior-driven approach, focusing on testing complete user journeys rather than isolated components. For testing the frontend we made use of Maestro.

We chose Maestro as our UI testing framework for several key reasons:

1. Simplicity: Maestro uses human-readable YAML syntax, making tests easy to write and maintain
2. Cross-platform: Seamlessly works across both iOS and Android platforms
3. Minimal setup: Requires minimal configuration to get started
4. Fast execution: Tests run quickly compared to other frameworks
5. Visual verification: Supports both element-based and visual testing approaches

Here is an example snippet of a test case for testing the functionality of logging a user into the app:

appId: com.odysseum.app

---

- launchApp

- tapOn: "Login"

- inputText:

id: "email-input"

text: "${TEST\_USER\_EMAIL}"

- inputText:

id: "password-input"

text: "${TEST\_USER\_PASSWORD}"

- tapOn: "Login"

- tapOn: "Create"

- assertVisible: "Create"

- assertVisible: "Share your adventures!"

- tapOn: "Create Post"

- assertVisible: "Create Post"

- back

- tapOn: "Create Itinerary"

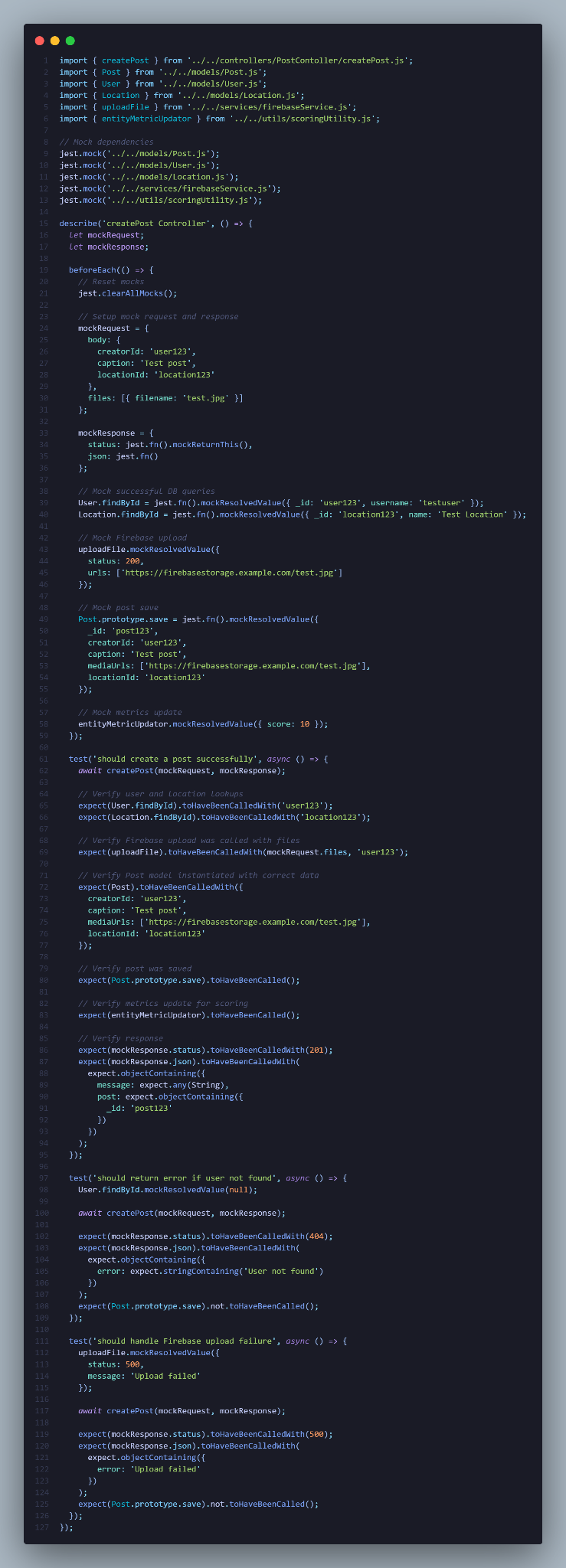
- assertVisible: "Enter Destination"

- back

Our backend testing strategy focuses on comprehensive testing of controller functions that handle critical business logic. We employ a combination of unit tests and integration tests to ensure controllers interact correctly with models, services, and external dependencies like Firebase. our backend testing relies on Jest because:

1. Jest: Perfect for JavaScript backend testing with support for mocking database connections and external services
2. Snapshot testing: Useful for API response validation
3. Async/await support: Excellent for testing Promise-based controller functions

Here is sample snippet of the post creation test case since it interacts with both MongoDB and Firebase for media storage:



# Deployment Guidelines

There are 2 different methods to deploy the backend of the project.

1. AWS Deployment on EC2 Instances
   1. In the “server/deployment” directory of the repository you will find a Terraform script that will deploy the entire backend application (Node Server and LLM Flask Server) on AWS. This requires your machine to have and AWS account, AWS CLI and Terraform CLI installed. One you have configured AWS CLI, run the following commands in the “server/deployment” directory.
      1. terraform init
      2. terraform plan
      3. terraform validate
      4. terraform apply –auto-approve
      5. If you wish to destroy the deployed architecture to save cost run:
         1. terraform destroy –auto-approve
2. Deployment on [Render.com](http://render.com) 
   1. This deployment is more simple and is the main deployed backend server in order to prevent build up of costs on AWS. It runs on the free tier instance so it is slower.
   2. To deploy the backend, we need to first make a Render account and connect our Github account to it. We can then access our project repository. Then we create a web service since we are deploying a server. We specify which branch and which directory our server files are in. Then we specify the start commands to run the server and finally click deploy.
   3. The URLS to access the servers are as follows:
      1. Node Server: <https://p10-odysseum.onrender.com>
      2. LLM Flask Server: <https://p10-odysseum-1.onrender.com>
3. There are 2 main frontends:
   1. React Native App
      1. Method 1:
         1. On your Android phone, download the Expo Go app from the following: <https://d1ahtucjixef4r.cloudfront.net/Exponent-2.31.2.apk>
         2. This will download an APK file and you must install it on your phone.
         3. Make sure the Expo Go app supports Expo SDK v51 by checking the settings tab..
         4. Now on your machine:
         5. For the frontend to work, clone the repository and navigate to Development/Sprint-4/Code/odysseum-frontend
         6. Install the required packages using "npm install".
         7. In the terminal, run "npx expo start" or "npm run start".
         8. This will start the app display a QR Code in the terminal.
         9. Open Expo App on you phone, select scan QR code and scan the code. This will bundle and open up the app and you will now be able to navigate and use it.
      2. Method 2:
         1. Download the APK file form the following URL: https://expo.dev/accounts/group10/projects/odysseum/builds/cc731acc-be9a-43e6-9d51-c985665f336f
         2. Install the file in you phone.
         3. For the frontend to work, clone the repository and navigate to Development/Sprint-4/Code/odysseum-frontend
         4. Install the required packages using "npm install".
         5. In the terminal, run "npx expo start" or "npm run start".
         6. This will start the app display a QR Code in the terminal.
         7. You must switch to development mode in order to run the APK app. You can do this by pressing the “S” key in the terminal.
         8. This will switch to development mode and display a new QR code.
         9. Open the installed apk in your phone and select to scan the QR code from your phone camera. Scan the displayed QR code in the terminal and this will bundle and open up the app and you will now be able to navigate and use it.
      3. To use the app you must login or register. If you do not wish to register you can use the following credentials to login:
         1. Email: usman1
         2. Password: Usm@n123
   2. Admin Dashboard
      1. This is also deployed as a static site on Render.
      2. URL: [odysseum-admin.onrender.com](http://odysseum-admin.onrender.com)
      3. Credentials to sign in:
         1. Email: affantest
         2. Password: Aff@n123

# Conclusion

## Summary

Odysseum was envisioned as a seamless travel companion that empowers users to discover, plan, and experience Pakistan like never before. From initial brainstorming to final deployment, we adopted an Agile (Scrum) development methodology, enabling iterative improvements based on continuous feedback. Our system architecture was carefully designed to separate concerns across layers — ensuring scalability, security, and maintainability. We successfully integrated a mobile application built on React Native with a robust backend powered by Node.js, MongoDB, and Firebase, while also leveraging Python Flask microservices for AI-based features such as itinerary generation and review summarization. Through this project, we enhanced our technical expertise in full-stack development, cloud deployment, AI integration, and project management, while learning valuable lessons about the complexities of building production-grade applications.

## Challenges

**LLM Integration:** Implementing accurate, context-aware responses from the itinerary generator and review summarizer posed a significant challenge. Fine-tuning prompts, handling token limits, and ensuring meaningful outputs for diverse user queries required rigorous experimentation and iterative refinement.

**Dual Storage Management (MongoDB + Firebase):** Managing two storage systems — MongoDB for structured data and Firebase for large media files — introduced synchronization complexities, particularly for associating user-uploaded images with posts, businesses, and locations without latency or data consistency issues.

**Real-Time Features:** Implementing efficient real-time communication (chats, notifications) via sockets while maintaining performance under load demanded careful server architecture optimization.

**Testing Across Devices:** Since Odysseum was developed as a mobile-first application, ensuring UI/UX consistency and responsiveness across different Android and iOS devices required extensive manual testing and adjustments.

## Future

**Augmented Reality (AR) Tours:** Allow users to experience virtual walkthroughs of historical landmarks and cultural sites using AR, enriching the travel planning phase.

**Gamification Elements:** Introduce travel badges, achievement points, and leaderboards to encourage user engagement and community building, much like platforms such as Reddit or Duolingo.

**AI Travel Assistant:** Further enhance the LLM capabilities to act as a full travel concierge, suggesting dynamic itineraries, updating plans in real-time based on weather or traffic conditions, and answering location-specific queries.

**Offline Mode:** Allow travelers to access their itineraries, maps, and essential business listings offline, catering to remote area travel where internet access may be limited.

**Expanded Multilingual Support:** Add more local and international languages to cater to a broader range of travelers and business owners, helping promote tourism inclusively.

**Personalized Marketing for Businesses:** Enable local businesses to run targeted promotions/offers to tourists based on user travel interests and itineraries, boosting local economies.

# Review checklist

Before submission of this report, the team must perform an internal review. Each team member will review one or more sections of the deliverable.

| **Chapter/Section Name** | **Reviewer Name(s)** |
| --- | --- |
| Section 1 | Shahraiz Chishty |
| Section 7 | Mohammad Haroon Khawaja |
| Section 10 | Shahrez Aezad |
| Section 11 | Muhammad Affan Naved |
| Section 12 | Luqman Aadil |
| All other sections were mostly a copy paste of previous documents submitted to the supervisor and are available on the repository with the exception of a few diagram updates. |  |

# References