# Software System Design Documentation

Project Name:

**Roamly - Generative AI Travel Itinerary Planner**

Instructor: Professor Santosh
Student: Liyan Alkhalidy

Table of Contents

# 1 Introduction

## 1.1 Purpose

The purpose of Roamly is to provide travelers with a smart, personalized, and effortless way to plan multi-day travel itineraries using generative AI technology. By leveraging user preferences such as destination, travel dates, budget, interests, and travel style, Roamly automatically generates customized day-by-day travel plans that include suggested activities, restaurants, hotels, and estimated costs.

This system eliminates the time-consuming manual process of trip planning by integrating with AI language models and third-party travel APIs to deliver intelligent, adaptive recommendations in real time. Roamly is designed to support both registered users who wish to save and manage multiple trips and guest users who simply want quick and simple itinerary generation.

The goal is to enhance the travel planning experience through automation, personalization, and usability, making travel more accessible, inspiring, and tailored to individual needs.

## 1.2 Scope

Roamly is a web-based and a mobile application travel planning system that uses generative AI to create personalized itineraries based on user input. The system allows travelers to specify their destination, travel dates, budget, interests, and travel style in order to receive a customized, day-by-day itinerary with suggested activities, hotels, restaurants, and estimated costs.

The scope of Roamly includes:

- Collecting and processing user travel preferences through an intuitive interface
- Generating personalized itineraries using an AI itinerary engine
- Integrating with external APIs to retrieve flight and hotel options
- Estimating trip costs based on selected components
- Allowing users to view, edit, and save generated itineraries
- Supporting both guest users and registered users with account-specific features
- Ensuring scalability, usability, and modularity to support future enhancements

Roamly does not handle actual travel bookings, real-time availability verification, or payment processing. Instead, it focuses on delivering accurate and intelligent recommendations that guide users in their trip planning process.

## 1.3 Stakeholders

The following stakeholders are involved in or affected by the development and use of the Roamly system:

- **Registered Travelers**
  Individuals who create accounts to access personalized features such as saving itineraries, editing trip plans, and tracking past trips. They are the primary users and directly interact with the system.
- **Guest Travelers**

Users who access the application without registering. They benefit from limited features like generating a one-time itinerary based on their preferences. Their needs help shape the system's accessibility and ease of use.

- **System Administrator**
  Responsible for managing system configurations, destination metadata, and ensuring that AI outputs and third-party integrations operate smoothly. The admin may also oversee feedback handling and usage analytics.
- **Product Owner / Business Sponsor**
  The person or entity funding and overseeing the Roamly project. They define the business goals, priorities, and long-term vision for the system.
- **Development Team**
  Software engineers, UX designers, and testers involved in building and maintaining Roamly. They use this document to understand requirements and translate them into a working system.
- **API Providers**
  External travel services like OpenAI (for AI generation), Skyscanner, Booking.com, and Google Maps that offer APIs integrated into Roamly. They are not users, but the system relies on their services.

# 2 Business Requirements

## 2.1 Problem Statement

Planning a multi-day trip is often time-consuming, overwhelming, and fragmented. Travelers must manually search for flights, hotels, and activities across multiple platforms while trying to organize everything into a coherent itinerary. This process can be especially frustrating for users who have limited travel experience, are short on time, or don’t know where to start.

While there are tools that offer static recommendations or booking options, they typically lack personalization, contextual understanding of preferences, and the ability to generate full-day schedules that align with budget, interests, and travel style.

There is a need for a smarter, faster, and more personalized solution that simplifies the travel planning process from start to finish. A system that leverages generative AI can address this gap by producing dynamic, custom itineraries based on user input, freeing travelers from manual research and helping them make the most of their trip.

## 2.2 Functional Capabilities

Roamly provides the following core functional capabilities to support intelligent and personalized travel planning:

**1. User Management**

- Allow users to register, log in, and manage their profiles
- Support guest access for one-time itinerary generation
- Save and retrieve past itineraries for registered users

**2. Trip Preferences Collection**

- Capture destination, travel dates, budget, interests, and travel style
- Support inputs like number of travelers, trip length, and preferred activity types

**3. AI-Powered Itinerary Generation**

- Use generative AI to create multi-day itineraries based on user preferences
- Suggest activities, restaurants, and time slots for each day
- Ensure suggestions align with budget, travel style, and timing

**4. Travel API Integration**

- Retrieve real-time flight options based on destination and dates
- Provide hotel options filtered by location, price, and preferences
- Optionally show distances and travel times via map services

**5. Itinerary Management**

- Let users view, edit, and fine-tune generated itineraries
- Allow users to remove or rearrange items in the itinerary
- Support export to PDF or sharing via email

**6. Budget Estimation**

- Calculate total estimated trip cost based on selected items
- Provide a detailed cost breakdown: flights, hotels, food, activities

## 2.3 Target Users and Their Needs

Roamly is designed for a broad range of travelers who seek a convenient, intelligent, and personalized way to plan their trips. The primary target users and their needs are outlined below:

**1. Casual Travelers**

Individuals or families planning occasional leisure trips that will need quick, personalized itinerary suggestions without doing extensive research. Needs**:**

- Easy-to-use interface
- Budget-friendly travel options
- Automatically generated daily plans

**2. Busy Professionals**

Users with limited time to plan trips who need a fast and organized travel planning experience. Needs:

- Fast itinerary generation based on minimal input (by entering only destination, travel dates, and budget)
- Smart recommendations aligned with available time and budget
- Exportable itineraries for quick access on the go (downloaded or emailed)

**3. First-Time Travelers**

Users with little travel experience who may not know where to start or what to include in an itinerary. Needs:

- Guided input forms with helpful defaults (common destinations, budget ranges, interests)
- Suggested destinations and activities
- Day-by-day structure with helpful tips

**4. Digital Nomads and Frequent Travelers**

Experienced users who travel often and want optimized plans based on their preferences. Needs:

- Saved profiles and travel history
- More control over editing and refining itineraries

- Destination variety (popular spots, lesser-known places) and advanced filtering (activity types, travel pace)

**5. Guest Users (Unregistered)**

Visitors who want to try out the tool before creating an account. Needs:

- Simple, no-login trip generation
- A sample itinerary to explore the system's capabilities (cannot save, edit, export itinerary)
- Quick preview of how personalized AI can help them

## 2.4 Business Goals

The primary goal of Roamly is to provide an intelligent, user-friendly platform that simplifies travel planning through the use of generative AI. The system aims to deliver real value to users by saving time, increasing personalization, and enhancing the overall travel experience.

**Key Business Goals:**

- **Streamline trip planning** by offering fast, AI-generated itineraries that reduce manual effort
- **Attract a wide user base** by supporting both guest access and account-based personalization
- **Enhance user engagement** with editable, exportable, and downloadable itineraries
- **Build trust and adoption** through high-quality recommendations and an intuitive interface
- **Differentiate from traditional tools** by offering real-time suggestions based on budget, interest, and travel style
- **Enable future monetization opportunities**, such as premium features, affiliate travel bookings, or user subscriptions
- **Ensure scalability** so the system can grow to support more destinations, languages, and advanced planning options

## 3 Non-Functional Requirements

### 3.1 Performance

Roamly must respond quickly and handle user load smoothly to ensure a seamless planning experience.

- The AI-generated itinerary should be returned within 10 seconds under normal load. This accounts for real-world delays in calling OpenAI and travel APIs.
- The system should comfortably handle 50 concurrent users in its first version, using a basic cloud backend (like Azure App Service).
- Travel APIs (flights, hotels) can be cached for a few minutes to reduce repeated lookups and improve response time.

### 3.2 Security

User data and interactions must be protected using standard, reliable security practices.

- All traffic will use HTTPS for secure transmission.
- User passwords will be hashed and salted (e.g., using ASP.NET Identity).
- Basic role-based access control will restrict guests from accessing user-only features.
- Secure token-based sessions (e.g., JWT) will be used to manage login session

### 3.3 Maintainability

The system should be easy to update, fix, and expand over time.

- The system will follow modular architecture, separating frontend, backend, and AI logic into services or layers.
- Clear naming conventions and basic documentation will be used to support future developers.
- Cloud deployment will be managed through scalable Azure services, such as Azure App Service and Azure Functions, for hosting backend logic and APIs.

### 3.4 Other Non-Functional Requirements

**Usability**

- The interface should be simple and enjoyable to use, even for first-time visitors.
- A mobile-first, responsive UI will support both desktop and smartphone users.
- Forms will include smart defaults, tooltips, and guided steps to simplify input.

**Scalability**

- Roamly should be able to grow its user base and data volume with minimal changes.
- Hosted on Azure Cloud platform that allows scaling on demand.
- Designed to handle growth in destinations, user profiles, and integrations.

**Availability**

- The system should be accessible and functioning most of the time, even during updates.
- The initial target is 95% uptime.
- Use of deployment pipelines and error tracking will ensure quick recovery from issues.

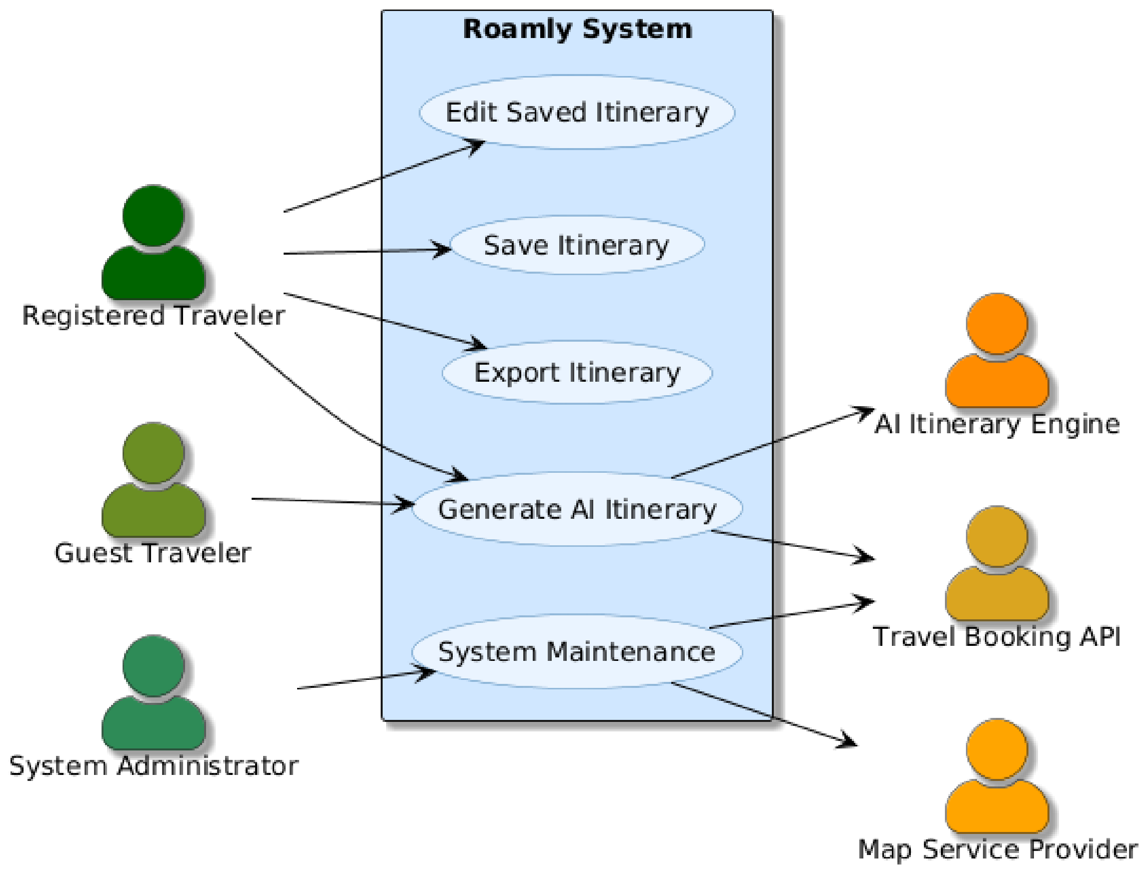
**Portability**

- Roamly should work across devices and browsers without compatibility issues.
- Accessible via all modern browsers.
- Built with responsive design for use on tablets and mobile devices.

# 4 System Design and Modeling

## 4.1 Use Case

### 4.1.1 Use Case Diagram



### 4.1.2 Actors Table

| Type | Actor | Goal Description |
|---|---|---|
| **Primary** | Registered Traveler | Logged-in user who can set preferences, generate, save, edit, and export personalized itineraries. |
| | Guest Traveler | Unregistered user with limited access. Can generate one-time itineraries and preview a sample itinerary. |
| | System Administrator | Maintains the platform, updates destination content, manages feedback, and monitors API health |
| **Supporting** | AI Itinerary Engine | Internal module that uses generative AI to build customized, day-by-day travel plans. |
| | Travel Booking API | Connects to third-party providers (e.g., Skyscanner, Booking.com) for real-time hotel and flight data. |
| **Offstage** | AI Provider | External service hosting the large language model (e.g., OpenAI or Azure OpenAI) used by the AI engine. |

| | | |
|---|---|---|
| | Map Service Provider | External mapping service (e.g., Google Maps API) that provides route, distance, and travel time information. |
| | Travel Companion | A user-invited friend, family member, or partner whose preferences influence the generated itinerary but who does not interact with the system directly. |

### 4.1.3 Use Case Requirements

| Use Case Name | Primary Actor(s) | Supporting Actor(s) | Brief Description |
|---|---|---|---|
| **1. Generate AI Itinerary** | Registered Traveler, Guest Traveler | AI Itinerary Engine, Travel Booking API | Users enter destination, dates, and preferences. The AI engine generates a personalized itinerary using real-time travel data and companion preferences. |
| **2. Save Itinerary** | Registered Traveler | - | Logged-in users save generated itineraries to their profile for future editing or export. |
| **3. Edit Saved Itinerary** | Registered Traveler | - | Users can modify saved itineraries by changing days, activities, or locations to fit evolving needs. |
| **4. Export Itinerary** | Registered Traveler | - | Users export their finalized itinerary as a PDF or shareable link for easy access while traveling. |
| **5. System Maintenance** | System Administrator | Travel Booking API, Map Service Provider | Admin monitors third-party API health, updates destination content, and ensures system functionality. |

### 4.1.4 Fully-Dressed Use Cases

| Use Case Section | Comment |
|---|---|
| **Use Case Name** | Generate AI Itinerary |
| **Goal in Context** | The Registered or Guest Traveler wants to generate a personalized, day-by-day travel itinerary by entering destination, dates, and preferences into Roamly. |
| **Scope** | Roamly System |
| **Level** | User-goal |
| **Primary Actor(s)** | Registered Traveler, Guest Traveler |
| **Supporting Actor(s)** | AI Itinerary Engine, Travel Booking API |
| **Stakeholders and Interests** | • Traveler wants a complete itinerary that fits their budget, interests, and timeline.<br>• AI Provider wants to ensure model output is accurate and relevant.<br>• Travel Booking Partners want their flight and hotel options to appear in Roamly so users can book through them.<br>• Travel Companion wants their preferences considered without direct interaction.<br>• System Admin wants smooth API integration and minimal errors. |

| | |
|---|---|
| **Preconditions** | • The system is online and AI engine is accessible.<br>• Guest user or logged-in user has entered required details (destination, date, preferences). |
| **Postconditions** | A detailed itinerary is generated and displayed to the user, ready for saving or editing. |
| **Main Success Scenario** | 1. The traveler opens the itinerary planner form.<br>2. They enter destination, travel dates, budget, and preferences.<br>3. The system validates inputs.<br>4. The AI engine builds a structured prompt using user and companion preferences.<br>5. The LLM (via AI Provider) returns an initial itinerary.<br>6. The system enriches it with real-time travel data via the Travel Booking API.<br>7. The system queries the Map Service to estimate travel durations.<br>8. The full itinerary is assembled and shown to the user. |
| **Extensions** | 8a. User chooses to share itinerary → System generates a shareable link.<br>8b. User requests PDF version → System prepares a downloadable itinerary in PDF format.<br>8c. User clicks 'Regenerate' → System re-runs AI with the same or modified preferences. |
| **Special Requirements** | • Itinerary must load within 5 seconds under normal load.<br>• AI prompts must be structured to prioritize personalization.<br>• Companion preferences must be merged logically with user preferences.<br>• Allow retry if generation fails. |

| Use Case Section | Comment |
|---|---|
| **Use Case Name** | Save Itinerary |
| **Goal in Context** | The Registered Traveler wants to save a generated itinerary to their account for future reference, editing, or export. |
| **Scope** | Roamly System |
| **Level** | User-goal |
| **Primary Actor(s)** | Registered Traveler |
| **Supporting Actor(s)** | - |
| **Stakeholders and Interests** | • Registered Traveler wants to access saved itineraries across devices and modify them later.<br>• System Admin wants consistent data storage and access reliability. |
| **Preconditions** | • The user is logged in.<br>• A generated itinerary is displayed and ready to be saved. |
| **Postconditions** | The itinerary is saved and stored in the user's account profile. |
| **Main Success Scenario** | 1. User views a generated itinerary.<br>2. User clicks "Save Itinerary".<br>3. System validates that the user is logged in.<br>4. System stores the itinerary in the user's account.<br>5. Confirmation message is shown. |

| | |
|---|---|
| **Extensions** | 5a. User chooses to name the itinerary → System saves it with a custom name.<br>5b. User adds tags (e.g., "honeymoon", "Europe") → System saves tags for filtering. |
| **Special Requirements** | • Saved itineraries must sync to the user profile and be available on both web and mobile.<br>• System must prevent duplicate save names per user. |

| **Use Case Section** | **Comment** |
|---|---|
| **Use Case Name** | Edit Saved Itinerary |
| **Goal in Context** | The Registered Traveler wants to make changes to a previously saved itinerary, such as updating dates, changing destinations, or modifying daily activities. |
| **Scope** | Roamly System |
| **Level** | User-goal |
| **Primary Actor(s)** | Registered Traveler |
| **Supporting Actor(s)** | - |
| **Stakeholders and Interests** | • Registered Traveler wants to personalize plans to fit changes in availability, interest, or companion preferences.<br>• System Admin wants edits to be stored correctly and not overwrite critical data. |
| **Preconditions** | • The user is logged in.<br>• A saved itinerary exists in the user's profile. |
| **Postconditions** | The updated itinerary is saved, replacing the older version or stored as a new version if selected. |
| **Main Success Scenario** | 1. User accesses their profile.<br>2. User selects a saved itinerary.<br>3. User clicks "Edit".<br>4. System loads itinerary in edit mode.<br>5. User makes changes to days, activities, or preferences.<br>6. User saves changes.<br>7. System updates the itinerary and confirms success. |
| **Extensions** | 6a. User saves as new version → System stores itinerary as a separate entry.<br>6b. User adds notes → System appends private notes to the day or activity. |
| **Special Requirements** | • All changes must auto-save in case of accidental browser closure.<br>• Itinerary edit history should be optionally accessible. |

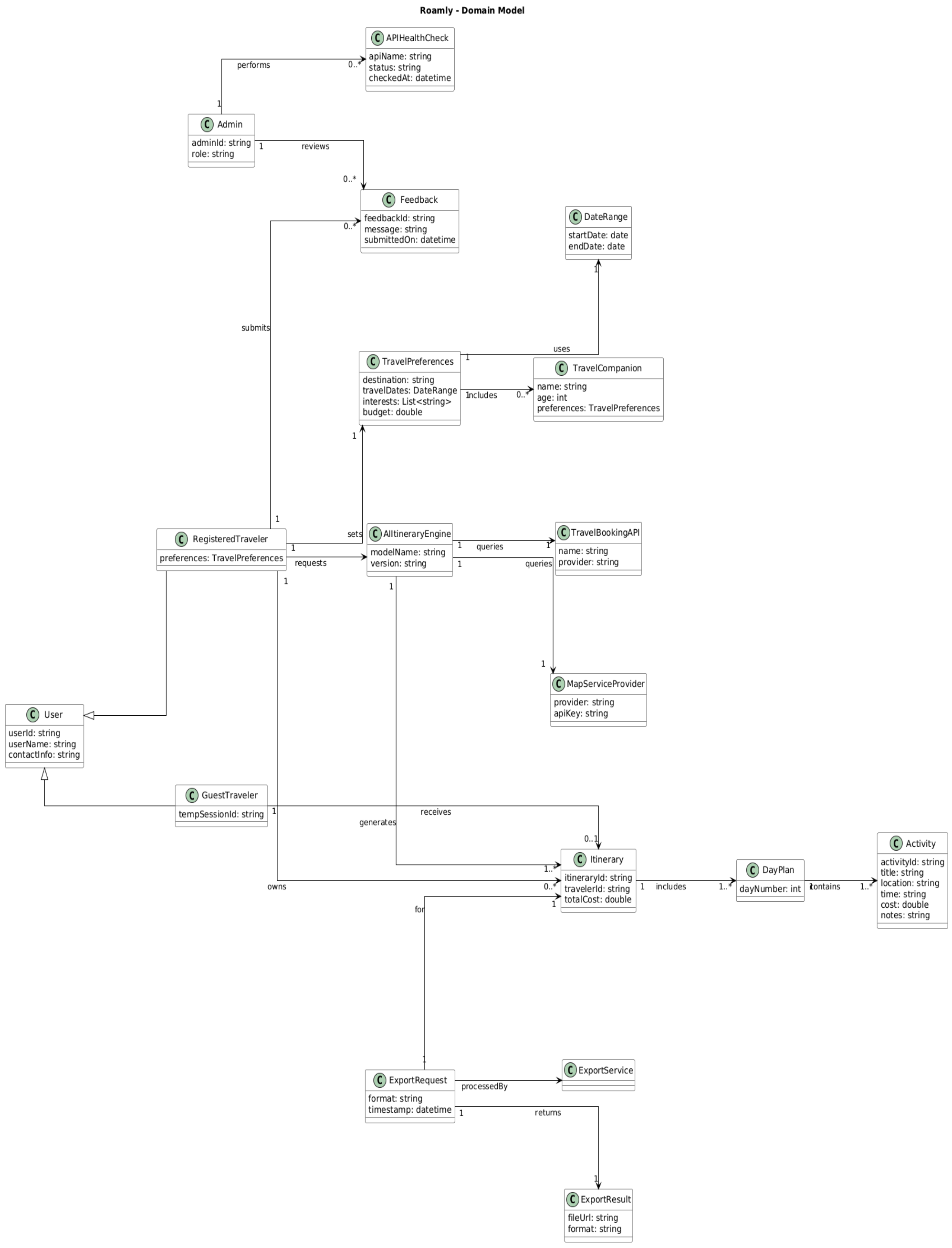
| **Use Case Section** | **Comment** |
|---|---|
| **Use Case Name** | Export Itinerary |
| **Goal in Context** | The Registered Traveler wants to download or share their itinerary in a convenient format (e.g., PDF, email, or shareable link) for offline access or to send to others. |
| **Scope** | Roamly System |
| **Level** | User-goal |

| | |
|---|---|
| **Primary Actor(s)** | Registered Traveler |
| **Supporting Actor(s)** | - |
| **Stakeholders and Interests** | • Registered Traveler wants easy access to their itinerary during travel or to send it to others.<br>• Travel Companion may want to review shared plans. |
| **Preconditions** | • User is logged in.<br>• At least one saved itinerary exists |
| **Postconditions** | The itinerary is downloaded, shared via link, or emailed. |
| **Main Success Scenario** | 1. User navigates to a saved itinerary.<br>2. User clicks "Export".<br>3. System prompts export options (PDF, email, link).<br>4. User selects desired format.<br>5. System generates and delivers the export. |
| **Extensions** | 5a. User adds a message to the email share → System includes custom message in email.<br>5b. User selects "Add to Calendar" → System provides calendar file download (ICS). |
| **Special Requirements** | • PDF export must include branding, dates, daily plans, and destination images.<br>• Export links should expire after 7 days if not accessed. |

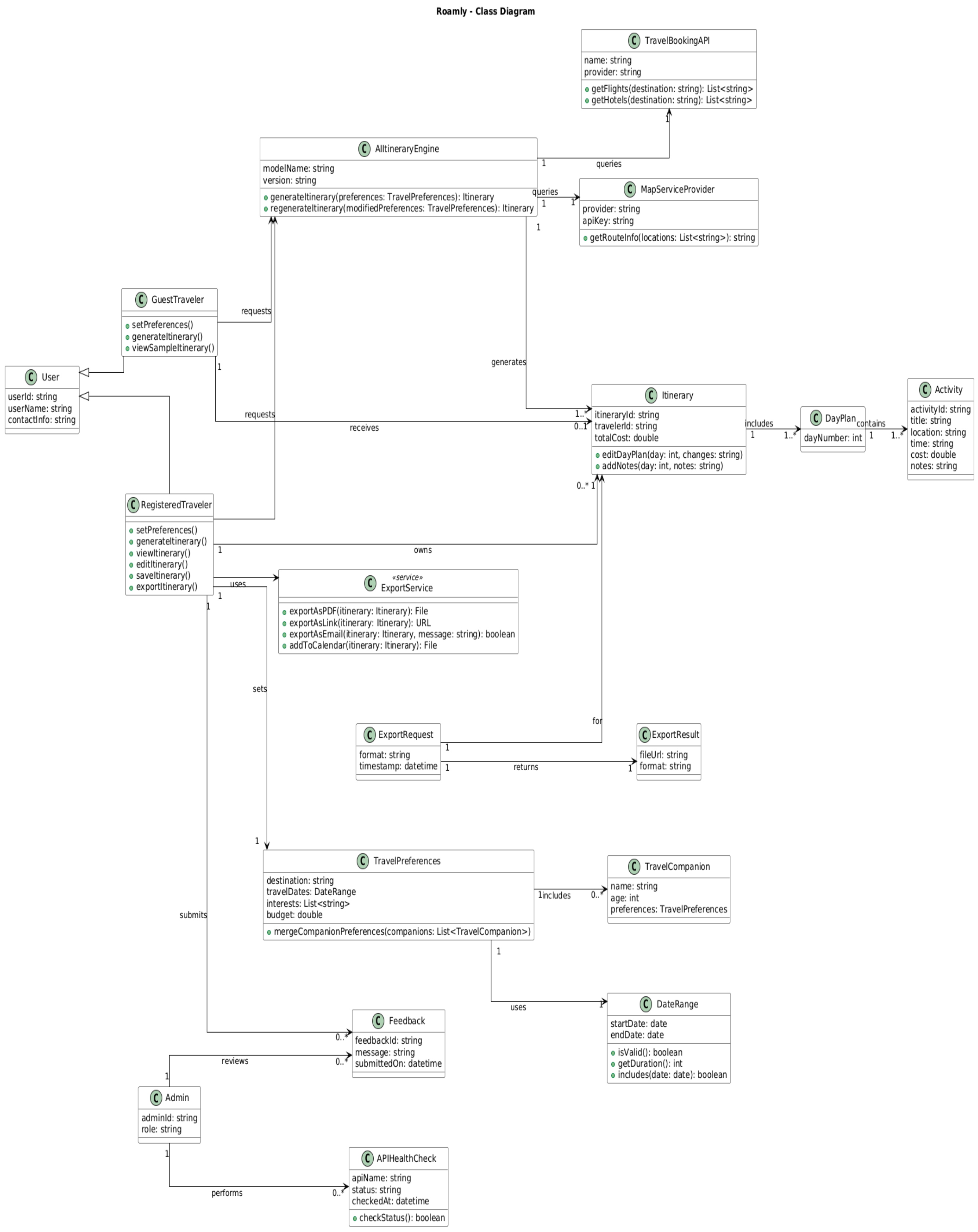
| Use Case Section | Comment |
|---|---|
| **Use Case Name** | System Maintenance |
| **Goal in Context** | The System Administrator wants to monitor system performance, manage feedback or error logs, and ensure third-party APIs (AI, map, and booking) are running smoothly. |
| **Scope** | Roamly System |
| **Level** | User-goal |
| **Primary Actor(s)** | System Administrator |
| **Supporting Actor(s)** | AI Itinerary Engine, Travel Booking API |
| **Stakeholders and Interests** | • System Administrator wants the system to run reliably and securely.<br>• End users expect minimal downtime and accurate results.<br>• Third-party API providers want to be alerted if their services are interrupted. |
| **Preconditions** | • Admin is authenticated and has access to monitoring tools.<br>• Logs and system health checks are enabled. |
| **Postconditions** | The system remains stable, or issues are detected and appropriate action is taken. |
| **Main Success Scenario** | 1. Admin logs into the admin dashboard.<br>2. Admin accesses monitoring tools.<br>3. Admin reviews system health metrics and API statuses.<br>4. Admin views or resolves any error logs or flagged issues.<br>5. Admin generates a status report or takes action (restart service, report issue). |
| **Extensions** | 3a. Admin schedules a maintenance window → System sends alerts to users in advance. |

| | |
|---|---|
| | 4a. Admin logs API timeout → System logs the issue for SLA tracking. |
| **Special Requirements** | • System status and logs should auto-refresh every 1–5 minutes.<br>• Alerts should be sent via email or Slack for critical failures. |

## 4.2 Domain Model

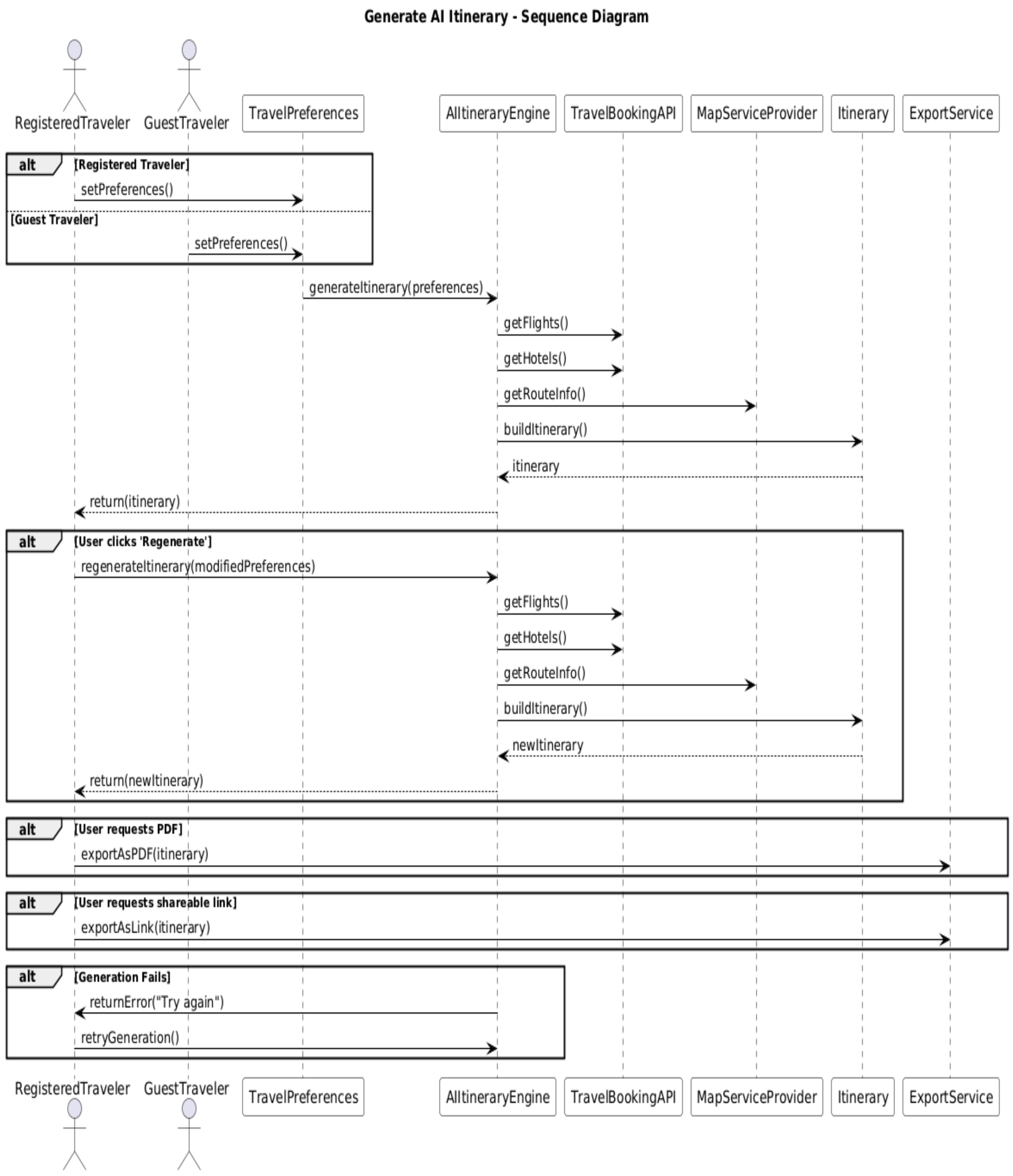


## 4.3 Class Diagram

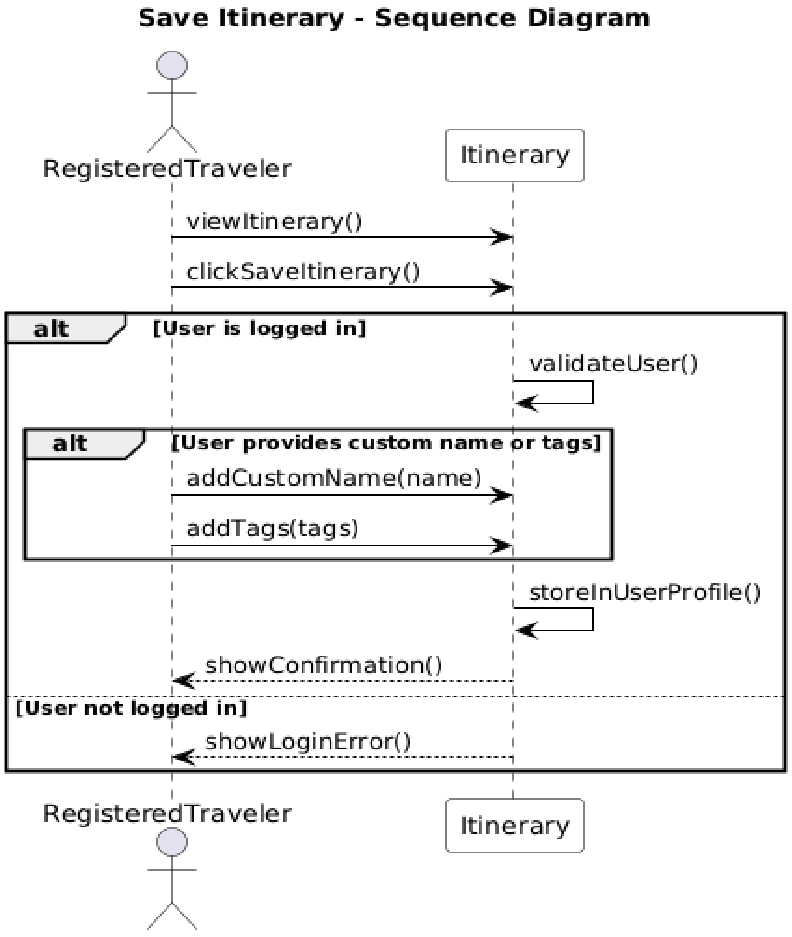


## 4.4 Sequence Diagrams

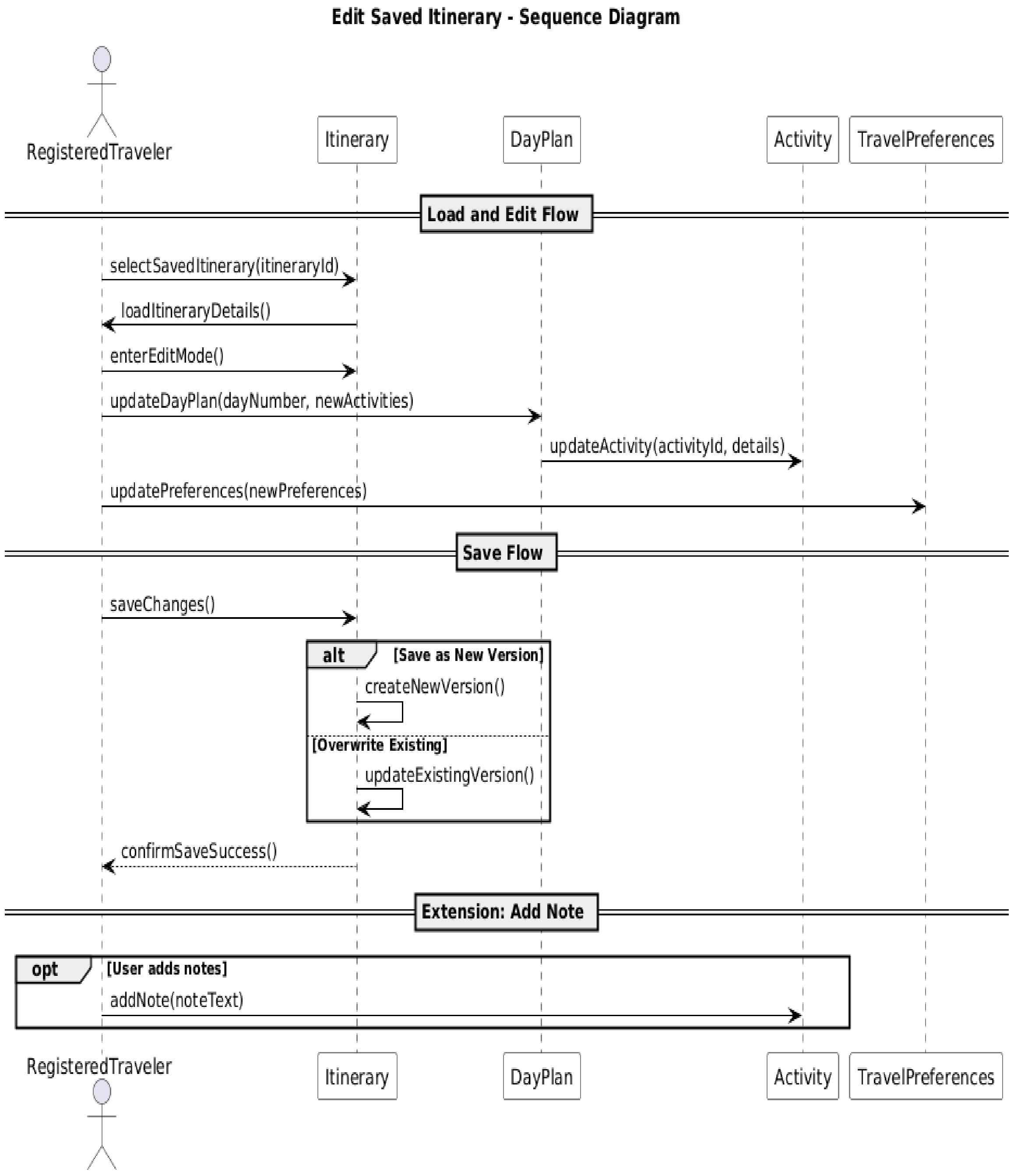
- Generate AI Itinerary Use Case:



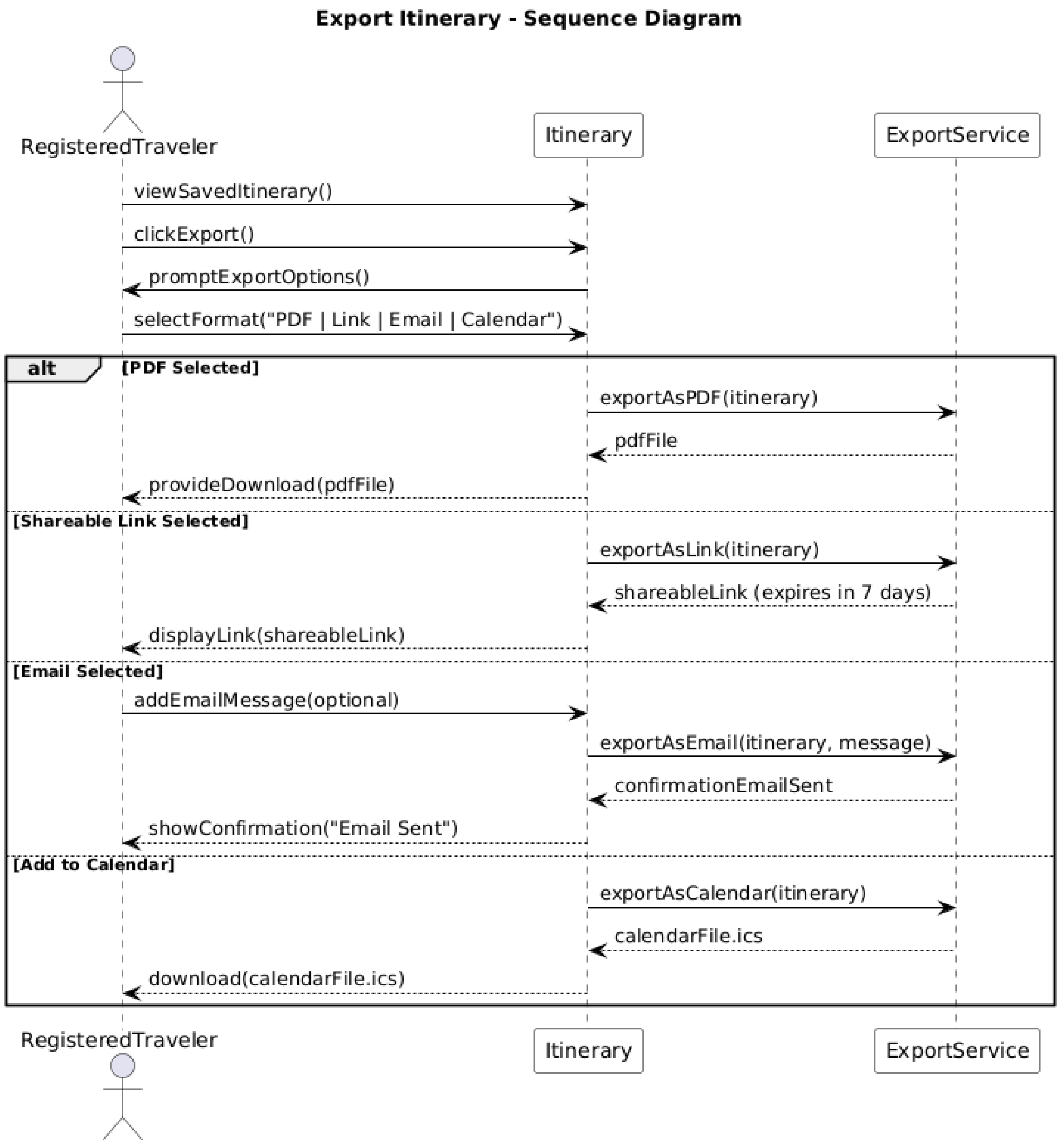
- Save Itinerary Use Case:



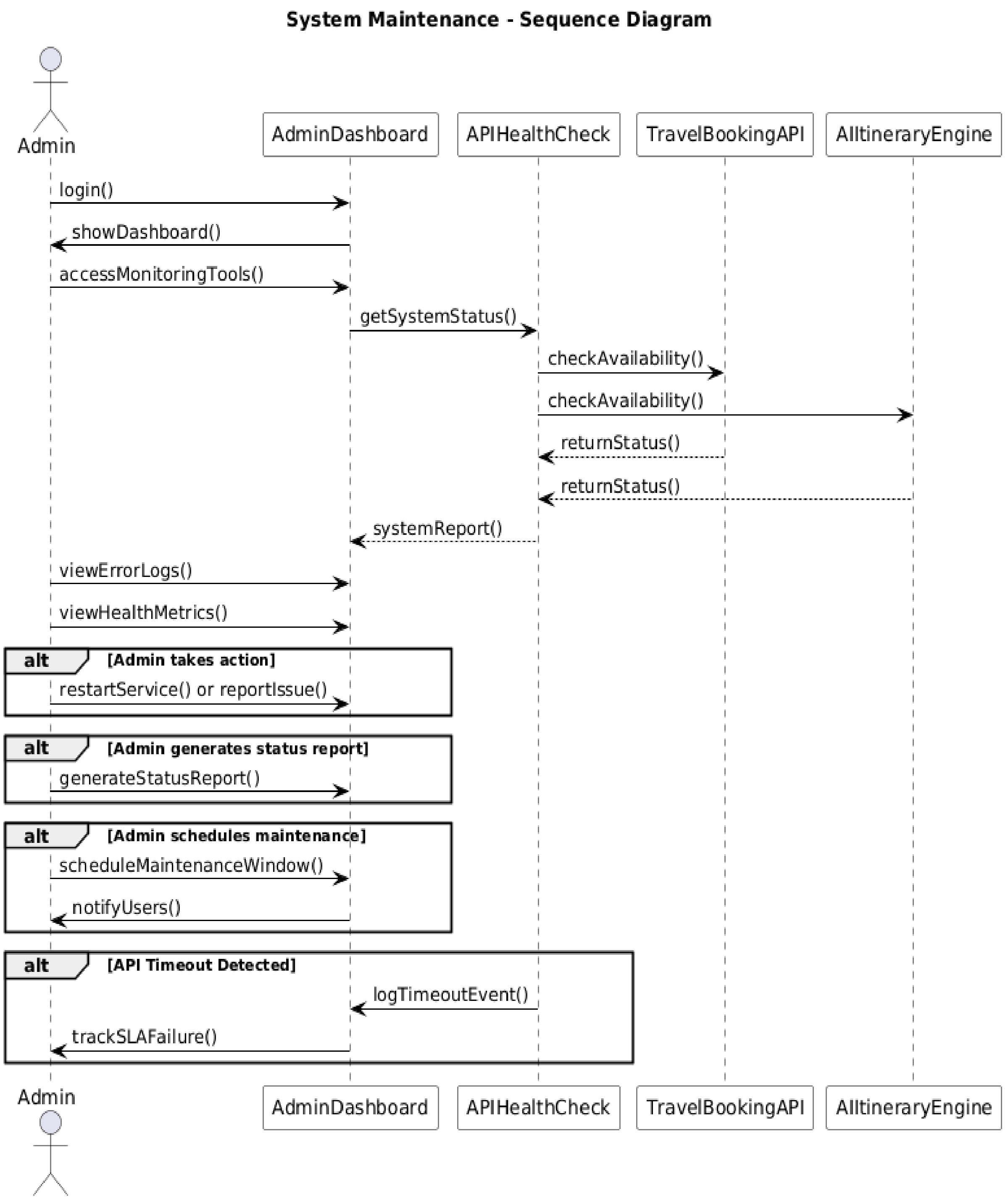
- Edit Saved Itinerary Use Case:



- Export Itinerary Use Case:


Export Itinerary - Sequence Diagram
RegisteredTraveler
Itinerary
ExportService
viewSavedItinerary()
clickExport()
promptExportOptions()
selectFormat("PDF | Link | Email | Calendar")
alt
[PDF Selected]
exportAsPDF(itinerary)
pdfFile
provideDownload(pdfFile)
[Shareable Link Selected]
exportAsLink(itinerary)
shareableLink (expires in 7 days)
displayLink(shareableLink)
[Email Selected]
addEmailMessage(optional)
exportAsEmail(itinerary, message)
confirmationEmailSent
showConfirmation("Email Sent")
[Add to Calendar]
exportAsCalendar(itinerary)
calendarFile.ics
download(calendarFile.ics)
RegisteredTraveler
Itinerary
ExportService

- System Maintenance Use Case:

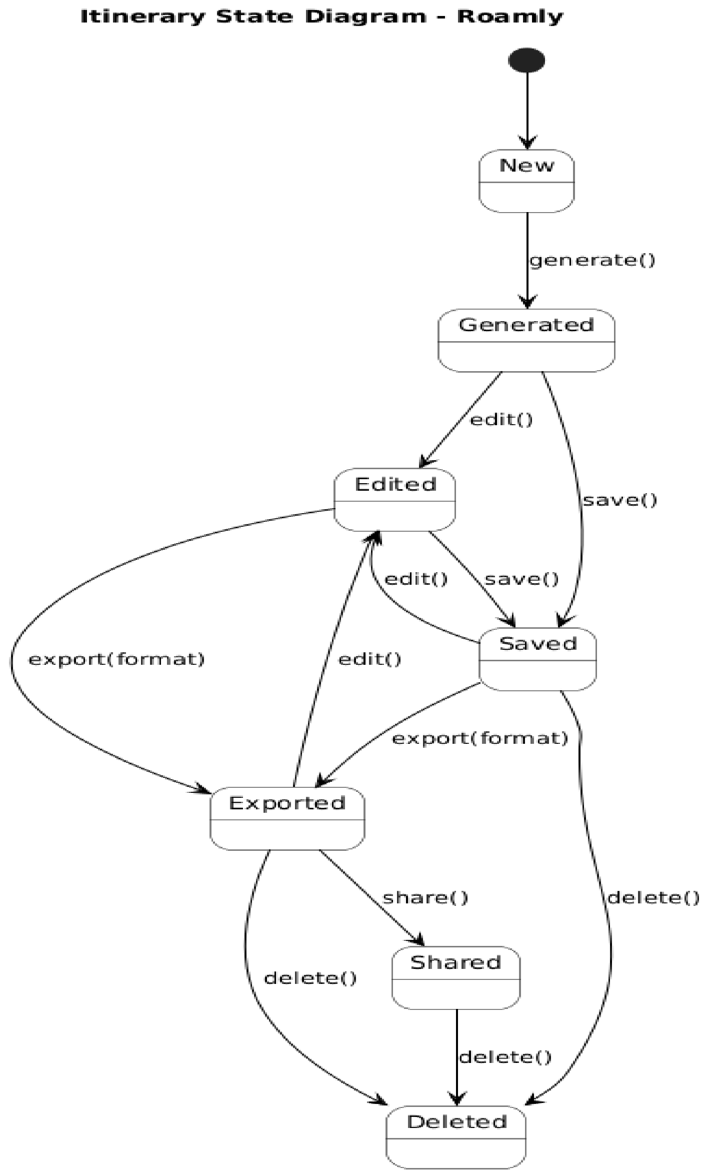


## 4.5 State Diagram

The state diagram below illustrates the lifecycle of an Itinerary object within the Roamly system. It captures the transitions between different states such as creation, editing, saving, exporting, and sharing, based on user actions and system behavior. This diagram aligns directly with the defined use cases and class diagram behavior:

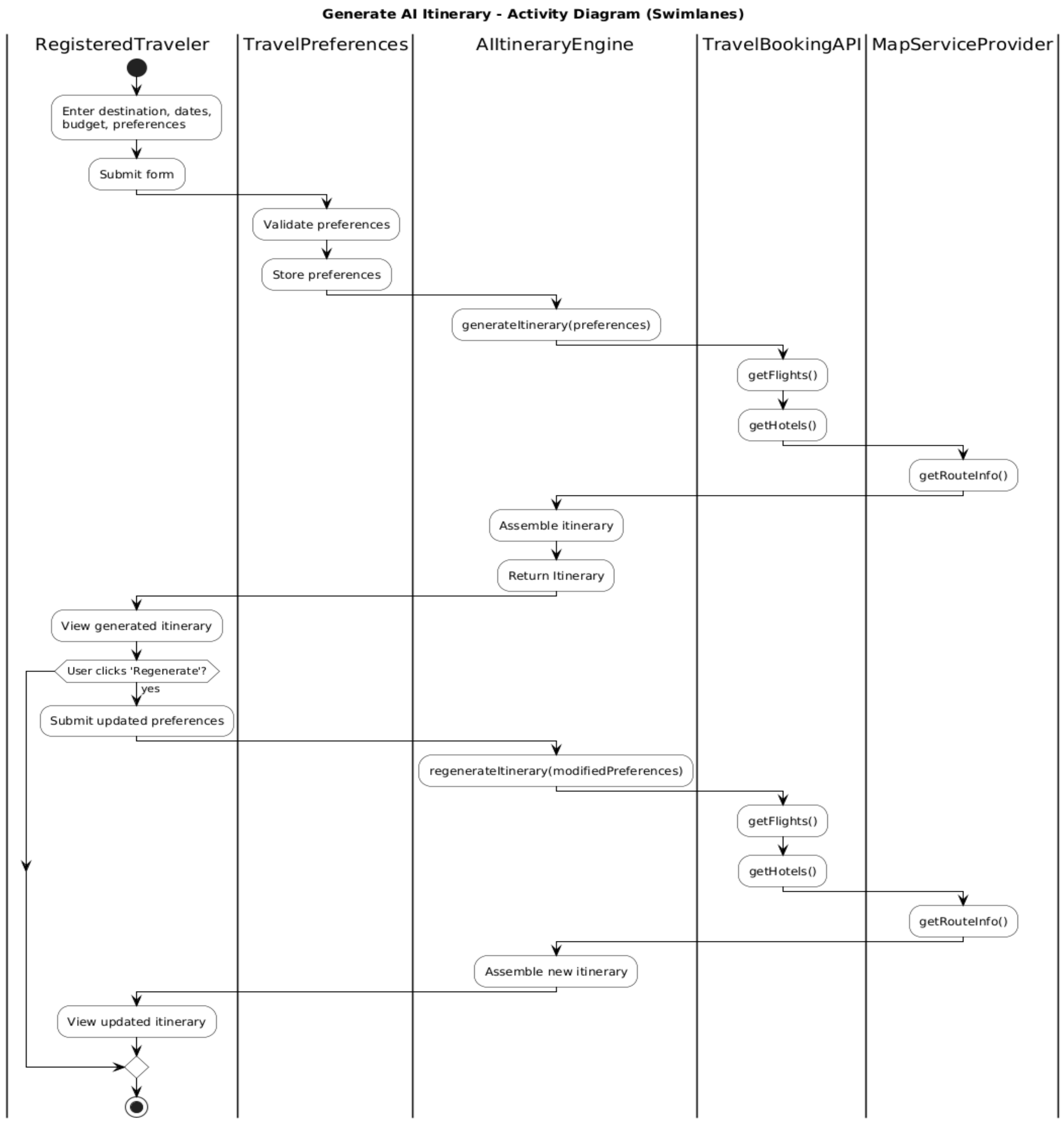
- generateItinerary() triggers the transition from New to Generated.
- editItinerary() and saveItinerary() are reflected in the transitions to Edited and Saved states.
- exportAsPDF() and exportAsLink() lead to the Exported and Shared states respectively.

By modeling these state changes, the diagram provides a clear understanding of how an itinerary evolves over time in response to user interactions and system operations

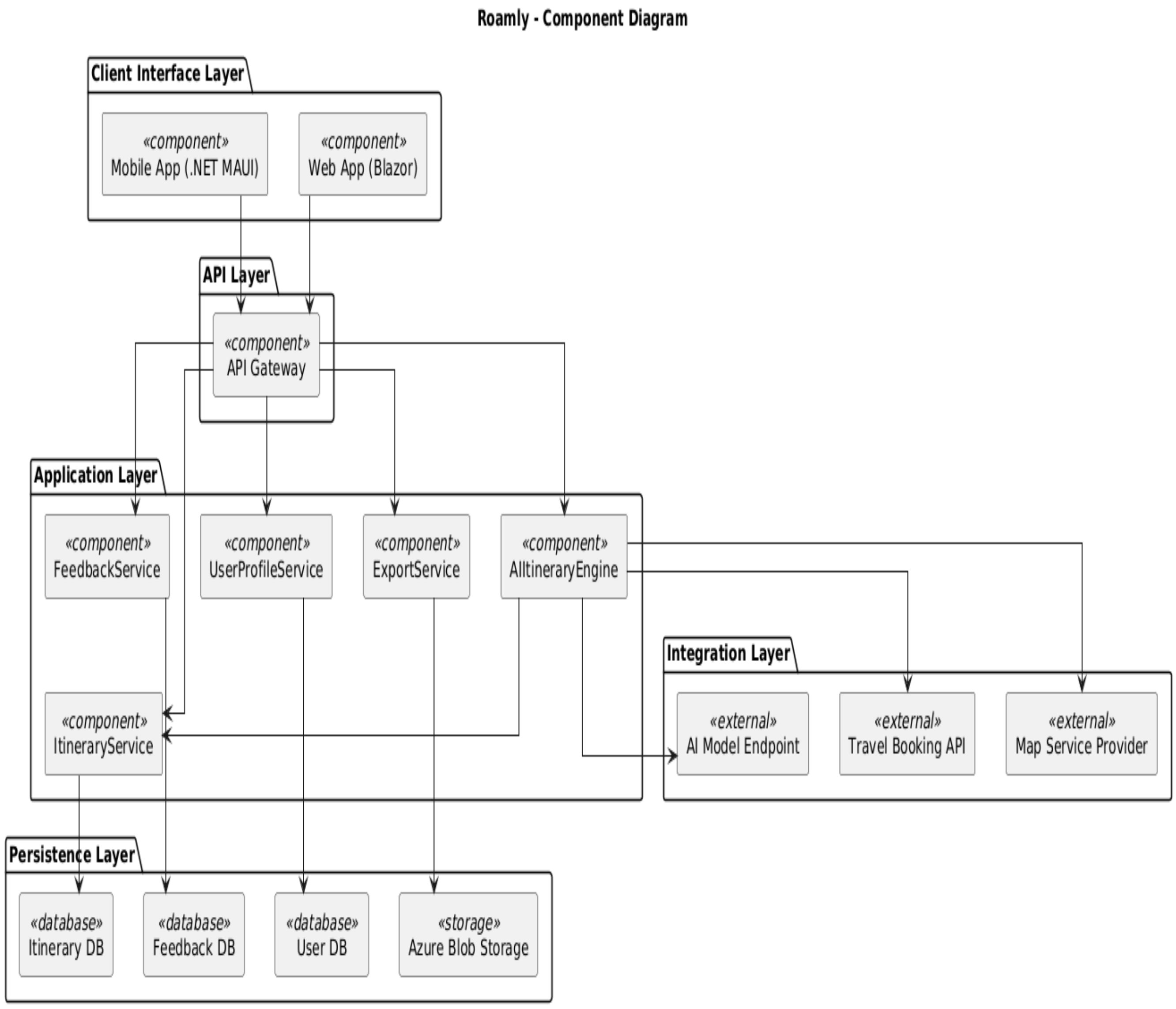


## 4.6 Activity Diagram (Swimlane)

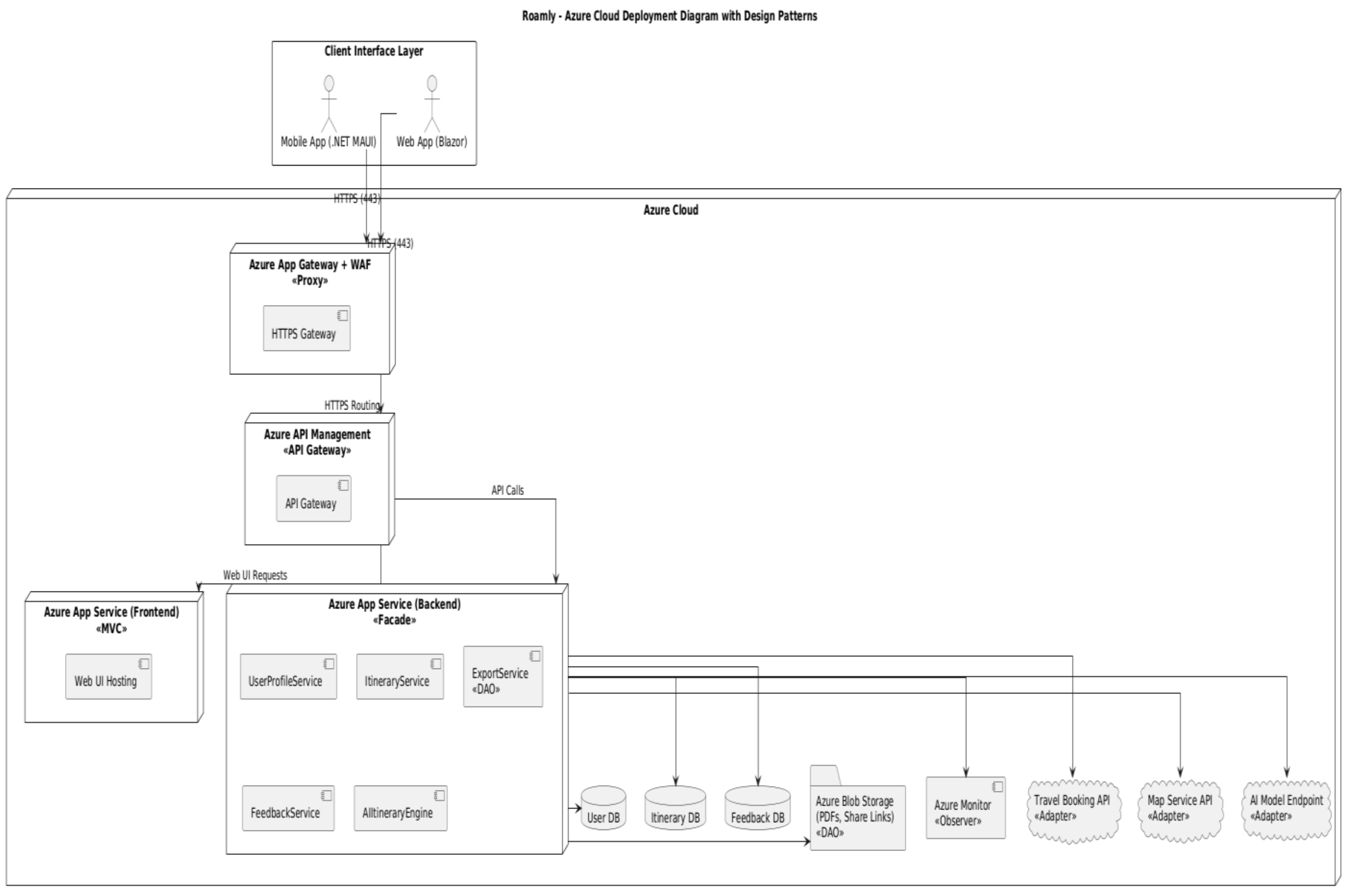
This activity diagram illustrates the step-by-step process of generating an AI-based travel itinerary in the Roamly system. It captures actions taken by both the user and system components, showing how a **RegisteredTraveler** enters trip details using the **TravelPreferences** form. The **AIItineraryEngine** then processes those preferences, queries the **TravelBookingAPI** and **MapServiceProvider**, and finally builds a personalized **Itinerary**. Each swimlane shows who's responsible for each action.



## 4.7 Component Diagram


Roamly - Component Diagram
Client Interface Layer
«component» Mobile App (.NET MAUI)
«component» Web App (Blazor)
API Layer
«component» API Gateway
Application Layer
«component» FeedbackService
«component» UserProfileService
«component» ExportService
«component» AIItineraryEngine
«component» ItineraryService
Integration Layer
«external» AI Model Endpoint
«external» Travel Booking API
«external» Map Service Provider
Persistence Layer
«database» Itinerary DB
«database» Feedback DB
«database» User DB
«storage» Azure Blob Storage

## 4.8 Cloud Deployment Diagram



## 4.9 Skeleton Classes and Database Tables

- Skeleton Classes: CLASS DEFINITIONS

```
public class User
{
   public string UserId { get; set; }
   public string UserName { get; set; }
   public string ContactInfo { get; set; }
}

public class RegisteredTraveler : User
{
   public TravelPreferences Preferences { get; set; }
   public void SetPreferences() { }
   public void ViewItinerary() { }
   public void EditItinerary() { }
   public void SaveItinerary() { }
   public void ExportItinerary() { }
}

public class GuestTraveler : User
{
   public string TempSessionId { get; set; }
   public void GenerateItinerary() { }
   public void ViewSampleItinerary() { }
}

public class TravelPreferences
{
   public string Destination { get; set; }
   public DateRange TravelDates { get; set; }
   public List<string> Interests { get; set; }
   public double Budget { get; set; }
   public List<TravelCompanion> Companions { get; set; }
}

public class TravelCompanion
{
   public string Name { get; set; }
   public int Age { get; set; }
   public TravelPreferences Preferences { get; set; }
}

public class Itinerary
```

```
{
    public string ItineraryId { get; set; }
    public string TravelerId { get; set; }
    public double TotalCost { get; set; }
    public List<DayPlan> Days { get; set; }
}

public class DayPlan
{
    public int DayNumber { get; set; }
    public List<Activity> Activities { get; set; }
}

public class Activity
{
    public string ActivityId { get; set; }
    public string Title { get; set; }
    public string Location { get; set; }
    public string Time { get; set; }
    public double Cost { get; set; }
}

public class Feedback
{
    public string FeedbackId { get; set; }
    public string Message { get; set; }
    public DateTime SubmittedOn { get; set; }
}
```

- Database Tables: SQL TABLE DEFINITIONS

```
CREATE TABLE Users (
  UserId VARCHAR(50) PRIMARY KEY,
  UserName VARCHAR(100),
  ContactInfo VARCHAR(255),
  UserType VARCHAR(20) -- 'Registered' or 'Guest'
);

CREATE TABLE TravelPreferences (
  PreferenceId INT PRIMARY KEY IDENTITY,
  UserId VARCHAR(50) FOREIGN KEY REFERENCES Users(UserId),
  Destination VARCHAR(100),
  StartDate DATE,
  EndDate DATE,
  Interests TEXT,
  Budget DECIMAL(10,2)
);

CREATE TABLE TravelCompanions (
  CompanionId INT PRIMARY KEY IDENTITY,
  PreferenceId INT FOREIGN KEY REFERENCES TravelPreferences(PreferenceId),
  Name VARCHAR(100),
  Age INT
);

CREATE TABLE Itineraries (
  ItineraryId VARCHAR(50) PRIMARY KEY,
  TravelerId VARCHAR(50) FOREIGN KEY REFERENCES Users(UserId),
  TotalCost DECIMAL(10,2)
);

CREATE TABLE DayPlans (
  DayPlanId INT PRIMARY KEY IDENTITY,
  ItineraryId VARCHAR(50) FOREIGN KEY REFERENCES Itineraries(ItineraryId),
  DayNumber INT
);

CREATE TABLE Activities (
  ActivityId VARCHAR(50) PRIMARY KEY,
  DayPlanId INT FOREIGN KEY REFERENCES DayPlans(DayPlanId),
  Title VARCHAR(100),
  Location VARCHAR(100),
  Time VARCHAR(50),
  Cost DECIMAL(10,2)
);

CREATE TABLE Feedback (
  FeedbackId VARCHAR(50) PRIMARY KEY,
  UserId VARCHAR(50) FOREIGN KEY REFERENCES Users(UserId),
  Message TEXT,
  SubmittedOn DATETIME
);
```

## 4.10 Design Patterns Used

- GRASP Patterns
    - **Controller**: We use service classes like ItineraryService, UserProfileService, and ExportService as Controllers. These classes handle system events triggered by the user and coordinate the logic needed, which helps separate UI concerns from backend logic.
    - **Creator**: The Itinerary class uses the Creator pattern to create and manage its associated DayPlan and Activity instances. This keeps object creation organized and encapsulated.
    - **Low Coupling and High Cohesion**: Each service and component (e.g., AIItineraryEngine, TravelBookingAPI) is self-contained and focused on a single responsibility, minimizing dependencies and improving testability.

- SOLID Principles
    - **Single Responsibility Principle (SRP)**: Every class in the system focuses on a single purpose. For instance, TravelPreferences holds user preferences, while Feedback simply captures user feedback — they don't mix responsibilities.
    - **Open/Closed Principle (OCP)**: The system is designed to be extendable. If we want to add a new export format (like calendar integration), we can do that without changing the core itinerary logic.
    - **Liskov Substitution Principle (LSP)**: Both RegisteredTraveler and GuestTraveler inherit from User, so we can treat them the same way in many parts of the system without worrying about breaking functionality.
    - **Interface Segregation Principle (ISP)**: In future implementations, we can define smaller interfaces (like IExportService or IFeedbackHandler) so that classes only depend on what they actually use.
    - **Dependency Inversion Principle (DIP)**: External services like the AI engine or travel APIs are accessed via abstractions (e.g., interfaces or integration layers), decoupling our core logic from specific implementations.

- GoF Patterns
    - **Strategy**: The system can use the Strategy pattern to switch between different itinerary generation approaches or export formats at runtime.
    - **Factory Method**: A Factory Method could be used within AIItineraryEngine to instantiate different types of itineraries (e.g., budget-friendly, luxury, adventure-based).
    - **Observer**: For system monitoring, we treat tools like Azure Monitor and App Insights as observers. They're notified when critical events happen, allowing admins to react in real time.
    - **Facade**: Services like ItineraryService and UserProfileService act as facades to hide underlying complexity from client apps (Web/Mobile).

- Microservices and Best Practices
    - **Service Decomposition**: Each major function (like itinerary generation, exporting, feedback) is handled by its own service. This makes the system easier to scale and update without affecting everything else.
    - **API Gateway**: The Azure App Service acts as a central entry point for mobile and web clients, routing requests to the correct services internally.
    - **External Services**: APIs like TravelBookingAPI, MapServiceProvider, and AI Model Endpoint are loosely coupled. This makes it easier to switch providers or handle failures without impacting the core system.
    - **Monitoring and Observability**: Built-in tools like Azure Monitor and App Insights give admins visibility into how the system is performing, helping with proactive issue detection and DevOps workflows.

# 5 Design Documentation and Explanation

## 5.1 Design Rationale

I designed the Roamly system to be clean, modular, and easy to maintain. From the start, I broke things down into clear layers like Client Interface, API, Application, and Integration so each part has a focused role. This makes it easier to manage and update the system as it grows. For example, the AI itinerary engine runs separately from export and feedback features, which keeps things flexible and avoids messy dependencies.

I followed GRASP and SOLID principles to help keep responsibilities clear and the code organized. I also used a few design patterns like Strategy and Facade to simplify complex parts, especially where users interact with different services. Since the system relies on several external APIs and needs to be reliable at scale, I chose Azure Cloud for deployment. It offers built-in monitoring, strong scalability, and solid integration with .NET technologies, which fit the project perfectly.

Overall, I tried to balance good design practices with real-world needs, so Roamly is both functional and ready to grow in the future.

## 5.2 Security, Scalability, and Performance Considerations

When designing Roamly, I made sure to consider not just how the system works now, but how it can stay secure, scale up, and perform well as it grows. Since users are sharing personal travel data, I prioritized security early on. All sensitive data is stored in secure Azure databases, and access is restricted based on user roles. Communication between services is done over HTTPS, and authentication is handled securely through Azure App Service with built-in identity options.

Scalability was also a big part of the design. By organizing the system into modular services like the AI engine, itinerary service, and export service it's easier to scale individual parts without affecting the rest. I chose Azure App Service and Azure SQL because they support automatic scaling, which helps the system handle more users during busy travel seasons.

For performance, I kept things lightweight and fast. The itinerary generation process offloads heavy AI and route calculations to external APIs, so the backend can stay responsive. I also included caching options and performance monitoring tools like Azure Monitor and Application Insights to track and improve speed as needed.

## 6 Appendix

### 6.1 References

N/A

### 6.2 Source Code or Repository Links

GitHub Link: https://github.com/Liyan5092/Roamly---SWENG-837

### 6.3 Presentation Link

Presentation can be viewed in the GitHub repo above.