Smart Travel and Itinerary Management System

By:

Iqra Yaqoob
44566
Aiman Shafeeq
44560
Mariam Sohail
45773
Samiyya Aftab
46891

Faculty of Computing
Riphah International University, Islamabad
Fall 2024

A Dissertation Submitted To

Faculty of Computing,
Riphah International University, Islamabad
As a Partial Fulfillment of the Requirement for the Course
Software Construction Development
Bachelors of Science in Software Engineering

Faculty of Computing
Riphah International University, Islamabad

Dedication/Acknowledgment

We are grateful to Allah Ta'ala for enabling us to finish this project report. Additionally, our course instructor helped us with this assignment. Everyone in the team who put in a lot of effort and diligence to finish this project.

Iqra Yaqoob

44566

Aiman Shafeeq

44560

Mariam Sohail

45773

Samiyya Aftab

46891

Abstract

The **Smart Travel and Itinerary Management System (STIMS)** is a user-focused platform designed to simplify travel planning by supporting multiple roles: Admin, Traveler, and Travel Agent. This paper outlines the development process, incorporating artifacts such as use case diagrams, sequence diagrams, and other UML models while adhering to the **MVC framework** and **GRASP principles** for robust design.

The **Use Case Diagram** highlights key functionalities: Admins manage users, oversee bookings, and generate reports; Travel Agents create itineraries, offer packages, and manage accommodation options; and Travelers customize itineraries, book trips, and receive reminders. **Sequence Diagrams** illustrate interactions for processes like booking approvals and itinerary updates. **Activity Diagrams** depict workflows such as trip customization and notification management.

STIMS is built using the **MVC framework**, ensuring a separation of concerns across Model, View, and Controller layers for maintainability. **GRASP principles** like Controller and Information Expert are applied to promote low coupling and high cohesion, enabling efficient and modular development.

Key features include secure role-based authentication, itinerary and accommodation management, booking notifications, and analytics on travel trends. The incorporation of structured artifacts ensures a clear and efficient development process, providing stakeholders with comprehensive insights into system functionality and design.

Table of Contents

Artifact-1
Project Proposal

Project Proposal

Project Title: Smart Travel and Itinerary Management System

Description:

The **Smart Travel and Itinerary Management System (STIMS)** is a comprehensive platform designed to simplify and enhance the travel experience for users by integrating advanced technologies and user-friendly features. This system connects **Travelers**, **Travel Agents**, and **Admins**, streamlining processes such as booking accommodations, planning itineraries, and receiving notifications. Travelers can create personalized profiles, customize itineraries, and receive reminders for important tasks. Travel Agents can offer travel packages, manage bookings, and provide accommodation options. Admins oversee the system, manage users, and generate reports on travel trends.

Existing travel management systems often lack efficiency, user accessibility, and personalized support, leading to a suboptimal experience for users. STIMS addresses these gaps by providing a faster, easier, and more organized way to manage travel plans. This system incorporates technologies such as mobile applications, intelligent reminders, and analytics, making it accessible, interactive, and effective for all stakeholders.

Feature:

- Login
- Traveler Login
- Travel Agent Login
- Admin Login
- Manage Users
- Approve Travel Agents
- Oversee Bookings
- Generate Travel Reports
- Export to Excel
- Export to PDF
- Add Accommodation Options
- Offer Packages
- Create Itineraries
- Approve Booking
- Check Availability
- Decline Booking
- Browse Destinations
- Book Trips
- Authenticate User
- Add Extra Services
- Manage Itineraries
- View Itineraries
- Customize Itineraries
- Add Itinerary
- Remove Itinerary

- Set Reminders
- Receive Notifications
- Manage Notifications
- Handle Reports
- Logout
- Email Verification
- Number Verification
- Login with Email
- Login with Number
- Approve Travel Agents
- Decline Travel Agents
- Generate Reports
- System Notifications
- Add Extra Services
- Set Reminders for Traveler

Artifact-2
Use case Diagram

In a use case diagram, **specification**, **generalization**, and **actor** are essential concepts that help to define the relationships and roles within the system.

Actors

- **Definition**: An actor represents any external entity that interacts with the system. This could be a person, group, or another system that performs actions in the system.

Actors in the Diagram:

- **Admin**: Manages users, approves travel agents, generates reports, oversees bookings, etc.
- **Travel Agent**: Creates itineraries, approves bookings, manages accommodations, etc.
- **Traveler**: Books trips, customizes itineraries, views destinations, sets reminders, etc.
- **System**: The backend system that supports functionalities such as authentication, notifications, and handling reports.

Generalization:

Generalization is a relationship where an actor or a use case is a specialized version of another actor or use case. A generalization represents inheritance, where the child (specialized actor or use case) inherits the behavior of the parent (generalized actor or use case) but may also have additional specific behaviors.

- **Actor Generalization**:

  Admin: The **Admin** actor can be generalized into specific roles like **User Administrator** and **Report Analyst**. These specialized roles share the core responsibilities of the Admin but have specific tasks:

  - **User Administrator**: Manages user accounts.
  - **Report Analyst**: Generates and exports reports.
- **Traveler Generalization**:

  The **Traveler** can be generalized into roles such as **Trip Planner** and **Itinerary Organizer**:

  - **Trip Planner**: Plans trips, browses destinations, books trips.
  - **Itinerary Organizer**: Customizes itineraries, sets reminders, and views itineraries.
- **Use Case Generalization**:

  In the system, **Create Itinerary** is a more general task that could be extended by a specific use case, like **Customize Itinerary**, where the traveler can customize their own itineraries. Similarly, **Generate Reports** might have specialized forms, such as **Export to Excel** or **Export to PDF**.

Specification:

Admin → Use Cases:

- Admin → Login
- Admin → Manage Users
- Admin → Approve Travel Agents
- Admin → Oversee Bookings
- Admin → Generate Travel Reports
- Admin → Export to Excel (Extended from Generate Travel Reports)
- Admin → Export to PDF (Extended from Generate Travel Reports)
- Admin → Send Email (Extend for sending reports via email)
- Admin → Send SMS/Number Notification (Extend for sending reports via phone number)

Travel Agent → Use Cases:

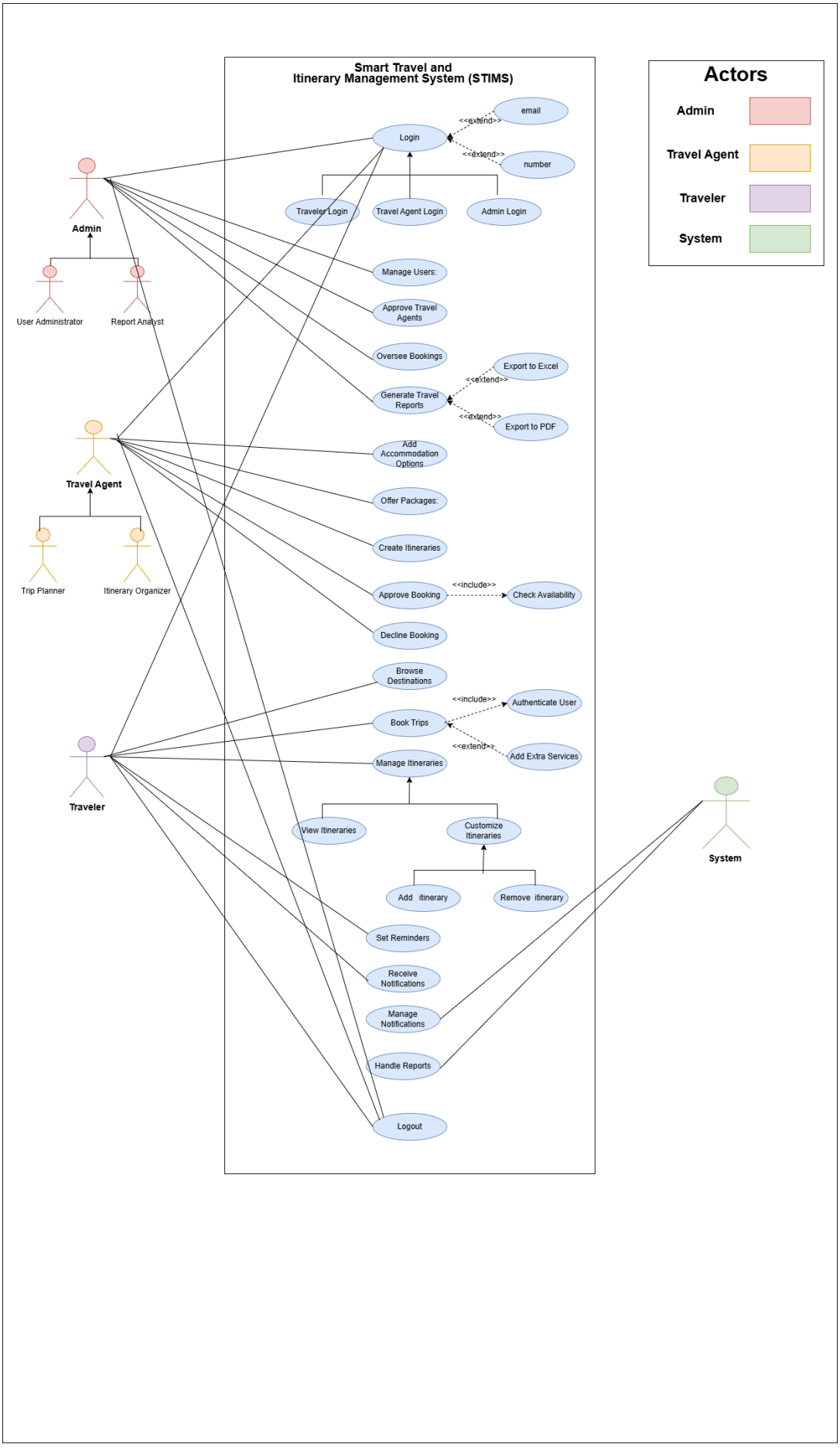
- Travel Agent → Login
- Travel Agent → Add Accommodation Options
- Travel Agent → Offer Packages
- Travel Agent → Create Itineraries
- Travel Agent → Approve Booking
- Travel Agent → Decline Booking

Traveler → Use Cases:

- Traveler → Login
- Traveler → Browse Destinations
- Traveler → Book Trips
    - Traveler → Authenticate User (Included as part of Book Trips)
    - Traveler → Add Extra Services
- Traveler → Manage Itineraries
    - Traveler → View Itinerary
    - Traveler → Customize Itinerary
- Traveler → Set Reminders
- Traveler → Receive Notifications

System → Use Cases:

- System → Authenticate User
- System → Handle Reports
- System → Send Notifications


Smart Travel and
Itinerary Management System (STIMS)
Actors
Admin
Travel Agent
Traveler
System
email
<<extend>>
Login
<<extend>>
number
Traveler Login
Travel Agent Login
Admin Login
Admin
User Administrator
Report Analyst
Manage Users:
Approve Travel Agents
Oversee Bookings
Export to Excel
<<extend>>
Generate Travel Reports
<<extend>>
Export to PDF
Travel Agent
Add Accommodation Options
Offer Packages:
Create Itineraries
Trip Planner
Itinerary Organizer
Approve Booking
<<include>>
Check Availability
Decline Booking
Browse Destinations
<<include>>
Authenticate User
Book Trips
<<extend>>
Add Extra Services
Manage Itineraries
Traveler
View Itineraries
Customize Itineraries
Add itinerary
Remove itinerary
Set Reminders
Receive Notifications
Manage Notifications
Handle Reports
Logout
System

Artifact-3
Fully Dress Format

A **fully dressed use case** is a detailed and comprehensive description of a use case. It provides all the necessary information about a specific use case in a structured format, ensuring clarity for developers, testers, and stakeholders.

UC-001: Generate Traveler Report

Section	Content
Designation	UC-1
Name	Generate Traveler Report
Authors	iqra
Priority	High
Criticality	Critical for system reporting and analytics
Source	System stakeholders
Responsible	Admin
Description	1. This functionality allows the Admin to generate a detailed report of Travelers in the system. 2. The report includes information about Traveler activity, bookings, itineraries, and overall system usage statistics. 3. The report can be exported in formats like PDF or Excel.
Trigger event	The Admin selects the **Generate Traveler Report** option from the system dashboard.
Actors	**Primary Actor**: Admin **Supporting Actor**: System (STIMS)
Precondition	Admin is logged into the system. Traveler data is available in the database. Reporting module is functional.
Postcondition	A detailed Traveler report is generated and ready for download. A copy of the report may be stored in the system for future reference.
Result	Admin has a comprehensive report on Traveler data, enabling system management and decision-making.
Main Scenario	• Admin logs into the system. • Admin navigates to the **Reports** section. • Admin selects the **Generate Traveler Report** option. • The system retrieves all Traveler data from the database. • The system organizes the data into a structured report format. • The report is displayed to the Admin for preview. • Admin downloads the report or views it in the system.
Alternative Scenario	The system detects that there is no Traveler data in the database. The system displays a message: "No data available to generate the Traveler report." Admin returns to the dashboard.

UC-002: Create Itinerary

Section	Content
Designation	UC-2
Name	Create Itinerary
Authors	Aiman
Priority	High
Criticality	high
Source	Old System
Responsible	Travel Agent
Description	The **"Create Itinerary"** feature enables **Travel Agents** to build customized travel itineraries for **Travelers** based on selected destinations, dates, and activities. The itinerary includes details like transportation, accommodation, activities, and reminders. Once created, the system generates and sends the itinerary to the Traveler.
Trigger event	The Traveler requests a customized itinerary. The Travel Agent selects the option to create a new itinerary.
Actors	**Primary Actor**: Travel Agent **Supporting Actor**: System (STIMS), Traveler
Precondition	Traveler has logged into the system and selected their desired destinations and services. Travel Agent is logged into the system with appropriate access rights. The System (STIMS) contains up-to-date information on available services, activities, and accommodations.
Postcondition	A Traveler itinerary is created and saved in the system. The Traveler receives a copy of their itinerary (via email or system notification). Traveler can view, modify, or cancel the itinerary. Travel Agent can view and modify the itinerary if needed.
Result	The Traveler receives a well-structured and detailed itinerary. Travel Agents can efficiently manage and customize multiple itineraries.
Main Scenario	• Traveler logs into the system and selects the Create Itinerary option. • System prompts the Traveler to input travel preferences (destination, travel dates, activities). • Traveler submits their preferences, which are passed to the Travel Agent. • Travel Agent logs into the system and opens the Create Itinerary option. • Travel Agent reviews the Traveler's submitted preferences and available options. • Travel Agent selects appropriate accommodation, transport, and activities based on the Traveler's preferences. • System validates availability for the selected services and option
Alternative Scenario	If the Traveler hasn't provided enough data (e.g., missing destination or date), the system prompts them to enter the missing information. The itinerary creation process cannot proceed until all required details are provided.

UC-003: Browse Destination

Section	Content
Designation	UC-3
Name	Browse Destination
Authors	Maryam
Priority	Medium
Criticality	Moderate
Source	Documentation
Responsible	Traveler
Description	The "**Browse Destination**" feature allows Travelers to explore available travel destinations within the system. Travelers can view destination details such as: • Location highlights (e.g., attractions, activities). • Accommodation options. • Available transportation.
Trigger event	The Traveler selects the **Browse Destination** option from the system menu or homepage.
Actors	**Primary Actor**: Traveler **Supporting Actor**: System (STIMS)
Precondition	Traveler is logged into the system. The **System** has a database of destinations with updated information. Internet connectivity is active.
Postcondition	The **Traveler** views a list of available destinations. The **System** displays relevant details for each destination (attractions, services, reviews). The **Traveler** can add destinations to their wish list or itinerary for future planning.
Result	The Traveler gains knowledge of destinations and their features. The System tracks browsing behavior for analytics and personalization
Main Scenario	• Traveler logs into the system. • Traveler selects the Browse Destination option from the dashboard or menu. • The System displays a list of destinations, categorized by region, popularity, or type (e.g., beaches, mountains).
Alternative Scenario	The System does not have any destinations matching the Traveler's filters or preferences. The **System** displays a message: "No destinations found. Please adjust your filters or preferences." The Traveler modifies their filters and retries.

Artifact-4
Domain Modeling

Domain modeling is the process of identifying and representing the key entities, their attributes, and relationships in a specific domain or system. It helps to provide a clear understanding of the system's structure and data flow, forming the foundation for system design and implementation.

Relationship:

Traveler ↔ Itinerary

- **Relationship**: A traveler can create or manage multiple itineraries.
- **Cardinality**: One **Traveler** can have multiple **Itineraries** (1..*), and each **Itinerary** belongs to one **Traveler** (1).

Travel Agent ↔ Accommodation

- **Relationship**: Travel agents add and manage multiple accommodation options.
- **Cardinality**: One **Travel Agent** can manage multiple **Accommodations** (1..*), and each **Accommodation** can be handled by one **Travel Agent** (1).

Itinerary ↔ Destination

- **Relationship**: An itinerary includes one or more destinations.
- **Cardinality**: One **Itinerary** can include multiple **Destinations** (1..*), and each **Destination** can appear in multiple **Itineraries** (..*).

Traveler ↔ Booking

- **Relationship**: A traveler can make multiple bookings for accommodations or itineraries.
- **Cardinality**: One **Traveler** can have multiple **Bookings** (1..*), and each **Booking** is associated with one **Traveler** (1).

Booking ↔ Accommodation

- **Relationship**: A booking is made for a specific accommodation.
- **Cardinality**: One **Booking** is linked to one **Accommodation** (1), and each **Accommodation** can have multiple **Bookings** (1..*).

Travel Agent ↔ Itinerary

- **Relationship**: A travel agent creates and manages multiple itineraries.
- **Cardinality**: One **Travel Agent** can create multiple **Itineraries** (1..*), and each **Itinerary** is associated with one **Travel Agent** (1).

Traveler ↔ Notification

- **Relationship**: Notifications are sent to travelers for reminders and updates.
- **Cardinality**: One **Traveler** can receive multiple **Notifications** (1..*), and each **Notification** is sent to one **Traveler** (1).

Admin ↔ Notification

- **Relationship**: Admins generate notifications for system-wide updates or alerts.

- **Cardinality**: One **Admin** can create multiple **Notifications** (1..*), and each **Notification** is managed by one **Admin** (1).

Admin ↔ Booking

- **Relationship**: Admins oversee and approve/review bookings.
- **Cardinality**: One **Admin** can oversee multiple **Bookings** (1..*), and each **Booking** is overseen by one **Admin** (1).

Traveler ↔ Authentication

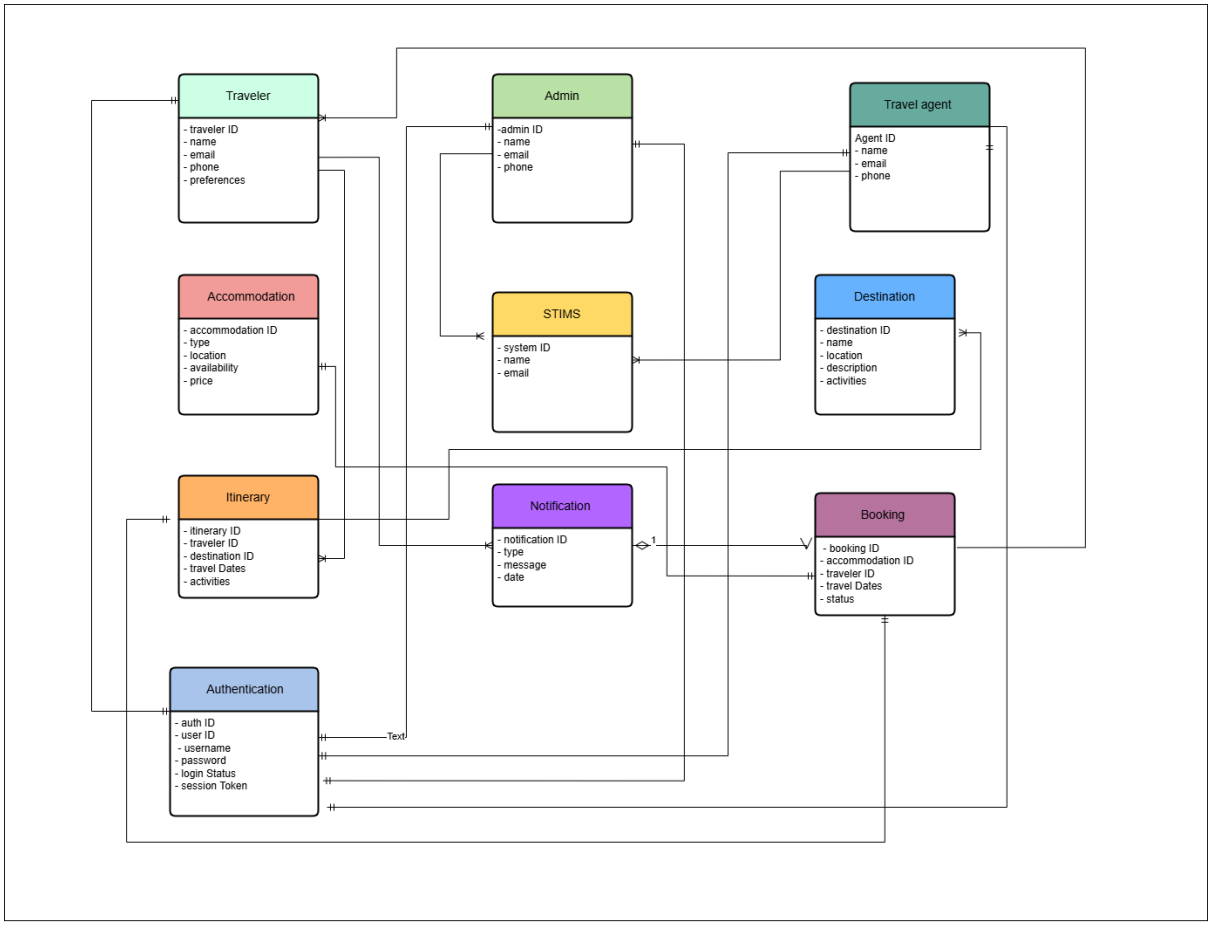
- **Relationship**: A traveler must authenticate to access the system.
- **Cardinality**: One **Traveler** corresponds to one **Authentication** (1), and each **Authentication** is linked to one **Traveler** (1).

Admin ↔ Authentication

- **Relationship**: Admins must authenticate to access admin-level controls.
- **Cardinality**: One **Admin** corresponds to one **Authentication** (1), and each **Authentication** is linked to one **Admin** (1).

Travel Agent ↔ Authentication

- **Relationship**: Travel agents must authenticate to perform their tasks.
- **Cardinality**: One **Travel Agent** corresponds to one **Authentication** (1), and each **Authentication** is linked to one **Travel Agent** (1).


Traveler
- traveler ID
- name
- email
- phone
- preferences
Admin
-admin ID
- name
- email
- phone
Travel agent
Agent ID
- name
- email
- phone
Accommodation
- accommodation ID
- type
- location
- availability
- price
STIMS
- system ID
- name
- email
Destination
- destination ID
- name
- location
- description
- activities
Itinerary
- itinerary ID
- traveler ID
- destination ID
- travel Dates
- activities
Notification
- notification ID
- type
- message
- date
1
Booking
- booking ID
- accommodation ID
- traveler ID
- travel Dates
- status
Authentication
- auth ID
- user ID
- username
- password
- login Status
- session Token
Text

Artifact-5
Class Diagram

A class diagram is a type of static structure diagram that represents the structure of a system by showing the relationships among classes. It's a fundamental object-oriented design tool used to visualize the design of a system.

Classes:

Represented by rectangles with three sections:

- **Class name (top section):** The name of the class.
- **Attributes (middle section):** The properties or data members of the class.
- **Operations (bottom section):** The methods or functions that can be performed on the class.

Relationships:

Represented by lines connecting classes:

Multiplicity:

Represented by numbers or symbols on the lines:

- 1: One instance.
- 0..1: Zero or one instance.
- 1...*: One or more instances.
- 0..*: Zero or more instances.

Admin ↔ Report

- **Relationship**: Admins generate reports based on system data and view travel trends.
- **Cardinality**: One **Admin** can generate multiple **Reports** (1..*), and each **Report** is associated with one **Admin** (1).
- **Functions**: generate Report (), viewTrends().

Admin ↔ User

- **Relationship**: Admins manage system users by approving or editing their details.
- **Cardinality**: One **Admin** can manage multiple **Users** (1..*), and each **User** belongs to one **Admin** (1).
- **Functions**: approveAgent(), manageUsers().

User ↔ Traveler

- **Relationship**: Travelers are specialized users with additional capabilities for booking and managing itineraries.
- **Cardinality**: One **User** corresponds to one **Traveler** (1), and each **Traveler** maps back to one **User** (1).

- **Functions**: login(), logout(), updateProfile().

User ↔ Travel Agent

- **Relationship**: Travel agents are specialized users who can add accommodations and create itineraries.
- **Cardinality**: One **User** corresponds to one **Travel Agent** (1), and each **Travel Agent** maps back to one **User** (1).
- **Functions**: createItinerary(), addAccommodation(), approveBooking().

Traveler ↔ Itinerary

- **Relationship**: Travelers create or manage itineraries for their trips.
- **Cardinality**: One **Traveler** can manage multiple **Itineraries** (1..*), and each **Itinerary** belongs to one **Traveler** (1).
- **Functions**: addActivity(), removeActivity(), viewItinerary().

Itinerary ↔ Activity

- **Relationship**: Each itinerary consists of one or more activities, such as sightseeing or scheduled events.
- **Cardinality**: One **Itinerary** can include multiple **Activities** (1..*), and each **Activity** belongs to one **Itinerary** (1).
- **Functions**: updateActivity(), deleteActivity().

Traveler ↔ Booking

- **Relationship**: Travelers make bookings for accommodations or itineraries.
- **Cardinality**: One **Traveler** can have multiple **Bookings** (1..*), and each **Booking** is associated with one **Traveler** (1).
- **Functions**: BookAccommodation(), SetReminder().

Booking ↔ Accommodation

- **Relationship**: Bookings are made for specific accommodations such as hotels or hostels.
- **Cardinality**: One **Booking** corresponds to one **Accommodation** (1), and each **Accommodation** can have multiple **Bookings** (1..*).
- **Functions**: approveBooking(), cancelBooking().

Traveler ↔ Reminder

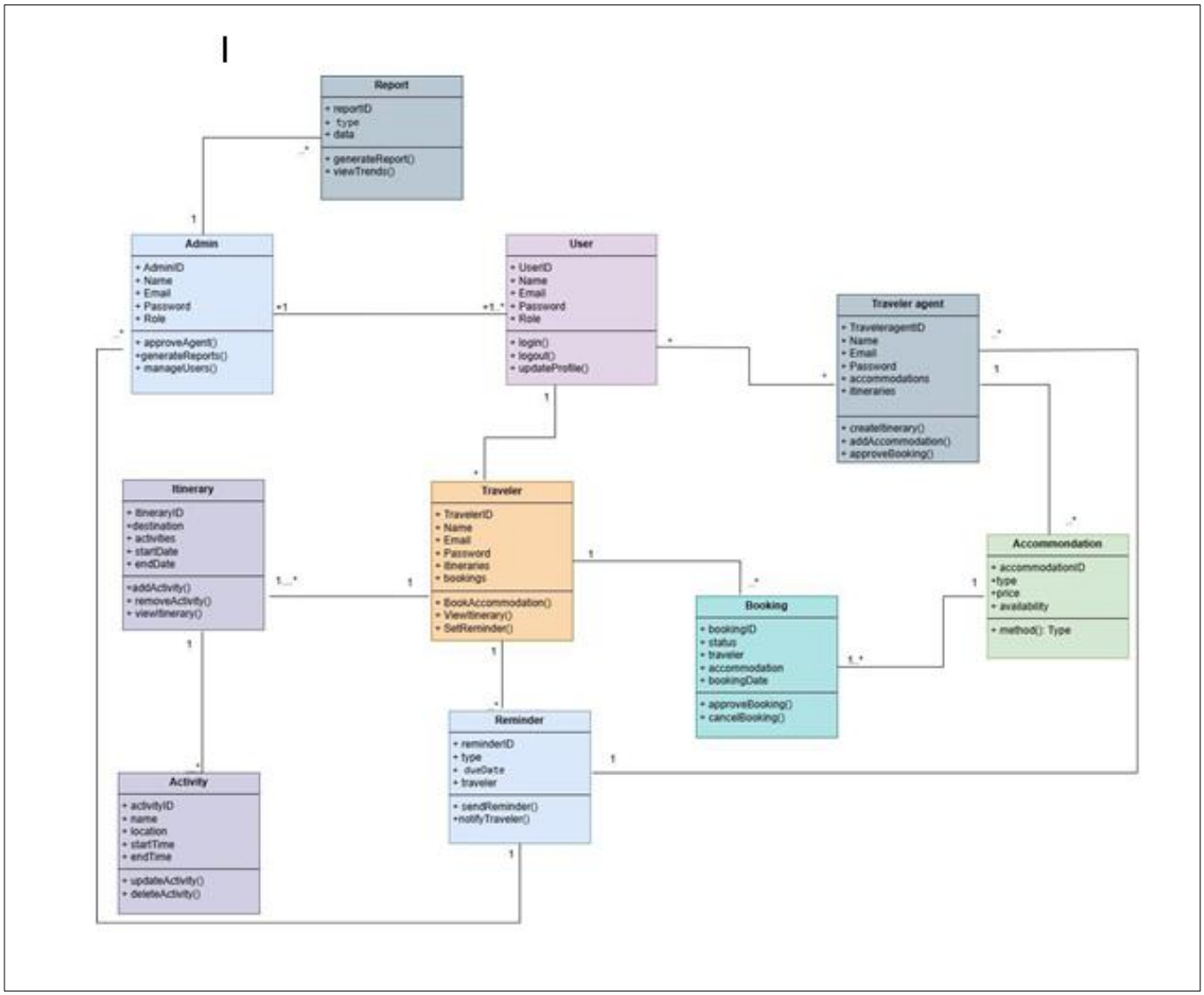
- **Relationship**: Travelers receive reminders for important travel-related tasks.
- **Cardinality**: One **Traveler** can receive multiple **Reminders** (1..*), and each **Reminder** belongs to one **Traveler** (1).
- **Functions**: sendReminder(), notifyTraveler().

Travel Agent ↔ Accommodation

- **Relationship**: Travel agents add and manage accommodations in the system.
- **Cardinality**: One **Travel Agent** can manage multiple **Accommodations** (1..*), and each **Accommodation** can be linked to one **Travel Agent** (1).
- **Functions**: addAccommodation().

Travel Agent ↔ Itinerary

- **Relationship**: Travel agents create and offer itineraries for travelers.
- **Cardinality**: One **Travel Agent** can create multiple **Itineraries** (1..*), and each **Itinerary** is managed by one **Travel Agent** (1).
- **Functions**: createItinerary()



Artifact-6
Activity Diagram

An **activity diagram** is a type of **UML (Unified Modeling Language)** diagram used to visually represent workflows of actions, activities, and decisions within a system or process. It helps analyze the flow of activities and their dependencies.

Symbols:

Oval (Black Circle): **Start Node**

Indicates the beginning of the workflow.

Example: Login process.

Rectangle with Rounded Edges: **Activity**

Represents a task or operation.

Example: "Request accommodation," "Generate report."

Diamond: **Decision Node**

Represents branching based on conditions.

Example: Checking the validity of accommodation or itinerary.

Arrow: **Transition**

Shows the flow between activities.

Example: Connecting "Receive login request" to "Send validation request."

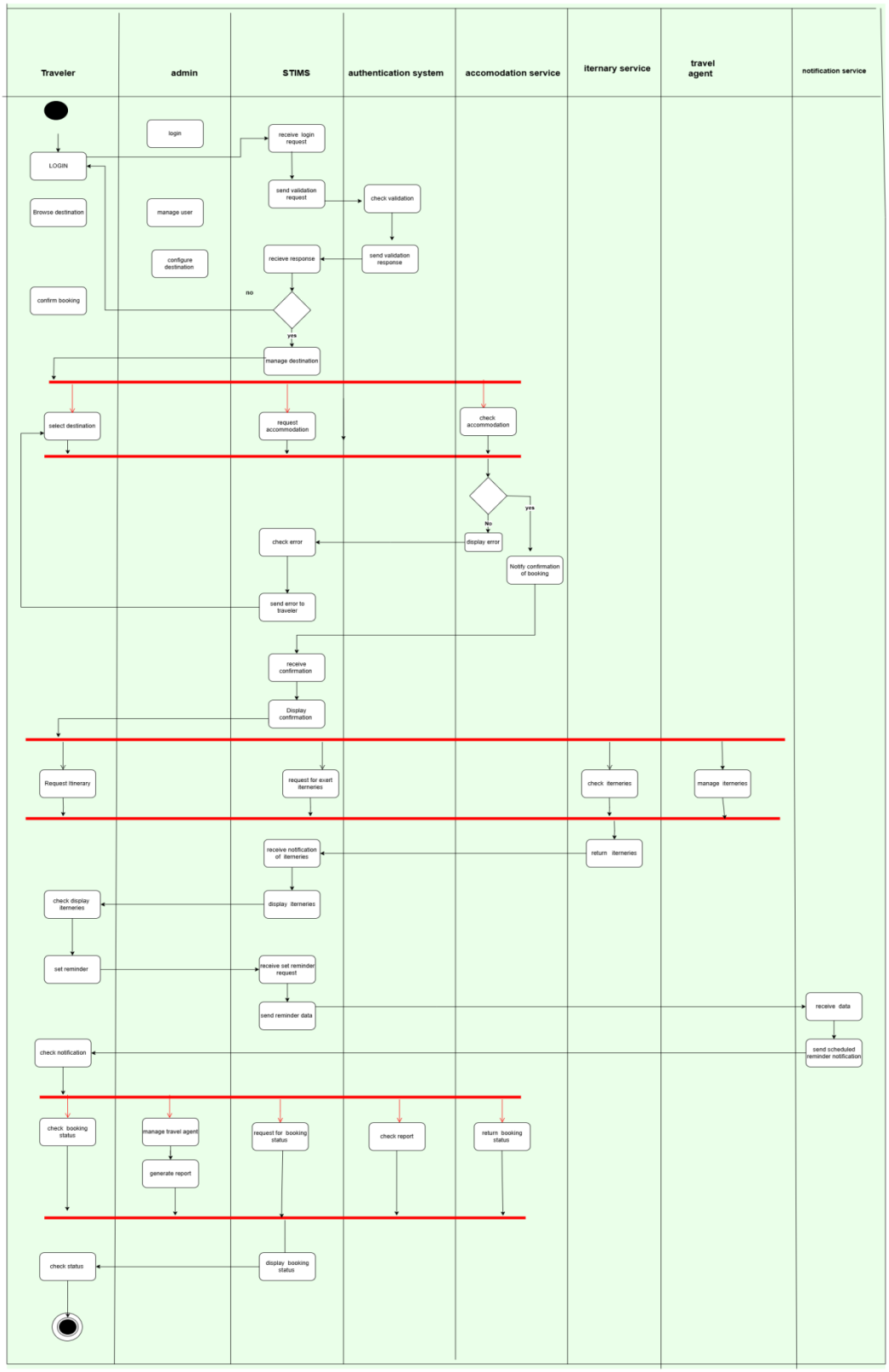
Vertical Red Lines: **Swim lane Divider**

Indicates separation of roles or services in the workflow.

Circle with a Black Border: **End Node**

Marks the termination of the process.

Example: "Check status."


Traveler
admin
STIMS
authentication system
accomodation service
iternary service
travel agent
notification service
login
receive login request
LOGIN
send validation request
check validation
Browse destination
manage user
send validation response
recieve response
configure destination
confirm booking
no
yes
manage destination
select destination
request accommodation
check accommodation
yes
No
check error
display error
Notify confirmation of booking
send error to traveler
receive confirmation
Display confirmation
Request itinerary
request for exert iterneries
check iterneries
manage iterneries
receive notification of iterneries
return iterneries
check display iterneries
display iterneries
set reminder
receive set reminder request
send reminder data
receive data
check notification
send scheduled reminder notification
check booking status
manage travel agent
request for booking status
check report
return booking status
generate report
check status
display booking status

Artifact-7
Sequence Diagram

A sequence diagram is a visual representation of interactions between different components or actors in a system. Here's a simplified explanation of its key elements:

1. **Actors and Components:**

   Shown as rectangles at the top of the diagram.

2. **Lifelines:**

   Vertical dashed lines under each actor or component that represent their existence during the process.

3. **Messages/Arrows:**

   Horizontal arrows between lifelines show communication or actions (e.g., requests, responses).

4. **Loops:**

   Represent repeated actions, such as validating data, enclosed in a labeled box.

5. **Conditions (Alt):**

   Show decision points, where different actions occur based on conditions.

Traveler Login:

- The **Traveler** logs in with credentials.
- The **Authentication Service** validates the login credentials.

Destination Selection:

- The **Traveler** selects a destination.
- The **System** requests destination data from the **Itinerary Service**.

Accommodation Availability:

- The **Traveler** checks accommodation availability.
- The **System** sends a request to the **Accommodation Service** for booking confirmation.

View Itinerary:

- The **Traveler** views the itinerary for the selected destination.
- The **System** retrieves and displays the itinerary data.

Set Reminders:

- The **Traveler** sets reminders for activities.
- The **System** sends reminder data to the **Notification Service** to schedule notifications.

Booking Status Check:

- The **Traveler** checks the current booking status.
- The **Accommodation Service** returns the status of the booking.

Admin Management:

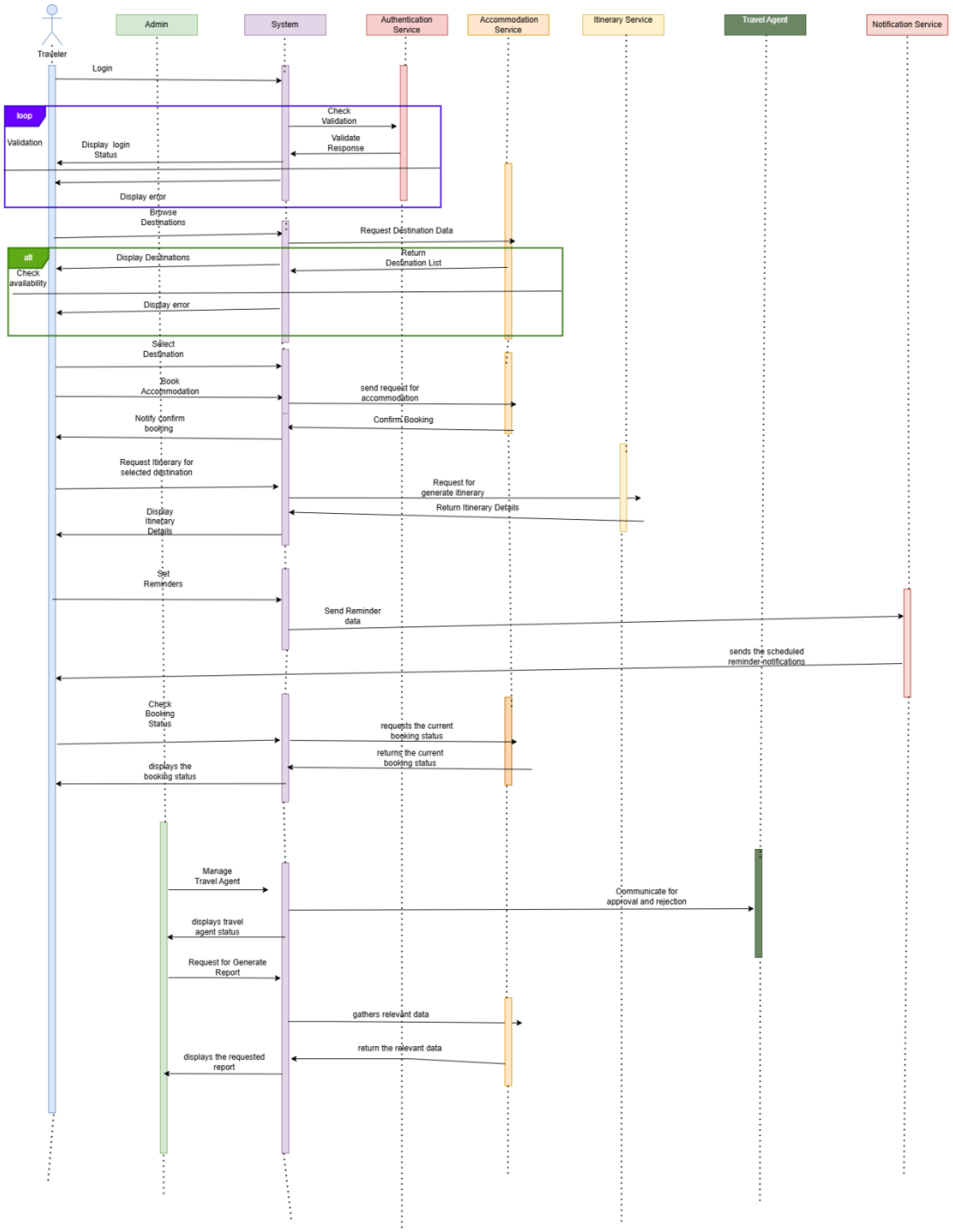
- The **Admin** manages travel agents.
- The **Admin** approves or rejects travel agent requests.

Report Generation:

- The **Admin** requests reports on travel trends or agent performance.
- The **System** gathers data and returns the requested report.

Approval/Reject Notification:

- The **Travel Agent** receives communication from the **Admin** for approval or rejection of bookings.


Traveler
Admin
System
Authentication Service
Accommodation Service
Itinerary Service
Travel Agent
Notification Service
Login
loop
Validation
Check Validation
Validate Response
Display login Status
Display error
Browse Destinations
Request Destination Data
alt
Check availability
Display Destinations
Return Destination List
Display error
Select Destination
Book Accommodation
send request for accommodation
Confirm Booking
Notify confirm booking
Request itinerary for selected destination
Request for generate itinerary
Return Itinerary Details
Display Itinerary Details
Set Reminders
Send Reminder data
sends the scheduled reminder notifications
Check Booking Status
requests the current booking status
returns the current booking status
displays the booking status
Manage Travel Agent
Communicate for approval and rejection
displays travel agent status
Request for Generate Report
gathers relevant data
return the relevant data
displays the requested report

Artifact-8
State Transition Diagram

A **state diagram** is a type of **UML (Unified Modeling Language)** diagram that shows how an object or system transitions between various states over time. It helps in modeling the dynamic behavior of a system by representing its different states and the transitions triggered by specific events.

- **States**: Represent the conditions or statuses of an object or system.
- **Transitions**: Indicate the movement from one state to another, often triggered by events or actions.
- **Events**: Occurrences that cause transitions between states.

Start:

The process begins when the system is turned on.

Request Scheme:

The customer requests a tour scheme.

Registration (New Customer):

The customer needs to provide their email and user ID for registration.

If the email is invalid, the process is halted.

If the email is valid, they proceed to browsing tour options.

Browse Tour Options:

After selecting a destination, the traveler browses available tour options.

Get Tour Detail:

The traveler selects a destination, and the system provides details about available tours related to the selected destination.

Agreement:

The traveler agrees to the terms and conditions of the tour package.

Personal Detail:

The traveler fills in their personal details for the booking process.

Submission:

The traveler submits their details to the traveling agency for processing.

Select Tour Package:

The traveler selects a tour package from the available options.

Booking Confirmation:

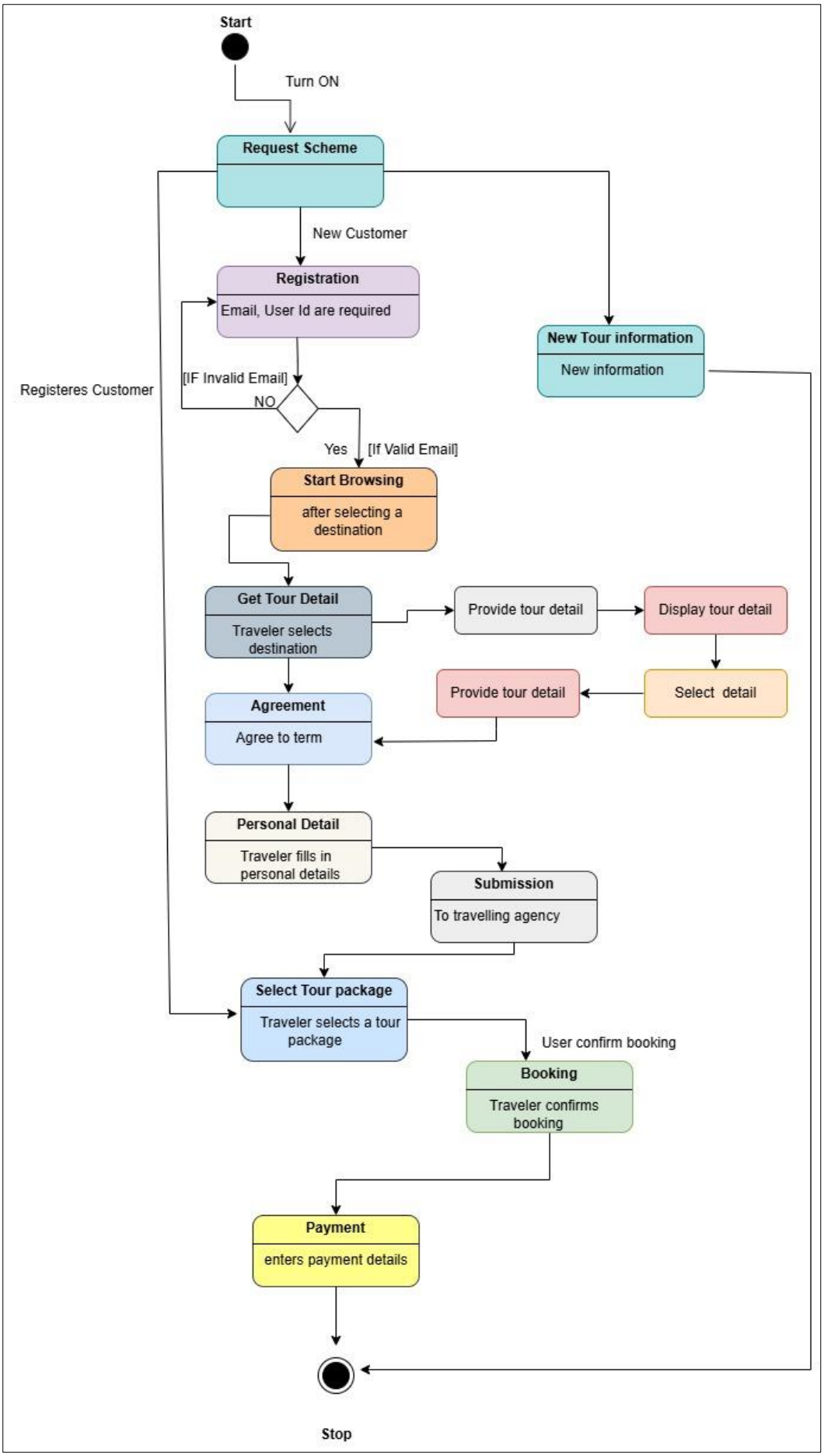
The traveler confirms their booking for the selected tour package.

Payment:

The traveler enters payment details to complete the booking.

End:

The process concludes after the payment is processed.


Start
Turn ON
Request Scheme
New Customer
Registration
Email, User Id are required
[IF Invalid Email]
NO
Yes
[If Valid Email]
Registeres Customer
New Tour information
New information
Start Browsing
after selecting a destination
Get Tour Detail
Traveler selects destination
Provide tour detail
Display tour detail
Select detail
Provide tour detail
Agreement
Agree to term
Personal Detail
Traveler fills in personal details
Submission
To travelling agency
Select Tour package
Traveler selects a tour package
User confirm booking
Booking
Traveler confirms booking
Payment
enters payment details
Stop

Artifact-9
MVC Framework &
GRASP Pattern

MVC Framework:

MVC stands for Model-View-Controller, a software architecture pattern that separates an application into three interconnected components. It's a framework that helps organize code, promote reuse, and make maintenance easier.

- **Model:** Represents the data and business logic, managing database interactions and data storage.
- **View:** Handles user interface and presentation, displaying data to the user.
- **Controller**: Acts as an intermediary, receiving input, processing requests, and updating the model and view accordingly.

GRASP Patterns:

GRASP (General Responsibility Assignment Software Patterns) is a set of principles that help assign responsibilities to classes and objects in object-oriented design. It's a way to make your code more maintainable, flexible, and scalable.

- **Information Expert:** Assigns responsibility to the class that has the most knowledge about the data or behavior.
- **Creator:** Assigns responsibility to the class that creates an object.
- **Controller**: Assigns responsibility to the class that manages and coordinates the behaviour of other classes.
- **Low Coupling:** Minimizes dependencies between classes.
- **High Cohesion:** Ensures classes have a clear, well-defined purpose.
- **Polymorphism:** Allows objects of different classes to be treated as if they were of the same class.
- **Pure Fabrication:** Creates classes that don't represent real-world entities but help with organization and structure.
- **Indirection:** Decouples objects by inserting an intermediate object to reduce direct dependencies.
- **Protected Variation:** Encapsulates elements of an object's behavior to allow for changes without affecting other parts of the system.

Output:

Welcome! Select your role:

1. Admin

2. Traveler

3. Travel Agent

4. Exit

Enter your choice: 1

Admin Dashboard:

1. Add Traveler

2. View All Travelers

3. Delete Traveler

4. Back to Main Menu

5. Exit Program

Enter your choice: 4

Welcome! Select your role:

1. Admin

2. Traveler

3. Travel Agent

4. Exit

Enter your choice: 2

Please enter your details to proceed:

Enter Your Name: ali

Enter Your Email: ass

Enter Your Contact Number: 23

Enter Your Address: 3

Traveler profile created successfully!

Traveler Dashboard:

1. View Traveler Details

2. Book a Seat

3. View Booked Seats

4. Cancel a Booking

5. Back to Main Menu

6. Exit Program

Enter your choice: 1

Traveler Details:

Name: ali, Email: ass, Contact: 23, Address: 3

Traveler Dashboard:

1. View Traveler Details

2. Book a Seat

3. View Booked Seats

4. Cancel a Booking

5. Back to Main Menu

6. Exit Program

Enter your choice: 2

Enter Seat Number to Book (e.g., A1, B2): A1

Seat A1 booked successfully!

Traveler Dashboard:

1. View Traveler Details

2. Book a Seat

3. View Booked Seats

4. Cancel a Booking

5. Back to Main Menu

6. Exit Program

Enter your choice: 3

Booked Seats:

- A1

Traveler Dashboard:

1. View Traveler Details

2. Book a Seat

3. View Booked Seats

4. Cancel a Booking

5. Back to Main Menu

6. Exit Program

Enter your choice:

4

Enter Seat Number to Cancel Booking: A1

Booking for seat A1 canceled successfully!

Traveler Dashboard:

1. View Traveler Details

2. Book a Seat

3. View Booked Seats

4. Cancel a Booking

5. Back to Main Menu

6. Exit Program

Enter your choice: 5

Welcome! Select your role:

1. Admin

2. Traveler

3. Travel Agent

4. Exit

Enter your choice: 2

Traveler Dashboard:

1. View Traveler Details

2. Book a Seat

3. View Booked Seats

4. Cancel a Booking

5. Back to Main Menu

6. Exit Program

Enter your choice: 1

Traveler Details:

Name: ali, Email: ass, Contact: 23, Address: 3

Traveler Dashboard:

1. View Traveler Details

2. Book a Seat

3. View Booked Seats

4. Cancel a Booking

5. Back to Main Menu

6. Exit Program

Enter your choice: 5

Welcome! Select your role:

1. Admin

2. Traveler

3. Travel Agent

4. Exit

Enter your choice: 3

Travel Agent Dashboard:

1. Create an Itinerary

2. Manage Bookings

3. Back to Main Menu

4. Exit Program

Enter your choice: 1

Enter Destination: 3

Enter Start Date (yyyy-mm-dd): 12345

Enter End Date (yyyy-mm-dd): 44

Itinerary created for destination: 3, from 12345 to 44

Travel Agent Dashboard:

1. Create an Itinerary

2. Manage Bookings

3. Back to Main Menu

4. Exit Program

Enter your choice: 2

Enter Booking ID: 4

Enter Traveler Name: ali

Booking ID: 4 for Traveler: ali has been managed.

Travel Agent Dashboard:

1. Create an Itinerary

2. Manage Bookings

3. Back to Main Menu

4. Exit Program

Enter your choice: 3

Welcome! Select your role:

1. Admin

2. Traveler

3. Travel Agent

4. Exit

Enter your choice: 5

Invalid choice. Please try again.

Welcome! Select your role:

1. Admin

2. Traveler

3. Travel Agent

4. Exit

Enter your choice: 4

Thank you for using the Travel Management System!

=== Code Execution Successful ===

Code:

```
import java.util.ArrayList;
import java.util.Scanner;

public class Main {

    // Traveler class
    static class Traveler {
        private String name;
        private String email;
        private String contactNumber;
        private String address;
        private ArrayList<String> bookedSeats;
```

```
    public Traveler(String name, String email, String contactNumber, String address) {
        // Creator: Traveler is responsible for creating and managing its own data.
        this.name = name;
        this.email = email;
        this.contactNumber = contactNumber;
        this.address = address;
        this.bookedSeats = new ArrayList<>(); // Information Expert: Manages its bookings.
    }

    // Information Expert: Provides access to traveler details and bookings.
    public String getName() { return name; }
    public String getEmail() { return email; }
    public String getContactNumber() { return contactNumber; }
    public String getAddress() { return address; }
    public ArrayList<String> getBookedSeats() { return bookedSeats; }

    // Information Expert: Handles booking and cancellation of seats.
    public void bookSeat(String seat) { bookedSeats.add(seat); }
    public boolean cancelSeat(String seat) { return bookedSeats.remove(seat); }

    @Override
    public String toString() {
        return "Name: " + name + ", Email: " + email + ", Contact: " + contactNumber + ", Address: "
+ address;
    }
  }

  // Admin class
  static class Admin {
    private ArrayList<Traveler> travelers = new ArrayList<>();

    // Information Expert: Admin manages the list of travelers.
    public void addTraveler(Traveler traveler) {
```

```
        travelers.add(traveler); // Low Coupling: Admin adds travelers without knowing their internal details.
        System.out.println("Traveler added successfully!");
    }

    public void viewAllTravelers() {
        // High Cohesion: Provides functionality specific to the Admin's role.
        if (travelers.isEmpty()) {
            System.out.println("No travelers available.");
        } else {
            System.out.println("\nList of Travelers:");
            for (int i = 0; i < travelers.size(); i++) {
                System.out.println((i + 1) + ". " + travelers.get(i).toString());
            }
        }
    }

    public void deleteTraveler(int index) {
        if (index >= 0 && index < travelers.size()) {
            travelers.remove(index); // Information Expert: Handles deletion from its managed list.
            System.out.println("Traveler deleted successfully!");
        } else {
            System.out.println("Invalid index. Please try again.");
        }
    }
}

// TravelAgent class
static class TravelAgent {
    // Controller: TravelAgent handles itinerary and booking management tasks.
    public void createItinerary(String destination, String startDate, String endDate) {
        System.out.println("Itinerary created for destination: " + destination + ", from " + startDate + " to " + endDate);
    }
```

```
    public void manageBookings(String bookingId, String travelerName) {
        // High Cohesion: TravelAgent focuses on booking management.
        System.out.println("Booking ID: " + bookingId + " for Traveler: " + travelerName + " has been managed.");
    }
}

// Polymorphism: Interface for shared behavior among roles
interface Role {
    void showDashboard(); // Polymorphic method to show role-specific dashboards
}

// (Indirection and Protected Variation)TravelService to manage interactions between roles
static class TravelService {
    private Admin admin;
    private TravelAgent travelAgent;

    public TravelService(Admin admin, TravelAgent travelAgent) {
        this.admin = admin;
        this.travelAgent = travelAgent;
    }

    public void addTraveler(Traveler traveler) {
        admin.addTraveler(traveler); // Delegate to Admin
    }

    public void createItinerary(String destination, String startDate, String endDate) {
        travelAgent.createItinerary(destination, startDate, endDate); // Delegate to TravelAgent
    }
}

// Pure Fabrication: Utility class for validating seat booking
static class SeatBookingValidator {
```

```
    public static boolean validateSeat(String seat) {
        return seat.matches("[A-Z][0-9]+"); // Example validation logic
    }
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    boolean isRunning = true;

    Traveler traveler = null; // Indirection: Traveler instance decouples role-specific responsibilities.
    Admin admin = new Admin(); // Creator: Main is responsible for creating Admin and TravelAgent objects.
    TravelAgent agent = new TravelAgent();
    TravelService travelService = new TravelService(admin, agent); // Indirection: Centralized service layer.

    while (isRunning) { // Controller: Main manages user role selection and program flow.
        System.out.println("\nWelcome! Select your role:");
        System.out.println("1. Admin");
        System.out.println("2. Traveler");
        System.out.println("3. Travel Agent");
        System.out.println("4. Exit");
        System.out.print("Enter your choice: ");
        int roleChoice = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        switch (roleChoice) {
            case 1: // Admin role
                while (true) { // Controller: Manages Admin-specific operations.
                    System.out.println("\nAdmin Dashboard:");
                    System.out.println("1. Add Traveler");
                    System.out.println("2. View All Travelers");
                    System.out.println("3. Delete Traveler");
                    System.out.println("4. Back to Main Menu");
```

```
                System.out.println("5. Exit Program");
                System.out.print("Enter your choice: ");
                int adminChoice = scanner.nextInt();
                scanner.nextLine();

                if (adminChoice == 1) {
                    System.out.print("Enter Traveler Name: ");
                    String name = scanner.nextLine();
                    System.out.print("Enter Email: ");
                    String email = scanner.nextLine();
                    System.out.print("Enter Contact Number: ");
                    String contact = scanner.nextLine();
                    System.out.print("Enter Address: ");
                    String address = scanner.nextLine();
                    travelService.addTraveler(new Traveler(name, email, contact, address)); // Use
service layer
                } else if (adminChoice == 2) {
                    admin.viewAllTravelers();
                } else if (adminChoice == 3) {
                    admin.viewAllTravelers();
                    System.out.print("Enter index of traveler to delete: ");
                    int index = scanner.nextInt() - 1;
                    scanner.nextLine();
                    admin.deleteTraveler(index);
                } else if (adminChoice == 4) {
                    break; // Exit to main menu
                } else if (adminChoice == 5) {
                    isRunning = false; // Exit program
                    break;
                } else {
                    System.out.println("Invalid choice.");
                }
            }
            break;
```

```
case 2: // Traveler role
    if (traveler == null) {
        System.out.println("Please enter your details to proceed:");
        System.out.print("Enter Your Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter Your Email: ");
        String email = scanner.nextLine();
        System.out.print("Enter Your Contact Number: ");
        String contact = scanner.nextLine();
        System.out.print("Enter Your Address: ");
        String address = scanner.nextLine();
        traveler = new Traveler(name, email, contact, address);
        System.out.println("Traveler profile created successfully!");
    }

    while (true) { // Controller: Manages Traveler-specific operations.
        System.out.println("\nTraveler Dashboard:");
        System.out.println("1. View Traveler Details");
        System.out.println("2. Book a Seat");
        System.out.println("3. View Booked Seats");
        System.out.println("4. Cancel a Booking");
        System.out.println("5. Back to Main Menu");
        System.out.println("6. Exit Program");
        System.out.print("Enter your choice: ");
        int travelerChoice = scanner.nextInt();
        scanner.nextLine();

        if (travelerChoice == 1) {
            System.out.println("\nTraveler Details:");
            System.out.println(traveler.toString());
        } else if (travelerChoice == 2) {
            System.out.print("Enter Seat Number to Book (e.g., A1, B2): ");
            String seat = scanner.nextLine();
```

```
            if (SeatBookingValidator.validateSeat(seat)) { // Use Pure Fabrication
                traveler.bookSeat(seat);
                System.out.println("Seat " + seat + " booked successfully!");
            } else {
                System.out.println("Invalid seat number format.");
            }
        } else if (travelerChoice == 3) {
            System.out.println("\nBooked Seats:");
            if (traveler.getBookedSeats().isEmpty()) {
                System.out.println("No seats booked yet.");
            } else {
                for (String seat : traveler.getBookedSeats()) {
                    System.out.println("- " + seat);
                }
            }
        } else if (travelerChoice == 4) {
            System.out.print("Enter Seat Number to Cancel Booking: ");
            String seat = scanner.nextLine();
            if (traveler.cancelSeat(seat)) {
                System.out.println("Booking for seat " + seat + " canceled successfully!");
            } else {
                System.out.println("No booking found for seat " + seat + ".");
            }
        } else if (travelerChoice == 5) {
            break; // Back to main menu
        } else if (travelerChoice == 6) {
            isRunning = false; // Exit program
            break;
        } else {
            System.out.println("Invalid choice.");
        }
    }
    break;
```

```
case 3: // Travel Agent role
    while (true) { // Controller: Manages Travel Agent-specific operations.
        System.out.println("\nTravel Agent Dashboard:");
        System.out.println("1. Create an Itinerary");
        System.out.println("2. Manage Bookings");
        System.out.println("3. Back to Main Menu");
        System.out.println("4. Exit Program");
        System.out.print("Enter your choice: ");
        int agentChoice = scanner.nextInt();
        scanner.nextLine();

        if (agentChoice == 1) {
            System.out.print("Enter Destination: ");
            String destination = scanner.nextLine();
            System.out.print("Enter Start Date (yyyy-mm-dd): ");
            String startDate = scanner.nextLine();
            System.out.print("Enter End Date (yyyy-mm-dd): ");
            String endDate = scanner.nextLine();
            travelService.createItinerary(destination, startDate, endDate); // Use service layer
        } else if (agentChoice == 2) {
            System.out.print("Enter Booking ID: ");
            String bookingId = scanner.nextLine();
            System.out.print("Enter Traveler Name: ");
            String travelerName = scanner.nextLine();
            agent.manageBookings(bookingId, travelerName);
        } else if (agentChoice == 3) {
            break; // Back to main menu
        } else if (agentChoice == 4) {
            isRunning = false; // Exit program
            break;
        } else {
            System.out.println("Invalid choice.");
        }
    }
```

```
                break;

            case 4: // Exit program
                isRunning = false;
                break;

            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }

    System.out.println("Thank you for using the Travel Management System!");
    scanner.close();
  }
}
```

Lesson Learned:

In this project, we learnt about time management, division of work and communication skills. We also learned how to work in a team. We also get more ideas and information that never known. For getting the ideas of one another, create a WhatsApp group and face to face communication in the university. We also learnt about different tool for diagram. Every team member focus on the work and help the other member if she doesn't understand. By doing this, our communication skill increased. The class discussion about the project is very helpful for the improvement.

Conclusion:

The **Smart Travel and Itinerary Management System (STIMS)** were developed to enhance the process of planning and managing travel more efficiently and effectively. It provides a user-friendly platform for travelers, travel agents, and administrators. The system streamlines booking accommodations, creating itineraries, and managing reminders, meeting the needs of all stakeholders. The role-based access ensures secure functionality tailored to users, while features like customizable itineraries, travel reminders, and analytics reports make the system comprehensive. With a structured architecture following the MVC framework and GRASP principles, the system achieves a robust, scalable, and maintainable design.