UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Group: Musketeers CSCI927: Service-Oriented Software Engineering

Spring 2024

Anika Salsabil (8764736)

Thisuru Deesan Karunathilaka Aitthappulige (8697127)

Yingqi Hao (8867951)

Shagun Shrestha (7739084)

Md Ab Hayat (8309346)

Md Tanvir Sazib (8064817)

Subject Coordinators:

Dr Jack (Jie) Yang

Dr Guoxin Su

PART A: Service Identification and Specification

For the first task, we identified five service providers (referred to as SPs). Under each SP, we identified three potential services that the Tourism Ecosystem platform offers. In the second task, we developed Business Service Representation Language (BSRL) specifications for each service associated with each SP.

Task (I): Identification of Services

Service Provider	Identified 3 services
1. Tourism Information Center (TIC)	▪ Visitor Information ▪ Tour Bookings ▪ Event Notifications
2. Local Attractions and Museums (LAM)	▪ Ticket Reservations ▪ Guided Tours ▪ Educational Content Delivery
3. Accommodation Providers (AP)	▪ Room Bookings ▪ Guest Services ▪ Feedback and Reviews
4. Restaurants and Cafes (RC)	▪ Table Reservations ▪ Menu Recommendations ▪ Online Ordering
5. Local Transportation Services (LTS)	▪ Ride Bookings ▪ Route Planning ▪ Real-Time Traffic Information

Task (II): BSRL specifications

This section presents each of the three services identified under each SP in the previous section, utilizing the Business Service Representation Language (BSRL).

SP 1: Tourism Information Center (TIC)

Service 1(i): Visitor Information

Service ID	TIC01
Service Name	Visitor Information
Goal	This service provides the necessary information on local culture, events, attractions and general travel tips to the visitors.
Preconditions	The user is logged in to the application and provides required information
Postconditions	Relevant and accurate information is provided to the tourist.
Assumptions	▪ User familiarity with technology and language ▪ Availability of up-to-date information
Inputs	▪ User queries such as location, interest ▪ Date, Time

	▪ Budget (optional)
Outputs	▪ List of Attraction, restaurants, accommodation ▪ Suggested itineraries and Event/ activities schedule ▪ Travel tips and recommendations ▪ Travel cost details
Resources	Tourism database, External API's or real-time update, Local maps
Key Steps	▪ User provides location and other preferences. ▪ System queries tourism database. ▪ Relevant information is retrieved and displayed.
QoS Factors	Information accuracy, Response time (under 2 seconds), Accessibility, Data freshness.

<u>Service 1(ii): Tour Bookings</u>

Service ID	TIC02
Service Name	Tour Bookings
Goal	This service allows tourists to book available tours and excursions based on their preference
Preconditions	The user is logged in to the application and provides required information
Postconditions	The selected tour is booked, and confirmation sent to the user along with relevant and accurate information
Assumptions	▪ User familiarity with technology and language ▪ Availability of up-to-date information for tour availability and pricing
Inputs	▪ Tour type ▪ Preferred Date, Time ▪ Number of Tourists ▪ Personal Information ▪ Payment Details
Outputs	▪ Booking confirmation ▪ Tour schedule details ▪ Payment Receipt ▪ Reminder
Resources	Tourism database, Booking engine, Payment gateway, Notification service
Key Steps	▪ User selects tour based on preferences. ▪ System checks availability. ▪ Payment is processed. ▪ Booking confirmation is generated.
QoS Factors	Ease of use, Secure payment processing, Response time (under 5 seconds), Booking accuracy

<u>Service 1(iii): Event Notifications</u>

Service ID	TIC03
Service Name	Event Notifications
Goal	This service notifies users about future local events and special offers relevant to their preference

Preconditions	The user has subscribed to notifications and already provided location preferences earlier
Postconditions	Users are promptly notified about upcoming events and special offers.
Assumptions	▪ User accessibility to the application and the registered email/ phone number. ▪ Availability of current event schedule and offers ▪ Real time notification sending functionality of the system
Inputs	▪ User location ▪ Event Preference ▪ Notification settings (on/off, frequency etc.)
Outputs	▪ Push notifications ▪ Email/ text message alerts with event details
Resources	Event database, Notification system, Communication APIs (email/ SMS gateway)
Key Steps	▪ User subscribes to event notifications and adds location preferences. ▪ The system checks for relevant events. ▪ Notifications are sent based on user preferences.
QoS Factors	Notification timeliness, Content relevance and correctness

SP 2: Local Attractions and Museums (LAM)

<u>Service 2(i): Ticket Reservations</u>

Service ID	LAM01
Service Name	Ticket Reservations
Goal	This service allows users to facilitate the reservation of tickets for local attractions and museums.
Preconditions	The user is logged in to the application and the local attractions/ museums have available ticket slots.
Postconditions	Ticket reservations are made successfully, and confirmation sent to the user.
Assumptions	▪ System having Real-time ticket availability ▪ Ticket pricing information
Inputs	▪ User selected attraction ▪ Visit Date ▪ Number of tickets ▪ Payment details
Outputs	▪ Reservation confirmation ▪ Ticket details ▪ Payment receipt
Resources	Attraction database, Ticket booking system, Payment Gateway, Notification service
Key Steps	▪ User selects preferred attraction and visit date. ▪ System verifies ticket availability for the chosen attraction. ▪ User provides payment details. ▪ Reservation is finalized and ticket details are sent.

QoS Factors	Fast response time (under 3 seconds), Secure payment processing, Reservation information accuracy, Ease of use

<u>Service 2(ii): Guided Tours</u>

Service ID	LAM02
Service Name	Guided Tours
Goal	This service allows users to book guided tours at local attractions and museums
Preconditions	The user is logged in to the system and the tour has available slots
Postconditions	A guided tour is booked, and confirmation with tour details is sent to the user. The tour should be conducted as scheduled.
Assumptions	▪ Availability of guides ▪ Real-time Tour schedule accessibility
Inputs	▪ User selected tour ▪ Tour Date ▪ Number of tourists ▪ Payment Details
Outputs	▪ Tour booking confirmation with schedule ▪ Guided tour details ▪ Payment receipt
Resources	Tour database, Scheduling system, Payment Gateway, Notification service
Key Steps	▪ User chooses the guided tour and preferred date. ▪ System checks availability. ▪ User enters payment details. ▪ Booking is confirmed, and tour details are sent. ▪ Conduct tour
QoS Factors	Accurate tour scheduling, Guide punctuality, Visitor satisfaction, secure payment,

<u>Service 2(iii): Educational Content Delivery</u>

Service ID	LAM03
Service Name	Educational Content delivery
Goal	This service allows users to deliver educational content related to the local attractions and museums
Preconditions	The user is logged in to the application and has subscribed to educational content, with pre-selected topic of interest
Postconditions	The relevant educational content is delivered to the users in the desired format (e.g., video, articles).
Assumptions	Availability of up-to-date educational materials, multimedia resources
Inputs	▪ User subscription data (selected topic of interest) ▪ Content type (e.g., video, article, blogposts) ▪ Age group (optional)
Outputs	The educational content (e.g., video, PDF or any interactive material) is delivered to the users.
Resources	Educational content database, multimedia streaming service, digital content platform

Key Steps	▪ User subscribed to educational content ▪ User selects a topic and content type. ▪ System retrieves relevant materials. ▪ Content is delivered to the user.
QoS Factors	Quality educational content, content delivery speed, accuracy, format compatibility based on different devices (mobile, desktop)

SP 3: Accommodation Providers (AP)

<u>Service 3(i): Room Bookings</u>

Service ID	AP01
Service Name	Room Bookings
Goal	Allow guests to search for, choose, and reserve rooms based on availability and preferences.
Preconditions	Rooms must be available, and guests must have access to a booking site that accepts the right payment methods.
Postconditions	The room is successfully reserved, and the guest receives confirmation with the booking information saved in the system.
Assumptions	▪ Guests have internet access. ▪ Rooms are available on specified dates. ▪ Payment methods work.
Inputs	▪ Room preferences (room type, dates) ▪ Guest details (name, contact information, id) ▪ Payment information
Outputs	▪ Booking confirmation. ▪ Booking reference number. ▪ Receipt for payment.
Resources	Booking system, payment gateway
Key Steps	▪ Guest searches for the available rooms in the hotel. ▪ They select a hotel and room and enter booking details. ▪ They confirm booking. ▪ They processed the payment. ▪ Confirmation is sent to the guest.
QoS Factors	Payment processing is reliable, Response period for booking confirmation.

<u>Service 3(ii): Guest Services</u>

Service ID	AP02
Service Name	Guest Services
Goal	Provide personalized services to guests during their stay, such as room service, transportation, and maintenance.
Preconditions	The guest must have a confirmed reservation and access to the service request platform.
Postconditions	The guest's request is completed, and the service is recorded for future billing or feedback.
Assumptions	▪ The guest is present at the lodging. ▪ Staff and resources are available to satisfy requests.

Inputs	▪ Guest location ▪ Guest room request (room services, transport)
Outputs	▪ Customer satisfaction feedback. ▪ Service satisfaction.
Resources	Guest service system, In-house staff, External partners like transport and tours.
Key Steps	▪ Guest makes a service request. ▪ Requests are forwarded to the proper departments. ▪ Services are provided. ▪ The request is marked as finished.
QoS Factors	Guest satisfaction, Timeliness of service provision.

Service 3(iii): Feedback and Reviews

Service ID	AP03
Service Name	Feedback and Reviews
Goal	Collect guest feedback and reviews to help improve services and boost customer happiness.
Preconditions	The guest must have completed their stay and accessed the feedback platform.
Postconditions	The guest's feedback is collected, saved, and optionally publicized to provide insights for service improvement.
Assumptions	▪ Guests are willing to give feedback. ▪ The feedback platform is easily accessible.
Inputs	▪ Guest review or feedback (ratings, comments).
Outputs	▪ Feedback is stored in the system. ▪ Insights into service enhancement.
Resources	Customer relations team, Feedback management system.
Key Steps	▪ Guests are requested to offer feedback. ▪ They complete the feedback form. ▪ Feedback is stored and analyzed. ▪ Practical suggestions are obtained.
QoS Factors	The pace at which guests respond, Easy to submit feedback.

SP 4: Restaurant and Cafes (RC)

Service 4(i): Table Reservations

Service ID	RC01
Service Name	Table Reservations
Goal	Enables users to reserve tables at local restaurants, ensuring they have a spot during their desired dining time.
Preconditions	The user is logged in to the application and provides required information such as the restaurant and number of guests.
Postconditions	A reservation confirmation is generated and sent to the user.
Assumptions	User familiarity with the application and language, availability of reservation slots at the restaurant.

Inputs	▪ Restaurant ID ▪ Date and Time ▪ Number of Guests ▪ User ID
Outputs	▪ Reservation Confirmation ▪ Restaurant Details ▪ Possible alternatives if the preferred time is unavailable
Resources	▪ Restaurant reservation system, ▪ External APIs for real-time availability, ▪ User database for preference tracking
Key Steps	▪ User selects a restaurant and provides reservation details. ▪ System checks availability in the restaurant reservation system. ▪ If available, the reservation is confirmed and stored. ▪ User receives confirmation details.
QoS Factors	▪ Confirmation response time (under 2 seconds) ▪ Reservation accuracy ▪ Accessibility for users with disabilities ▪ Real-time updates on reservation status

Service 4(ii): Menu Recommendations

Service ID	RC02
Service Name	Menu Recommendations
Goal	This service provides personalized menu recommendations based on user preferences and dietary restrictions.
Preconditions	The user is logged into the application and provides dietary preferences.
Postconditions	A personalized list of recommended menu items is generated.
Assumptions	User familiarity with the application and availability of up-to-date menu information.
Inputs	▪ User ID ▪ Dietary Preferences (e.g., vegetarian, allergies) ▪ Restaurant ID
Outputs	▪ Recommended menu items list ▪ Details for each recommended dish
Resources	▪ Restaurant menu database ▪ User preference records
Key Steps	▪ User inputs dietary preferences and selects a restaurant. ▪ System queries the menu database for relevant dishes. ▪ Displays recommended menu items to the user.
QoS Factors	▪ Recommendation accuracy ▪ Response time (under 2 seconds) ▪ Accessibility ▪ Data freshness

Service 4(iii): Online Ordering

Service ID	RC03
Service Name	Online Ordering

Goal	This service allows users to place orders online for delivery or pickup.
Preconditions	The user is logged into the application and has selected a restaurant and menu items.
Postconditions	An order confirmation is generated and sent to the user.
Assumptions	User familiarity with the online ordering process and that the restaurant can handle online orders.
Inputs	▪ User ID ▪ Restaurant ID ▪ Menu Items ▪ Quantity ▪ Delivery Address (optional)
Outputs	▪ Order Confirmation ▪ Estimated Delivery Time ▪ Total Cost
Resources	▪ Restaurant order management system ▪ Payment processing system
Key Steps	▪ User selects menu items and enters order details. ▪ System processes the order and confirms availability. ▪ Generates order confirmation and sends it to the user.
QoS Factors	▪ Order processing time (under 2 seconds) ▪ Order accuracy ▪ Accessibility ▪ Data security

SP 5: Local Transportation Services (LTS)

Service 5(i): Ride Bookings

Service ID	LTS01
Service Name	Ride Bookings
Goal	This service allows users to book local transportation, providing reliable travel options to their desired destinations.
Preconditions	The user is logged into the application and provides necessary details for the ride request.
Postconditions	A ride confirmation is generated, and the user is notified of the ride details.
Assumptions	User familiarity with the application and availability of rides in the area.
Inputs	▪ Pickup Location ▪ Destination ▪ Date and Time ▪ User ID ▪ Type of Vehicle (optional)
Outputs	▪ Ride Confirmation ▪ Estimated Arrival Time ▪ Driver Details (name, vehicle, contact) ▪ Fare Estimate
Resources	▪ Local transportation database ▪ Real-time vehicle tracking systems

	▪ Payment processing system
Key Steps	▪ User enters pickup and destination details. ▪ System queries local transportation database for available rides. ▪ User selects a ride option, and the system confirms the booking. ▪ User receives ride confirmation and driver details.
QoS Factors	▪ Booking response time (under 2 seconds) ▪ Ride availability accuracy ▪ Accessibility ▪ Data security

<u>Service 5(ii): Route Planning</u>

Service ID	LTS02
Service Name	Route Planning
Goal	This service provides users with optimal route planning to help them reach their destinations efficiently.
Preconditions	The user is logged into the application and provides starting and ending locations.
Postconditions	The best route and related information are generated.
Assumptions	User familiarity with the application and accurate location information.
Inputs	▪ Starting Point ▪ Destination ▪ Date and Time
Outputs	▪ Best Route Details ▪ Estimated Travel Time ▪ Traffic Conditions
Resources	▪ Map service APIs ▪ Real-time traffic data
Key Steps	▪ User inputs starting point and destination. ▪ System queries map services for the optimal route. ▪ Displays planned route and estimated time to the user.
QoS Factors	▪ Route planning accuracy ▪ Response time (under 2 seconds) ▪ User interface friendliness ▪ Real-time traffic updates

<u>Service 5(iii): Real time Traffic Information</u>

Service ID	LTS03
Service Name	Real-Time Traffic Information
Goal	This service provides real-time traffic updates to help users choose the best travel times and routes.
Preconditions	The user is logged into the application and provides route information.
Postconditions	The user receives the latest traffic condition information.
Assumptions	User familiarity with the application and the system's ability to fetch real-time data.
Inputs	Route ID

Outputs	▪ Real-Time Traffic Updates ▪ Adjusted Estimated Arrival Times
Resources	▪ Traffic monitoring systems ▪ Map service APIs
Key Steps	▪ User inputs travel route. ▪ System queries real-time traffic monitoring data. ▪ Displays the latest traffic information and any adjusted arrival times.
QoS Factors	▪ Data accuracy ▪ Update frequency (real-time) ▪ Response time (under 2 seconds) ▪ Accessibility

PART B: Enterprise Architecture (EA) Design using ArchiMate

Task (I):

In the Enterprise Architecture (EA) Design of the Tourism Ecosystem platform, we structured the architecture across three essential layers: the Business Layer, Application Layer, and Technology Layer. Each layer plays a pivotal role in ensuring the platform functions seamlessly and delivers an optimized user experience.

Business Layer

The Business Layer focuses on defining the core services and business processes that the platform offers to its users and service providers. This includes:

- **Tourism Services**: The primary business services offered through the platform, including room bookings, table reservations, and route planning for local transportation.
- **Customer and Provider Interactions**: The platform facilitates interactions between customers, who use the platform to make bookings and reservations, and service providers, who offer accommodations, dining, and transportation options.

These services are interconnected, ensuring a smooth flow of information from the customer to the providers, enabling real-time bookings, feedback collection, and service management.

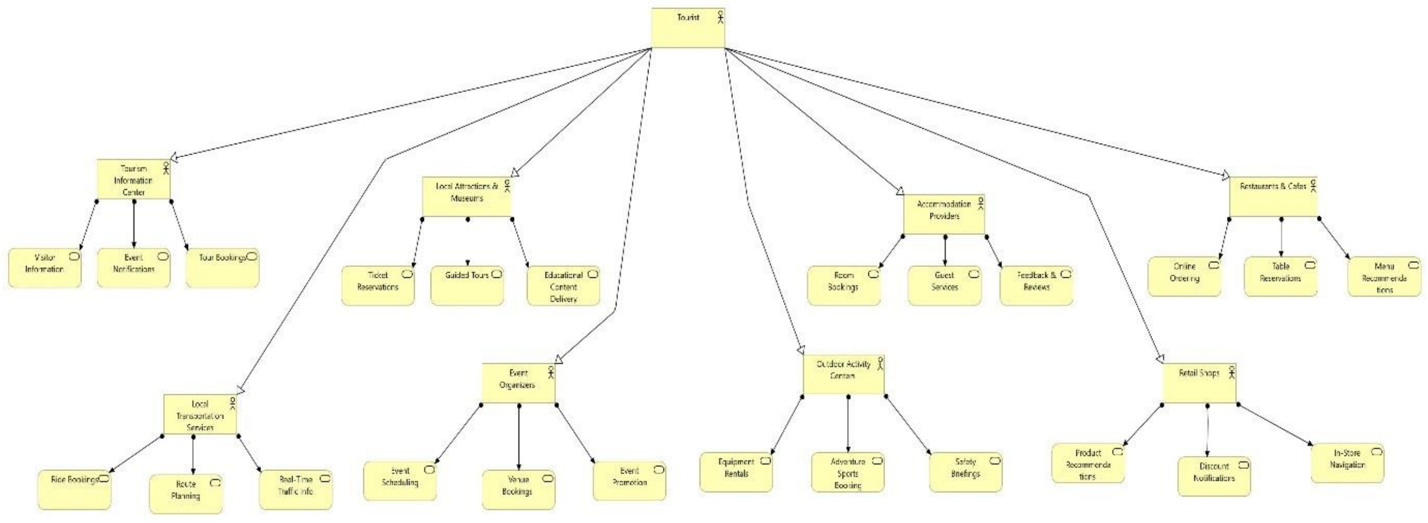


Figure: Business layer

Application Layer

The Application Layer represents the software systems and components that implement the business functions. This layer comprises:

- **Booking Management System**: Responsible for handling room and table bookings. It processes user inputs, checks availability, confirms bookings, and sends notifications.
- **Transportation System**: Manages the route planning feature by integrating real-time traffic data to suggest optimized routes for users.
- **Feedback Collection System**: Allows customers to submit feedback, which is collected and made available to service providers to improve service quality.

These application components are critical in ensuring that business processes run smoothly and are supported by automated software systems.

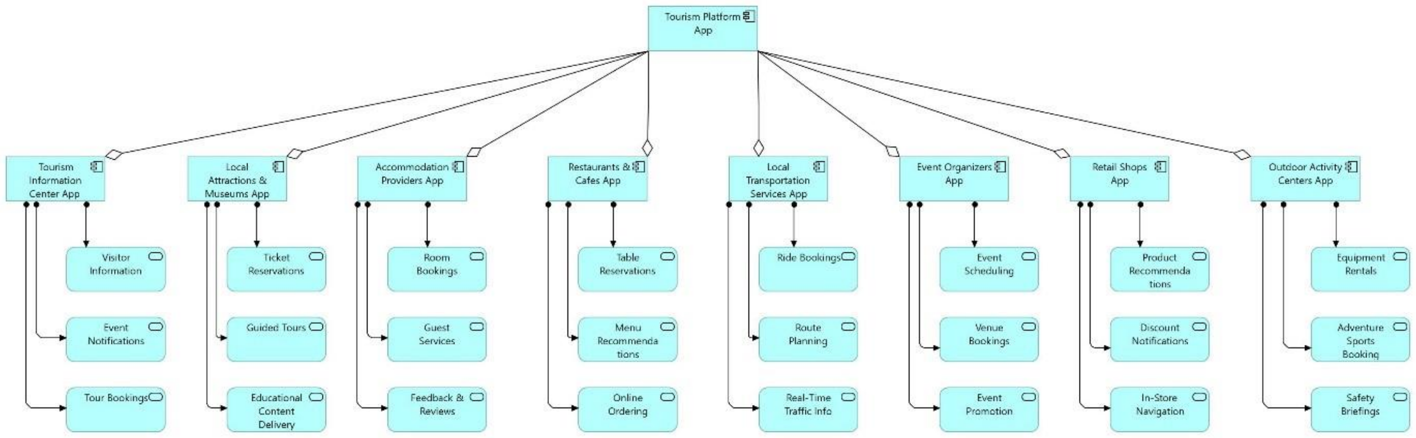


Figure: Application layer

Technology Layer

The **Technology Layer** supports the platform through its underlying infrastructure, ensuring that the application systems are secure, scalable, and reliable. It consists of:

- **Cloud Hosting and Servers**: The platform is hosted on cloud infrastructure, enabling it to scale based on demand and maintain high availability.
- **External APIs**: The platform uses APIs (such as Google Maps API) for real-time traffic data integration, which powers the route optimization in the Transportation System.
- **Database Systems**: Secure databases store all critical data, including customer details, booking information, and feedback from users.

This technology infrastructure enables the application components to function efficiently while maintaining security, speed, and reliability.

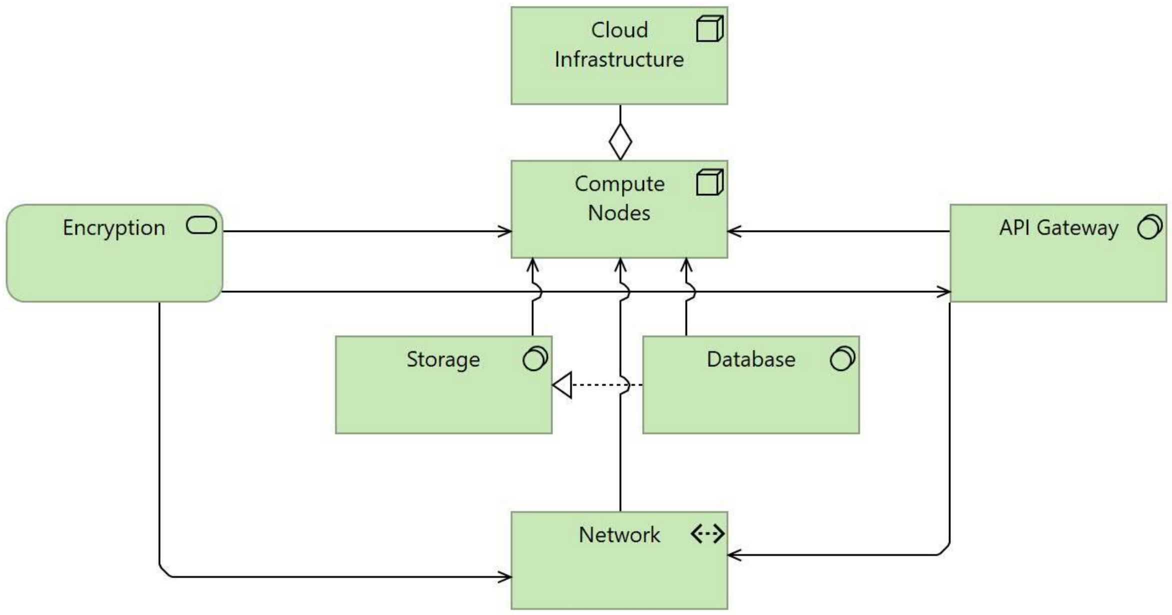
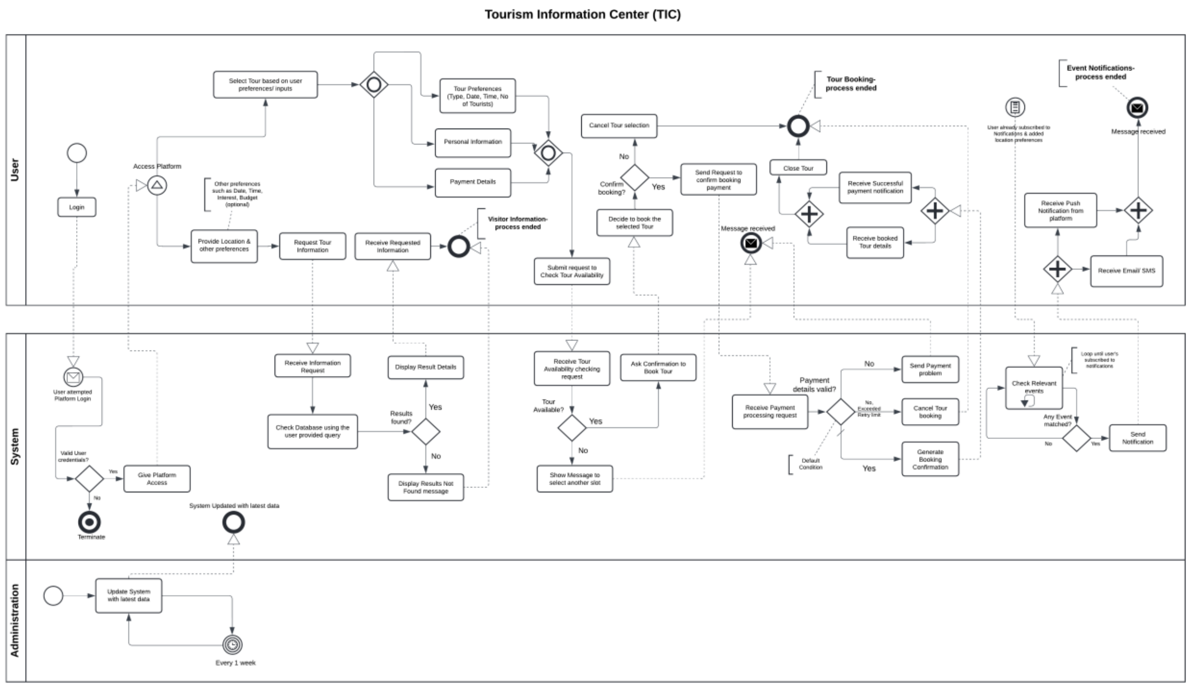


Figure: Technology Layer

PART C: Business Process Design and Analysis

Task (I):

To begin the Business Process Design and Analysis for the Tourism Ecosystem project, we must build BPMN (Business Process Model and Notation) diagrams for at least five of the services listed. Here's a breakdown on how to proceed with each:


Tourism Information Center (TIC)
User
System
Administration
Login
Access Platform
Select Tour based on user preferences/ inputs
Tour Preferences (Type, Date, Time, No of Tourists)
Personal Information
Payment Details
Other preferences such as Date, Time, Interest, Budget (optional)
Provide Location & other preferences
Request Tour Information
Receive Requested Information
Visitor Information- process ended
Submit request to Check Tour Availability
Cancel Tour selection
Confirm booking?
No
Yes
Decide to book the selected Tour
Send Request to confirm booking payment
Tour Booking- process ended
Close Tour
Receive Successful payment notification
Receive booked Tour details
Message received
User already subscribed to Notifications & added location preferences
Event Notifications- process ended
Message received
Receive Push Notification from platform
Receive Email/ SMS
User attempted Platform Login
Valid User credentials?
Yes
No
Give Platform Access
Terminate
Receive Information Request
Check Database using the user provided query
Display Result Details
Results found?
Display Results Not Found message
System Updated with latest data
Receive Tour Availability checking request
Tour Available?
Ask Confirmation to Book Tour
Show Message to select another slot
Receive Payment processing request
Payment details valid?
Send Payment problem
No, Exceeded Retry limit
Cancel Tour booking
Default Condition
Generate Booking Confirmation
Check Relevant events
Loop until user's subscribed to notifications
Any Event matched?
Send Notification
Update System with latest data
Every 1 week

SP 1: Tourism Information Center (TIC)

Source for the Tourism Information Center (TIC) Service Process Model

The BPMN model for the TIC system, which includes **Visitor Information**, **Tour Bookings**, and **Event Notifications**, is based on various online booking systems. Below is the source explanation for the key steps in the model:

Visitor Information:

Visit London is the main reference inspired for the Visitor information module. It offers information to customers about places of interest, maps, and means of transportation.

- **Reference**: Visit London

Tour Bookings:

Viator inspires the Tour Booking module. This reference explains how customers can search and book city tours based on user preferences, guided tour experience, etc.

- **Reference**: Viator

Event Notifications:

Eventbrite is the main example which was considered when developing the Event Notification module. This platform allows to search for and sign up for upcoming events, such as concerts.

Reference: Eventbrite

Annotation:

User Swimlane

1. Log In

- **Semantic Annotation**: The user logs in to access the tourism platform.
- **Immediate Effect**:
- The user is successfully authenticated, and the system grants access.
- **FOL**: LogIn(User) -> ValidLogin(System)
- **Cumulative Effect**:
- The successful login allows the user to proceed with tour queries and booking services.
- **FOL**: ValidLogin -> AccessTourServices(User)

2. Select Tour Based on User Preferences

- **Semantic Annotation**: The user selects a tour based on preferences such as location, time, and tour type.
- **Immediate Effect**:
- The system displays available tour options for the user.
- **FOL**: SelectTour(User, Preferences) -> TourOptionsDisplayed(System)
- **Cumulative Effect**:
- The selected tour will affect the subsequent steps of checking availability and making a reservation.
- FOL: TourOptionsDisplayed -> InitiateBookingRequest(User)

3. Request Tour Information

- **Semantic Annotation**: The user requests detailed information about the selected tour.
- **Immediate Effect**:
- The system provides the requested tour information to the user.
- **FOL**: RequestTourInfo(User) -> TourInfoProvided(System)
- **Cumulative Effect**:
- The provided information helps the user decide whether to proceed with the booking.
- **FOL**: TourInfoProvided -> MakeBookingDecision(User)

4. Personal Information & Payment Details

- **Semantic Annotation**: The user enters personal information and payment details to complete the booking.
- **Immediate Effect**:
- The system receives and processes the user's personal and payment details.
- **FOL**: SubmitPersonalInfo(User) -> PersonalInfoReceived(System)
- **Cumulative Effect**:
- Valid personal information and payment details allow the system to proceed with payment validation and booking confirmation.
- **FOL**: PersonalInfoReceived -> ProcessPaymentAndBooking(System)

System Swimlane

1. Validate User Login

- **Semantic Annotation**: The system verifies the user's login credentials to grant access to the platform.
- **Immediate Effect**:
- Upon successful authentication, the system grants platform access.
- **FOL**: ValidateLogin(System, User) -> AccessGranted(User)
- **Cumulative Effect**:
- The system can track and log the user's activity for further processing.
- **FOL**: AccessGranted -> UserActivityRecorded(System)

2. Check Tour Availability

- **Semantic Annotation**: The system checks the availability of the selected tour based on the user's input.
- **Immediate Effect**:
- The system confirms whether the tour is available or not.
- **FOL**: CheckAvailability(Tour) -> (Available ∨ Unavailable)
- **Cumulative Effect**:
- The availability result determines whether the booking process can proceed.
- **FOL**: Available -> ContinueBookingProcess(User) ∨ Unavailable -> SendUnavailableMessage(User)

3. Payment Validation

- **Semantic Annotation**: The system validates the user's payment information for the booking.
- **Immediate Effect**:

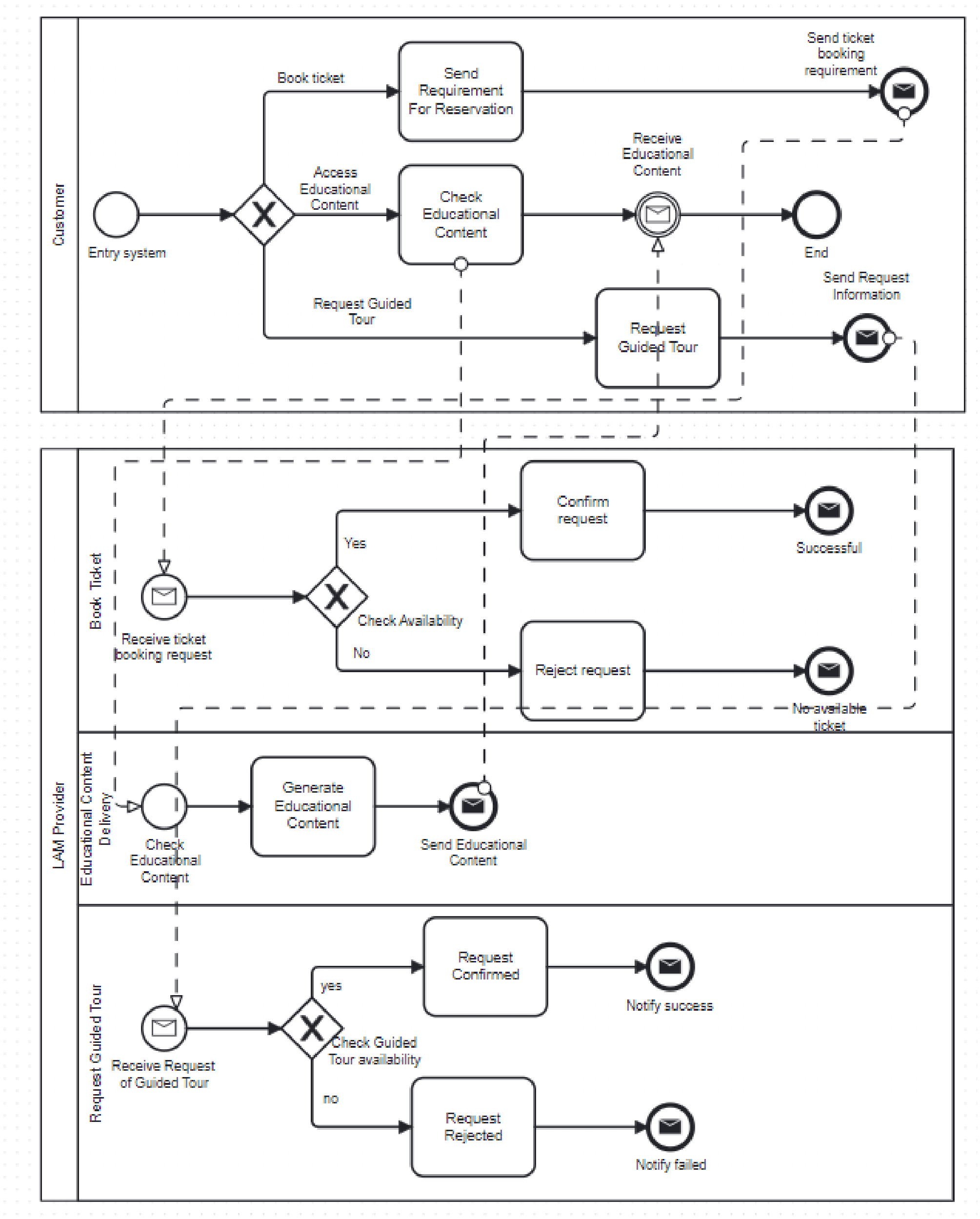
- If the payment is valid, the system allows the booking to proceed.
- **FOL**: ValidatePayment(System) -> PaymentConfirmed(User)
- **Cumulative Effect**:
- Validated payment leads to booking confirmation and completes the payment process.
- **FOL**: PaymentConfirmed -> ConfirmBooking(User)

Administration Swimlane

1. Update System with Latest Data

- **Semantic Annotation**: The administration regularly updates the system with the latest tour information.
- **Immediate Effect**:
- The updated data is uploaded into the system for users to query.
- **FOL**: UpdateSystem(Admin) -> DataUpdated(System)
- **Cumulative Effect**:
- The updated data ensures users have access to the most current tour information for accurate bookings.
- **FOL**: DataUpdated -> AccurateTourInfoDisplayed(User)

SP 2: Local Attractions and Museums (LAM)


Customer
Entry system
Book ticket
Send Requirement For Reservation
Send ticket booking requirement
Access Educational Content
Check Educational Content
Receive Educational Content
End
Request Guided Tour
Request Guided Tour
Send Request Information
LAM Provider
Book Ticket
Receive ticket booking request
Check Availability
Yes
No
Confirm request
Successful
Reject request
No available ticket
Educational Content Delivery
Check Educational Content
Generate Educational Content
Send Educational Content
Request Guided Tour
Receive Request of Guided Tour
Check Guided Tour availability
yes
no
Request Confirmed
Notify success
Request Rejected
Notify failed

Source for the Local Attractions and Museums (LAM) Service Process Model

The BPMN model for the LAM system, which includes Ticket Reservations, Guided Tours, and Educational Content Delivery, is based on various online booking systems and content delivery platforms. Below is the source explanation for the key steps in the model:

- Ticket Reservation Process:

This step references the API documentation and booking processes used in **Eventbrite** and similar online ticketing platforms. It details how customers can select available dates, check ticket availability, and make reservations for museum visits or events. The system processes the request, confirms availability, and generates a booking confirmation.

- Guided Tour Scheduling:

The guided tour process is modeled on tour scheduling systems such as those used by **GetYourGuide**. This reference explains how customers can request a tour based on available time slots and how the provider confirms or rejects the request depending on availability.

- Educational Content Access:

This step is inspired by Coursera’s educational content delivery model, where users can request access to specific learning materials. The system grants access based on content availability and provides users with downloadable materials or links to view content online.

- Reference: Coursera Content Delivery GetYourGuide Tour Booking System

Annotation

1. Send Requirement for Reservation

- **Semantic Annotation:** The customer initiates a ticket reservation request, and the system processes it.
- **Immediate Effect:**
- **Effect:** The ticket reservation request is sent to the attraction provider for processing.
- **FOL:** SendReservationRequest(Customer) -> ReservationRequestSent(Provider)
- **Cumulative Effect:**
- **Effect:** The reservation request enters the processing flow, where ticket availability will be checked.
- **FOL:** ReservationRequestSent -> CheckTicketAvailability

2. Request Guided Tour

- **Semantic Annotation:** The customer initiates a request for a guided tour, and the system processes it.
- **Immediate Effect:**
- **Effect:** The guided tour request is sent to the attraction provider for processing.
- **FOL:** RequestGuidedTour(Customer) -> GuidedTourRequestSent(Provider)
- **Cumulative Effect:**
- **Effect:** The request enters the processing flow, and the system will check the availability of the guided tour.
- **FOL:** GuidedTourRequestSent -> CheckGuidedTourAvailability

3. Access Educational Content

- **Semantic Annotation:** The customer accesses the educational content provided by the attraction.
- **Immediate Effect:**
- **Effect:** The customer requests educational content, and the system generates and sends the content.
- **FOL:** AccessEducationalContent(Customer) -> EducationalContentGenerated(Provider)
- **Cumulative Effect:**
- **Effect:** The educational content is generated and sent to the customer, moving the process to the content delivery stage.
- **FOL:** EducationalContentGenerated -> EducationalContentSent(Customer)

4. Check Ticket Availability

- **Semantic Annotation:** The attraction provider checks the availability of tickets to determine if the reservation can be confirmed.
- **Immediate Effect:**
- **Effect:** If tickets are available, the system confirms the reservation; otherwise, the reservation is rejected.
- **FOL:** CheckTicketAvailability(Provider) -> (ConfirmReservation ∨ RejectReservation)
- **Cumulative Effect:**
- **Effect:** Based on ticket availability, the system determines whether the customer's reservation is successful, and the process continues.
- **FOL:** TicketAvailable -> ConfirmReservation ∨ TicketUnavailable -> RejectReservation

5. Confirm Reservation

- **Semantic Annotation:** If tickets are available, the attraction provider confirms the reservation and notifies the customer.
- **Immediate Effect:**
- **Effect:** The reservation is successful, and the customer receives confirmation.
- **FOL:** ConfirmReservation(Provider) -> ReservationConfirmed(Customer)
- **Cumulative Effect:**
- **Effect:** The customer's reservation is confirmed, and the process continues to completion.
- **FOL:** ReservationConfirmed(Customer) -> ReadyForArrival(Customer)

6. Reject Reservation

- **Semantic Annotation:** If tickets are unavailable, the attraction provider rejects the reservation and notifies the customer.
- **Immediate Effect:**

- **Effect:** The reservation fails, and the customer receives a rejection notice.
- **FOL:** RejectReservation(Provider) -> ReservationRejected(Customer)
- **Cumulative Effect:**
- **Effect:** The reservation process ends, and the customer may choose to try again or cancel.
- **FOL:** ReservationRejected -> EndOrRetry

7. Check Guided Tour Availability

- **Semantic Annotation:** The attraction provider checks the availability of guided tours to determine if the request can be confirmed.
- **Immediate Effect:**
- **Effect:** If the guided tour is available, the system confirms the request; otherwise, the request is rejected.
- **FOL:** CheckGuidedTourAvailability(Provider) -> (ConfirmTour ∨ RejectTour)
- **Cumulative Effect:**
- **Effect:** Based on tour availability, the system determines whether the customer’s request is successful, and the process continues.
- **FOL:** TourAvailable -> ConfirmTour ∨ TourUnavailable -> RejectTour

8. Confirm Guided Tour

- **Semantic Annotation:** If the guided tour is available, the attraction provider confirms the service and notifies the customer.
- **Immediate Effect:**
- **Effect:** The guided tour request is successful, and the customer receives confirmation.
- **FOL:** ConfirmTour(Provider) -> TourConfirmed(Customer)
- **Cumulative Effect:**
- **Effect:** The guided tour has been confirmed, and the process continues.
- **FOL:** TourConfirmed(Customer) -> ReadyForGuidedTour(Customer)

9. Reject Guided Tour

- **Semantic Annotation:** If the guided tour is unavailable, the attraction provider rejects the request and notifies the customer.
- **Immediate Effect:**
- **Effect:** The guided tour request fails, and the customer receives a rejection notice.
- **FOL:** RejectTour(Provider) -> TourRejected(Customer)
- **Cumulative Effect:**
- **Effect:** The guided tour request process ends, and the customer may choose to try again or cancel.
- **FOL:** TourRejected -> EndOrRetry

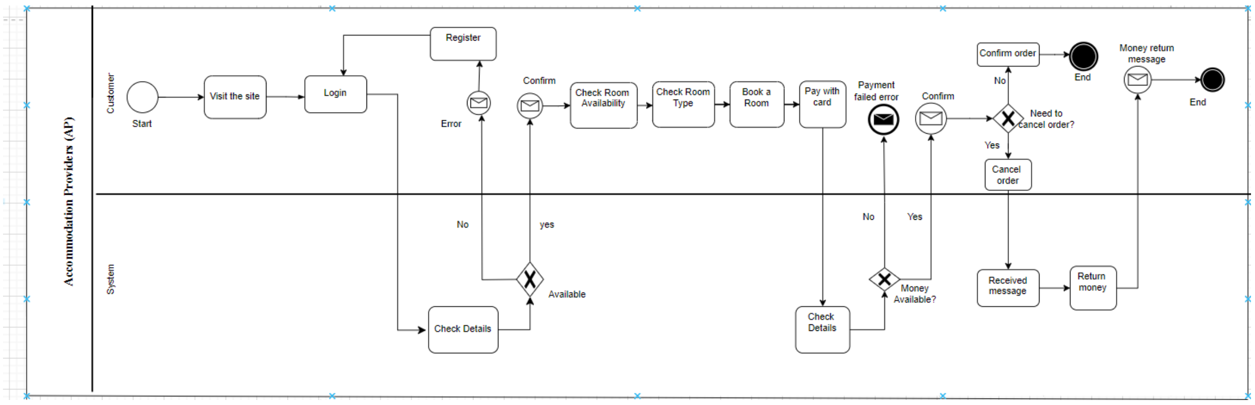
10. Generate Educational Content

- **Semantic Annotation:** The attraction provider generates educational content and sends it to the customer.
- **Immediate Effect:**
- **Effect:** The educational content is generated and sent to the customer.
- **FOL:** GenerateEducationalContent(Provider) -> EducationalContentGenerated
- **Cumulative Effect:**
- **Effect:** The educational content is generated and sent, moving the process into the delivery stage.
- **FOL:** EducationalContentGenerated -> EducationalContentSent(Customer)

11. End

- **Semantic Annotation:** The process concludes, and all tasks are completed.
- **Immediate Effect:**
- **Effect:** The system completes all customer requests and ends the process.
- **FOL:** ProcessEnd

SP 3: Accommodation Providers (AP)



Source for the Accommodation Providers (AP) Service Process Model

The BPMN model for the AP system, which includes Room Bookings, Guest Services, and Feedback and Reviews, is based on various hotel booking systems and hospitality management platforms. Below is the source explanation for the key steps in the model:

- Room Booking Process:

This method refers to the booking systems used by Booking.com and Expedia. These systems enable users to browse for available rooms, choose accommodation kinds, and complete the booking process by making online payments. The system confirms the booking and issues a confirmation receipt to the customer.

Reference**:** Booking.com, Expedia

- Guest Services:

The guest service model is based on Hilton's Digital Key and Marriott's Mobile App offerings. These platforms enable hotel customers to check in, access rooms, and request additional services such as housekeeping or transportation using mobile apps, which are all monitored in real time by the hotel's system.

Reference**:** Hilton Digital Key, Marriott Mobile App

- Feedback and Reviews:

This phase is modelled after TripAdvisor and Google Reviews' feedback and review systems, which allow customers to rate their stay and submit comments about their experience. The system verifies the submission, publishes the comments, and may utilize the reviews to make suggestions to prospective consumers.

Reference: TripAdvisor, Google Reviews

Annotation

1. Visit the site

- **Semantic Annotation**: The customer opens the website to browse accommodation options.
- **Immediate Effect**: The customer accesses the site and the system loads available room options.
  - **FOL**: VisitSite(Customer) -> LoadRoomOptions(System)
- **Cumulative Effect**: The system has successfully loaded room options for the customer to view.
  - **FOL**: LoadRoomOptions(System) -> ReadyForSelection(Customer)

2. Login

- **Semantic Annotation**: The customer logs in to access personalized services, including previous bookings.
- **Immediate Effect**: The system verifies the customer's credentials and grants access to their account.
  - **FOL**: Login(Customer) -> AccountAccessGranted(System)
- **Cumulative Effect**: The customer is now able to proceed with making a reservation.
  - **FOL**: AccountAccessGranted -> ReadyForReservation(Customer)

3. Check Room Availability

- **Semantic Annotation**: The customer checks the availability of rooms on the desired dates.
- **Immediate Effect**: The system checks the availability of rooms for the requested date and notifies the customer.
  - **FOL**:CheckRoomAvailability(Customer)->(RoomsAvailable∨ RoomsUnavailable)
- **Cumulative Effect**: Based on availability, the customer can either proceed with the booking or search for other options.
  - **FOL**: RoomsAvailable -> ProceedToBooking(Customer) ∨ RoomsUnavailable -> SearchNewDates(Customer)

4. Select Room Type

- **Semantic Annotation**: The customer selects a specific room type based on availability and preferences.
- **Immediate Effect**: The system reserves the chosen room type for a short period until the payment is confirmed.
  - **FOL**: SelectRoomType(Customer) -> RoomTypeReserved(System)
- **Cumulative Effect**: The room type is temporarily held while the payment process is initiated.
  - **FOL**: RoomTypeReserved -> AwaitingPayment(Customer)

5. Book a Room

- **Semantic Annotation**: After selecting a room type, the customer proceeds to reserve the room.
- **Immediate Effect**: The system temporarily reserves the room for the customer.
- **FOL**: BookRoom(Customer) -> RoomReservedTemporarily(System)
- **Cumulative Effect**: The room is reserved, and the customer proceeds to payment.
- **FOL**: RoomReservedTemporarily -> ProceedToPayment(Customer)

6. Make Payment

- **Semantic Annotation**: The customer makes the payment to confirm the reservation.
- **Immediate Effect**: The system processes the payment and confirms the room booking upon successful transaction.
  - **FOL**: MakePayment(Customer) -> (PaymentSuccess ∨ PaymentFailure)
- **Cumulative Effect**: If payment is successful, the room booking is finalized and confirmation is sent to the customer.

    - **FOL**:PaymentSuccess->BookingConfirmed(System)∨PaymentFailure -> RetryPayment(Customer)

7. **Pay with Card**

- **Semantic Annotation**: The customer enters payment details to complete the booking.
- **Immediate Effect**: The system processes the payment and verifies if the transaction is successful.
    - **FOL**: PayWithCard(Customer) -> PaymentProcessed(System)
- **Cumulative Effect**: If the payment is successful, the booking is confirmed.
    - **FOL**: PaymentProcessed -> (PaymentSuccessful ∨ PaymentFailed)

8. Payment Failed (Error Case)

- **Semantic Annotation**: If the payment fails, the system prompts the customer to retry or choose another method.
- **Immediate Effect**: The system displays an error message and asks the customer to retry the payment.
    - **FOL**: PaymentFailed -> ErrorMessageDisplayed(System)
- **Cumulative Effect**: The customer retries the payment or cancels the booking process.
    - **FOL**:ErrorMessageDisplayed -> (RetryPayment(Customer) ∨ BookingCancelled(Customer))

9. Booking Confirmation

- **Semantic Annotation**: The system confirms the booking and sends a confirmation message to the customer.
- **Immediate Effect**: A confirmation email or message is sent to the customer with booking details.
    - **FOL**: ConfirmBooking(System) -> ConfirmationSent(Customer)
- **Cumulative Effect**: The customer has now successfully booked a room and is ready for check-in on the scheduled date.
    - **FOL**: ConfirmationSent -> BookingFinalized(Customer)

10. Confirm Check-out

- **Semantic Annotation**: The customer confirms their room booking after completing their stay.
- **Immediate Effect**: The system marks the booking as complete and processes check-out.
    - **FOL**: ConfirmCheckout(Customer) -> BookingCompleted(System)
- **Cumulative Effect**: The booking is completed, and the customer's stay is officially concluded.
    - **FOL**: BookingCompleted -> StayFinalized(Customer)

11. Request a Refund (Cancellation)

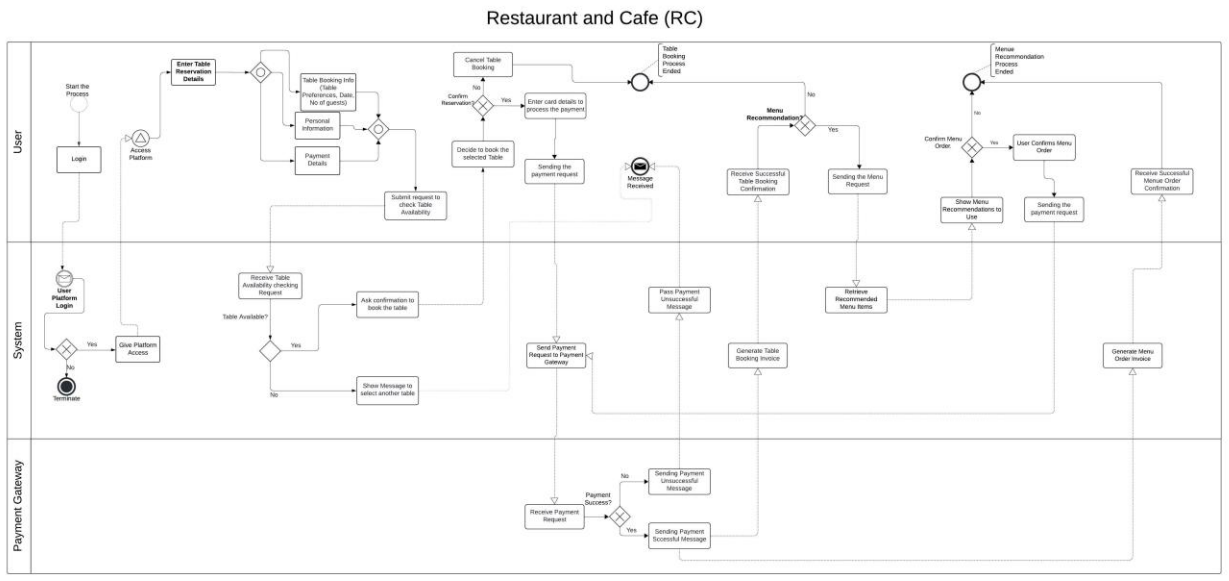
- **Semantic Annotation**: If the customer cancels the booking, they request a refund.
- **Immediate Effect**: The system processes the refund request.

        - **FOL**: RequestRefund(Customer) -> RefundProcessed(System)
    - **Cumulative Effect**: The refund is processed, and the booking is cancelled.
        - **FOL**: RefundProcessed -> BookingCancelled(Customer)

12. End

- **Semantic Annotation**: The process concludes with the booking either confirmed or cancelled.
- **Immediate Effect**: The system terminates the process after confirmation or cancellation.
    - **FOL**: ProcessEnd -> ProcessTerminated(System)
- **Cumulative Effect**: The customer's booking journey has ended, with either a confirmed stay or a cancellation and refund.
    - **FOL**: ProcessTerminated -> BookingJourneyCompleted(Customer)

SP 4: Restaurant and Cafes (RC)


Restaurant and Cafe (RC)
User
System
Payment Gateway
Start the Process
Login
Access Platform
Enter Table Reservation Details
Table Booking Info (Table Preferences, Date, No of guests)
Personal Information
Payment Details
Submit request to check Table Availability
Cancel Table Booking
No
Confirm Reservation?
Yes
Enter card details to process the payment
Decide to book the selected Table
Sending the payment request
Message Received
Table Booking Process Ended
No
Menu Recommondation?
Yes
Receive Successful Table Booking Confirmation
Sending the Menu Request
Menue Recommondation Process Ended
No
Confirm Menu Order.
Yes
User Confirms Menu Order
Show Menu Recommendations to Use
Sending the payment request
Receive Successful Menue Order Confirmation
User Platform Login
Yes
Give Platform Access
No
Terminate
Receive Table Availability checking Request
Table Available?
Yes
Ask confirmation to book the table
No
Show Message to select another table
Send Payment Request to Payment Gateway
Pass Payment Unsuccessful Message
Generate Table Booking Invoice
Retrieve Recommended Menu Items
Generate Menu Order Invoice
Receive Payment Request
Payment Success?
No
Sending Payment Unsuccessful Message
Yes
Sending Payment Scoessful Message

Source for the Restaurants and Cafes (RC) Service Process Model

The BPMN model for the RC system, which includes Table Reservations, Menu Recommendations, and Online Ordering, is based on various online booking systems. Below is the source explanation for the key steps in the model:

- Table Reservation Process:

OpenTable is the main example of the Table Reservation Process. It is the most widely used reservation platform and offers real-time table booking and much more. The system processes the request, confirms availability, and generates a booking confirmation.

- Menu Recommendations:

UberEats Menu Personalization inspires this step. This reference explains how customers can get menu recommendations based on user preferences, order history, and popular trends. Then, Customers can confirm the menu or decline it and make their own personal choices.

Reference: UberEats Menu Personalization

- Online Ordering:

DoorDash is the main example of an online ordering system. It offers the customer with flexible delivery options. The system process allows the users to select foods from their preferred restaurant and then offers the payment gateway to make the payment and generate the booking confirmation.

Reference: DoorDash

Annotation

1. User Logs In

- **Semantic Annotation:** The user logs into the system, and the system verifies the credentials.
- **Immediate Effect:**

  Effect: The system provides platform access if the credentials are valid.

  FOL: UserLogin(User) -> PlatformAccessGranted(System) ∨ Terminate(System)

- **Cumulative Effect:**

  Effect: Based on whether the login is successful or not, the system either terminates the process or proceeds to allow access.

  FOL: PlatformAccessGranted(System) -> AccessPlatform(User)

2. Enter Table Reservation Details

- **Semantic Annotation:** The user enters reservation details, including table preferences, date, and number of guests.
- **Immediate Effect:**

  Effect: The reservation details are sent to the system for further processing.

  FOL: EnterReservationDetails(User) -> ReservationDetailsSent(System)

- **Cumulative Effect:**

  Effect: The system now has all the necessary details for table availability check.

  FOL: ReservationDetailsSent(System) -> CheckTableAvailability(System)

3. Check Table Availability

- **Semantic Annotation:** The system checks for table availability based on the user's reservation preferences.
- **Immediate Effect:**

  Effect: If a table is available, the system proceeds to the booking confirmation step; otherwise, the user is asked to select another table.

  FOL: CheckTableAvailability(System) -> (TableAvailable -> AskConfirmation(User)) ∨ (TableUnavailable -> SelectAnotherTable(User))

- **Cumulative Effect:**

  Effect: The table reservation process either moves forward to booking or returns to modify table preferences.

  FOL: TableAvailable->AskConfirmation(User)∨TableUnavailable-> RetryTableSelection

4. Confirm Reservation

- **Semantic Annotation:** The user confirms the reservation after verifying table availability.
- **Immediate Effect:**

  Effect: The system submits the table reservation request for booking confirmation.

  FOL: ConfirmReservation(User) -> ReservationRequestSubmitted(System)

- **Cumulative Effect:**

  Effect: Upon user confirmation, the system proceeds to process the reservation.

  FOL: ReservationRequestSubmitted -> ProceedToPayment

5. Process Payment

- **Semantic Annotation:** The user enters payment details, and the system processes the payment through the payment gateway.
- **Immediate Effect:**

  Effect: The system sends a payment request to the payment gateway.

  FOL: EnterPaymentDetails(User) -> PaymentRequestSent(PaymentGateway)

- **Cumulative Effect:**

  Effect: The system awaits a payment success or failure response from the gateway.

  FOL: PaymentRequestSent -> PaymentSuccess ∨ PaymentFailure

6. Payment Successful

- **Semantic Annotation:** The payment gateway confirms successful payment, and the system completes the reservation.
- **Immediate Effect:**

  Effect: A success message is generated, and the reservation is confirmed.

  FOL: PaymentSuccess(PaymentGateway) -> ReservationConfirmed(System)

- **Cumulative Effect:**

  Effect: The system confirms the table reservation and notifies the user of success.

  FOL: ReservationConfirmed(System) -> NotifyUserOfSuccess

7. Menu Recommendation Decision

- **Semantic Annotation:** The user is presented with an option for menu recommendations after the table reservation is completed.
- **Immediate Effect:**

Effect: The system either provides menu recommendations or ends the process based on user input.

FOL: MenuRecommendationOption(User) -> (RequestMenuRecommendations -> MenuRecommendationRequested) ∨ EndProcess

- **Cumulative Effect:**

  Effect: The system either proceeds to recommend menu items or concludes the process.

  FOL: MenuRecommendationRequested -> RecommendMenuItems ∨ EndProcess

8. Send Menu Recommendation

- **Semantic Annotation:** The system retrieves and sends menu recommendations to the user.
- **Immediate Effect:**

  Effect: The user receives the recommended menu items**.**

  FOL: SendMenuRecommendation(System) -> MenuRecommendationReceived(User)

- **Cumulative Effect:**

  Effect: The system presents the menu recommendations, allowing the user to place an order.

  FOL: MenuRecommendationReceived -> ConfirmMenuOrder(User)

9. Confirm Menu Order

- **Semantic Annotation:** The user confirms the menu order after reviewing the recommended items**.**
- **Immediate Effect:**

  Effect: The menu order request is sent to the system for payment processing.

  FOL: ConfirmMenuOrder(User) -> MenuOrderRequestSent(System)

- **Cumulative Effect:**

  Effect: The system proceeds to payment after confirming the menu order.

FOL: MenuOrderRequestSent -> ProceedToMenuPayment

10. Generate Invoices (Reservation and Menu)

- **Semantic Annotation:** The system generates separate invoices for the table booking and the menu order.
- **Immediate Effect:**

  Effect: Invoices for the table and menu are generated and sent to the user.

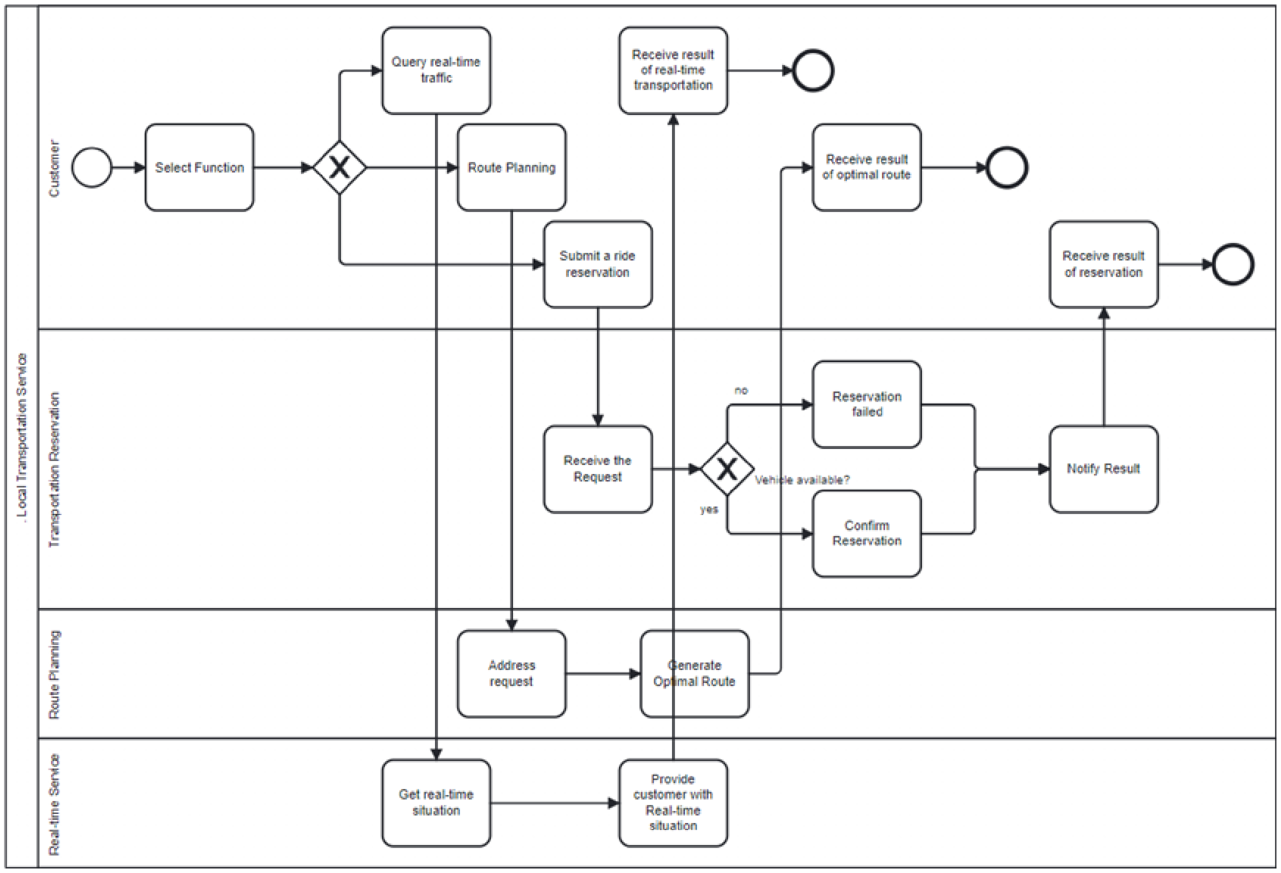
  FOL: GenerateInvoice(System) -> InvoiceGenerated(User)

- **Cumulative Effect:**

  Effect: The process of reservation and menu order is finalized with successful invoice generation.

FOL: InvoiceGenerated -> EndProcess

SP 5: Local Transportation Services (LTS)



Source for the Route Planning Service (LTS) Process Model

The BPMN model for the Route Planning Service is based on several technical documents and API references from navigation and traffic management systems. These resources detail how to process user input, generate multiple route options, and optimize routes based on real-time traffic information. Below is the source explanation for the key steps in the model:

- User Inputs Destination:

This step references the API documentation of a well-known navigation app, Google map. Which describes how the system generates route options based on the location information input by the user. The system analyzes different route choices and optimizes them according to user preferences and real-time traffic data.

- Generate and Optimize Routes:

This step draws from the real-time data processing mechanisms of traffic management systems. The documentation notes that the system adjusts route choices based on real-time traffic information and recommends the best route to the user.

- Route Confirmation and Feedback:

After the user selects the final route, the system generates a confirmation path and displays it on the map, while also updating the user's departure time and estimated arrival time. This part of the process closely aligns with the operation process of Google System

Reference: https://developers.google.com/maps/documentation/routes?hl=zh-cn (Google API documentation).

Annotation

- **Select Function**
- **Semantic Annotation**: The customer selects a function (e.g., query real-time traffic or submit a reservation).
- **Immediate Effect**:
- **Effect**: The customer selects the function to execute, triggering the corresponding subsequent process.
- **FOL**: SelectFunction(Customer) -> (QueryTraffic ∨ RoutePlanning ∨ SubmitReservation)
- **Cumulative Effect**:
- **Effect**: The selected function determines the branching of the subsequent processes, influencing the overall progress.
- **FOL**: SelectFunction(Customer) -> TriggerNextStepBasedOnFunction
- **Query Real-Time Traffic**
- **Semantic Annotation**: The customer queries current real-time traffic information.
- **Immediate Effect**:
- **Effect**: The system retrieves and returns the customer's queried real-time traffic information.
- **FOL**: QueryTraffic(Customer) -> RealTimeInfoRetrieved(TrafficData)
- **Cumulative Effect**:
- **Effect**: The real-time traffic information is used to update the customer’s decisions, such as route planning or adjusting reservation requests.
- **FOL**: RealTimeInfoRetrieved(TrafficData) -> AvailableForRoutePlanning
- **Route Planning**
- **Semantic Annotation**: The system plans a route based on the customer’s input of the start and end locations.
- **Immediate Effect**:
- **Effect**: The system generates a route based on the input and the current traffic data.
- **FOL**: PlanRoute(System, Start, End) -> RouteGenerated(RouteID)
- **Cumulative Effect**:
- **Effect**: The planned route will be used for subsequent steps such as reservation confirmation or traffic updates.
- **FOL**: RealTimeInfoRetrieved ∧ PlanRoute -> ReadyForReservation(RouteID)
- **Submit a Ride Reservation**
- **Semantic Annotation**: The customer submits a ride reservation request.
- **Immediate Effect**:
- **Effect**: The reservation request is submitted and recorded in the system, awaiting further processing.
- **FOL**: SubmitReservation(Customer) -> ReservationSubmitted(ReservationID)
- **Cumulative Effect**:
- **Effect**: The submitted reservation will proceed through the next steps, including vehicle availability check and route optimization.

- **FOL**: ReservationSubmitted(ReservationID) -> TriggerRequestProcessing
- **Receive the Request**
- **Semantic Annotation**: The system receives and processes the customer's reservation request.
- **Immediate Effect**:
- **Effect**: The system successfully receives the reservation request and begins processing it.
- **FOL**: ReceiveRequest(System, ReservationID) -> RequestProcessingStarted
- **Cumulative Effect**:
- **Effect**: The receipt of the reservation request initiates the system's processing workflow, including vehicle availability check and route confirmation.
- **FOL**: RequestReceived(ReservationID) -> ProceedWithVehicleCheck
- **Vehicle Available?**
- **Semantic Annotation**: The system checks if there are available vehicles to fulfill the customer's reservation request.
- **Immediate Effect**:
- **Effect**: The system determines whether the reservation is successful based on vehicle availability.
- **FOL**: CheckAvailability(Vehicle) -> (VehicleAvailable ∨ VehicleNotAvailable)
- **Cumulative Effect**:
- **Effect**: Vehicle availability determines whether the reservation is confirmed or fails, directly affecting the customer experience.
- **FOL**: VehicleAvailable -> ProceedToReservationConfirmation ∧ VehicleNotAvailable -> TriggerFailureNotification
- **Reservation Failed**
- **Semantic Annotation**: If no vehicles are available, the reservation fails, and the system notifies the customer.
- **Immediate Effect**:
- **Effect**: The system informs the customer that the reservation has failed, and the process ends.
- **FOL**: ReservationFailed(ReservationID) -> NotifyFailure(Customer)
- **Cumulative Effect**:
- **Effect**: The reservation failure prevents the customer from proceeding with the reservation process, terminating the workflow.
- **FOL**: ReservationFailed -> ProcessTerminated
- **Confirm Reservation**
- **Semantic Annotation**: If a vehicle is available, the system confirms the reservation and notifies the customer.
- **Immediate Effect**:
- **Effect**: The reservation is successful, and the customer receives the confirmation notification.
- **FOL**: ConfirmReservation(System, ReservationID) -> ReservationConfirmed
- **Cumulative Effect**:
- **Effect**: Upon confirming the reservation, the process proceeds to notify the customer and prepare for subsequent actions.
- **FOL**: ReservationConfirmed -> ProceedToNotifyCustomer

- **Notify Result**
- **Semantic Annotation**: The system informs the customer of the final reservation result (success or failure).
- **Immediate Effect**:
- **Effect**: The customer is informed of the reservation result, ending the process or preparing for further actions.
- **FOL**: NotifyResult(System, ReservationID) -> CustomerInformed
- **Cumulative Effect**:
- **Effect**: The system notification ensures the customer is aware of the reservation status, enabling the smooth completion of the process.
- **FOL**: NotifyResult -> ProcessCompletion
- **Receive Result of Reservation**
- **Semantic Annotation:** The customer receives the final result of the reservation (confirmed or failed).
- **Immediate Effect:**
- **Effect:** The customer receives the result of the reservation, and the process ends.
- **FOL:** ReceiveReservationResult(Customer, ReservationID) -> (ReservationConfirmed ∨ ReservationFailed)
- **Cumulative Effect:**
- **Effect:** The result of the reservation affects the customer’s subsequent actions, such as proceeding with the service after confirmation or retrying after failure.
- **FOL:** ReservationConfirmed -> ProceedToServiceUsage ∧ ReservationFailed -> EndOrRetry
- **Address Request**
- **Semantic Annotation:** The system processes the customer’s request in preparation for generating the optimal route.
- **Immediate Effect:**
- **Effect:** The request is processed, and the system is ready to generate the optimal route.
- **FOL:** ProcessRequest(System, RequestID) -> RequestProcessed(RequestID)
- **Cumulative Effect:**
- **Effect:** The system has processed the request and is prepared with real-time traffic data to generate a more accurate route.
- **FOL:**RequestProcessed(RequestID)∧RealTimeDataReady-> ReadyForOptimalRouteGeneration
- **Generate Optimal Route**
- **Semantic Annotation**: The system generates the optimal route based on real-time traffic data.
- **Immediate Effect**:
- **Effect**: The optimal route is generated and passed to the next task.
- **FOL**:GenerateOptimalRoute(System, TrafficData) -> OptimalRouteGenerated(RouteID)
- **Cumulative Effect**:
- **Effect**: Combining the processed request and real-time traffic data, the system now generates the best route for the current conditions.

- **FOL**:RequestProcessed(RequestID)∧RealTimeDataReady-> OptimalRouteGenerated(RouteID)
- **Get Real-Time Situation**
- **Semantic Annotation**: The system retrieves the latest traffic data from real-time traffic services and updates the route.
- **Immediate Effect**:
- **Effect**: Real-time traffic data is updated, providing the customer with better route suggestions.
- **FOL**: RetrieveRealTimeTrafficInfo(TrafficService) -> RealTimeDataUpdated(RouteID)
- **Cumulative Effect**:
- **Effect**: With each update of real-time traffic data, the system's route recommendations are gradually optimized, ensuring the information provided to the customer is always current.
- **FOL**:OptimalRouteGenerated(RouteID)∧RealTimeDataUpdated-> OptimizedRouteAvailable(RouteID)
- **Provide Customer with Real-Time Situation**
- **Semantic Annotation**: The system provides the customer with the latest route information based on real-time traffic conditions.
- **Immediate Effect**:
- **Effect**: The customer receives updated traffic information.
- **FOL**: ProvideRealTimeData(System, Customer) -> RealTimeDataReceived(Customer, RouteID)
- **Cumulative Effect**:
- **Effect**: The customer has received the latest route and traffic information, ensuring the flow of information remains consistent and updated throughout the process.
- **FOL**:RealTimeDataUpdated(RouteID)∧OptimalRouteGenerated(RouteID)-> RealTimeInfoDelivered(Customer)
- **Receive Result of Real-Time Transportation**
- **Semantic Annotation**: The customer receives the final real-time traffic and route information, completing the process.
- **Immediate Effect**:
- **Effect**: The customer receives all updated traffic and route information, and the process concludes.
- **FOL**: ReceiveRealTimeRouteInfo(Customer, RouteID) -> ProcessCompleted
- **Cumulative Effect**:
- **Effect**: The customer, having received multiple updates and optimizations throughout the process, is now equipped with the most accurate route and traffic data, ensuring correct execution and information delivery.
- **FOL**:RealTimeDataReceived(Customer)∧ OptimizedRouteAvailable(RouteID) -> CustomerHasFinalOptimalRoute

PART D: Service Design and Analysis using SoaML

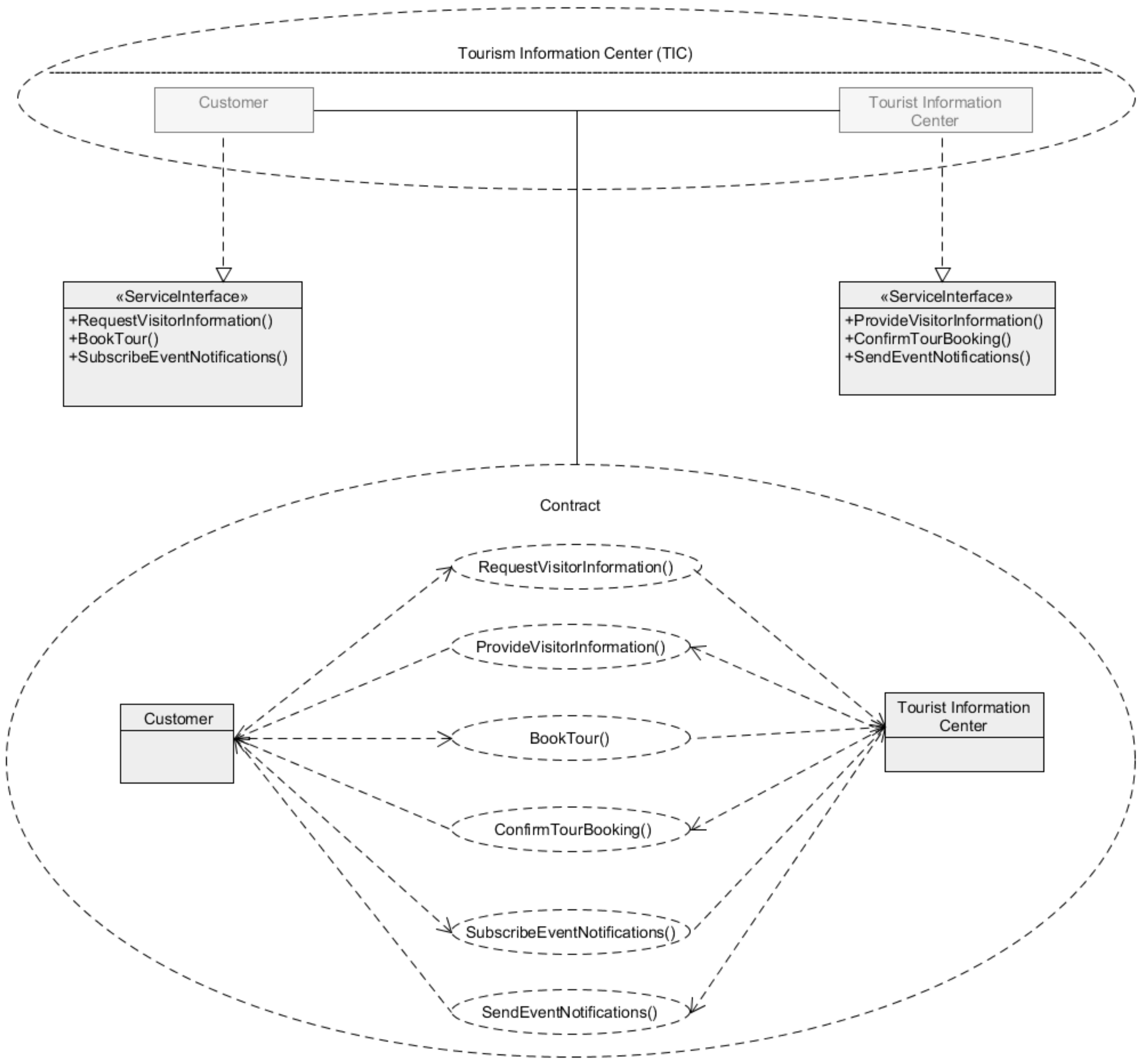
Task (I):

Conduct service design and analysis for your platform using SoaML (based on the service participants and service providers you identified in Part A). Your design should cover at least service architectures, service interfaces, service contracts (including their behaviour), and service compositions.

Tourism Information Center (TIC):

- ***Service Architecture Design***
- Service Participants:
- Tourist (Customer): Engages with the TIC for visitor information, tour bookings, and event notifications. They can ask for specific details, book tours, and subscribe to event notifications.
- Tourism Information Center (TIC): Provides information, processes tour bookings, and sends event notifications. Also handles inquiries and feedback.
- ***Service Interface Design***
- Tourist Interface:
- RequestVisitorInformation(): Tourist requests information about local attractions and maps.
- BookTour(): Tourist books a tour based on the provided information.
- SubscribeEventNotifications(): Tourist subscribes to event notifications.
- TIC Interface:
- ProvideVisitorInformation(): TIC provides information about attractions, maps, and tours.
- ConfirmTourBooking(): TIC processes the tour booking and confirms availability.
- SendEventNotifications(): TIC sends event notifications to subscribed tourists.
- ***Service Contract Design***
- Tourist-TIC Contract:
- The tourist requests visitor information, books tours, and subscribes to notifications.
- The TIC provides the requested information, processes bookings, and sends notifications.
- Tourists can ask questions or provide feedback, which the TIC answers or acknowledges.
- Contract Behavior:
- **Visitor Information**:
- Tourist sends a request for information.
- TIC responds with details about attractions, maps, and tours.
- **Tour Bookings**:
- Tourist submits a booking request.
- TIC confirms availability and processes the booking.
- **Event Notifications**:
- Tourist subscribes to notifications.
- TIC sends event updates and reminders to the tourist.
- ***Service Behaviour***
- **Visitor Information**:

- **Request-Response**: Tourist sends a request for information using RequestVisitorInformation(). TIC responds using ProvideVisitorInformation().
- **Tour Bookings**:
- **Booking Confirmation**: Tourist books a tour using BookTour(). TIC processes and confirms the booking using ConfirmTourBooking().
- **Event Notifications**:
- **Notification Subscription**: Tourist subscribes using SubscribeEventNotifications(). TIC sends updates via SendEventNotifications().
-
- ***Service Contract Behavior and Protocols***
- For **Visitor Information**, the tourist requests details using RequestVisitorInformation(), and TIC provides the needed information using ProvideVisitorInformation().
- For **Tour Bookings**, the tourist submits a booking using BookTour(), and TIC processes the booking with ConfirmTourBooking().
- For **Event Notifications**, the tourist subscribes via SubscribeEventNotifications(), and TIC sends event updates through SendEventNotifications().


Tourism Information Center (TIC)
Customer
Tourist Information Center
«ServiceInterface»
+RequestVisitorInformation()
+BookTour()
+SubscribeEventNotifications()
«ServiceInterface»
+ProvideVisitorInformation()
+ConfirmTourBooking()
+SendEventNotifications()
Contract
RequestVisitorInformation()
ProvideVisitorInformation()
BookTour()
ConfirmTourBooking()
SubscribeEventNotifications()
SendEventNotifications()
Customer
Tourist Information Center

LAM (Local Attractions and Museums) Service Design

1. Service Architecture Design

- Service Participants:
- **Customer**: Initiates ticket reservations, requests guided tours, and accesses educational content.
- **Attraction Provider**: Handles ticket reservations, manages guided tours, and provides educational content.

2. Service Interface Design

- Customer Interface:

- **SendTicketReservationRequest()**: The customer sends a ticket reservation request, including details like date, time, and number of visitors.
- **RequestGuidedTour()**: The customer sends a request for a guided tour at the museum or attraction.
- **AccessEducationalContent()**: The customer requests access to educational content such as museum guides or learning materials.

- Attraction Provider Interface:

- **ConfirmTicketReservation()**: The attraction provider confirms the ticket reservation.
- **RejectTicketReservation()**: The attraction provider rejects the ticket reservation if no tickets are available.
- **ScheduleGuidedTour()**: The attraction provider schedules a guided tour and confirms the tour request.
- **RejectGuidedTour()**: The attraction provider rejects the tour request if no guides are available.
- **ProvideEducationalContent()**: The attraction provider grants access to educational content or delivers learning materials.

3. Service Contract Design

- Customer-Attraction Provider Contract:

- **Contract Details**: The customer sends a ticket reservation request, and the attraction provider checks availability and confirms or rejects the reservation. The customer can also request a guided tour, and the provider either schedules or rejects the request based on availability. Finally, the customer can access educational content provided by the attraction.

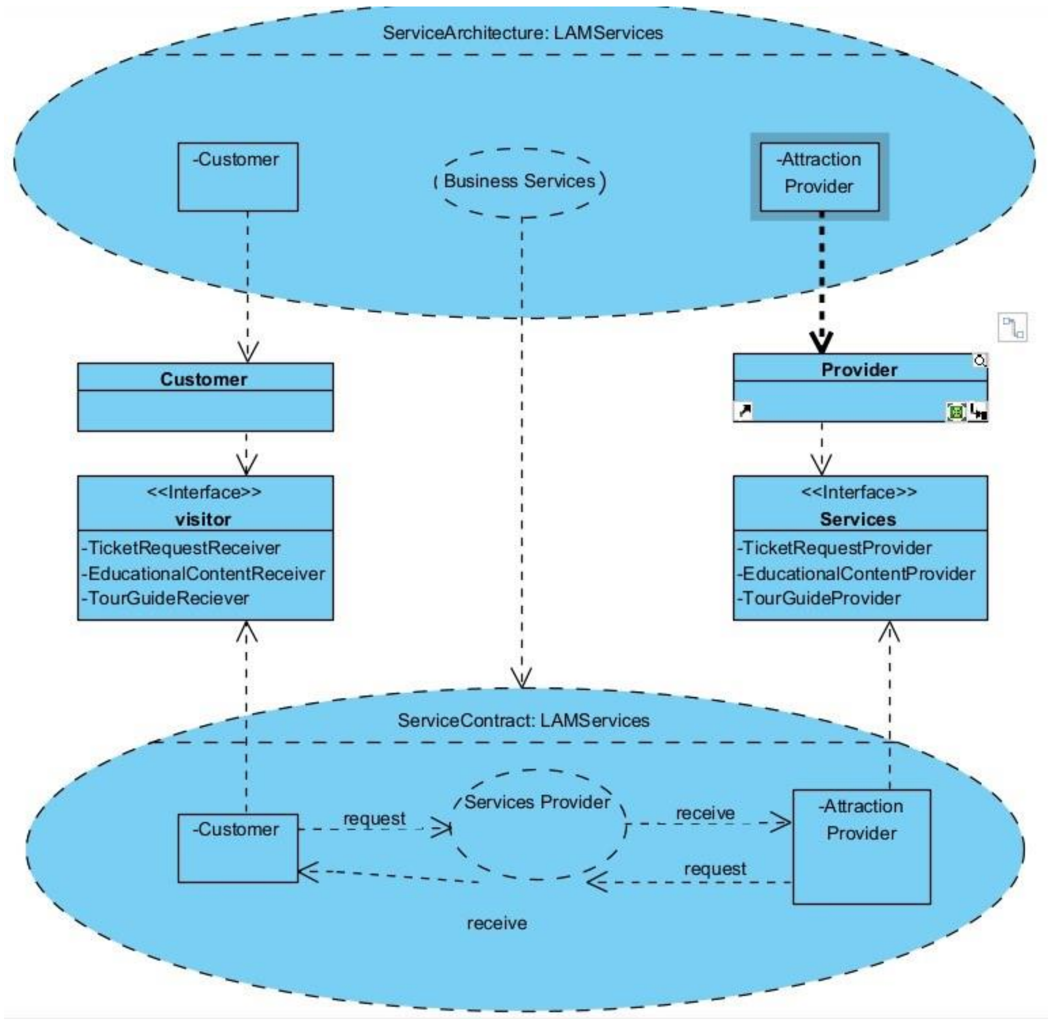
- **Contract Behavior**:
- **Ticket Reservation**: The customer sends a ticket reservation request, and the provider either confirms or rejects the reservation based on availability.
- **Guided Tour Request**: The customer requests a guided tour, and the provider responds by scheduling the tour or rejecting the request.
- **Educational Content Access**: The customer requests access to educational content, and the provider delivers the requested content.

4. Service Behavior

- Service Workflow:

- **Ticket Reservation**:
- The customer sends a ticket reservation request via SendTicketReservationRequest(), and the provider checks ticket availability.
- If tickets are available, the provider confirms the reservation via ConfirmTicketReservation().
- If tickets are unavailable, the provider sends a rejection message via RejectTicketReservation().
- **Guided Tour Request:**
- The customer requests a guided tour via **RequestGuidedTour()**, and the provider checks availability.
- If a guide is available, the provider schedules the tour via **ScheduleGuidedTour()** and sends a confirmation.
- If no guide is available, the provider sends a rejection message via **RejectGuidedTour()**.
- **Educational Content Access:**

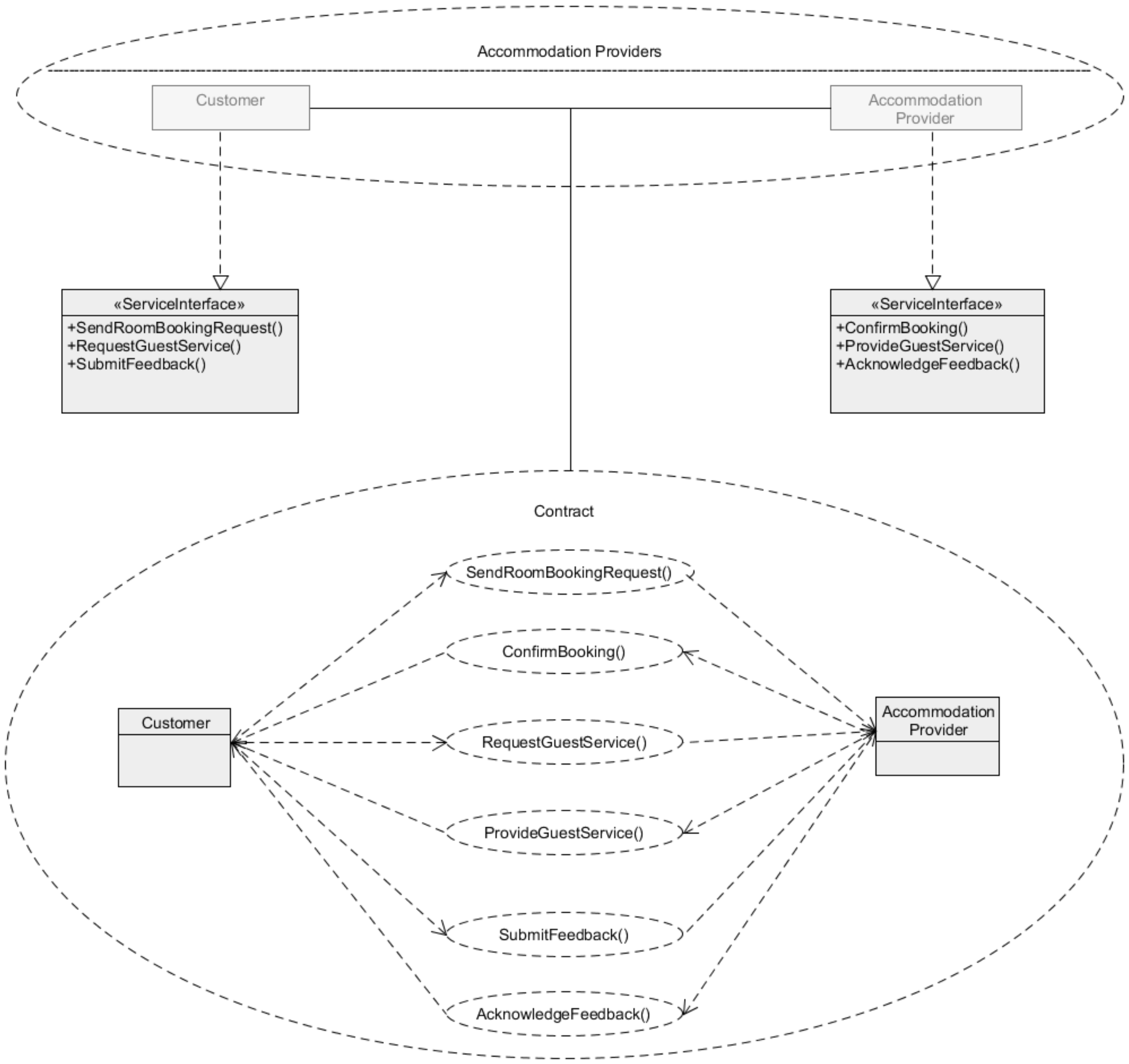
  The customer requests access to educational content via **AccessEducationalContent()**, and the provider delivers the content via **ProvideEducationalContent()**.


ServiceArchitecture: LAMServices
-Customer
Business Services
-Attraction Provider
Customer
Provider
<<Interface>>
visitor
-TicketRequestReceiver
-EducationalContentReceiver
-TourGuideReciever
<<Interface>>
Services
-TicketRequestProvider
-EducationalContentProvider
-TourGuideProvider
ServiceContract: LAMServices
-Customer
request
Services Provider
receive
-Attraction Provider
request
receive

AP (Accommodation Providers) Design:

- ***Service Architecture Design***
- Service Participants:
- Customer (Tourist): Initiates room booking requests, requests guest services, and submits feedback.
- Accommodation Provider (AP): Manages room booking, handles guest service requests, provides personalized room services, and collects feedback.
- ***Service Interface Design***
- **Customer Interface:**
- SendRoomBookingRequest(): The customer sends a request for room availability, including details like date, room type, and duration of stay.
- RequestGuestService(): The customer sends a request for guest services, such as housekeeping or room service.
- SubmitFeedback(): The customer submits feedback regarding their stay experience.
- **Accommodation Provider Interface:**
- ConfirmBooking(): The AP confirms the room booking based on availability of room.
- RejectBooking(): The AP rejects the booking request if no rooms are available or the requested dates are unavailable.
- ProvideGuestService(): The AP processes and fulfills guest service requests, like housekeeping or room service.
- AcknowledgeFeedback(): The AP acknowledges and records the feedback submitted by the customer.
- ***Service Contract Design***
- **Customer-Accommodation Provider Contract:**
- Contract Details:
  - The customer submits a room booking request, and the AP checks room availability before confirming or denying the booking. Once validated, the customer may request extra guest services, which the AP will process. After the stay, the consumer can submit feedback on the service, which the AP will acknowledge.
- Contract Behavior:
  - The customer provides a room booking request. The AP examines availability and either confirms or denies the booking. Once the reservation is confirmed, the customer can make requests for guest services, which the AP will fulfil. Following the stay, feedback is submitted and acknowledged by the AP.
- ***Service Behavior***
- Service Behavior:
- The customer sends a room booking request via SendRoomBookingRequest(), and the AP checks availability. If the booking is successful, the AP confirms the booking via ConfirmBooking(). If no rooms are available, the AP sends a rejection message via RejectBooking().

- The customer can request guest services via RequestGuestService(), and the AP processes the request by sending a service fulfillment confirmation via ProvideGuestService().
- The customer provides feedback using SubmitFeedback(), and the AP acknowledges receipt of the feedback via AcknowledgeFeedback().
- ***Service Contract Behavior and Protocols***
- **Service Protocol:**
- The customer sends a room booking request -> The AP checks availability and either confirms or rejects the request -> The customer requests guest services -> The AP provides the requested services -> The customer submits feedback -> The AP acknowledges the feedback.



RC (Restaurant and Cafe) Service Design

1. Service Architecture Design

- **Service Participants**:
- **Customer**: Initiates table reservation, checks menu recommendations, and places food orders.
- **Restaurant**: Handles reservation requests, provides menu recommendations, confirms reservations, and processes food orders.

2. Service Interface Design

- **Customer Interface**:
- SendReservationRequest(): The customer sends a table reservation request, including details like date, time, and number of people.
- CheckRecommendation(): The customer checks the restaurant's recommended menu.
- OrderFood(): The customer places a food order.
- **Restaurant Interface**:
  - ConfirmReservation(): The restaurant confirms the table reservation.
  - RejectReservation(): The restaurant rejects the reservation if no tables are available.
  - GenerateMenu(): The restaurant generates the recommended menu and sends it to the customer.
  - ConfirmOrder(): The restaurant confirms the customer's food order.
  - SuggestSubstitute(): The restaurant suggests alternative dishes if the ordered items are unavailable.

3. Service Contract Design

- **Customer-Restaurant Contract**:
- **Contract Details**: The customer initiates a reservation request, and the restaurant checks availability and confirms or rejects the reservation. If successful, the customer can check the menu recommendations and place a food order. The restaurant processes the order and may suggest substitutes if items are unavailable.
- **Contract Behavior**: The customer sends a reservation request, and the restaurant responds by either confirming or rejecting the reservation. After the reservation is confirmed, the customer can order food, and the restaurant processes the order.

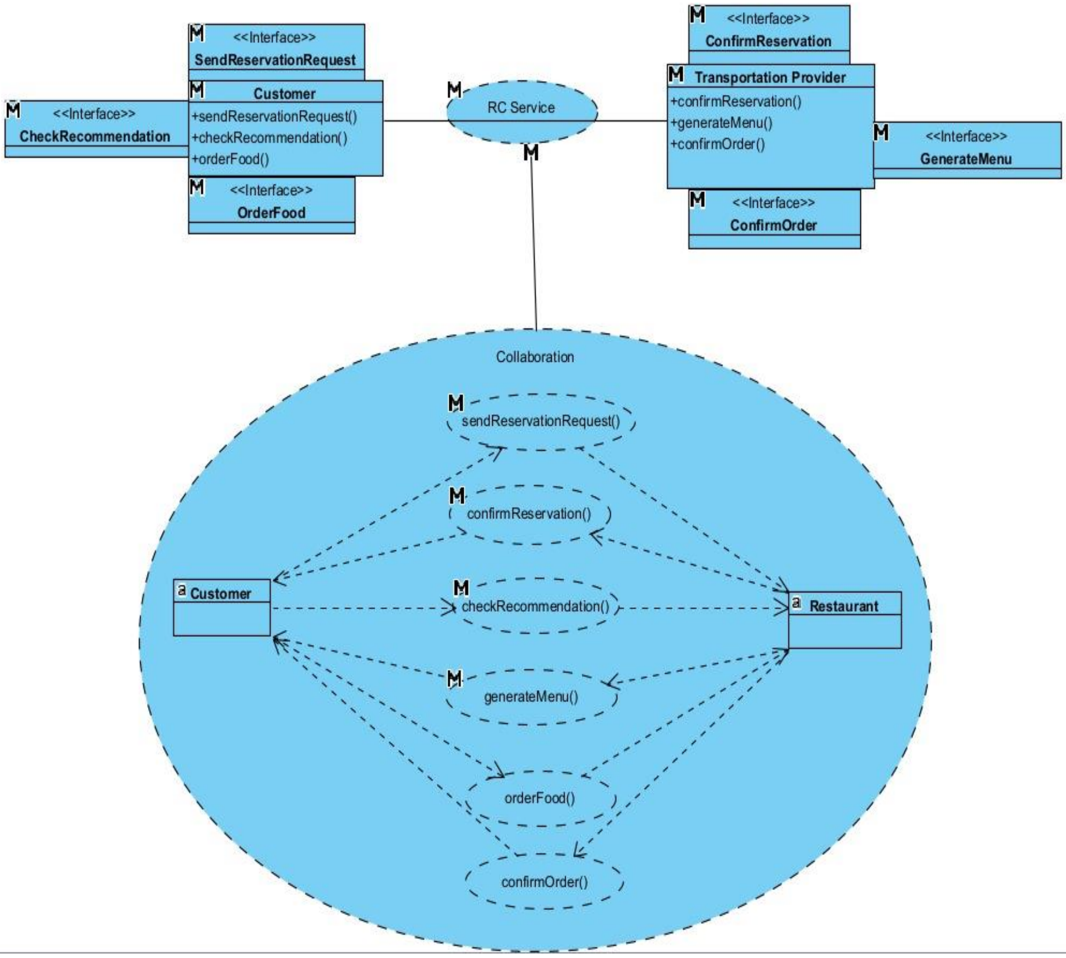
4. Service Behavior

- **Service Behavior**:
  - The customer sends a reservation request via SendReservationRequest(), and the restaurant checks table availability. If the reservation is successful, the restaurant confirms the reservation via ConfirmReservation(). If the reservation is rejected, the restaurant sends a rejection message via RejectReservation().
  - The customer can check the recommended menu via CheckRecommendation(), and the restaurant sends the recommended menu via GenerateMenu().

- The customer places a food order via OrderFood(), and the restaurant processes the order. If the ordered items are available, the restaurant confirms the order via ConfirmOrder(). If items are unavailable, the restaurant suggests alternatives via SuggestSubstitute().

5. Service Contract Behavior and Protocols

- **Service Protocol**:
- The customer sends a reservation request -> The restaurant checks availability and either confirms or rejects the reservation -> The customer checks menu recommendations -> The customer places a food order -> The restaurant confirms the order or suggests substitutes if necessary.



-

LTS (Local Transportation Service) Design

1. Service Architecture Design

- **Service Participants**:
- **Customer**: Initiates ride reservation and route planning requests.
- **Transportation Provider**: Provides route planning, ride reservation, and real-time traffic information services.

2. Service Interface Design

- **Customer Interface**:
  - RequestRoutePlanning(): The customer sends a route planning request.
  - RequestRideReservation(): The customer sends a ride reservation request.
  - ReceiveTrafficUpdate(): The customer receives real-time traffic updates.
- Transportation Provider Interface:
  - ProvideRouteOptions(): The transportation provider returns possible route options.
  - ConfirmRideReservation(): The transportation provider confirms the ride reservation.
  - UpdateRealTimeTraffic(): The transportation provider provides real-time traffic updates.

3. Service Contract Design

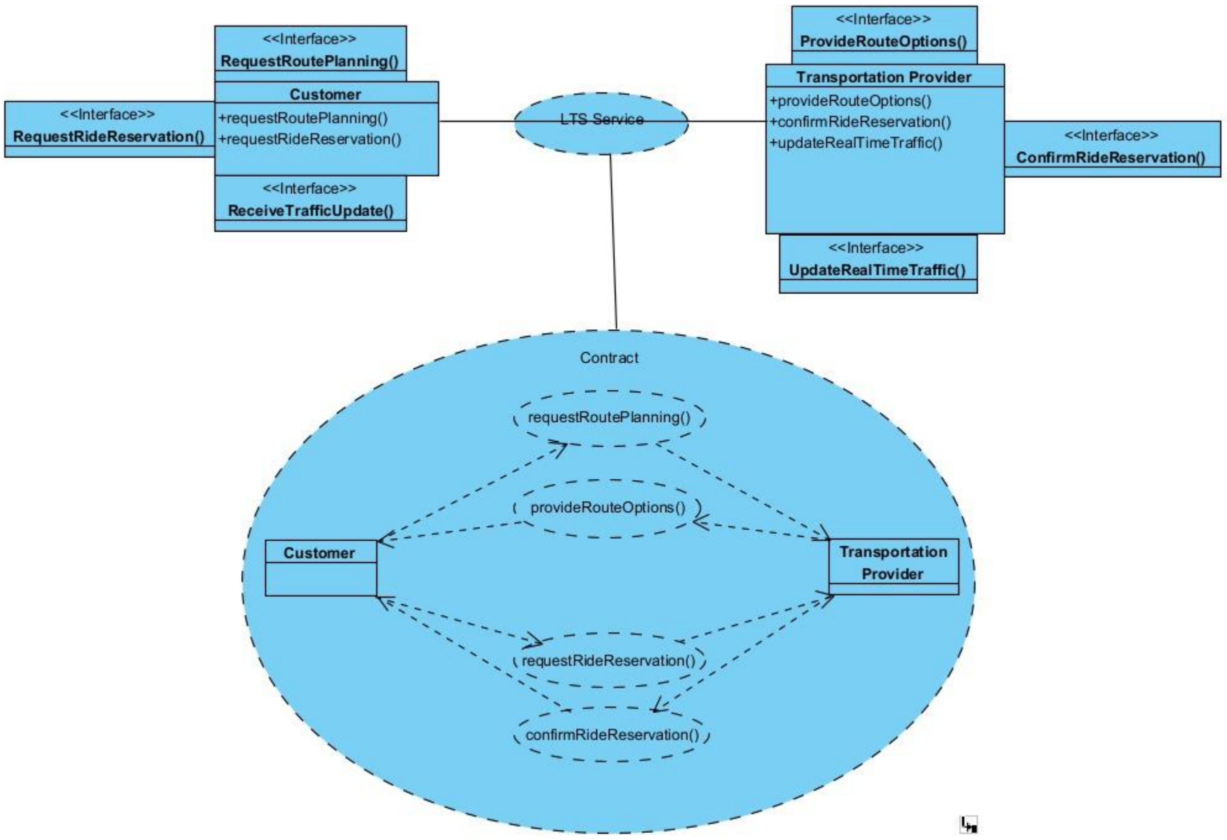
- **Customer-Transportation Provider Contract**:
- **Contract Details**: The customer initiates a ride reservation, and the transportation provider confirms vehicle availability and plans the route. The provider also updates real-time traffic information.
- **Contract Behavior**: The customer requests route planning, and the provider returns multiple route options, offering real-time traffic updates.

4. Service Behavior

- The customer sends a route planning request through RequestRoutePlanning(), and the transportation provider returns route options using ProvideRouteOptions().
- The customer reserves a ride using RequestRideReservation(), and the transportation provider confirms the reservation via ConfirmRideReservation().
- The transportation provider updates real-time traffic information via UpdateRealTimeTraffic().

5. Service Contract Behavior and Protocols

- **Service Protocol**: The customer requests a route -> The transportation provider offers route options -> The customer selects a route and makes a reservation -> The transportation provider confirms the reservation and updates real-time traffic information.



<<Interface>>
RequestRoutePlanning()
Customer
+requestRoutePlanning()
+requestRideReservation()
<<Interface>>
RequestRideReservation()
<<Interface>>
ReceiveTrafficUpdate()
LTS Service
<<Interface>>
ProvideRouteOptions()
Transportation Provider
+provideRouteOptions()
+confirmRideReservation()
+updateRealTimeTraffic()
<<Interface>>
ConfirmRideReservation()
<<Interface>>
UpdateRealTimeTraffic()
Contract
requestRoutePlanning()
provideRouteOptions()
Customer
Transportation Provider
requestRideReservation()
confirmRideReservation()

PART E: Microservices design and implementation

In this part, we selected the following three services from the Tourism Ecosystem Platform for design and implementation using the microservices approach:

- Tour Booking service from TIC (Tourism Information Center)
- Room Booking service from AP (Accommodation Provider)
- Ride Booking service from LTS (Local Transportation Service)

Task (I):

We have implemented the corresponding services of the Tourism Ecosystem using the microservices approach. Each service is developed as an independent, loosely coupled module using a microservices architecture.

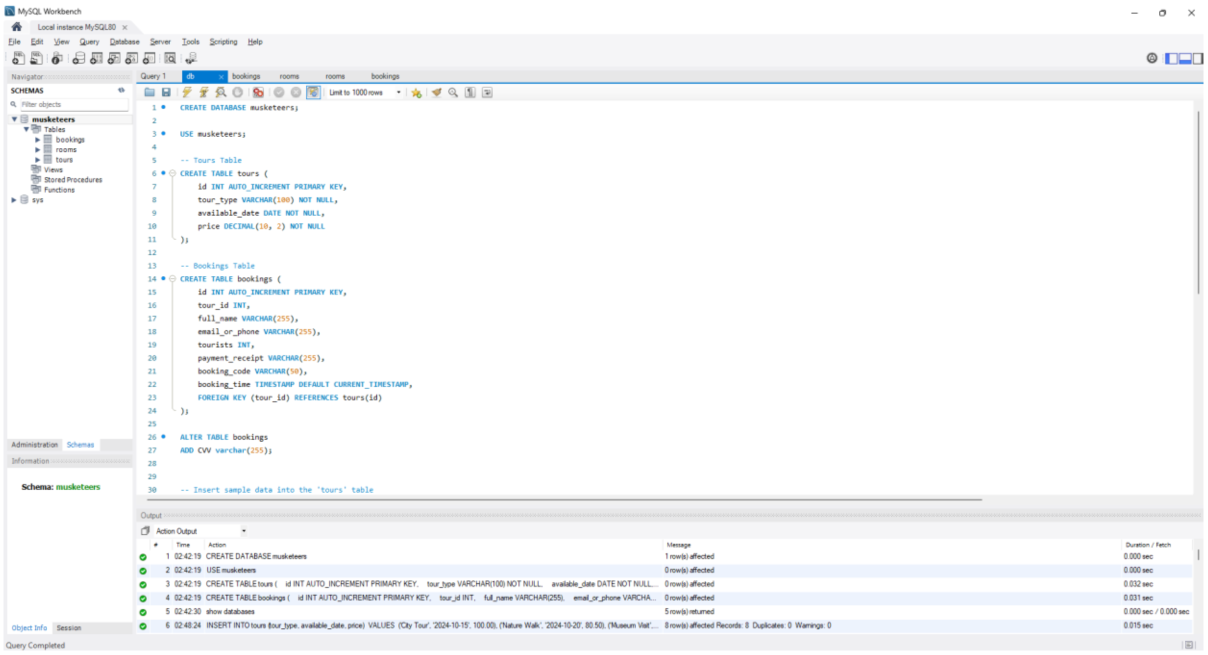
Tools & technologies:

- **IDEs used:** PyCharm, MySQL Workbench
- **Backend**: Implemented in Python using the Flask (Web framework)
- **Frontend**: Built using HTML, CSS, JavaScript (for handling user inputs and interactivity)
- **Database**: MySQL
- **Third-party services**:
- **Payment API:** Stripe (for payment processing)
- **Notifications:**
- Mailtrap SMTP for mocking email functionality.
- Twilio for SMS notifications.

Step by Step Project Plan:

After distributing the corresponding microservices among the team-members, we followed the following approach for design and development:

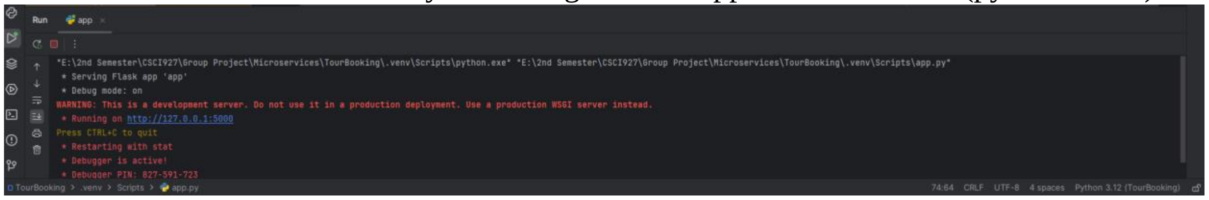
- Install Required packages in your system/ development environment such as Flask, flask-mysqldb, stripe, flask-mail, twilio, smtplib etc.
- Set up the third-party (Stripe, Twilio, Mailtrap) test accounts to process mock payments, email/sms. Use the necessary credentials to configure inside the code.
- Create the MySQL database and necessary tables (such as tours, bookings, rooms etc.) using MySQL query. Insert some sample data inside the required tables to test the code. Please refer to the following screenshot of MySQL Workbench for better understanding:



```sql
CREATE DATABASE musketeers;

USE musketeers;

-- Tours Table
CREATE TABLE tours (
    id INT AUTO_INCREMENT PRIMARY KEY,
    tour_type VARCHAR(100) NOT NULL,
    available_date DATE NOT NULL,
    price DECIMAL(10, 2) NOT NULL
);

-- Bookings Table
CREATE TABLE bookings (
    id INT AUTO_INCREMENT PRIMARY KEY,
    tour_id INT,
    full_name VARCHAR(255),
    email_or_phone VARCHAR(255),
    tourists INT,
    payment_receipt VARCHAR(255),
    booking_code VARCHAR(50),
    booking_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (tour_id) REFERENCES tours(id)
);

ALTER TABLE bookings
ADD CVV varchar(255);


-- Insert sample data into the 'tours' table
```

#	Time	Action	Message	Duration / Fetch
1	02:42:19	CREATE DATABASE musketeers	1 row(s) affected	0.000 sec
2	02:42:19	USE musketeers	0 row(s) affected	0.000 sec
3	02:42:19	CREATE TABLE tours (id INT AUTO_INCREMENT PRIMARY KEY, tour_type VARCHAR(100) NOT NULL, available_date DATE NOT NULL...	0 row(s) affected	0.032 sec
4	02:42:19	CREATE TABLE bookings (id INT AUTO_INCREMENT PRIMARY KEY, tour_id INT, full_name VARCHAR(255), email_or_phone VARCHA...	0 row(s) affected	0.031 sec
5	02:42:30	show databases	5 row(s) returned	0.000 sec / 0.000 sec
6	02:48:24	INSERT INTO tours (tour_type, available_date, price) VALUES ('City Tour', '2024-10-15', 100.00), ('Nature Walk', '2024-10-20', 80.50), ('Museum Visit'...	8 row(s) affected Records: 8 Duplicates: 0 Warnings: 0	0.015 sec

- Implement the core backend logic with Python using Flask framework.
- Create a simple HTML form to accept booking details and a confirmation webpage (regardless of tour, room or ride).
- Start the Flask server by running the application file (python file)



```
"E:\2nd Semester\CSCI927\Group Project\Microservices\TourBooking\.venv\Scripts\python.exe" "E:\2nd Semester\CSCI927\Group Project\Microservices\TourBooking\.venv\Scripts\app.py"
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 827-591-723
```

- Access the booking services at http://127.0.0.1:5000 in the browser. Make sure to modify the URL accordingly based on the selected service.
- After implementing the microservice's core functionalities, create necessary external CSS files to style webpages as per preference.

Project Repository Link:

Please refer to the project repository link for further reference of the microservices implementation of Tourism Ecosystem:

- OneDrive

Task (II):

In this task, we will demonstrate the design and implementation of the corresponding microservices: Tour booking, Room booking and Ride booking.

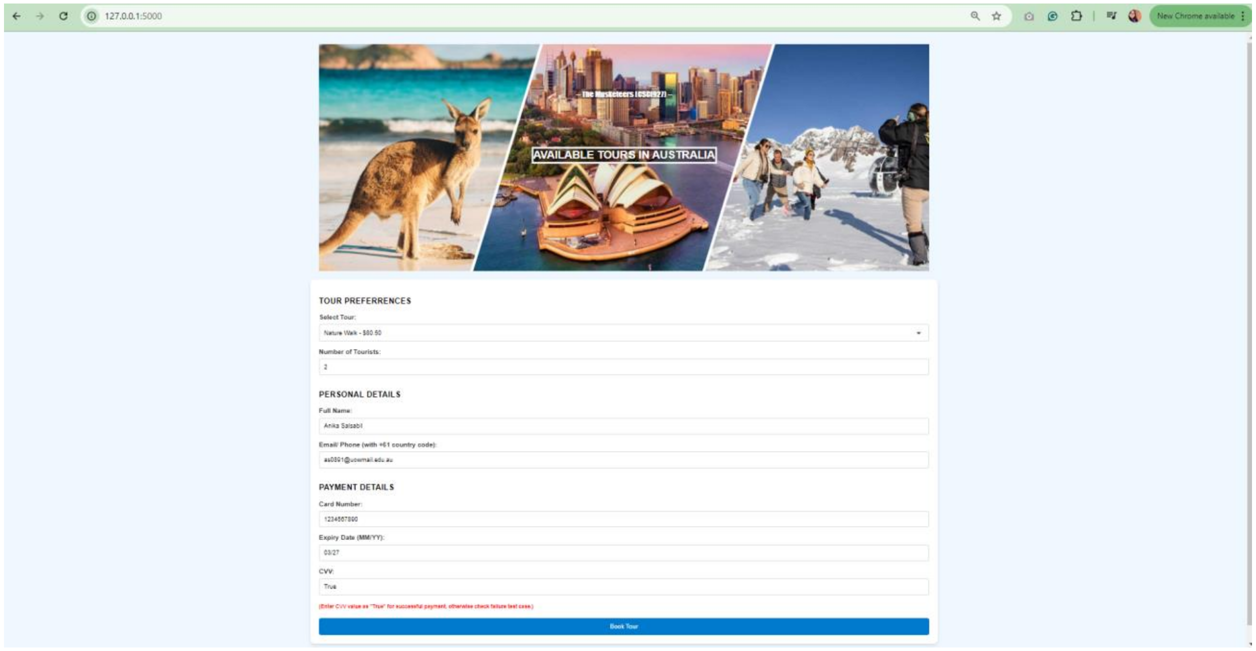
1. Tour Booking Service

Design

The Tour Booking service allows users to book different types of tours listed on the platform. The system fetches the available tours from the database, lets the user choose a tour, and processes payments using Stripe. After successful payment, a booking record is created, and the user receives confirmation via email or SMS, depending on their provided contact details.

Please check the following screenshots of the Tour Booking service's concerned webpages for reference:

<u>Tour Booking Form:</u>



TOUR PREFERRENCES

Select Tour:
Nature Walk - $80.50

Number of Tourists:
2

PERSONAL DETAILS

Full Name:
Anika Salsabil

Email/ Phone (with +61 country code):
as0891@uowmail.edu.au

PAYMENT DETAILS

Card Number:
1234567890

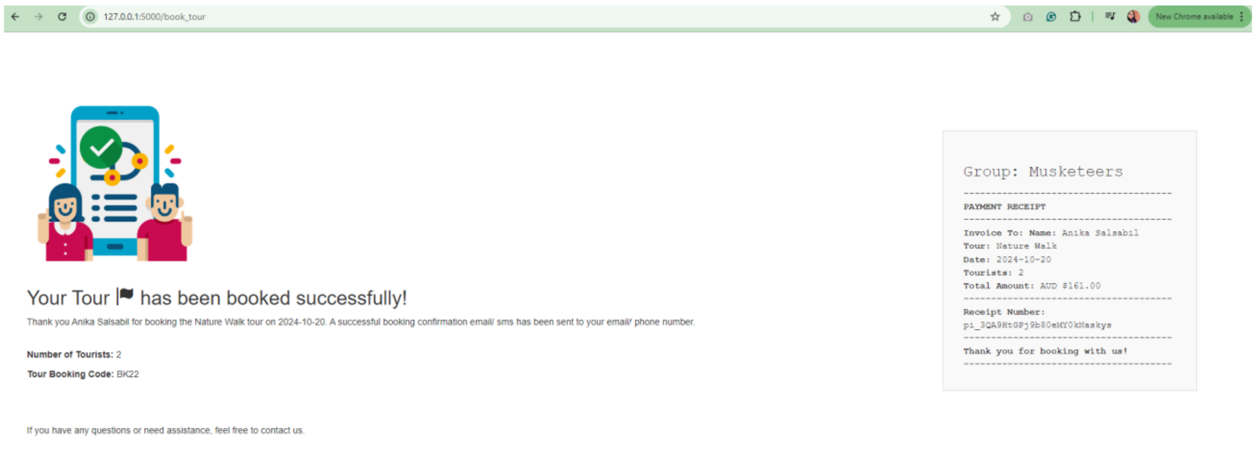
Expiry Date (MM/YY):
03/27

CVV:
True

(Enter CVV value as "True" for successful payment, otherwise check failure test case.)

Figure: Tour Booking form webpage

<u>Tour Booking Confirmation</u>:



Your Tour 🏴 has been booked successfully!

Thank you Anika Salsabil for booking the Nature Walk tour on 2024-10-20. A successful booking confirmation email/ sms has been sent to your email/ phone number.

Number of Tourists: 2

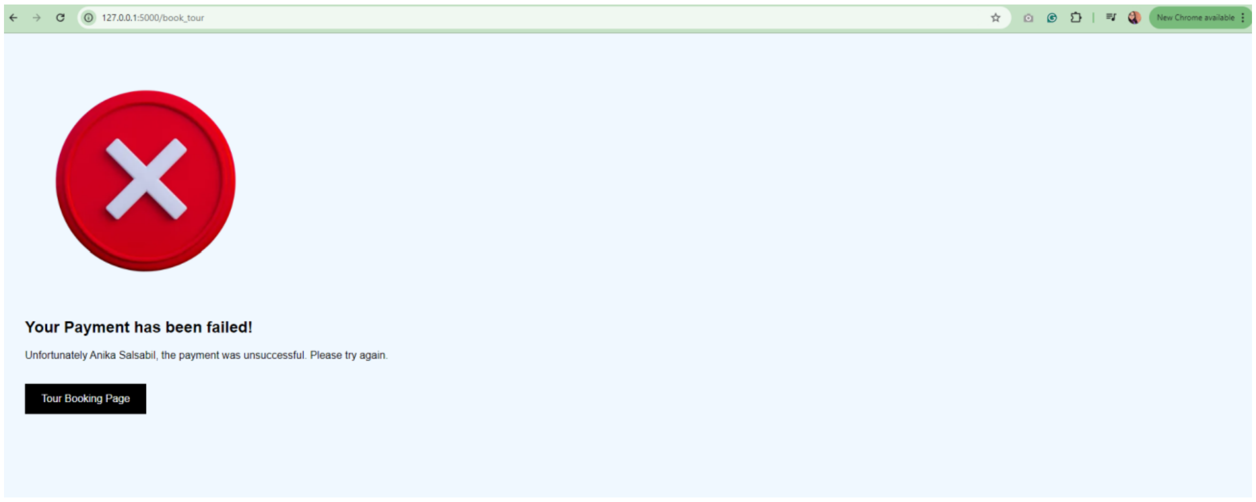
Tour Booking Code: BK22

If you have any questions or need assistance, feel free to contact us.

```
Group: Musketeers
---------------------------------------
PAYMENT RECEIPT
---------------------------------------
Invoice To: Name: Anika Salsabil
Tour: Nature Walk
Date: 2024-10-20
Tourists: 2
Total Amount: AUD $161.00
---------------------------------------
Receipt Number:
pi_3QA9HtGPj9b80eMY0kNaskys
---------------------------------------
Thank you for booking with us!
---------------------------------------
```

Figure: Tour Booking successful webpage

Tour Booking Failure:



Your Payment has been failed!

Unfortunately Anika Salsabil, the payment was unsuccessful. Please try again.

Figure: Tour booking payment failure webpage

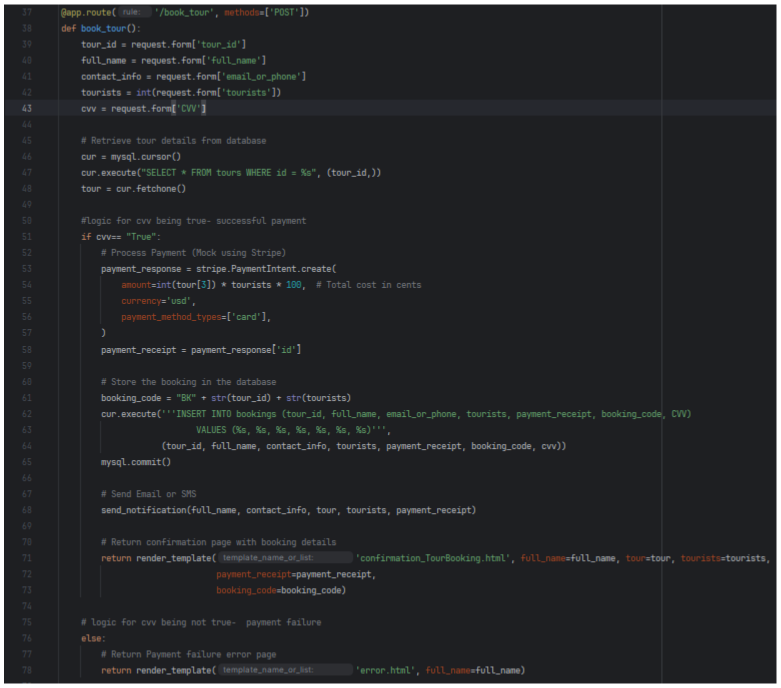
Implementation

- **Backend**: Implemented using Flask.
- The application is connected to a MySQL database to fetch available tours and store bookings.
- For payment processing, we used Stripe. Payments are mocked with a condition where-providing the CVV as "True" is triggered as a successful payment.
- The payment details are stored in the bookings table, and a confirmation email/SMS is sent.
- **Mocking Payment**: Stripe's API is used with a secret key for processing payments. In the mock setup, if the CVV field contains the value "True," the payment is processed; otherwise, an error page is shown.
- **Mocking Email and SMS Notifications**:

- Mailtrap is used to simulate email sending. The system uses SMTP configuration with credentials (host address, usable port number, username, password) provided by Mailtrap for email dispatch.
- Twilio is used to mock SMS notifications. We configured a Twilio trial account with an account SID and authentication key to send SMS messages to registered phone numbers.
- **Database Design**:
- A tours table stores tour details, and a bookings table stores booking records along with payment receipts.
- **Frontend**: Built with HTML and CSS. The booking form is presented to users, allowing them to choose a tour, enter personal details, and proceed to payment.

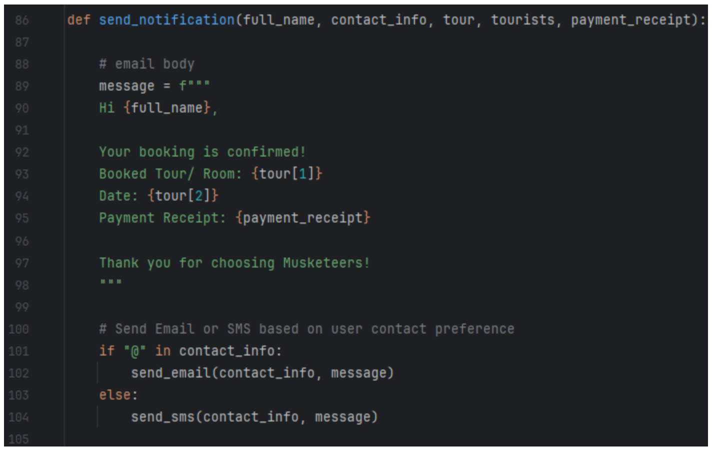
Key sections of the code for the Tour Booking Service include:

Book Tour function:



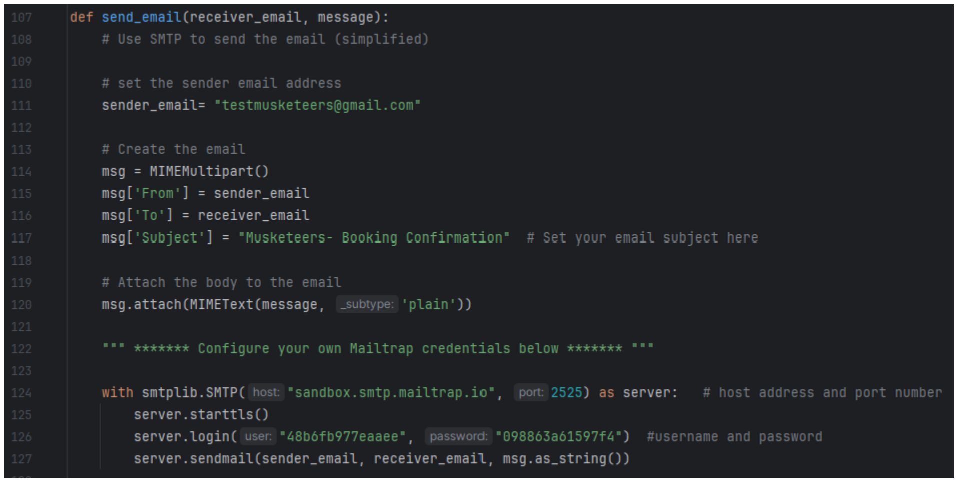
```
@app.route( rule: '/book_tour', methods=['POST'])
def book_tour():
    tour_id = request.form['tour_id']
    full_name = request.form['full_name']
    contact_info = request.form['email_or_phone']
    tourists = int(request.form['tourists'])
    cvv = request.form['CVV']

    # Retrieve tour details from database
    cur = mysql.cursor()
    cur.execute("SELECT * FROM tours WHERE id = %s", (tour_id,))
    tour = cur.fetchone()

    #logic for cvv being true- successful payment
    if cvv== "True":
        # Process Payment (Mock using Stripe)
        payment_response = stripe.PaymentIntent.create(
            amount=int(tour[3]) * tourists * 100,  # Total cost in cents
            currency='usd',
            payment_method_types=['card'],
        )
        payment_receipt = payment_response['id']

        # Store the booking in the database
        booking_code = "BK" + str(tour_id) + str(tourists)
        cur.execute('''INSERT INTO bookings (tour_id, full_name, email_or_phone, tourists, payment_receipt, booking_code, CVV)
                           VALUES (%s, %s, %s, %s, %s, %s, %s)''',
                    (tour_id, full_name, contact_info, tourists, payment_receipt, booking_code, cvv))
        mysql.commit()

        # Send Email or SMS
        send_notification(full_name, contact_info, tour, tourists, payment_receipt)

        # Return confirmation page with booking details
        return render_template( template_name_or_list: 'confirmation_TourBooking.html', full_name=full_name, tour=tour, tourists=tourists,
                                payment_receipt=payment_receipt,
                                booking_code=booking_code)

    # logic for cvv being not true-  payment failure
    else:
        # Return Payment failure error page
        return render_template( template_name_or_list: 'error.html', full_name=full_name)
```

Notification function:



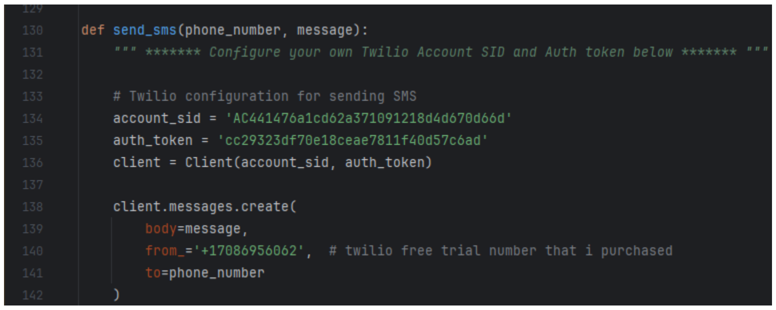
```python
def send_notification(full_name, contact_info, tour, tourists, payment_receipt):

    # email body
    message = f"""
    Hi {full_name},

    Your booking is confirmed!
    Booked Tour/ Room: {tour[1]}
    Date: {tour[2]}
    Payment Receipt: {payment_receipt}

    Thank you for choosing Musketeers!
    """

    # Send Email or SMS based on user contact preference
    if "@" in contact_info:
        send_email(contact_info, message)
    else:
        send_sms(contact_info, message)
```

<u>Send Email function</u>:



```python
def send_email(receiver_email, message):
    # Use SMTP to send the email (simplified)

    # set the sender email address
    sender_email= "testmusketeers@gmail.com"

    # Create the email
    msg = MIMEMultipart()
    msg['From'] = sender_email
    msg['To'] = receiver_email
    msg['Subject'] = "Musketeers- Booking Confirmation"  # Set your email subject here

    # Attach the body to the email
    msg.attach(MIMEText(message, _subtype: 'plain'))

    """ ******* Configure your own Mailtrap credentials below ******* """

    with smtplib.SMTP( host: "sandbox.smtp.mailtrap.io", port: 2525) as server:   # host address and port number
        server.starttls()
        server.login( user: "48b6fb977eaaee", password: "098863a61597f4")  #username and password
        server.sendmail(sender_email, receiver_email, msg.as_string())
```

<u>Send SMS function</u>:



```python
def send_sms(phone_number, message):
    """ ******* Configure your own Twilio Account SID and Auth token below ******* """

    # Twilio configuration for sending SMS
    account_sid = 'AC441476a1cd62a371091218d4d670d66d'
    auth_token = 'cc29323df70e18ceae7811f40d57c6ad'
    client = Client(account_sid, auth_token)

    client.messages.create(
        body=message,
        from_='+17086956062',  # twilio free trial number that i purchased
        to=phone_number
    )
```

2. Room Booking Service

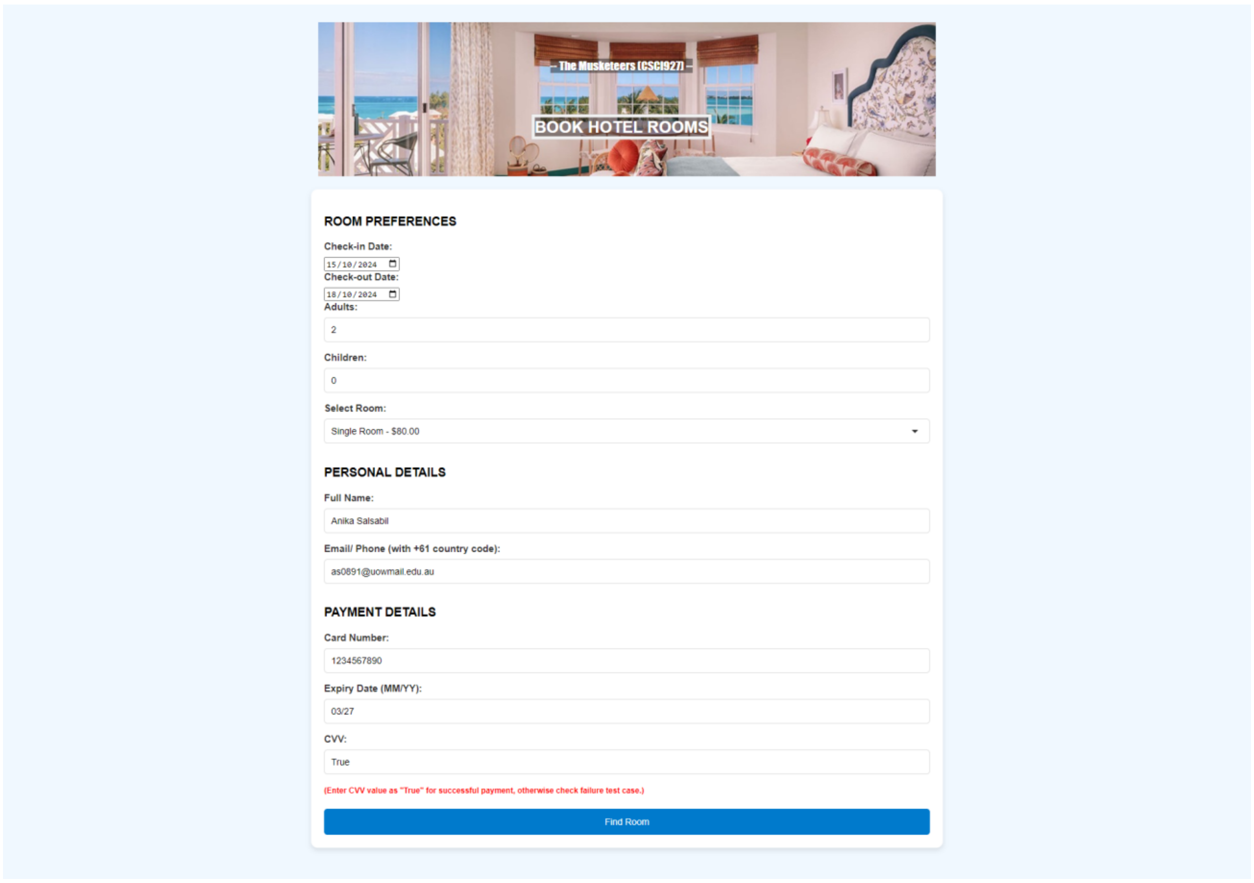
Design

The Room Booking service allows users to book rooms based on availability. The user can select check-in and check-out dates, specify the number of adults and children, and proceed to payment.

After successful payment, a booking record is created, and the user receives confirmation via email or SMS, depending on their provided contact details.

Please check the following screenshots of the Room Booking service's concerned webpages for reference:

<u>Room Booking form webpage:</u>



ROOM PREFERENCES

Check-in Date: 15/10/2024

Check-out Date: 18/10/2024

Adults: 2

Children: 0

Select Room: Single Room - $80.00

PERSONAL DETAILS

Full Name: Anika Salsabil

Email/ Phone (with +61 country code): as0891@uowmail.edu.au

PAYMENT DETAILS

Card Number: 1234567890

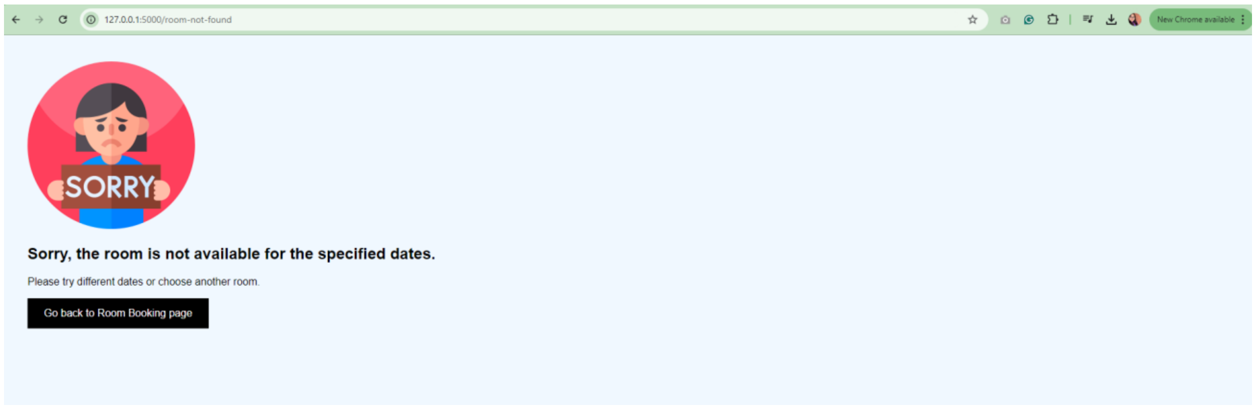
Expiry Date (MM/YY): 03/27

CVV: True

(Enter CVV value as "True" for successful payment, otherwise check failure test case.)

Figure: Room booking form

<u>Room Not Available case</u>:

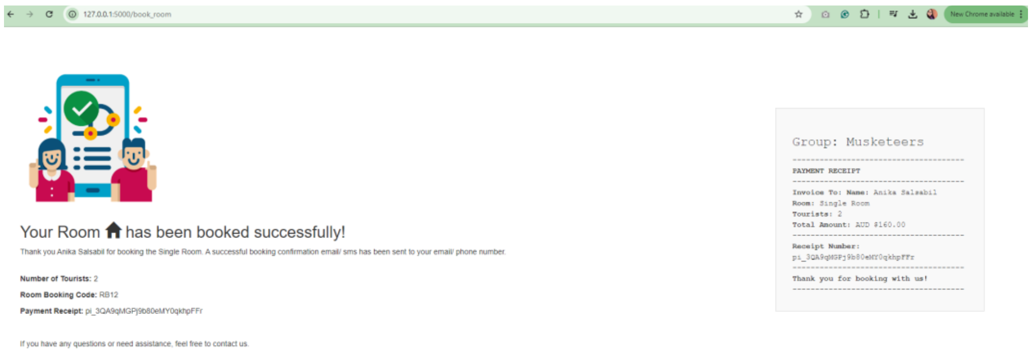


Sorry, the room is not available for the specified dates.

Please try different dates or choose another room.

Figure: Room not found page

<u>Room Booking Confirmation</u>:



Your Room has been booked successfully!

Thank you Anika Salsabil for booking the Single Room. A successful booking confirmation email/ sms has been sent to your email/ phone number.

Number of Tourists: 2

Room Booking Code: RB12

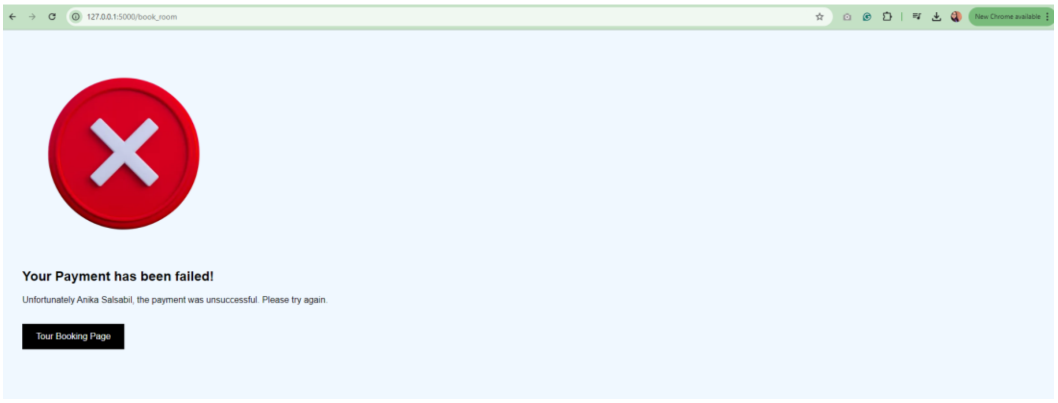
Payment Receipt: pi_3QA9qMGPj9b80eMY0qkhpFFr

If you have any questions or need assistance, feel free to contact us.

```
Group: Musketeers
------------------------------------
PAYMENT RECEIPT
------------------------------------
Invoice To: Name: Anika Salsabil
Room: Single Room
Tourists: 2
Total Amount: AUD $160.00
------------------------------------
Receipt Number:
pi_3QA9qMGPj9b80eMY0qkhpFFr
------------------------------------
Thank you for booking with us!
------------------------------------
```

Figure: Room booking payment confirmation

<u>Room Booking payment failure</u>:



Your Payment has been failed!

Unfortunately Anika Salsabil, the payment was unsuccessful. Please try again.

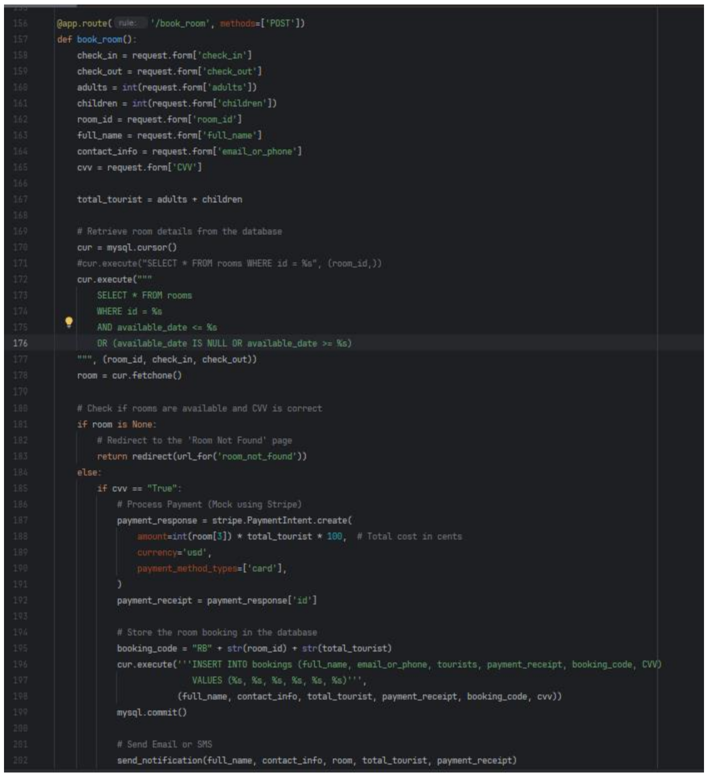
Figure: Room booking payment failure

Implementation

- **Backend**: Implemented using Flask.
- The available rooms are fetched from a MySQL database.
- Payment processing and email/SMS notifications follow the same logic as the Tour Booking service.
- Bookings are stored on the bookings table, with additional details for room types and tourists.
- **Mocking Payment, Email, and SMS Notifications**: Same as Tour Booking service. We have used the same functions for sending notifications to users.
- **Database Design**:
- A rooms table stores available rooms with prices, and a bookings table records room bookings.

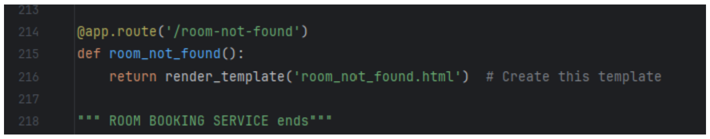
Key sections of the code for the Room Booking Service include:

Room Book function:



```python
@app.route('/book_room', methods=['POST'])
def book_room():
    check_in = request.form['check_in']
    check_out = request.form['check_out']
    adults = int(request.form['adults'])
    children = int(request.form['children'])
    room_id = request.form['room_id']
    full_name = request.form['full_name']
    contact_info = request.form['email_or_phone']
    cvv = request.form['CVV']

    total_tourist = adults + children

    # Retrieve room details from the database
    cur = mysql.cursor()
    #cur.execute("SELECT * FROM rooms WHERE id = %s", (room_id,))
    cur.execute("""
        SELECT * FROM rooms
        WHERE id = %s
        AND available_date <= %s
        OR (available_date IS NULL OR available_date >= %s)
    """, (room_id, check_in, check_out))
    room = cur.fetchone()

    # Check if rooms are available and CVV is correct
    if room is None:
        # Redirect to the 'Room Not Found' page
        return redirect(url_for('room_not_found'))
    else:
        if cvv == "True":
            # Process Payment (Mock using Stripe)
            payment_response = stripe.PaymentIntent.create(
                amount=int(room[3]) * total_tourist * 100,  # Total cost in cents
                currency='usd',
                payment_method_types=['card'],
            )
            payment_receipt = payment_response['id']

            # Store the room booking in the database
            booking_code = "RB" + str(room_id) + str(total_tourist)
            cur.execute('''INSERT INTO bookings (full_name, email_or_phone, tourists, payment_receipt, booking_code, CVV)
                           VALUES (%s, %s, %s, %s, %s, %s)''',
                        (full_name, contact_info, total_tourist, payment_receipt, booking_code, cvv))
            mysql.commit()

            # Send Email or SMS
            send_notification(full_name, contact_info, room, total_tourist, payment_receipt)
```

Room Not found function:



```python
@app.route('/room-not-found')
def room_not_found():
    return render_template('room_not_found.html')  # Create this template

""" ROOM BOOKING SERVICE ends"""
```

Please note that we have used the same send notification, email and SMS functions that we used in the Tour booking service.

3. Ride Booking Service

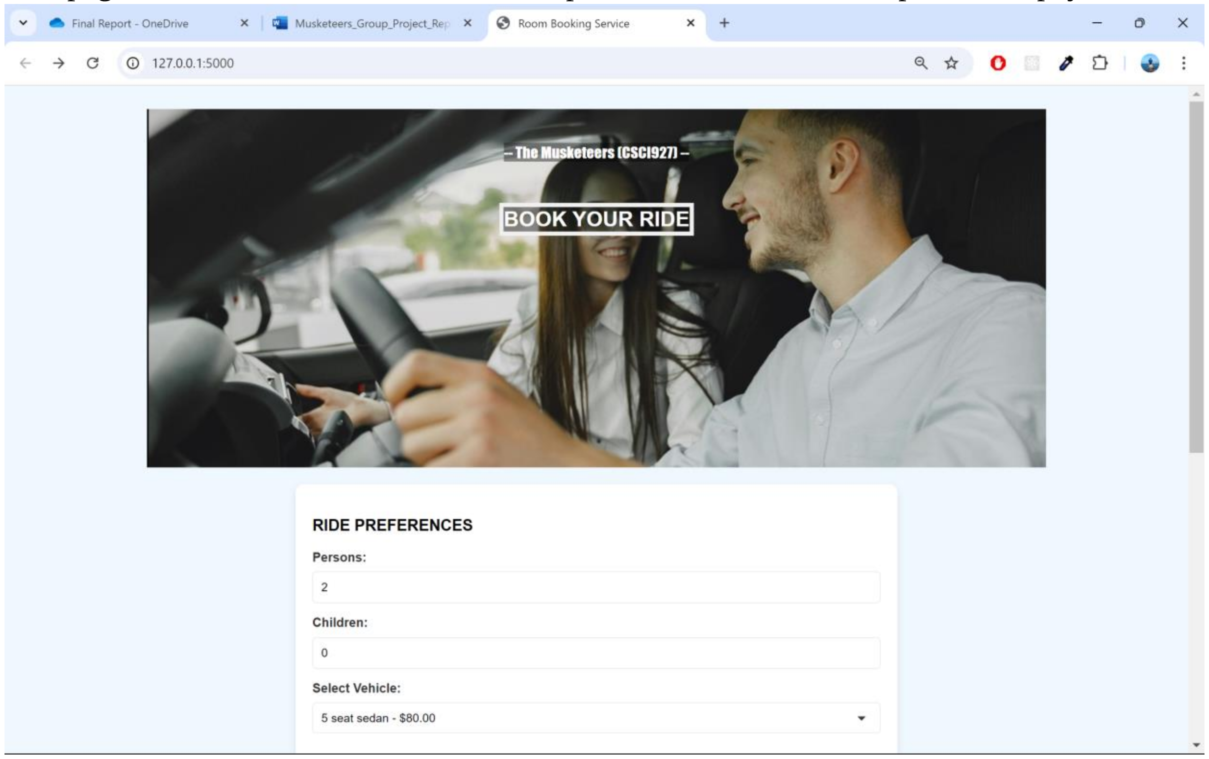
The **Ride Booking Service** gives users an easy way to book vehicles for transportation. It pulls available rides from the system's database and lets users select their preferred ride. Payment is handled using **Stripe**, and once it goes through successfully, the system saves the booking details and sends a confirmation to the user via email or SMS, depending on the contact details provided. If the payment fails, the user is notified with an error message and can try again.

User Experience

The following are the key steps users go through when booking a ride:

- **Ride Booking Form**:

- This page lets the user select a vehicle, enter personal information, and proceed to payment.

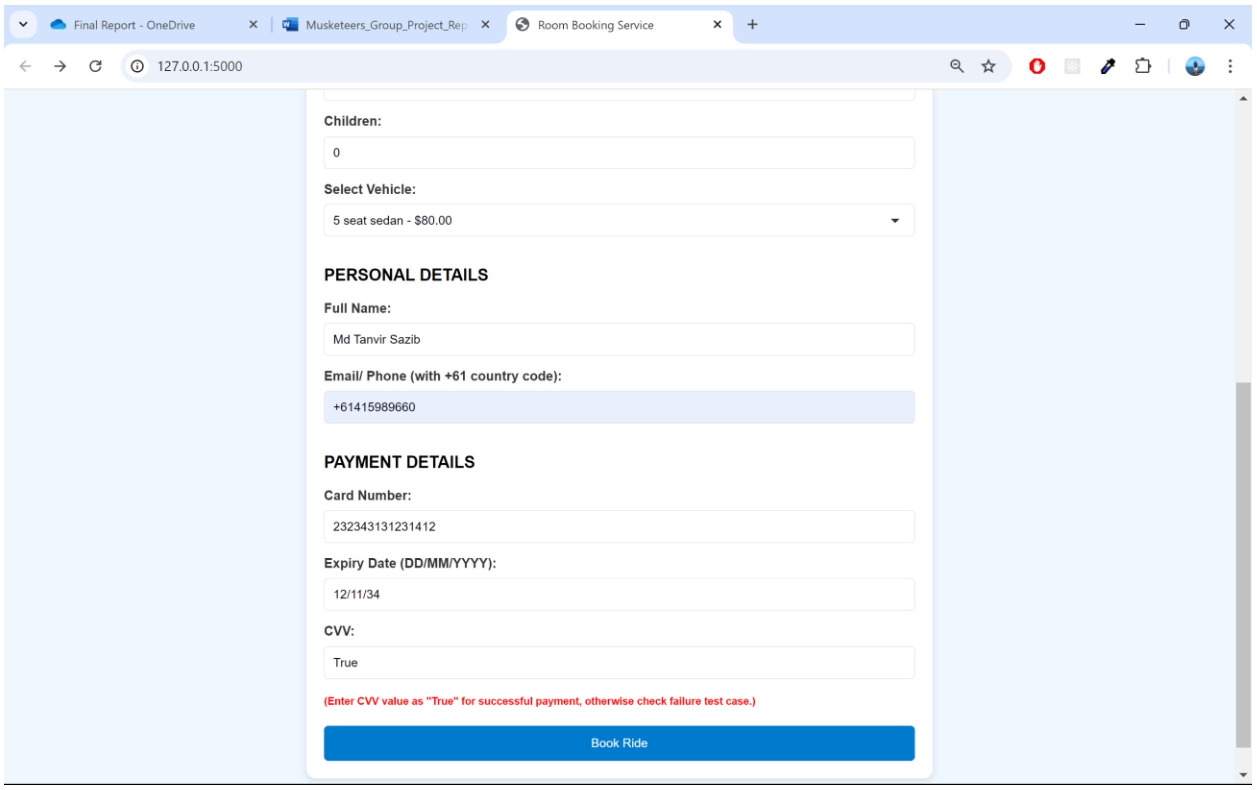


RIDE PREFERENCES

Persons: 2

Children: 0

Select Vehicle: 5 seat sedan - $80.00


Final Report - OneDrive
Musketeers_Group_Project_Rep
Room Booking Service
127.0.0.1:5000

Children:

0

Select Vehicle:

5 seat sedan - $80.00

PERSONAL DETAILS

Full Name:

Md Tanvir Sazib

Email/ Phone (with +61 country code):

+61415989660

PAYMENT DETAILS

Card Number:

232343131231412

Expiry Date (DD/MM/YYYY):

12/11/34

CVV:

True

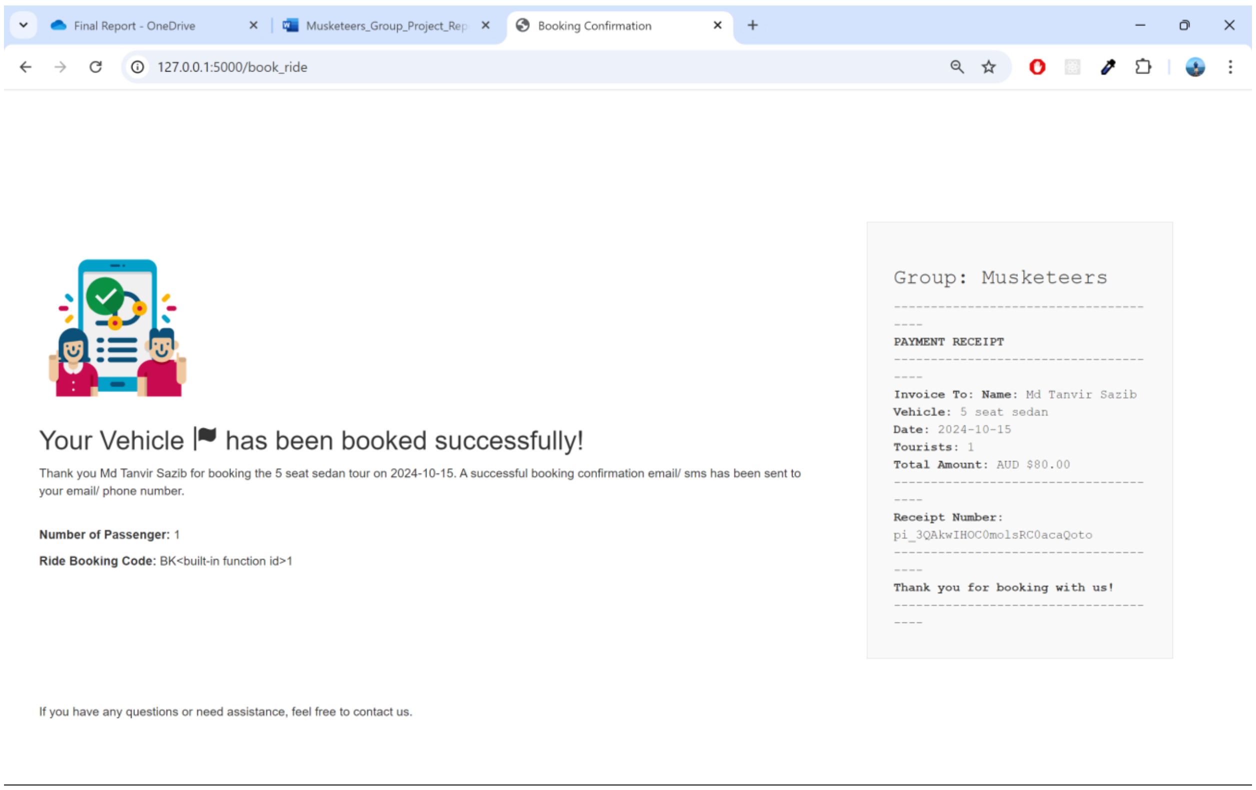
(Enter CVV value as "True" for successful payment, otherwise check failure test case.)

Book Ride

Figure: Ride Booking form webpage

- **Booking Confirmation**:

After a successful payment, the system shows a confirmation message, and the user gets a confirmation via email or SMS.



Your Vehicle |⚑ has been booked successfully!

Thank you Md Tanvir Sazib for booking the 5 seat sedan tour on 2024-10-15. A successful booking confirmation email/ sms has been sent to your email/ phone number.

Number of Passenger: 1

Ride Booking Code: BK<built-in function id>1

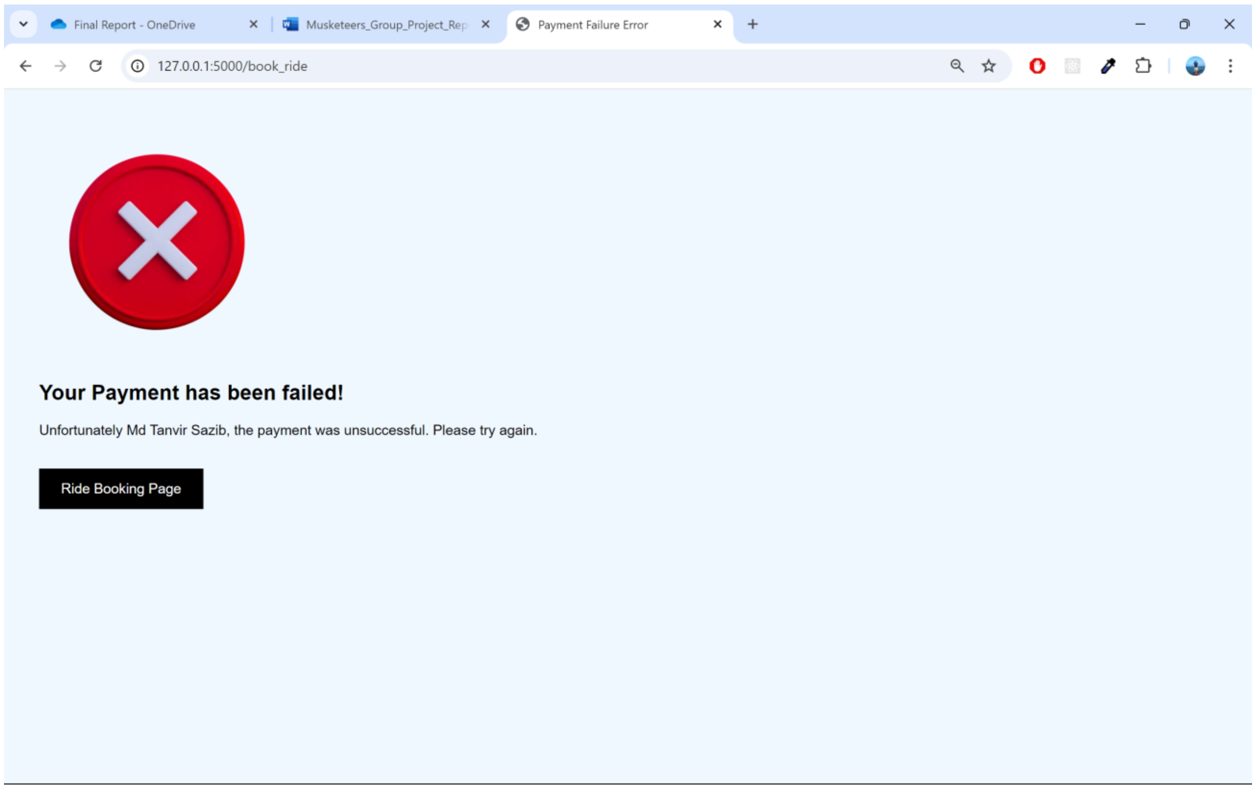
If you have any questions or need assistance, feel free to contact us.

```
Group: Musketeers
----------------------------------
----
PAYMENT RECEIPT
----------------------------------
----
Invoice To: Name: Md Tanvir Sazib
Vehicle: 5 seat sedan
Date: 2024-10-15
Tourists: 1
Total Amount: AUD $80.00
----------------------------------
----
Receipt Number:
pi_3QAkwIHOC0molsRC0acaQoto
----------------------------------
----
Thank you for booking with us!
----------------------------------
----
```

Figure: Ride Booking successful webpage

- **Booking Failure**:

If the payment fails, the user sees an error message and can try again.



Your Payment has been failed!

Unfortunately Md Tanvir Sazib, the payment was unsuccessful. Please try again.

Figure: Ride booking payment failure webpage

How It Works

Backend Implementation

The backend of the Ride Booking Service is built using **Flask**. It connects to a **MySQL database**, which stores the details of available rides and keeps track of bookings. Payments are processed securely through **Stripe**, and we've set up a simple way to simulate payments for testing. If the user enters **"True"** as the CVV during payment, it's treated as successful. If not, the system shows a payment failure message.

Payment Processing

For payments, we integrated **Stripe**, using a secret API key for authentication. In our mock setup:

- If the user enters **"True"** in the CVV field, the payment succeeds.
- If the CVV is anything else, an error page appears, and the booking isn't saved.

The booking details are stored in the **bookings table**, and confirmation is sent to the user automatically.

Notifications

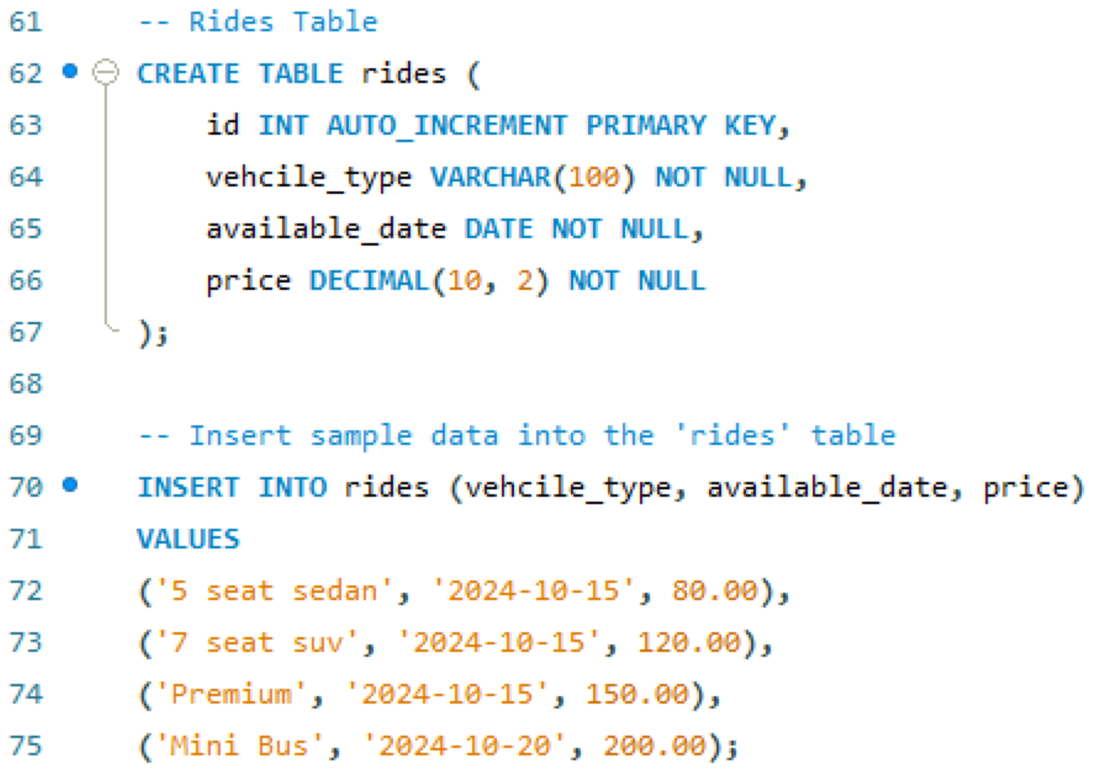
We've added both email and SMS notifications so users get immediate updates:

- **Email**: We use **Mailtrap** to simulate sending confirmation emails. The system connects to Mailtrap using SMTP settings, including the server address and credentials provided by Mailtrap.
- **SMS**: We also configured **Twilio** to send text messages. A trial Twilio account, with the **SID** and **authentication key**, lets us send messages to users with registered phone numbers.

Database Design

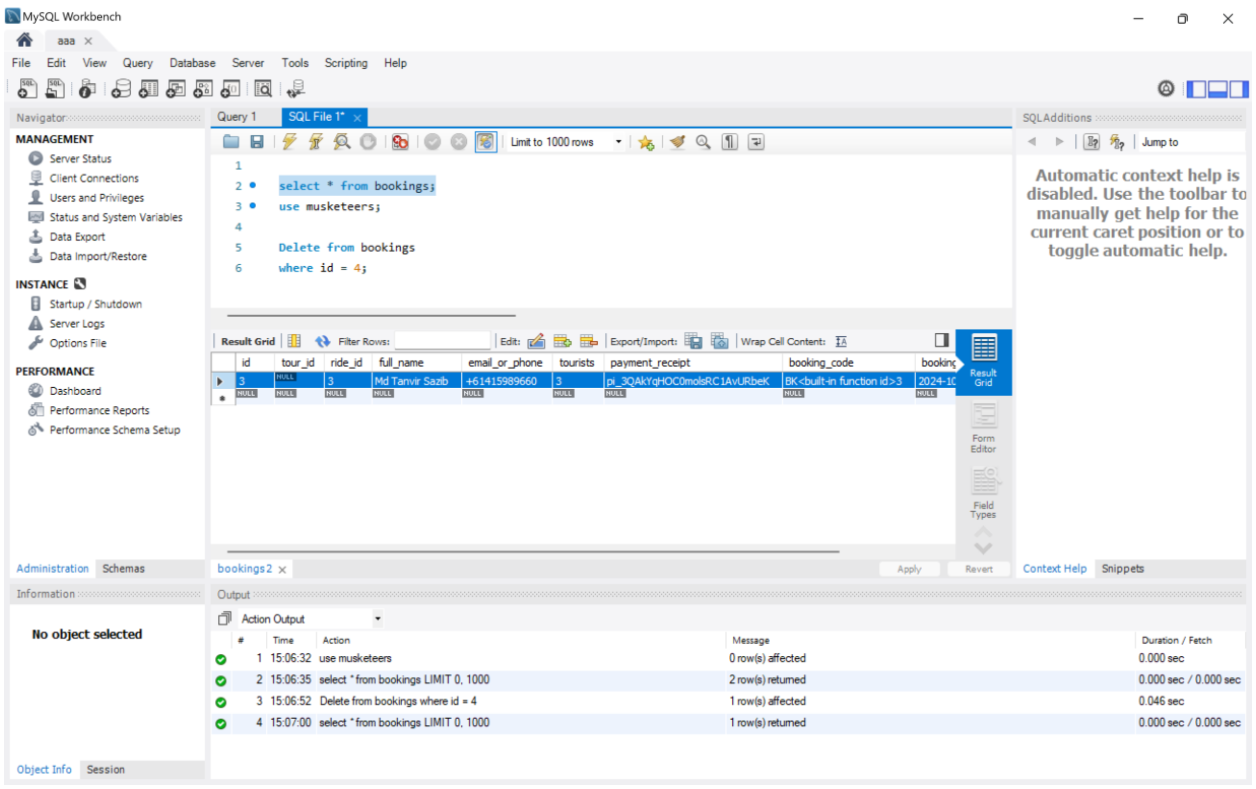
To manage the data efficiently, we use two key tables:

- **rides table**: This table keeps all the information about the available vehicles, such as their type, seating capacity, and status.
- **bookings table**: This table stores all booking records, including payment status and the contact details needed for notifications.



```sql
-- Rides Table
CREATE TABLE rides (
    id INT AUTO_INCREMENT PRIMARY KEY,
    vehcile_type VARCHAR(100) NOT NULL,
    available_date DATE NOT NULL,
    price DECIMAL(10, 2) NOT NULL
);

-- Insert sample data into the 'rides' table
INSERT INTO rides (vehcile_type, available_date, price)
VALUES
('5 seat sedan', '2024-10-15', 80.00),
('7 seat suv', '2024-10-15', 120.00),
('Premium', '2024-10-15', 150.00),
('Mini Bus', '2024-10-20', 200.00);
```

-



```
select * from bookings;
use musketeers;

Delete from bookings
where id = 4;
```

id	tour_id	ride_id	full_name	email_or_phone	tourists	payment_receipt	booking_code	booking
3	NULL	3	Md Tanvir Sazib	+61415989660	3	pi_3QAkYqHOC0molsRC1AvURbeK	BK<built-in function id>3	2024-1
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
#  Time      Action                                Message              Duration / Fetch
1  15:06:32  use musketeers                        0 row(s) affected    0.000 sec
2  15:06:35  select * from bookings LIMIT 0, 1000  2 row(s) returned    0.000 sec / 0.000 sec
3  15:06:52  Delete from bookings where id = 4     1 row(s) affected    0.046 sec
4  15:07:00  select * from bookings LIMIT 0, 1000  1 row(s) returned    0.000 sec / 0.000 sec
```

Figure: Database tables.

Frontend Design

We built the frontend with **HTML and CSS**, keeping things simple and user-friendly. The booking form allows users to browse available vehicles, fill in their details, and make a payment with minimal steps. After submitting their information, users either see a confirmation page or a payment failure message if something goes wrong.

Key Code Functions

- **Book Ride Function**

This function handles the main booking logic. It retrieves available vehicles from the **rides table** and processes the user's booking. If the payment is successful, the booking is saved in the **bookings table**, and notifications are triggered.

- **Notification Function**

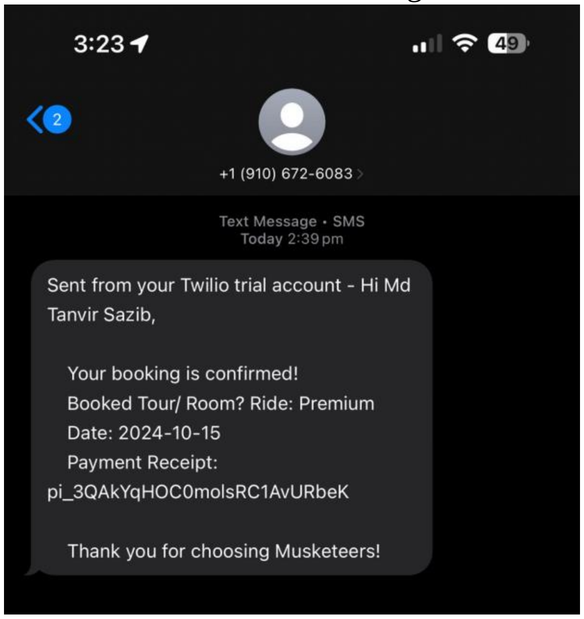
The notification function sends updates to users via email or SMS once the booking is confirmed. It uses **Mailtrap** and **Twilio** to manage these notifications.

- **Send Email Function**

This function connects to **Mailtrap** using the SMTP configuration and sends confirmation emails to users after a successful booking.

- **Send SMS Function**

Using **Twilio**, this function sends SMS confirmation messages to users. It pulls the user's phone number from the booking information and sends the message automatically.



Sent from your Twilio trial account - Hi Md Tanvir Sazib,

Your booking is confirmed!
Booked Tour/ Room? Ride: Premium
Date: 2024-10-15
Payment Receipt:
pi_3QAkYqHOC0molsRC1AvURbeK

Thank you for choosing Musketeers!

Figure: Ride booking sms confirmation

Benefits of Microservices for the Tourism Ecosystem

Breaking down the Tourism Ecosystem platform into microservices, such as Tour Booking, Room Booking, and Ride Booking, has offered several key benefits, both for current implementation and future scalability:

- **Improved Agility**
- Teams can work more independently and collaboratively on specific microservices.
- Each of the six members can focus on different services (Tour, Room, and Ride Booking), reducing development cycle times and increasing overall productivity.
- **Flexible Scaling**
- Microservices allow us to scale individual services like Tour, Room, and Ride Booking based on specific demand.
- Infrastructure can be right sized for each service, ensuring cost efficiency and maintaining service availability during peak loads (e.g., holiday seasons or major events).
- **Simplified Deployment**

- Continuous integration and deployment enable fast updates and experimentation.
- New features in Tour, Room, or Ride Booking can be quickly deployed, tested, and rolled back if necessary, improving time-to-market and reducing the cost of failure.
- **Technological Freedom**
- Each service can use the most suitable tools and technologies for its specific needs.
- For instance, the Ride Booking service could use a different database or technology stack compared to the Tour Booking service, allowing for customized solutions.
- **Reusable Code and Modular Design**
- Microservices promote the reuse of functionalities across different modules.
- For example, payment processing or user authentication services developed for Room Booking can be reused in Tour or Ride Booking, reducing development effort and enhancing consistency across services.
- **Enhanced Resilience**
- The independence of each service ensures that the failure of one (e.g., Ride Booking) does not crash the entire system.
- Service failures degrade functionality instead of causing a complete shutdown, improving overall reliability.

Future Scope with Microservices

As we expand the Tourism Ecosystem to include more services (e.g., event management, ticket reservations), microservices will:

- Support parallel development of new modules, allowing multiple teams to work simultaneously on different parts of the ecosystem.
- Simplify the integration of advanced technologies such as AI-driven recommendations or mobile app support for booking services.
- Facilitate long-term scaling and adaptability, making it easier to introduce new features without disrupting existing services.

The microservices design and implementation of the **Tour, Room and Ride Booking** services provide a scalable and flexible architecture. By using Flask, we modularized the functionalities, and by integrating third-party services like Stripe, Mailtrap, and Twilio, we effectively mocked payment, email, and SMS functionalities. This approach enhances the modularity and reusability of services, making the system easily extendable for future features.

PART F: Service/ Process analytics

Task (I):

In this task, we generated event logs for three key business processes from Part C: **Tour Booking**, **Room Booking and Ride Booking**. These business processes are part of the Tourism Ecosystem platform, which provides services for booking tours, accommodations and rides. The purpose of this task is to capture the execution of each process step-by-step, recording the timestamps, tasks, and resources involved. These event logs will later be used to analyze the performance of each process and identify areas for optimization.

Selected Business Processes

The following business processes were selected for event log generation:

- **Tour Booking**: Managed by the Tourism Information Center (TIC), this process lets customers book tours by selecting time slots, checking availability, and confirming with a tour guide.
- **Room Booking**: This process, part of the Accommodation Providers (AP) services, allows customers to book rooms by selecting dates, checking availability, confirming bookings, and processing payments.
- **Ride Booking**: Offered by the Local Transportation Services (LTS), this process allows users to book rides by specifying pickup and drop-off locations, checking availability, and assigning drivers.

Steps Involved in Each Process

The key steps in each process are as follows:

- **Tour Booking**:
- Start Tour Booking
- Select Tour and Time Slot
- Check Tour Availability
- Confirm Tour
- Assign Tour Guide
- Tour Confirmed

- **Room Booking**:
- Start Booking Process
- Select Location and Dates
- Check Room Availability

- Select Room
- Confirm Booking
- Payment Processed
- Send Confirmation
- **Ride Booking**:
- Start Ride Booking
- Input Pickup and Drop-off
- Check Ride Availability
- Confirm Ride
- Assign Driver
- Ride Started
- Ride Completed

Method of Event Log Generation

To generate event logs for each of the processes, Python simulations were developed. The simulations were run in Google Colab, allowing the automatic generation of event logs, which were saved as CSV files. These logs capture:

- **Case-ID**: A unique identifier for each instance of the process (e.g., Booking No. 001).
- **Task-ID**: The specific task being performed (e.g., "Check Room Availability").
- **Timestamp**: The time at which each task was performed.
- **Resource**: The entity (e.g., customer, system) responsible for performing the task.

Below is the Python code used to simulate the **Room Booking** process and generate event logs:

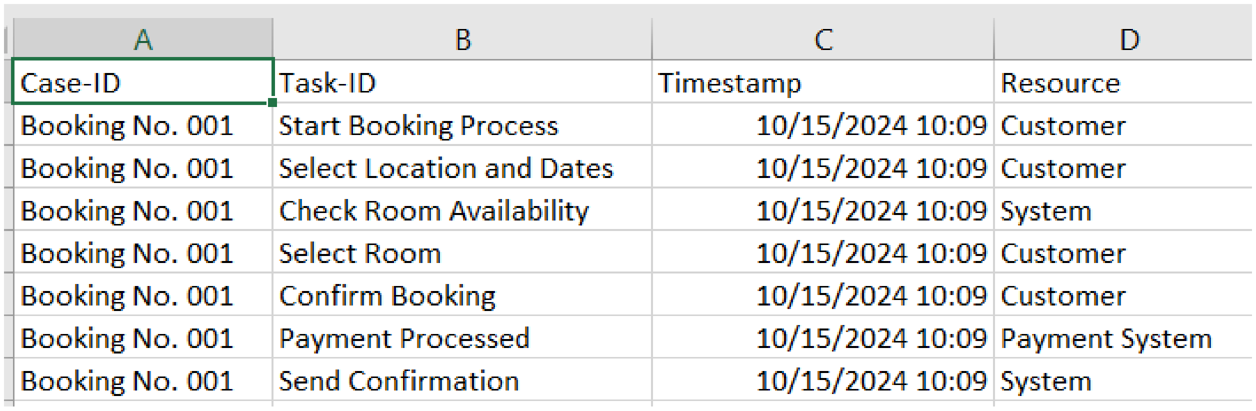
```python
import time
import csv
from datetime import datetime
from google.colab import files

# Define the room booking process steps
room_booking_steps = [
    {"task": "Start Booking Process", "resource": "Customer"},
    {"task": "Select Location and Dates", "resource": "Customer"},
    {"task": "Check Room Availability", "resource": "System"},
    {"task": "Select Room", "resource": "Customer"},
    {"task": "Confirm Booking", "resource": "Customer"},
    {"task": "Payment Processed", "resource": "Payment System"},
    {"task": "Send Confirmation", "resource": "System"}
]
```

```python
# Function to simulate event log generation
def generate_event_log(case_id, process_steps):
    event_log = []
    for step in process_steps:
        # Capture the current time as the timestamp
        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        # Append the log entry with Case-ID, Task-ID, Timestamp, and Resource
        event_log.append([case_id, step["task"], timestamp, step["resource"]])
        # Simulate time delay between steps
        time.sleep(1)
    return event_log

# Function to write event logs to a CSV file
def write_event_log_to_csv(event_log, filename="room_booking_event_log.csv"):
    # Open a new CSV file to write the event log
    with open(filename, mode="w", newline="") as file:
        writer = csv.writer(file)
        # Write the headers
        writer.writerow(["Case-ID", "Task-ID", "Timestamp", "Resource"])
        # Write each event log entry
        writer.writerows(event_log)
    print(f"Event log saved to {filename}")
    # Download the file to your local machine
    files.download(filename)

# Simulate the booking process and generate the event log
case_id = "Booking No. 001"
event_log = generate_event_log(case_id, room_booking_steps)

# Write the event log to a CSV file and download it
write_event_log_to_csv(event_log)
```

Similarly, simulations were developed for **Ride Booking** and **Tour Booking** to capture the step-by-step process of each service. The CSV files generated for each business process contain detailed logs of the tasks performed, including timestamps and resource involvement.

Event Log Examples

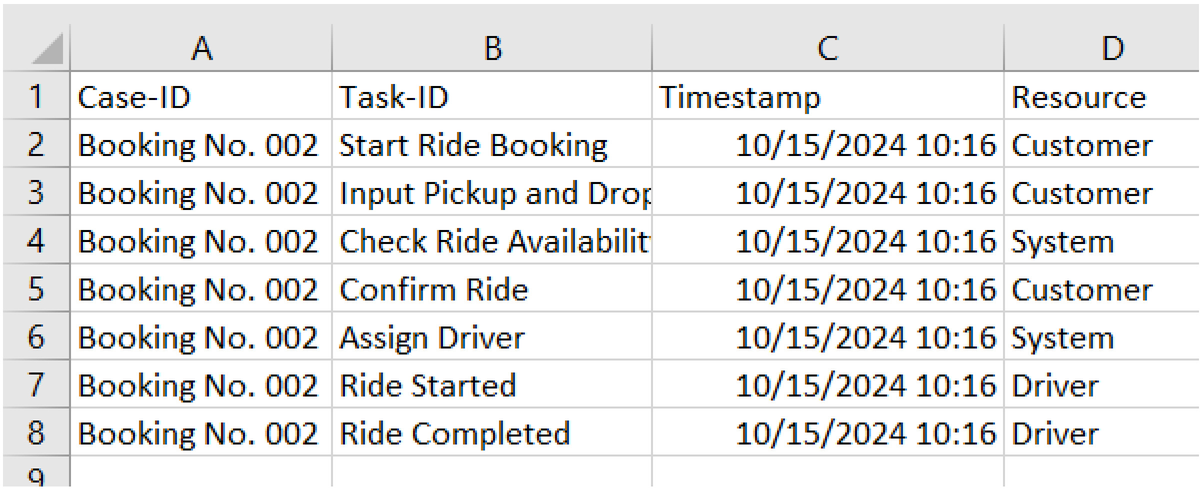
For the **Room Booking process**, a **CSV file** was generated that records the execution of each task, including the **timestamp and the resource responsible** for performing the task. Below is a sample screenshot of the Room Booking event log.



A	B	C	D
Case-ID	Task-ID	Timestamp	Resource
Booking No. 001	Start Booking Process	10/15/2024 10:09	Customer
Booking No. 001	Select Location and Dates	10/15/2024 10:09	Customer
Booking No. 001	Check Room Availability	10/15/2024 10:09	System
Booking No. 001	Select Room	10/15/2024 10:09	Customer
Booking No. 001	Confirm Booking	10/15/2024 10:09	Customer
Booking No. 001	Payment Processed	10/15/2024 10:09	Payment System
Booking No. 001	Send Confirmation	10/15/2024 10:09	System

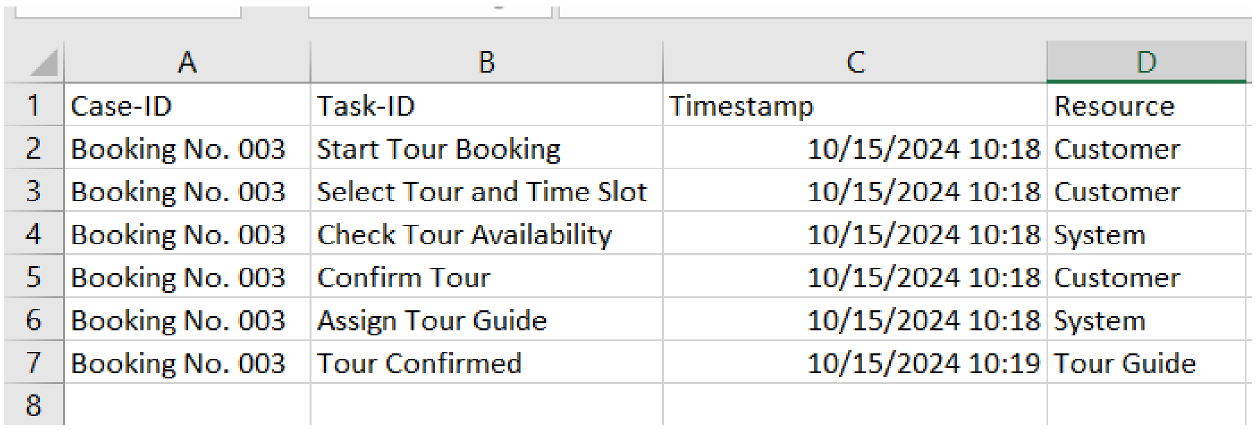
The same approach was used for Ride Booking and Tour Booking, where event logs were generated and saved as CSV files, capturing the sequence of tasks, timestamps, and the resources involved.

For the Ride Booking process, the event log records tasks such as specifying pickup and drop-off locations, checking ride availability, assigning a driver, and completing the ride. Each step was logged with a timestamp to ensure accurate tracking of the process flow, helping to identify any delays that might occur.



	A	B	C	D
1	Case-ID	Task-ID	Timestamp	Resource
2	Booking No. 002	Start Ride Booking	10/15/2024 10:16	Customer
3	Booking No. 002	Input Pickup and Drop	10/15/2024 10:16	Customer
4	Booking No. 002	Check Ride Availabilit	10/15/2024 10:16	System
5	Booking No. 002	Confirm Ride	10/15/2024 10:16	Customer
6	Booking No. 002	Assign Driver	10/15/2024 10:16	System
7	Booking No. 002	Ride Started	10/15/2024 10:16	Driver
8	Booking No. 002	Ride Completed	10/15/2024 10:16	Driver
9				

For the Tour Booking process, the event log tracks the steps involved in selecting a tour and time slot, checking tour availability, assigning a tour guide, and confirming the booking. These logs provide detailed insight into the efficiency of the booking process and any potential bottlenecks related to guide assignment, especially during high-demand periods.



	A	B	C	D
1	Case-ID	Task-ID	Timestamp	Resource
2	Booking No. 003	Start Tour Booking	10/15/2024 10:18	Customer
3	Booking No. 003	Select Tour and Time Slot	10/15/2024 10:18	Customer
4	Booking No. 003	Check Tour Availability	10/15/2024 10:18	System
5	Booking No. 003	Confirm Tour	10/15/2024 10:18	Customer
6	Booking No. 003	Assign Tour Guide	10/15/2024 10:18	System
7	Booking No. 003	Tour Confirmed	10/15/2024 10:19	Tour Guide
8				

The event logs generated for Room Booking, Ride Booking, and Tour Booking provide a comprehensive view of the processes, capturing each step with timestamps and resource involvement. These logs not only facilitate accurate tracking but also enable detailed analysis to identify inefficiencies, such as payment delays, driver shortages, and guide assignment issues. With these insights, the project lays a solid foundation for improving service delivery and optimizing resource allocation across the platform.

Task (II):

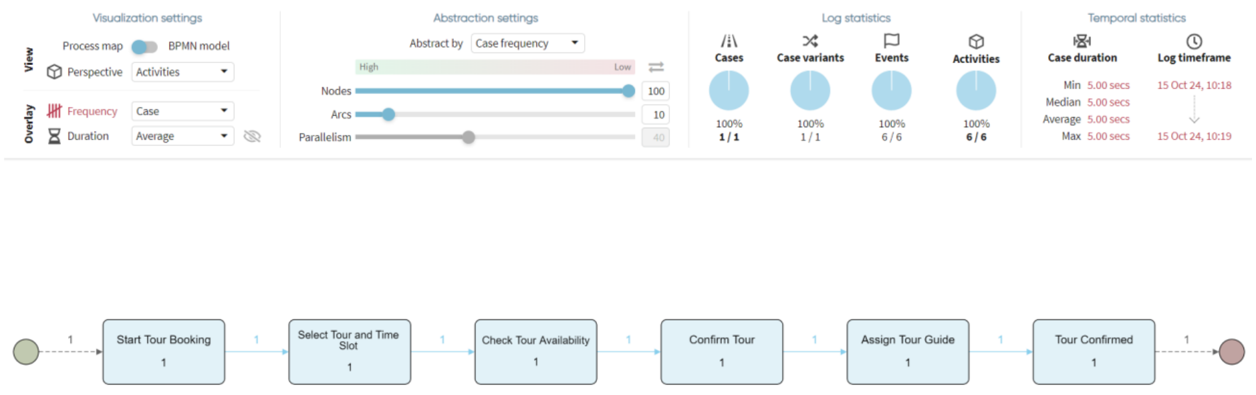
The event logs for the three business processes—**R**oom Booking, Ride Booking, and Tour Booking—were analyzed using the process mining tool **Apromore**. This analysis aimed to visualize the workflows, assess performance metrics, and identify bottlenecks for each process.

Tour Booking Process

The Tour Booking model is depicted in **Figure 1**. This model shows a streamlined workflow, with fewer steps compared to the other two processes.

Key Performance Metrics:

- **Case Duration**: The average duration for the Tour Booking process is slightly shorter at **5 seconds**, indicating a more efficient workflow with less complexity.
- **Bottlenecks**: While the Tour Booking process is generally efficient, the **Assign Tour Guide** task may introduce delays if guide availability becomes an issue, particularly during peak booking times.


Visualization settings
View: Process map / BPMN model
Perspective: Activities
Overlay: Frequency: Case
Duration: Average

Abstraction settings
Abstract by: Case frequency
Nodes: 100
Arcs: 10
Parallelism: 40

Log statistics
Cases: 100% 1 / 1
Case variants: 100% 1 / 1
Events: 100% 6 / 6
Activities: 100% 6 / 6

Temporal statistics
Case duration: Min 5.00 secs, Median 5.00 secs, Average 5.00 secs, Max 5.00 secs
Log timeframe: 15 Oct 24, 10:18 – 15 Oct 24, 10:19

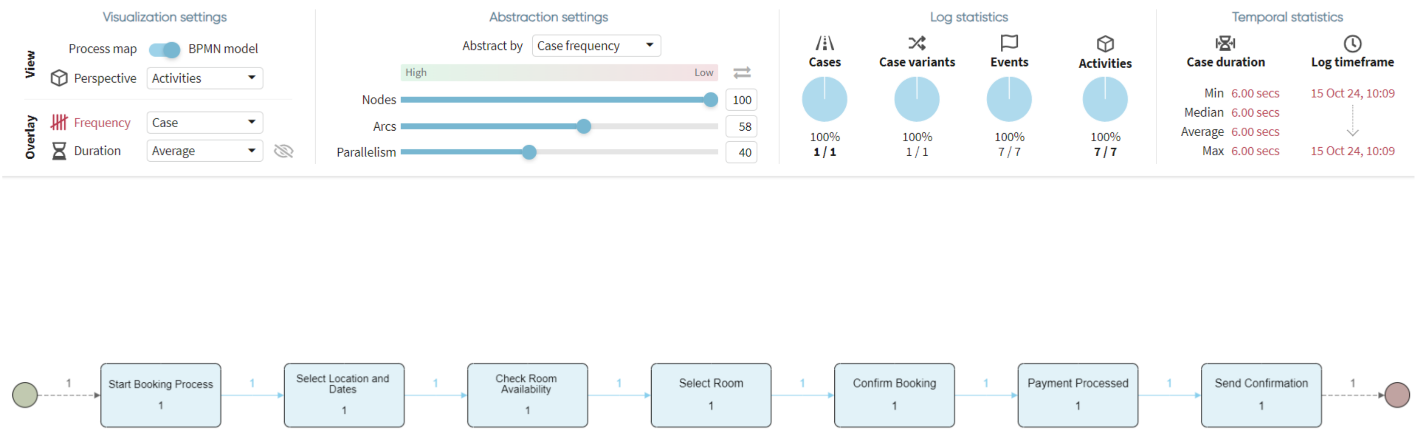
Figure 1

Room Booking Process

The process model for the **Room Booking** process is depicted in **Figure 2**. This model illustrates the sequential activities involved in booking a room, beginning from the initiation of the booking to the final confirmation.

Key Performance Metrics:

- **Case Duration**: The average time taken to complete the process is consistently **6 seconds**, with a minimum and maximum also recorded at **6 seconds**. This uniformity indicates a stable and efficient process.
- **Bottlenecks**: The **Payment Processed** task is a potential bottleneck. Even though the duration is stable, any issues during payment could disrupt the flow and affect user satisfaction.


Visualization settings
View: Process map / BPMN model; Perspective: Activities
Overlay: Frequency: Case; Duration: Average

Abstraction settings
Abstract by: Case frequency
Nodes: 100
Arcs: 58
Parallelism: 40

Log statistics
Cases: 100% 1 / 1
Case variants: 100% 1 / 1
Events: 100% 7 / 7
Activities: 100% 7 / 7

Temporal statistics
Case duration: Min 6.00 secs; Median 6.00 secs; Average 6.00 secs; Max 6.00 secs
Log timeframe: 15 Oct 24, 10:09 – 15 Oct 24, 10:09

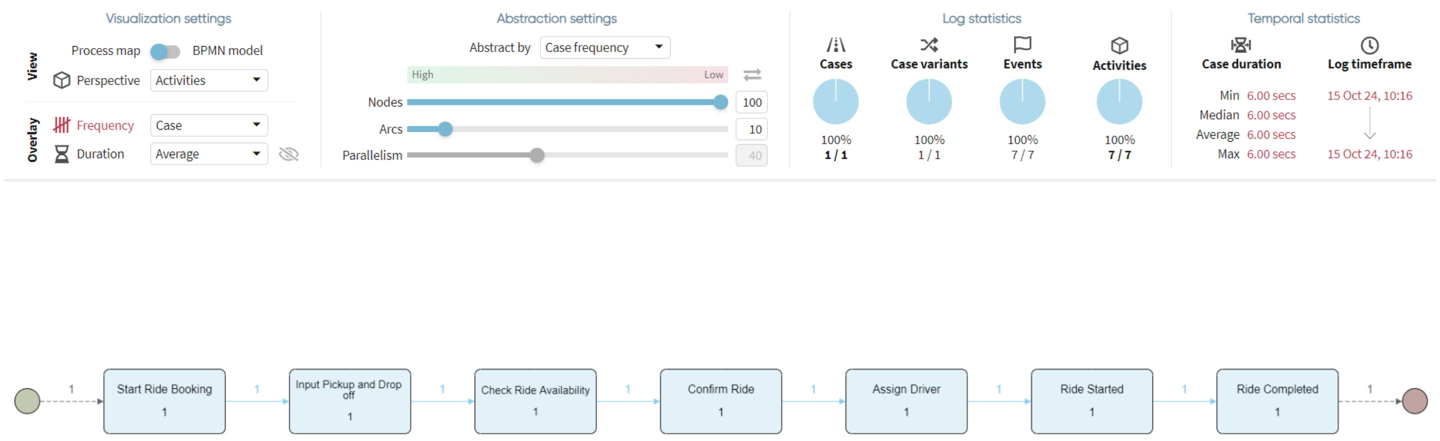
Figure 2

Ride Booking Process

The process model for **Ride Booking** is presented in **Figure 3**. This model highlights the flow of tasks from the initiation of the ride booking to the completion of the ride.

Key Performance Metrics:

- **Case Duration**: The average case duration is again noted at **6 seconds**, consistent with the Room Booking process. However, the actual duration may vary depending on real-time factors like traffic and driver availability.
- **Bottlenecks**: The **Assign Driver** task could represent a bottleneck, especially during high-demand periods. If there are limited drivers available, this could lead to delays in confirming rides, which may not be evident in the static average duration.


Visualization settings

View: Process map / BPMN model
Perspective: Activities

Overlay: Frequency: Case
Duration: Average

Abstraction settings

Abstract by: Case frequency
High — Low
Nodes: 100
Arcs: 10
Parallelism: 40

Log statistics

Cases	Case variants	Events	Activities
100%	100%	100%	100%
1 / 1	1 / 1	7 / 7	7 / 7

Temporal statistics

	Case duration	Log timeframe
Min	6.00 secs	15 Oct 24, 10:16
Median	6.00 secs	
Average	6.00 secs	
Max	6.00 secs	15 Oct 24, 10:16

Figure 3

Comparative Insights and Recommendations

The analysis of the Room Booking, Ride Booking, and Tour Booking processes reveals both strengths and areas for improvement:

- **Efficiency**: All three processes exhibit a strong average case duration, indicating effective workflow design. However, the reliance on payment processing in Room Booking and driver assignment in Ride Booking highlights potential vulnerabilities in maintaining efficiency.
- **Identified Bottlenecks**: Each process has potential bottlenecks that could be addressed:
- **Room Booking**: Investigate improving the payment processing system to reduce any possible delays.
- **Ride Booking**: Develop strategies to enhance driver availability during peak times, such as dynamic driver allocation.
- **Tour Booking**: Implement contingency plans for guide assignments to minimize delays.
- **Data-Driven Improvements**: The insights gathered from this analysis provide a strong foundation for targeted improvements. Implementing changes based on actual data can lead to significant enhancements in service delivery and user experience.

Task (III):

This task demonstrates the execution of Tasks 1 and 2, focusing on the analysis and design of services within the Tourism Ecosystem platform. The objective is to provide a comprehensive overview of how event logs were generated and analyzed, and how these findings inform service design.

Execution of Task (I): Event Log Generation

In Task 1, we focused on generating event logs for three key business processes within the Tourism Ecosystem. These processes were essential for understanding user interactions and operational flow.

Selected Business Processes

- **Room Booking**: This process allows customers to book accommodations by selecting dates, checking room availability, confirming bookings, and processing payments. It is a crucial aspect of the platform as it directly impacts user satisfaction and revenue generation.
- **Ride Booking**: This service enables users to specify pickup and drop-off locations, check ride availability, and assign drivers. It plays a vital role in enhancing the overall travel experience by ensuring that transportation needs are met efficiently.
- **Tour Booking**: Managed by the Tourism Information Center, this process allows customers to select time slots for tours, check availability, and confirm bookings with a tour guide. This service enriches the user experience by offering personalized travel experiences.

Steps Involved in Each Process

For each process, we broke down the activities into clear steps, ensuring a thorough understanding of the workflow. This structured approach facilitated comprehensive data capture during the event log generation. The steps included:

Room Booking:

- Start Booking Process
- Select Location and Dates
- Check Room Availability
- Select Room
- Confirm Booking
- Payment Processed
- Send Confirmation

Ride Booking:

- Start Ride Booking
- Input Pickup and Drop-off
- Check Ride Availability
- Confirm Ride

- Assign Driver
- Ride Started
- Ride Completed

Tour Booking:

- Start Tour Booking
- Select Tour and Time Slot
- Check Tour Availability
- Confirm Tour
- Assign Tour Guide
- Tour Confirmed

Method of Event Log Generation To capture the execution of each process step-by-step, Python simulations were developed to automate the event log creation. This method ensured that logs were generated accurately and efficiently, with minimal manual intervention. The logs included:

- **Case-ID**: A unique identifier for each instance of the process (e.g., Booking No. 001). This helps in tracking individual user interactions and provides a way to analyze specific cases in detail.
- **Task-ID**: The specific task being performed (e.g., "Check Room Availability"). This categorization allows for granular analysis of where time and resources are being allocated.
- **Timestamp**: The precise time at which each task was performed. This data is critical for measuring performance and identifying delays in the process.
- **Resource**: The entity (e.g., customer, system) responsible for performing the task. Understanding which resources are involved helps in identifying potential areas for improvement in task execution.

Execution of Task 2: Process Analysis Using Apromore

In Task 2, we analyzed the event logs generated in Task 1 using the process mining tool Apromore. This analysis aimed to visualize workflows, assess performance metrics, and identify bottlenecks for each process.

Analysis of Each Process

- **Room Booking Process**:
- **Model Visualization**: A process model was created showing the sequential activities involved in booking a room, from initiation to final confirmation.
- **Key Metrics**: The analysis identified an average case duration of 6 seconds, indicating a stable and efficient process. However, the **Payment Processed** task was identified as a potential bottleneck, as any issues during payment could disrupt the flow and affect user satisfaction.

- **Ride Booking Process**:
- **Model Visualization**: The task flow from the initiation of the ride booking to completion was illustrated, providing a clear understanding of the user journey.
- **Key Metrics**: The average case duration was also noted at 6 seconds, similar to Room Booking. The **Assign Driver** task could represent a bottleneck, especially during high-demand periods when driver availability may be limited.
- **Tour Booking Process**:
- **Model Visualization**: A streamlined workflow was depicted, highlighting the steps involved in confirming a tour.
- **Key Metrics**: The average duration for the Tour Booking process was slightly shorter at 5 seconds, indicating a more efficient workflow. However, potential delays were noted in the **Assign Tour Guide** task, particularly during peak booking times.

Comparative Insights and Recommendations The analysis highlighted the efficiency of all three processes while identifying specific bottlenecks that could be improved. Recommendations for enhancements include:

- **Room Booking**: Investigate improving the payment processing system to reduce possible delays and enhance user satisfaction.
- **Ride Booking**: Develop strategies to enhance driver availability during peak times, such as dynamic driver allocation.
- **Tour Booking**: Implement contingency plans for guide assignments to minimize delays and improve the booking experience.

Outcomes and Integrative Analysis

The execution of these tasks has provided valuable insights into the operational efficiency of the Tourism Ecosystem.

- **Comprehensive Data Capture**: The event logs captured detailed execution data, allowing for precise analysis of each process.
- **Visual Process Models**: These models enabled a clear understanding of workflows and performance, facilitating better decision-making.
- **Identification of Improvements**: The insights highlighted specific areas for optimization, supporting data-driven strategies to enhance service delivery.

This task successfully integrates the outcomes of Task 1 and Task 2, demonstrating how event logs were generated and analyzed to improve service design within the Tourism Ecosystem platform. The insights gained will inform future enhancements to the user experience, ensuring a seamless and efficient booking process.

PART G: Member Contribution Acknowledgement

We, **Group Musketeers** have assigned each service provider to a group member, ensuring that everyone could collaborate on the same tasks together. Tasks were divided equally to keep all members engaged and prevent anyone from being overwhelmed. We held regular meetings to stay aligned on the project's progress. If any challenges arose, we supported each other to ensure the project was completed on time.

Contribution Details

Serial No.	Module Name	Sub-module Name	Contributor Name (ID)	Contribution Rating
A (Task I)	Service Identification & Specification	-	Anika Salsabil (8764736) Yingqi Hao (8867951)	CONTRIBUTED
A (Task II)		Tourism Information Center (TIC)	Anika Salsabil (8764736)	CONTRIBUTED
		Local Attractions and Musuems (LAM)	Anika Salsabil (8764736)	
		Accommodation Providers (AP)	Shagun Shrestha (7739084)	
		Restaurants & Cafes (RC)	Thisuru Deesan Karunathilaka Aitthappulige (8697127)	
		Local Transportation Services (LTS)	Yingqi Hao (8867951)	
B	Enterprise Architecture	Using Archimate	Md Ab Hayat (8309346) Md Tanvir Sazib (8064817)	CONTRIBUTED
C (Task I & II)	Business Process Design & Analysis	Tourism Information Center (TIC)	Anika Salsabil (8764736) Yingqi Hao (8867951) Thisuru Deesan Karunathilaka Aitthappulige (8697127)	CONTRIBUTED
		Local Attractions and Musuems (LAM)	Yingqi Hao (8867951)	
		Accommodation Providers (AP)	Shagun Shrestha (7739084)	

		Restaurants & Cafes (RC)	Thisuru Deesan Karunathilaka Aitthappulige (8697127)	
		Local Transportation Services (LTS)	Yingqi Hao (8867951)	
D	Service Design and Analysis using SoaML	Tourism Information Center (TIC)	Shagun Shrestha (7739084)	CONTRIBUTED
		Local Attractions and Musuems (LAM)	Yingqi Hao (8867951)	
		Accommodation Providers (AP)	Shagun Shrestha (7739084)	
		Restaurants & Cafes (RC)	Thisuru Deesan (8697127) Yingqi Hao (8867951)	
		Local Transportation Services (LTS)	Yingqi Hao (8867951)	
E (Task I & II)	Microservices Design & Implementation	Tour Booking service from TIC	Anika Salsabil (8764736)	CONTRIBUTED
		Ride Booking service from LTS	Md Tanvir Sazib (8064817)	
		Room Booking service from AP	Anika Salsabil (8764736) Thisuru Deesan Karunathilaka Aitthappulige (8697127)	
F (Task I, II, III)	Service/ Process Analytics	Tour Booking service from TIC	Md Ab Hayat (8309346) Anika Salsabil (8764736)	CONTRIBUTED
		Ride Booking service from LTS	Md Ab Hayat (8309346)	
		Room Booking service from AP	Md Ab Hayat (8309346) Anika Salsabil (8764736)	
G	Member Contribution Acknowledgement	-	Anika Salsabil (8764736)	CONTRIBUTED

	Project Presentation & Demonstration	-	Anika Salsabil (8764736)	CONTRIBUTED
	Weekly Meeting Reports	-	Anika Salsabil (8764736) Shagun Shrestha (7739084) Thisuru Deesan Karunathilaka Aitthappulige (8697127)	CONTRIBUTED

Here's a detailed explanation of the team member-wise contribution for the group project ***(this portion is compiled by each of the member himself/ herself)***:

- **Anika Salsabil (8764736)**: Anika led the design and implementation of microservices architecture. She took the lead in developing the tour booking service, including the front-end, back-end, database integration, and third-party services, which provided a foundation for the other team members to build additional microservices. Anika also organized and led weekly meetings, managed task planning, and ensured coordination within the team. She continuously shared feedback and knowledge after reviewing members' tasks. Additionally, she prepared the presentation slides and demonstration for the project. Anika also merged the whole implementation of the microservices and uploaded it with proper instructions. She basically conducted the overall project management, review, coordination, planning and execution (including documentation & development)

- **Thisuru Deesan Karunathilaka Aitthappulige (8697127)**: Thisuru designed and implemented the microservice architecture for room booking using different coding frameworks for the backend and front end. Thisuru has developed the backend using Node JS, setting up the database using MongoDB to store the room and booking data, and integrated Stripe for secure payment processing. Whole code is updated in the following repository for the reference.
  Code: https://github.com/Thisuru/Hotel-Room-Booking

  He developed most of the functionalities (Payment integration, Database integration, User Interface and response view part and etc.) and then compared it to Anika's tour booking service, which includes email and SMS sending features. After team discussions, they decided to implement the room-booking service using the same coding frameworks as the tour-booking architecture. Thisuru also contributed by creating weekly meeting reports, actively participating in all weekly meetings, and sharing feedback with the team. And Thisuru has also designed the BPMN for Restaurant and Cafes (RC) service providers and Semantic and Cumulative Effect Annotation part reflecting real-world processes as well for the RC and given references with real-world examples for RC. These real-world examples were directly used when developing the BPMN and annotations for RC. And he has done the Tourism and Information Center (TIC) reference part and helped the team to achieve the final goal.

- **Yingqi Hao (8867951)**:
  YingQI contributed significantly to the design and analysis of the Local Transportation Services (LTS), Restaurants and Cafes (RC), and Local Attractions and Museums (LAM) system, focusing on service architecture, interfaces, and contracts, as well as integrating real-time traffic updates for smoother route planning. He was involved in service design and analysis, contributing to the overall system architecture and service contracts.
  In the Local Attractions and Museums (LAM) system, he worked on guided tour reservations and educational content delivery, ensuring workflows accurately represented user interactions. YingQI also helped with BPMN modeling for Tourism Information Centers (TIC), Restaurants and Cafes (RC), and LTS, with semantic annotations reflecting real-world processes.

- **Shagun Shrestha (7739084)**:
  Shagun played an essential role in the project by contributing to the identification and specification of services, as well as the design and analysis of the Tourism Information Center (TIC) and Accommodation Providers (AP). She worked on the service architecture, interfaces, and contracts for both TIC and AP. Additionally, Shagun contributed to BPMN modeling and semantic annotations for these services. She also managed weekly meetings with Anika, ensuring smooth communication and collaboration. Shagun was also responsible for creating weekly meeting reports and actively participated in all meetings, sharing valuable feedback to the team.

- **Md Ab Hayat (8309346):**
  Hayat contributed to the project by leading the Enterprise Architecture (EA) design using the ArchiMate framework and conducting detailed process analytics. His work on the EA design focused on developing models across the business, application, and technology layers to ensure seamless alignment between business goals, software components, and the required infrastructure. He mapped key business processes, such as booking systems, and integrated them with microservices, ensuring scalability and efficient communication across the platform. In the process analytics phase, Hayat developed Python simulations to generate event logs for Room Booking, Ride Booking, and Tour Booking. These

simulations captured essential workflow steps such as availability checks, payment processing, driver assignment, and tour confirmations, storing the results in CSV files. Using Apromore, he analyzed the event logs to visualize workflows, assess performance metrics, and identify bottlenecks. Key bottlenecks identified included delays in payment processing for room bookings, driver shortages during peak hours for ride bookings, and guide assignment issues for tour bookings. His insights provided actionable recommendations to improve process efficiency and resource allocation. Additionally, he documented the EA models, process simulations, and mining results for the final report, ensuring that all components were clearly explained with code, screenshots, and diagrams to support the team's work and align with project objectives.

- **Md Tanvir Sazib (8064817):**
  Md Tanvir Sazib contributed significantly to the project by designing the Business, Application, and Technology layers using the ArchiMate framework. These layers were developed to ensure smooth alignment between business objectives, software components, and the supporting infrastructure. He worked closely with AB Hayat on the Enterprise Architecture (EA) design to ensure consistency across all models and seamless integration of the platform's services.
  In addition to EA design, He took the lead on implementing the Ride Booking Service. This involved developing the backend using Flask, setting up the MySQL database to store vehicle and booking data, and integrating Stripe for secure payment processing. To enhance user experience, He configured Mailtrap for email notifications and Twilio for SMS alerts, ensuring users received real-time booking confirmations.
  Md Tanvir Sazib also collaborated with Hayat on Task E: Microservices Design and Implementation, helping to design and deploy microservices for various platform components. The Ride Booking Service was seamlessly integrated within this architecture, focusing on scalability and efficient communication.
  For the final submission, Md Tanvir Sazib documented the Ride Booking Service architecture, process flows, and code with relevant screenshots and diagrams. This ensured the team's efforts were well-documented and aligned with the project goals.

Team Member Statements:

Student Name (ID)	**Statement**	**Student Signature**
Anika Salsabil (8764736)	I agree with my group member contribution report and ratings.	Anika

Thisuru Deesan Karunathilaka Aitthappulige (8697127)	I agree with my group member contribution report and ratings.	Thisuru
Yingqi Hao (8867951)	I agree with my group member contribution report and ratings.	Yingqi
Shagun Shrestha (7739084)	I agree with my group member contribution report and ratings.	Shagun
Md Ab Hayat (8309346)	I agree with my group member contribution report and ratings.	Hayat
Md Tanvir Sazib (8064817)	I agree with my group member contribution report and ratings.	Tanvir

All team members have reviewed and approved the contribution report, final report, and presentation. The project report accurately reflects the team's work, contributions, and progress. We are pleased to submit it for approval.