Enterprise Systems Architecture & BPMN Workflow Optimization

Table of Content

Introduction

The tourism industry has evolved a lot with the inclusion of technologies that need adequate architecture to integrate multiple service providers into it in a seamless manner. This is one of the areas where rapid expansion has occurred and is happening through the adoption of SOA concepts and their recent offshoots, microservices, in order to develop highly scalable, flexible, integrated platforms for tourism ecosystems.

The project covers the design and analysis of the service-oriented tourism ecosystem by incorporating TICs, APs, LTSs, and other concerned stakeholders in the tourism industry. The proposed platform is intended to make it easier for tourists, accommodation providers, transportation services, and other local businesses to experience integrated service facilitation from a common interface.

It involved the identification of services, their specification, architecture design, and the actual implementation of the service, using up-to-date service-oriented tools and standards, such as SoaML. In designing, it has complied with the basic principles of SOA, which include loose coupling among services to ensure reusability for easy scalability and maintenance capability. This will be ideal for the dynamic nature of the needs in the tourism industry.

The current work focuses on main principles of both SOA and microservices, aiming at a strong, scalable, and flexible basis that should be able to support the rich diversity of tourist needs. The proposed architecture, in this report, demonstrates how the latest principles of software engineering can change the way tourism services are offered and consumed, ensuring a seamless and enriched experience to all stakeholders involved.

TASK A. Service Identification and Specification

Service provider 1:Tourism Information Center (TIC)

Service Name	**Visitor Information**
Service ID	TIC_01
Goal	Provide accurate, timely and personalised tourism information, including attractions, accommodation, transportation, and so on.
Preconditions	● Users must have logged in ● A valid source location and destination must be provided. ● Specific departure time must be entered.
Postconditions	● Recommend the best travel information based on destination and user preferences.
Assumptions	Recommend the best travel information based on destination and user preferences.
Inputs	● Destination <string>: The location the user wants to travel to. ● Location<string>: User's current location. ● Project<string>: Which project are users looking for(attractions, accommodation, transportation, and so on) ● Transportation ● Type<string>: What kinds of attractions are users looking for.(Entertainment, attractions, restaurant
Outputs	● Price Details <string>: The detailed information of the final price should be displayed on the website according to the users' input. ● Information<string>:Travel information required by users
Key steps	Step 1: Users login to the Tourism Information Center with their account. Step 2: The user selects or searches activities based on their preferences. Step 3: The user selects the desired activity, date, and number of participants. Step 4: The service checks availability for the scheduled activities submitted by the user.
QoS factors	● Response Time ≤ 2 seconds: The time between the query sent by the user and the result returned. ● Stability ≥ 99.99%: The Uptime for the service is quite stable.
Delivery schedules	N/A
Payment schedules	N/A
Penalties	N/A
Value model	● Goal ○ Provide users with accurate travel information. ○ Recommend suitable places based on user preferences. ● Objective functions

	○ Improve user satisfaction with query results ○ Increase the probability of users booking directly on the platform after searching for information ● Plan ○ Provide accurate and personalized recommendation information
Resources	● Front-end system: The user interface that displays tourism information and captures user inputs. ● Back-end system: Supports the querying and recommendation of tourism information, including user preference analysis and real-time updates of tourism resources. ● Tourism Information Database: Stores detailed data related to attractions, accommodation, restaurants, transportation, etc. ● User Database: Stores user personal information, historical queries, preferences, and login details. ● Recommendation System: Analyzes user inputs and preferences to generate personalized tourism recommendations. ● Notification System: Used to push notifications to users, such as suggested attractions or activity confirmations. ● Availability Check Module: Ensures real-time availability of selected activities, attractions, or accommodations. ● API Integration: Designed and implemented to exchange data with external services (e.g., transportation, accommodation providers).

Service Name	**Tour Bookings**
Service ID	TIC_02
Goal	Provide users with flexible travel booking services and comprehensive tour information, enabling them to book the most suitable tour for themselves.
Preconditions	● Users can access the web pages
Postconditions	● Reservation is confirmed ● If the users pay upfront, then payment is processed and receipt is issued
Assumptions	● Users have valid contact details ● The network is accessible ● The payment system is operational and secure
Inputs	● Destination<string>: Users select the places they want to go or the activities that they want to take part in ● Date<date>: The start date of the tour ● Time<int>: There may be different time periods in some tours, in that case, the users should choose the time that they prefer ● Travellers: ○ Type<string>: The type of travellers, including “adult”, “child” and “infant”

	○ Number<string>: The number of different types of travellers ● Promotion Code<string>: If the users have some promotion codes they can use them to get a discount. ● First Name<string>: The first name of the user ● Last Name<string>: The last name of the user ● Email<string>: Users' emails ● Phone Number<string>: Users' phone numbers ● Special Requirements<string>: Users can input their special needs in terms of diet and disability facilities. ● Payment Method<string>: Users can choose to pay in different ways. ● CardInformation(If users choose to pay with Debit/Credit card, then the information below is required): ○ Card Number<string>: The number of the card ○ Expiration Date<date>: The expiration date of the card ○ Security Code<string>: The card's security code ○ Country<string>: The country of the card, in order to check out the bills using different currencies, usually can be generated automatically by the card number.
Outputs	● Price Details <string>: The detailed information of the final price should be displayed on the website according to the users' input. ● Reservation/Payment Confirmation<string>: Whether the user chooses to pay upfront or just makes a reservation, the information after reserving or paying should be shown on the website, including the tour information, the travellers' information and/or the payment receipt. ● Confirmation email<string>: The Reservation/Payment Confirmation information will be sent to the users' email automatically.
Key steps	Step1: Users choose a destination Step2: Users choose their preferred tour Step3: Users input some contact information
QoS factors	● Response Time ≤ 2 seconds: The time between the query sent by the user and the result returned. ● User Satisfaction ≥ 90%: User satisfaction with the booking experience is high. ● Stability ≥ 99.99%: Stability of complete booking service.
Delivery schedules	● <BookingConfirmationTime, ≤ 2 seconds>: A confirmation notification should be sent within 2 seconds of a successful booking. This notification should include booking details and payment receipt.

	● <NotificationTime, 24 hours>: This system shall notify the customer 24 hours before the start of the event. The notification should include event details and safety guidance.
Payment schedules	● <PaymentTime, ≤ 15 minutes>: Payment must be completed within 15 minutes of order submission. If payment is not completed within this time limit, the order will be automatically cancelled.
Penalties	● Cancellation penalty: The user cancels the reservation within 48 hours before the tour starts, minus 20% of the tour payment. ● Change penalty: Users do not support changes within 24 hours before the start of the tour, and changes at other times will be deducted 10% of the tour payment. ● Non-attendance penalty: If the user fails to attend the event on time and did not cancel the reservation, the full 100% of the event payment will be deducted.
Value model	● Goal ○ Provide a convenient tour booking experience for customers. ○ Increase bookings and customer satisfaction. ○ Increase the participation of tour organisers and suppliers. ● Objective functions ○ Maximise booking conversion rate. ○ Maximise customer satisfaction rating. ○ Minimise booking cancellation rate. ● Plan ○ Provide comprehensive and accurate activity information, and users' reviews to help users make decisions. ○ Provide up-to-date tour information, reflect the latest tour status. ○ Provide the "reserve" method so the users don't have to pay upfront, giving users more flexible booking options ○ Simplify booking steps and form filling to optimise the online booking process to reduce user booking time. ○ Phone 24/7 customer support to resolve scheduled issues.
Resources	● Booking system: Software application ● Reservation Database: Store user information and booking history data. ● Customer service support and reviews: Manage bookings and provide customer support.

Service Name	**Event Notifications**
Service ID	TIC_03

Goal	Provide accurate and timely notifications for those who have booked a trip, and send new campaign promotions for those who have not. Enhance visitor engagement by keeping them informed of relevant events.
Preconditions	● The user has registered for an account on the platform. ● Users asked for event notifications.
Postconditions	● Confirm user event information. ● Timely synchronisation of event promotions.
Assumptions	● The user's schedule has not changed. ● The user has turned on notification permissions for the application. ● The notification can be viewed only when the user is logged in the application.
Inputs	● Preferences<string>: Types of user interest. ● Event<string>: Serial number of the event that has been booked by the user(If the user booked an event)
Outputs	● Latest news on events, price changes, etc. ● Countdown to event opening. ● Reminders of itinerary details.
Key steps	Step 1: Users login to the Tourism Information Center with their account. Step 2: User input serial number of the event(If the user booked an event). Step 3: The platform continuously monitors changes to the project and notifies.
QoS factors	● Response Time ≤ 2 seconds: The time between the query sent by the user and the result returned. ● User Satisfaction ≥ 90%: User satisfaction with the booking experience is high. ● Stability ≥ 99.99%: Stability of complete booking service.
Delivery schedules	N/A
Payment schedules	N/A
Penalties	N/A
Value model	● Goal ○ Provide detailed information about the event, ensure that users remain informed about the event on an ongoing basis. ○ Making supplier marketing more efficient ● Objective functions ○ Increase the frequency of users booking events on the platform after receiving promotional push notifications ○ Reduce the possibility of users missing out on activities ● Plan ○ Determine a reasonable notification frequency.

	○ Strengthen the communications with suppliers to ensure timely access to the latest information.
Resources	● Database System: A database is required to store user event data, preferences, and notification history, ensuring timely and accurate event tracking. ● Back-end Service: Handles the processing of user preferences, event monitoring, and real-time updates. It ensures that notifications are sent based on the latest information. ● Front-end Interface: A user-friendly interface for guests to manage their event bookings, receive notifications, and adjust their preferences. ● Event Monitoring System: Continuously tracks event changes and updates, ensuring that users receive timely and relevant notifications. ● Notification System: Sends out reminders and alerts to users about upcoming events, promotions, and changes. This includes email, SMS, and in-app notifications.

Service provider 2:Accommodation Providers (AP)

Service Name	**Room Booking**
Service ID	AP_01
Goal	Enable guests to choose rooms, check prices, and make reservations based on their desired destination.
Preconditions	● Users must have an account of our ecosystem and have logged in. ● Users have a destination where they want to book a room beside it. ● Valid guest information(name, contact phone number). ● Room must be available for booking, including datetime ● Payment approved.
Postconditions	● Users have booked rooms, Booking is confirmed and saved in the system. ● User receives the confirmation with booking details.
Assumptions	● The guest provides valid payment information. There's enough money in users' accounts to pay the fees. ● The user's booking information is complete and correct(NoFakeID). ● The users want to live beside the destination.
Inputs	● Destination <string>: Where you want to live beside. ● Room Date <datetime>: check-in and check-out time for room. ● Occupancy and bed requirements<integer> ● Conditions selection<document>:

	○ Facilities<optional> ○ Hotel star rating<integer> ○ Aera<string> ● Guests information<document> ○ Name<string>:An identifier ○ Contact information<string>:email, phone number ○ Preferences<string>:Non-smoking or smoking rooms ● Payment information <list>: ○ Credit card number <string> ○ Cardholder Name <string> ○ Expiration Date <date> ○ CVV <string> ○ Billing address <string> ○ Save the card information<boolean>: The user can choose whether the system should save their card information except CVV for future use so they don't have to input anymore.
Outputs	● Booking Confirmation <document>: ○ Hotel name <string> ○ Hotel address <string> ○ Room Date <datetime> ○ Room information <string> ○ Guest name <string> ○ Meals<string>:such as 'no meals included' ○ Room amenities<string> ○ Cancellation Policy<string> ● Payment receipt <document>: ○ Payment amount (float): Total cost you have paid ○ Payment method (string) ○ Reference number (string): Unique identifier for the booking payment.
Key steps	Step 1: Users login in the ecosystem with their account. Step 2: Users select destinations where they want to travel. Step 3: Users select suitable rooms nearby. Step 4: The system checks availability for the room submitted by the user. Step 5: Users pay online. Step 6: Hotel System confirms the booking. Step 7: Enter user details（The system automatically obtains user information). Step 8: Users receive a confirmation notification with details.
QoS factors	● Booking Confirmation Time ≤ 3 seconds: Send confirmation booking request to return result. ● Stability ≥ 99.99%: Stability of complete reservation service.

	● Security 99.99% : Measures to ensure that the service is protected from unauthorized access and data breaches.
Delivery schedules	● <Booking Confirmation Time, ≤ 5 seconds>: A confirmation notification should be sent within 5 seconds of a successful booking, including booking details and payment receipt.
Payment schedules	● <Payment time limit, ≤ 20mins>: After the user submits the order, the payment must be completed within 20 minutes. The order will be automatically cancelled if the timeout occurs, and the room is released again.
Penalties	● Cancellation penalty: This booking cannot be modified, and no refund will be given if you cancel it. If you fail to check in, a penalty equivalent to the cancellation fee will be charged. You won't be able to cancel or modify your booking after the cancellation time. ● Change penalty: Additional changes only supported within 24 hours of the booking. Changes after 24 hours will incur an additional fee of 10% of the total payment.
Value model	● Goal: ○ Customer Value: ■ Provide convenient room booking experience for guests,Increase bookings and customer satisfaction with the system, high efficiency for booking. ○ Supplier Value: ■ Provide increasing occupancy and increasing revenue for hotels. ■ Give a convenient system to the AP administrator to manage the hotel systems. ○ Platform Value: ■ The value of our platform as an intermediary platform lies in connecting customers and suppliers. ■ Make a profit. ● Objective functions: ○ Minimize booking cancellation rate ○ Maximize operational efficiency. ○ Minimize latency. ● Plan: ○ To achieve Customer Value: ■ Offering lower prices, a wider range of options and more detailed hotel information for guests.

	■ Provide detailed room types' description and customer reviews ○ To achieve Supplier Value: ■ Our platform can help hotels achieve these goals by increasing their exposure, streamlining the booking process . ○ To achieve our platform Value: ■ We could consider realising the value of the platform by charging commissions, offering value-added services (e.g., insurance, travel guides), or leveraging customer data to provide personalised recommendations. ■ ad revenue.(How our platform can monetize)
Resources	● Front-end: For users ● Back-end: ○ Users Booking system: Software application. ○ Administration system ○ Notification system: resources used to notify customers of successful reservations and send confirmation information, such as email systems and SMS services. ● Database: ○ Reservation Database: Store users' name and booking history data. ○ Guests Database: Store users' information. ○ Hotel Information Resources: This includes all data related to the hotel, such as the hotel's name, location, room type, number of rooms, room availability, reviews, etc. ○ Customer information resources: including personal information of customers, historical booking records, payment information, preferences, etc. ○ Room Inventory Resources: This involves room inventory management for each hotel. For example, the number of available rooms, availability on a specific date, room type (single room, double room, suite, etc.). ● API: ○ Design and implement API to allow data exchange between different modules. ● Payment Gateway Integration: Integrates with multiple payment gateways to process various types of payments (credit cards, netbanking, e-wallets).

Service Name	**Guest Services**
Service ID	AP_02

Goal	Offer services to guests during their stay, and improve guests' satisfactions.
Preconditions	● Users checked into a room. ● The service request is within our Guest service list, or it is legal and reasonable.
Postconditions	● Service completion recorded in the system. ● Send a completion notification to guests when staff finish the service.
Assumptions	● Staff are available at the requested time. ● Room service ingredients are in stock. (If certain ingredients are not available, the service may offer alternatives or cancel the request.)
Inputs	● Guest services type <string>：Users need services types, such as housekeeping, laundry services etc. ● Guest details <string>: Users' name, room number. ● Services prefer time <datetime>: ○ Start time <datetime>: The time of the service starts. ○ End time <datetime>: The deadline for service.(eg: clean up before guests go back) ● Payment information <string, date>:(if it is a paid-service.) ○ Credit card number <string> ○ Cardholder Name <string> ○ Expiration Date <date> ○ CVV <string> ○ Billing address <string>
Outputs	● Confirmation <document>: ○ service Type <string> ○ service Date <datetime> ○ service total price<float>(if it is a paid service) ● Service completion notification<string> ● Payment receipt <document>: (if it is a paid service) ○ Payment amount (float) ○ Payment method (string)
Key steps	Step 1: Receive service booking 24h. Step 2: Offer services when guests need it. Step 3: Satisfy requirements which come from guests. Step4: Do the service. If it is a paid service, deduct from guests' accounts . Step5: Send a completed notification to guests. Step6: Send receipts to guests.
QoS factors	● High Availability ≥ 98% ● Performance: ○ low Response Time 1 second

	○ High Throughput 1ms ○ Low Latency 2-5 ms
Delivery schedules	● <Confirmation Time, ≤ 5 seconds>: A confirmation notification should be sent within 5 seconds of a successful booking, including service types and payment receipt.
Payment schedules	● <Payment time limit, ≤ 1 hour>:There's 1 hour for guests to pay for the service after pressing the 'reserve' button.
Penalties	N/A
Value model	● Goal: ○ Customer Value: ■ Provide a comfortable stay experience for guests. ■ Increase customer satisfaction during stay duration. ○ Supplier Value: ■ Give guests the best stay experiences. ■ Increase good comments rate. ● Objective functions: ○ Maximize the rate of services completed on schedule. ○ Maximize customer satisfaction rating ● Plan: To achieve Customer Value and Supplier Value, suppliers must give patient and on-time services to guests.
Resources	● Front-end ● Back-end: ○ Guest service system:Handle guest inquiry and issues and provide support in time. Arrange staff to help guests to solve problems. ○ Administration system ○ Inventory system: This system manages the inventory of food, ensuring stock levels are maintained for guest services. ● Database: ○ Reservation Database: Store users' name and services booking history data. ○ Guests Database: Store users' information. ● Payment system: Addressing fee payment, including credit card, posit card, and other methods. ○ Payment Gateway Integration**:** Integrates with multiple payment gateways to process various types of payments (credit cards, netbanking, e-wallets). ● API: Design and implement API to allow data exchange between different modules.

	● Staff resources

Service Name	**Feedbacks and Reviews**
Service ID	AP_03
Goal	Evaluate the quality of our guest services and identify areas for improvement.
Preconditions	● Guest has checked out.
Postconditions	● Feedback recorded in the system. ● Responses to feedback if necessary.
Assumptions	● Our staff ask for feedback from guests. ● The guest has checked out and is willing to provide feedback.If the guest has not checked out or is unwilling to provide feedback, the system will not request feedback. ● The guest is willing to provide contact information for follow-up on feedback. ● Guests are giving objective evaluations based on facts.
Inputs	● Feedback <document> : ○ Comments for the service <string>(good or bad, star-standard ○ Feeling about stay <string> ○ Suggestions <string[optional]>
Outputs	● Response for comments <string> eg:“Thanks for your feedback.”
Key steps	Step 1: Users check out. Step 2: Ask guests to write feedback or comments online. And our system will send a Questionnaire about their stay automatically. Step 3: Guests fulfil the Questionnaire file or give some comments for our hotel. Step 4: Response for the comments.
QoS factors	● Performance: ○ Response rate ≥ 80%: Response rate to feedback within 24 hours. ● Security: 99.99%
Delivery schedules	N/A
Payment schedules	N/A
Penalties	N/A
Value model	● Goal: ○ Collect the feedback to find out which parts we can improve to give better experiences for guests when they stay.

	○ Increase bookings and customer satisfaction in the future based on the feedback. ● Objective functions: ○ Maximize customer satisfaction rating. ○ Minimize booking cancellation rate in the future. ● Plan: ○ Send a link of a file which contains stay satisfaction to guests to fulfil. ○ Reform or improve poor services.
Resources	● Database System: A database is needed to store customer reviews and feedback. ● Back-end service: php ● Front-end interface: Design a user-friendly interface to allow customers to submit reviews conveniently. ● Review Management System: A management interface that allows administrators to view, review and manage customer reviews. Some analytical tools can be integrated to help understand customer feedback data. ● API: Design and implement API to allow data exchange between different modules. In this way, the rating system can be seamlessly integrated with hotel reservation systems, customer service systems, etc. ● Notification system: After customers complete check-in or check-out, remind customers to submit reviews through email or in-app notifications. ● Staff resources :Staff for replying to some comments.

Service provider 3:Local Transportation Services (LTS)

Service Name	**Ride Booking**
Service ID	RB_01
Goal	To allow users to book cabs, flight, train and bus tickets.
Preconditions	● Users must have valid login credentials. ● A valid source location and destination must be provided. ● Specific departure time must be entered. ● Affiliation of the Traffic service provider with cab, train, flight and bus providers for booking. ● Enablement of online wallet or other e-payment options.
Postconditions	● Estimation of trip duration, that is, time of arrival based on departure. ● Best route and mode of transport recommendation based on booking cost comparison, road closures and earliest arrival time. ● Confirmation of the chosen mode of transport.

	● Online payment and ticket booking confirmation. ● Saving of client information in the system.
Assumptions	● The Traffic Service Provider has real-time access to GPS for location tracking, traffic intensity prediction and closures. ● List of third-party ride booking applications such as Uber. ● Users' devices have GPS enabled. ● Users have access to the internet, online payment options and can interact with the booking system online.
Inputs	● Current location <string>: Source from which trip is to be planned i.e. current location of the user or a location entered by the user. ● Destination<string>: The final location to be travelled to. ● Date<date>: Desired date for travelling. ● Time<time>: Desired time for travelling. ● First name<string> : User's first name. ● Last name<string>: User's last name. ● Email<string>: User's email address for logging in. ● Phone number<string>: User's contact details. ● Payment type<string>: Whether the user wants to pay using netbanking or an e-wallet. ● Payment Details <string, date, integer>: ○ Credit card number <string> ○ Cardholder Name <string> ○ Expiration Date <date> ○ CVV <integer> ○ Billing address <string>
Outputs	● Booking Confirmation <document>: ○ Current location <string> ○ Destination<string> ○ Ride Arrival time<datetime>: Time for arrival of the ride booked ○ Booking Reference Number <string>: Unique identifier for the booking. ● Booking Details <document>: Description of the booked mode of transport, ticket number or vehicle number, arrival and departure time, contact details,terms and conditions. ● Payment receipt <document>: ○ Payment amount (float) ○ Payment method (string) ○ Reference number (string): Unique identifier for the booking payment.
Key steps	Step 1: User logs in the Traffic Service Provider with their account. Step 2: User enters current location, destination and time for departure.

	Step 3: The system displays various available transport options based on the entered trip details.. Step 4: The user selects their preferred mode of transport (e.g., cab, train, bus, flight). Step 5: The user enters payment details to complete the booking. Step 6: The system processes the payment through the chosen payment gateway. Step 7: Upon successful payment, the system confirms the booking and generates a booking reference number. Step 8: The system sends a booking confirmation notification to the user via email, SMS, or app notification. Step 9: The user can track the real-time status of their ride.
QoS factors	● Response Time ≤ 2 seconds: The time between the query sent by the user and the result returned. ● Booking Confirmation Time ≤ 5 seconds: Send confirmation booking request to return result. ● Uptime ≥ 99.99%: The Uptime for the service is quite stable.
Delivery schedules	● <Ride Booking Confirmation Time, ≤ 5 seconds>: A confirmation notification should be sent within 5 seconds of a successful booking, including ride details and payment receipt.
Payment schedules	● <Payment processing time, ≤ 1 minute >: Once the user has requested the ride, the payment must be completed within 1 minute, and the request will be automatically cancelled if the timeout occurs. ● <Payment response, ≤ 5 seconds>: After the user makes a payment, the system processes and generates a payment receipt within 5 seconds of confirmation.
Penalties	● Cancellation penalty: The user needs to cancel the booking within 1 minute of the request otherwise a penalty for full payment is applied. ● Waiting charges: The user needs to pay an extra 2$ per minute after the first 2 minutes of ride arrival, for waiting.
Value model	● Goal: ○ Provide a simplified and satisfactory ride booking experience. ○ Provide customers with multiple options for transport providers. ● Objective functions: (elaborate the objective the functions) ○ Minimise Booking time. ○ Minimise Travel cost. ○ Minimise waiting time. ○ Minimise Travel time. ○ Maximise ride availability for booking.

	○ Minimise Cancellation rate ○ Minimise Payment Processing Time. ● Plan: ○ Provide route and transport selection. ○ Optimise the online booking process to reduce user booking time ○ 24/7 customer support to resolve scheduled issues.
Resources	● Ride Booking Application: A web and mobile application where users can log in, enter trip details, and complete the booking process. ● User Authentication System: Handles user login and registration, ensuring secure access to the booking system. ● Payment Gateway Integration: Integrates with multiple payment gateways to process various types of payments (credit cards, netbanking, e-wallets). ● GPS and Mapping Software: Provides real-time location tracking, route planning, and map display functionalities. ● Database System (DBMS): Manages user data, booking details, payment information, and ride history. ● Notification System: Sends booking confirmations, reminders, and real-time updates via email, SMS, or push notifications.

Service Name	**Route Planning**
Service ID	RP_02
Goal	Prediction of best route based on shortest arrival time, mode of transport selected, traffic closures and congestion.
Preconditions	● A valid source location and destination or choose current location as source. ● A specific time for departure.
Postconditions	● Estimation of best route based on time, traffic and mode of transport. ● Estimation of time duration from a particular source to a destination. ● Live navigation using arrows or voice systems. ● Recommendation for nearby restaurants, petrol pumps, stores and so on.
Assumptions	● The Traffic Service Provider has real-time access to GPS for location tracking, traffic intensity prediction and closures. ● Users have access to the internet. ● GPS service is enabled for users for location accessibility.
Inputs	● Current location <string>: Source from which trip is to be planned i.e. current location of the user or a location entered by the user. ● Destination<string>: The final location to be travelled to.

	● Departure Time<datetime>: Desired day and time for travelling. ● Travel mode<string>: Preferred mode of transport for travelling. ● Pit stops<string>(optional): Stops on the route such as washrooms, petrol pumps, restaurants, cafes and hotels.
Outputs	● Arrival<time>: Time of arrival at a particular destination. ● Distance<string>: Trip distance based on source and destination. ● Trip Duration<datetime>: The time taken to cover the specific route. ● Best Route<string>: Shortest route with the least arrival time based on entered mode of transport. ● Amenities list<array>: A list of washrooms, petrol pumps, restaurants, cafes and hotels.
Key steps	Step 1: Users log in to the Traffic Service Provider with their account. Step 2: User enters current location, pit stops (if any) and destination. Step 3: The system processes the entered trip details using real-time data to determine the best possible routes. Step 4: The system evaluates the potential routes based on criteria such as travel time, distance, traffic congestion, and available amenities. Step 5: The system displays the recommended routes to the user, highlighting the best option based on the evaluation. Step 6: The user selects their preferred route from the recommended options. Step 7: The system provides detailed navigation instructions for the selected route, including turn-by-turn directions, estimated arrival time, and traffic updates. Step 8: The system continuously monitors the selected route for any changes in traffic conditions, road closures, or other disruptions and provides real-time updates to the user.
QoS factors	● Response Time ≤ 2 seconds: The time between the query sent by the user and the result returned. ● Uptime ≥ 99.99%: The Uptime for the service is quite stable. ● Accuracy≥99%: Result and condition prediction is highly accurate.
Delivery schedules	N/A
Payment schedules	N/A
Penalties	N/A
Value model	● Goal: ○ Provide the best travel route based on user requirement. ○ Provide a time estimation based on traffic conditions. ● Objective functions: ○ Minimise Travel time. ○ Minimise travel distance.

	○ Maximise route safety. ○ Maximise convenience of amenities. ● Plan: ○ Provide route and transport selection. ○ Optimise the online booking process to reduce user booking time ○ 24/7 customer support to resolve scheduled issues.
Resources	● GPS and Mapping Software: Provides real-time location tracking, route planning, and map display functionalities. ● User Authentication System: Handles user login and registration.

Service Name	**Real-time Traffic Information**
Service ID	RTI_03
Goal	Live information for road closures, high/medium/low traffic congestion and estimation of travel duration.
Preconditions	● Source and destination location to be entered by the user.
Postconditions	● Live information about traffic congestion, road closures and diversions.
Assumptions	● The Traffic Service Provider has real-time access to GPS for location tracking, traffic intensity prediction and road closures. ● Users have internet access. ● Traffic information is updated via GPS at regular intervals. ● GPS turned on on user devices for live location tracking.
Inputs	● Current location <string>: Source from which trip is to be planned i.e. current location of the user or a location entered by the user. ● Destination<string>: The final location to be travelled to. ● Time<datetime>: The current time or the time for which the traffic details are to be viewed for.
Outputs	● Traffic Congestion<string>: Intensity of traffic at a particular route. ● Road closures<array>: List of routes closed or have construction going on.
Key steps	Step 1: Users log in in the Traffic Service Provider with their account. Step 2: User enters current location and destination. Step 3: The system processes the request by collecting real-time traffic data from various sources. Step 4: Users receive updates on traffic conditions such as traffic congestion, road closures and multiple routes. Step 5: Users receive navigation assistance based on real-time traffic data. Step 6: Users can look up another destination, quit the application or provide feedback.
QoS factors	● Response Time ≤ 2 seconds: The time between the query sent by the user and the result returned.

	● Uptime ≥ 99.99%: The Uptime for the service is quite stable. ● Accuracy≥99%: Result and condition prediction is highly accurate.
Delivery schedules	**N/A**
Payment schedules	**N/A**
Penalties	**N/A**
Value model	● Goal: ○ Provide users with accurate and up-to-date traffic information ● Objective functions: ○ Maximise accuracy of Traffic Information. ○ Minimise Response Time. ○ Maximise Coverage of Traffic Data across different regions ○ Minimise System Downtime ● Plan: ○ Integrate data from various sources into a centralised system. ○ Ensure continuous and reliable traffic data feeds. ○ Integrate traffic updates with navigation systems and third-party apps. ○ Implement redundancy and failover mechanisms to minimise downtime.
Resources	● GPS and Mapping Software: Provides real-time location tracking, route planning, and map display functionalities. ● User Authentication System: Handles user login and registration.

Service provider 4:Restaurants and Cafes (RC)

Service Name	**Table Reservations**
Service ID	TR_01
Goal	Efficiently manage table reservations to maximise seating capacity.
Preconditions	● Users must have an account with the restaurant reservation platform and must be logged in (optional). ● The selected restaurant must be open and tables must be available for reservation at the selected time and date.
Postconditions	● Reservation is confirmed, a unique booking confirmation number is generated and saved in the system. ● The reserved table is marked as unavailable in the system for the specified time slot. ● Customers are given the confirmation and reservation details.

Assumptions	● The availability status for all participating restaurants and cafes is regularly updated. ● All the information provided by the customer is complete and accurate (mainly the email address).
Inputs	● *RestaurantID<string>*: Unique identifier for the selected restaurant. ● *ResturantName<string> :* Name of restaurant or cafe where the reservation is being made. ● *ResturantLocation<string> :* Location of the restaurant or cafe (city, country). ● *TableNumber<integer>*: Table number selected by a customer. ● *ReservationDate<date> :* Preferred date of the reservation ● *ReservationTime<time> :* Preferred time of the reservation ● *NumberOfGuest<integer> :* Number of guests attending the reservation. ● *CustomerName<string> :* Name of customer who made reservation. ● *Email<string> :* Customer email address for contact. ● *PhoneNumber<string> :* Customer phone number for contact. ● *SpecialRequests<string> :* Any special request from a customer(like, highchair, table near window, some celebration occasion). ● *PaymentType<string> :* Type of payment(e.g., Master, Visa). ● *CardHolderName<string> :* Name in the card. ● *CardNumber<integer> :* Account card number. ● *ExpirationDate<date> :* Expiration date of card. ● *CVC<string> :* Credit card verification code.
Outputs	● *ReservationID<string>*: Unique identifier for the reservation. ● *ReservationDetails<string>*: Summary of the reservation, included date, time, number of people, and special request (if any). ● *Status<string>*: The status of the reservation (e.g., confirmed, pending, failed). ● *Message <string>*: Message for confirmation or cancellation of the reservation. ● *TotalCost <string>*: Total amount charged for the reservation, including any discounts or promotions. ● *PaymentReceipt <string>*: URL where the user can view or download the receipt for the reservation.
Key steps	Step 1: The user logs in to the restaurant or cafe booking system website with their account. Step 2: The user searches for a restaurant by name or filters by location, cuisine, or other preferences. Step 3: The user selects a restaurant from the search result.

	Step 4: The reservation system checks the table availability on that particular date and time. Step 5: If available, the system asks a user to confirm the reservation. Step 6: Users confirm the reservation. Step 7: The reservation system stores the reservation details and sends them to the customer. Step 8: The user gets a confirmation notification with all the reservation details.
QoS factors	● **Response Time**: ≤ 2 seconds. The time between the submission of the reservation request and the confirmation result. ● **Accuracy**: 100%. The reservation details must be accurate and match the customer's.
Delivery schedules	● **<ConfirmationTime, ≤ 1 minute>** : Confirmation of the reservation is sent immediately after booking. ● **<ReminderTime, 1 hour before>**: A reminder notification is sent 1 hour before the reservation time. ● **<CancellationNotificationTime, ≤ 1 minutes>**: A notification is sent immediately upon cancellation, confirming the cancellation.
Payment schedules	● **<PaymentTime, ≤ 1 minutes>**: If prepayment is required, the system should process and confirm the payment within a minute of the reservation request.
Penalties	● **Late cancellation fee: I**f the customer cancels their reservation less than twenty-four hours prior to the scheduled time, there can be a charge. ● **No-show fee**: If the client cancels their reservation without giving notice, there can be a penalty.
Value model	● **Goal:** ○ Improve or enhance customer satisfaction by efficiently managing table reservation of restaurants or cafes. ● **Objective Function**: ○ Increase the amount of reservations while reducing mistakes and maximizing table utilization. ● **Plan**: ○ Install a user friendly reservation system that communicates with real-time table availability and notifies users in a timely manner.
Resources	● Reservation System: Software and infrastructure for managing reservations. ● Customer support staff: Support customers with inquiries and complaints.

	● Database: System for storing reservation details and table availability.

Service Name	**Menu Recommendation**
Service ID	MR_02
Goal	To present the available menu in an organised and visually appealing way, helping customers easily choose dishes.
Preconditions	● Users must have an account with the restaurant reservation platform and must be logged in (optional). ● The user must select a restaurant or cafe.
Postconditions	● The user views the menu and can make a decision about which dishes they would like to order. ● Menu items are highlighted or marked as popular or chef's recommendations.
Assumptions	● The restaurant or cafe regularly updates their menu in the system including adding new items or some specials (lunch special, weekend special, etc) ● Users can make their own decision based on the menu presented without the need for personlized suggestions.
Inputs	● *RestaurantID <string>*: Unique identifier for the selected restaurant. ● *ResturantName<string> :* Name of restaurant or cafe where the reservation is being made. ● *ResturantLocation<string> :* Location of the restaurant or cafe (city, country). ● *MenuItems<List<string>>*: A list of available menu items at the restaurant.
Outputs	● *DisplayedMenu<List<string>>*: The menu is presented to the user, optionally highlighting popular or chef-recommended items.
Key steps	Step 1: The user logs in to the restaurant portal with their account . Step 2: The user searches for a restaurant by name or filters by location, cuisine, or other preferences. Step 3: The user selects a restaurant from the search results. Step 4: The user navigates to the “Menu” tab to view available dishes. Step 5: The system displays the entire menu, optionally highlighting popular or chef-recommended items. Step 6: The user reviews the menu and mentally notes or decides on their preferred items. Step 7: The system provides additional details about the selected dishes, such as ingredients, preparation time, and nutritional information (if available).

QoS factors	● **Response Time**: ≤ 2 seconds. The time between the user's request to view the menu and the display of the menu items. ● **Accuracy**: 100%. The menu displayed should be up-to-date and accurately reflect the restaurant's offerings. ● **Reliability**: 99.99% accuracy in recommendation.
Delivery schedules	● **<MenuDisplayTime, ≤ 2 seconds>**: The menu is displayed within 2 seconds of the user's request.
Payment schedules	N/A
Penalties	N/A
Value model	● **Goal**: ○ Provide customers an easy-to-use method for exploring the menu and choosing dishes ● **Objective Function**: ○ Improve the clarity and accessibility of displaying the menus while preserving the time taken to display it. ● **Plan**: ○ Implement a system for displaying the restaurant's menu in a simple to use layout, possibly highlighting popular or chef-recommended items.
Resources	● Menu Display System: Software that organizes and presents the menu items. ● Menu Database: Manage up-to-date menu information at each restaurant.

Service Name	**Online Ordering**
Service ID	OO_03
Goal	Provide users with a convenient way to place food and beverage orders online, facilitating both delivery and takeaway options.
Preconditions	● The restaurant or cafe must have an active online ordering system integrated with the platform. ● The user must be logged in to their account to place an order. ● The menu and ordering system must be up-to-date and functioning.
Postconditions	● The user successfully places an order for delivery or takeaway. ● The system processes the payment and confirms the order. ● The user receives a confirmation notification with details of the order and estimated pickup or delivery time.
Assumptions	● The restaurant or cafe maintains accurate and current menu information. ● The payment processing system is secure and operational. ● Users have access to the online ordering system and can complete their orders.

Inputs	● ResturantID<string>: Unique identifier for the selected restaurant. ● ResturantName<string> : Name of restaurant or cafe where the reservation is being made. ● Location<string> : Location of the restaurant or cafe (city, country). ● CustomerName<string> : Name of customer who made reservation. ● Email<string> : Customer email address for contact. ● PhoneNumber<string> : Customer phone number for contact. ● CustomerAddress<Address>: The full address of the customer, including street, suburb, city, state, postal code, and country (if order is for delivery). ○ Country<string> ○ Street<string> ○ Suburb<string> ○ State<string> ○ PostalCode<integer> ● OrderDetailsList<List<string>>: ○ ItemId<string> : Unique identifier for the selected item. ○ ItemName<string>: Name of an item selected. ○ Quantity<integer>: Total quantities of selected item. ○ Notes<string>: Any special instruction from a customer. ● PaymentType<string> : Type of payment(e.g., Master, Visa). ● CardHolderName<string> : Name in the card. ● CardNumber<integer> : Account card number. ● ExpirationDate<date> : Expiration date of card. ● CVC<string> : Credit card verification code.
Outputs	● OrderConfirmationID<string>: Unique identifier for the order confirmation. ● OrderConfirmationDetails<string>: Confirmation of the order including a unique confirmation number, order summary, and estimated pickup or delivery time.
Key steps	Step 1: The user logs in to the ordering portal with their account. Step 2: The user searches for a restaurant by name or filters by location, cuisine, or other preferences. Step 5: The system displays the menu items. Step 6: The user selects items for delivery or takeaway and adds them to their order. Step 7: The user proceeds to checkout, reviews the order, and provides any special instructions (if needed).

	Step 8: The user enters payment details (if applicable) and confirms the order. Step 9: The system processes the payment and finalises the order. Step 10: The user receives an order confirmation with details and estimated pickup or delivery time.
QoS factors	● **Response Time: ≤ 5 seconds.** The time between the user's request to view the menu and the display of the menu items. ● **Order Processing Time: ≤ 10 seconds.** The time between order confirmation and processing. ● **Accuracy: 100%.** The order details and payment processing should be accurate and reflect the user's selections.
Delivery schedules	● **<MenuDisplayTime, ≤ 5 seconds>**: The menu is displayed within 5 seconds of the user's request.
Payment schedules	● **<PaymentTime, ≤ 10 minutes>**: The system should process and confirm the payment within 10 minutes of the order.
Penalties	● **Late Cancellation Fee**: Users are not allowed to cancel their dining orders after the confirmation for payment. And there is no charge before payment for the ordering.
Value model	● Goal ○ Provide users with convenient food ordering services to improve user satisfaction. ○ Increase restaurant orders and revenue. ● Objective functions ○ Maximise customer satisfaction rating. ○ Minimise booking and payment errors rate. ● Plan ○ Implement a friendly and interactive online ordering system. ○ Restaurant operation and payment are seamlessly connected to ensure a smooth ordering process. ○ Establish a sound notification system, covering order confirmation, receipt notification and user feedback.
Resources	● Online Ordering System: Manages order placement, updates, and payment processing. ● Menu Display System: Provides up-to-date menu information. ● Payment Processing System: Handles secure transactions.

Service provider 5: Outdoor Activity Centers (OAC)

Service Name	**Outdoor Activity Centre**
Service ID	OAC_01

Goal	Customers can book various outdoor activities based on their preferences and availability at the outdoor activity centre.
Preconditions	● Users must have an account with the Tourism Ecosystem and have logged in. ● Activity must be available for booking, such as time based, number of people limited.
Postconditions	● Booking is confirmed and saved in the system. ● User receives the confirmation with booking details.
Assumptions	● Event information is kept up to date. ● The user's booking information is complete and correct.
Inputs	● ActivityType<string>: User selected or searched activity type (e.g., hiking, cruise). ● GeographicPosition<string>: Geographic location of the activity. ● ActivityDate<date>: Desired date for the activity. ● NumberOfParticipants<integer>: Number of people participating in the event. ○ Adult<integer> ○ Children<integer> ○ Infants<integer> ● FirstName<string>: The first name of all participants. ● LastName<string>: The last name of all participants. ● Email<string>: Email address for contact. ● PhoneNumber <string>: Phone number for contact. ● ReceiveSMSUpdates<boolean>: whether the user wants to receive SMS updates about the booking. ● PromoCode<string>: promo code for discounts or offers. ● PaymentType<string>: Type of payment method (e.g., credit_card, paypal). ● CardholderName<string>: Name of the cardholder. ● CardNumber<string>: Credit card number. ● ExpirationDate<date>: Credit card expiration date. ● CVC<string>: Credit card verification code, typically 3 or 4 digits. ● Country<string>: Country of the billing address. ● ZipCode<string>: Postal code for billing address.
Outputs	● BookingConfirmationNumber<string>: Unique identifier for the booking. ● Status<string>: the status of the booking (e.g., confirmed, pending, failed). ● Message<string>: Description of additional details about booking status or errors. ● TotalCost<decimal>: Total amount charged for the booking, including any discounts or promotions.

	● Payment receipt <string>: URL where the user can view or download the receipt for the booking.
Key steps	Step1: User login in the Outdoor Activity Center with their account. Step2: Users select or search activities based on their preferences. Step3: The user selects the desired activity, date, and number of participants. Step4: User input contact details and payment information. Step5: User submitted orders. Step6: The service checks availability for the scheduled activities submitted by the user. Step7: The service returns a message to users about the status of this booking, such as successful or not.
QoS factors	● Response Time ≤ 2 seconds: The time between the query sent by the user and the result returned. ● User Satisfaction ≥ 90%: User satisfaction with the booking experience is high. ● Stability ≥ 99.99%: Stability of complete reservation service.
Delivery schedules	● <BookingConfirmationTime, ≤ 2 seconds>: A confirmation notification should be sent within 2 seconds of a successful booking. This notification should include booking details and payment receipt.
Payment schedules	● <PaymentTime, ≤ 15 minutes>: Payment must be completed within 15 minutes of order submission. If payment is not completed within this time limit, the order will be automatically cancelled.
Penalties	● Cancellation penalty: The user cancels the reservation within 48 hours before the event starts, minus 20% of the event payment.
Value model	● Goal ○ Provide convenient event booking experience for customers. ○ Increase bookings and customer satisfaction. ○ Increase the participation of event organisers and suppliers. ● Objective functions ○ Maximise booking conversion rate. ○ Maximise customer satisfaction rating. ○ Minimise booking cancellation rate. ● Plan ○ Provide comprehensive and accurate activity information, and users' reviews to help users make decisions. ○ Provide up-to-date activity information, reflect the latest activity status. ○ Simplify booking steps and form filling to optimise the online booking process to reduce user booking time.

	○ Phone 24/7 customer support to resolve scheduled issues.
Resources	● User Authentication System: Handles user login and registration. ● Database System (DBMS): Manages user data, booking details, payment information, and ride history. ● Payment Gateway: Integrates with multiple payment gateways to process various types of payments (credit cards, netbanking, e-wallets). ● Notification System: Sends booking confirmations, reminders, and real-time updates via email, SMS, or push notifications.

Service Name	**Equipment Lease**
Service ID	EL_02
Goal	Providing users with convenient equipment rental ensures that users can easily obtain high-quality and well-maintained equipment.
Preconditions	● Equipment must be available for lease. ● Users must have an account of the Tourism Ecosystem, and login in. ● Rental duration and equipment type must be specified.
Postconditions	● Lease agreement is generated ● Equipment is reserved and prepared for pickup or delivery ● Payment is processed and receipt is issued
Assumptions	● Equipment inventory is up-to-date and accurately reflects availability ● Users are familiar with basic terms and conditions of equipment rental ● The payment system is operational and secure. ● The customer has an accurate self-perception of their own health and physical condition.
Inputs	● Feature<string>: Users expect to rent equipment types, such as tents, bicycles, skis. ● UserType<string>: User categories, such as male, female and kids, if relevant to the rental equipment. ● Size<string>: Size requirements for clothing or equipment, such as S/M/L for clothing or specific dimensions for equipment. Optional if not relevant to the rental equipment. ● Clothing size and equipment size, such as clothing size S/M/L, and equipment size such as length, bicycle size. It is an optional if related rental equipment. ● Colour<string>: Preferred colour for the equipment, such as black or white. Optional if not relevant to the rental equipment. ● LeaseTerm<integer>: The period from the start time to the end time of the rental equipment. ● StartDate<dateTime>: The date of the lease starts.

	● EndDate<dateTime>: The deadline for device return. ● PickupWay<string>: Pickup on site or delivery. ● FirstName<string>: The first name of all participants. ● LastName<string>: The last name of all participants. ● Email<string>: Email address for contact. ● PhoneNumber <string>: Phone number for contact. ● ReceiveSMSUpdates<boolean>: whether the user wants to receive SMS updates about the booking. ● ShippingAddress<string>: Where do users require the equipment to be sent. ● PaymentType<string>: Type of payment method (e.g., credit_card, paypal). ● CardholderName<string>: Name of the cardholder. ● CardNumber<string>: Credit card number. ● ExpirationDate<date>: Credit card expiration date. ● CVC<string>: Credit card verification code, typically 3 or 4 digits. ● Country<string>: Country of the billing address. ● ZipCode<string>: Postal code for billing address.
Outputs	● BookingConfirmationNumber<string>: Unique identifier for the rental equipment. ● Status<string>: The status of the booking (e.g., confirmed, pending, failed). ● Message<string>: Description of additional details about booking status or errors. ● RentalAgreement<document>: According to user information to generate the rental contract. ● TotalCost<decimal>: The total cost of the rental, including delivery fee. ● PaymentReceipt <string>: URL where the user can view or download the receipt. ● Instruction<document>: Instructions and safety guidelines for the correct and safe use of the equipment.
Key steps	Step 1: Check availability for equipment. Step 2: Generate rental agreement, including equipment details and instruction of use and safety. Step 3: Process payments to confirm payment amount is correct, and send the payment receipt to the customer. Step4: Schedule equipment pickup or delivery. Step5: Manage follow-up services. Process request of equipment return, repair and replacement, as well as update inventory status and adjust inventory records.
QoS factors	● Availability ≥ 98%: The equipment must be provided as scheduled.

	● Response Time ≤ 2 seconds: The time between the query sent by the user and the result returned. ● Confirm time ≤ 15 minutes: Users submit reservations until they receive confirmation. ● Accuracy ≥ 99%: Correctness of equipment details and lease terms.
Delivery schedules	● <Pickup/delivery time, ≤ 24 hours>: After the rental goes to the person, the user can choose to pick up the item immediately or arrange the delivery. Pick-up and delivery will be arranged within 24 hours of confirmation. ● <Equipment return, end of lease time>: Equipment needs to be returned before the end of the lease. Late return will incur additional costs.
Payment schedules	● <PaymentTime, ≤ 15 minutes>: Payment must be completed within 15 minutes of order submission. If payment is not completed within this time limit, the order will be automatically cancelled.
Penalties	● Cancellation penalty: The user cancels the reservation within 48 hours before the rental starts, 20% of the rental fee will be deducted. ● Late return penalty: The user does not return the equipment on time, 20% of the equipment rental fee will be charged as a late return penalty per day. ● Loss/damage penalty: Equipment is lost or damaged during the lease and the user is required to pay for replacement or repair.
Value model	● Goal: ○ Provide consumers with high-quality and professional outdoor equipment. ○ Ensure the safety and availability of equipment for customers. ○ Enhance customer satisfaction and loyalty. ● Objective functions: ○ Maximise booking conversion rate. ○ Minimise booking cancellation rate. ○ User repeat booking rate. ● Plan: ○ Provide detailed equipment description and customer reviews to assist users in making informed decisions. ○ Optimise the online booking process to reduce the time required for users to complete booking. ○ Ensure accurate and comprehensive equipment specifications, including size, material, and the features, are provided to users.

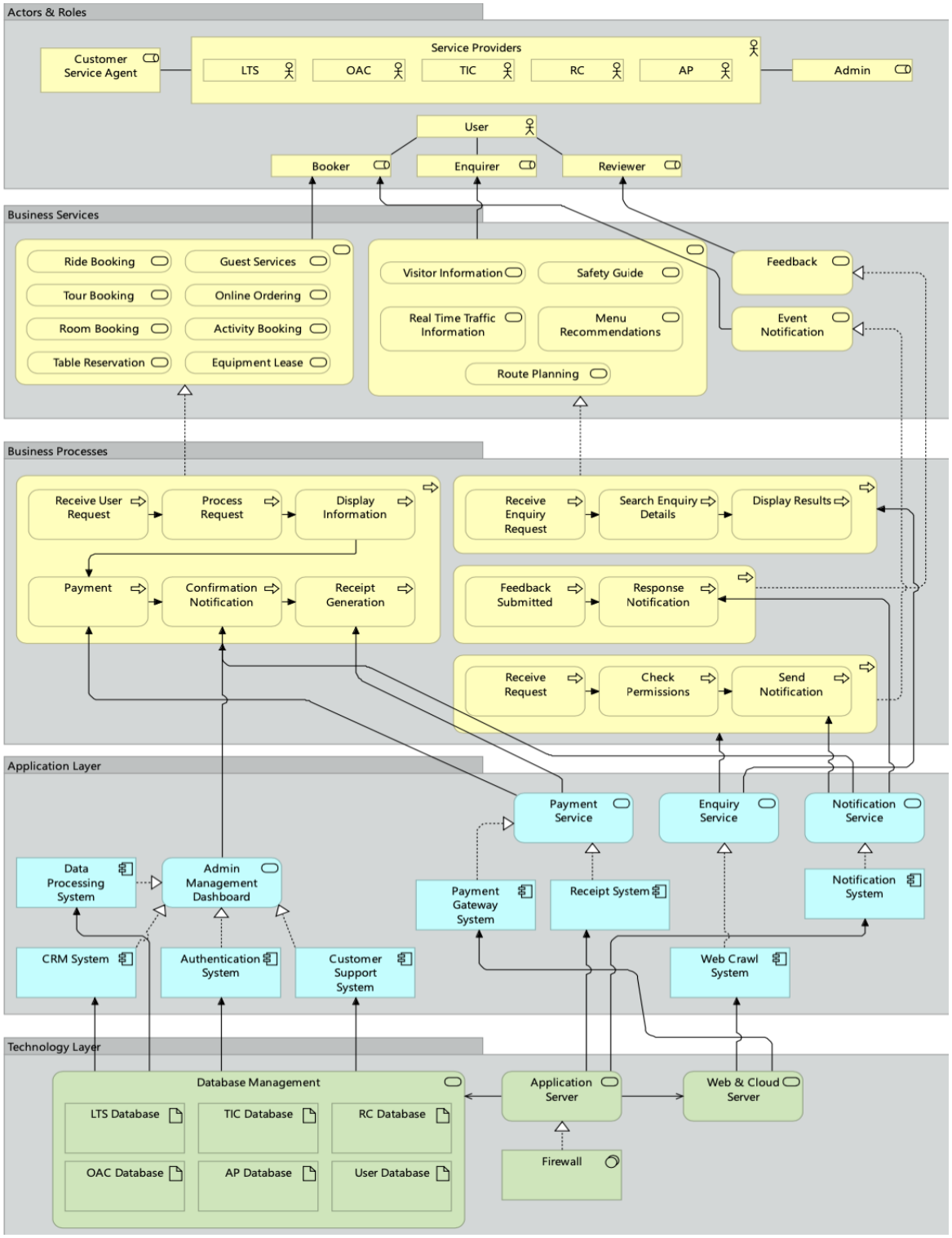
	○ Utilise user data to make personalised recommendations and offers, aiming to improve conversion rates and enhance user experience.
Resources	● User Authentication System: Handles user login and registration. ● Database System (DBMS): Manages user data, booking details, payment information, and ride history. ● Payment Gateway Integration: Integrates with multiple payment gateways to process various types of payments (credit cards, netbanking, e-wallets). ● Notification System: Sends booking confirmations, reminders, and real-time updates via email, SMS, or push notifications. ● Equipment Storage Database: Record and store device details such as device type, size, colour, lease status, maintenance records, usage guidelines, inventory quantity, and location.

Service Name	**Safety Guide**
Service ID	SG_03
Goal	Provide detailed safety guidelines and emergency guidance to help ensure the safety of participants during outdoor activities.
Preconditions	● There are safety guidelines and emergency measures for the specific activities provided by the system. ● Safety guidance in accordance with legal requirements and industry standards.
Postconditions	● Emergency contact information is provided. ● Customers receive detailed safety guidance.
Assumptions	● Customers have a comprehensive knowledge and understanding of the safety guidance. ● Safety guidance and emergency measures are activity-specific and in line with current, effective and up-to-date content.
Inputs	● ActivityType<string>：Types of outdoor activities, such as rock climbing, wild animals etc. ● GeographicPosition<string>: Geographic location of the activity.
Outputs	● Guidance on safety <document>: Detailed documentation of safety instructions and emergency measures for the activity. ● Emergency contact information <document>: Emergency numbers and local emergency services relevant to the activity location. ● Security briefing <document>: Schedule information about the time, place and participants of the security briefing, if applicable, such as high risk area.
Key steps	Step 1: Receive activity type and geographic position. Step 2: Prepare security guidance to the activity type and location.

	Step 3: Provide security guidelines to users, ensuring they understand the necessary precautions and procedures. Step 4: Provide emergency contact information to users. Step 5: Arrange and conduct a safety briefing (If applicable).
QoS factors	● Accuracy = 100%: Safety guidance and emergency measures provided must be accurate and up to date. ● Query response time ≤ 2 seconds: the time from the request to display the result.
Delivery schedules	N/A
Payment schedules	N/A
Penalties	N/A
Value model	● Goal: ○ To popularise the safety knowledge and precautions of outdoor activities for users. ○ Provide effective emergency measures for users. ○ Reduce the risk of outdoor activities for users. ● Objective functions: ○ Emergency response time. ○ Accident rate. ○ User complaint rate. ● Plan: ○ Provide professional, effective and detailed safety guidelines. ○ Ensure that applicable emergency measures and guidelines are available for provided activities. ○ The Hazard program provides specialised training (Pay item). ○ Ensure the validity of emergency contact information. ○ Identify the inadequacies of safety guidelines through data analysis and make targeted improvements.
Resources	● Database System (DBMS): This database includes information such as activity type, geographic location, safety guidelines, emergency contact information, risk levels, and safety training courses.

TASK B. Enterprise Architecture (EA) Design and Analysis

Enterprise Architecture of the Tourism Ecosystem



Description of the Tourism Ecosystem EA

- **Goal:**

Transform the way tourists experience travel by leveraging advanced technologies to create seamless, personalised, and enriching journeys.Develop a Comprehensive Tourism Ecosystem, Integrate AI and mobile applications to connect various service providers, offering a unified platform that caters to all aspects of the tourism experience.

Strategic Goals:

- Enhance User Experience through Personalization
- Create a Seamless and Integrated Platform
- Facilitate Efficient Digital Transactions
- Empower Service Providers
- Foster Collaboration and Innovation
- Enhance Customer Satisfaction and Engagement

- **Assumption:**

  The assumptions layer outlines the basic premises and environmental factors considered in the design and implementation of the tourism ecosystem. These assumptions are critical to align expectations and plan for potential risks or challenges.
  - Adoption of advanced technologies:
    - Artificial intelligence and mobile technologies are mature enough to provide reliable and scalable solutions.
    - Necessary technology infrastructure (smartphones, internet connectivity) is available to users and service providers.
  - Users' preference for digital solutions:
    - Tourists prefer to use digital platforms to plan and manage their travel experiences.
    - Users are willing to share personal data in exchange for personalized services.
  - Market demand for a unified platform:
    - There is a huge market demand for an integrated tourism ecosystem that simplifies travel planning.
  - Compliance with legal standards:
    - Platforms comply with all data protection regulations (e.g. GDPR, CCPA).
    - Digital transactions are allowed to be conducted securely within the target market.
  - Supportive policies:
    - Government and regulators support technological innovation in the tourism industry.
  - Service provider participation:
    - Service providers are willing to cooperate and update their information on the platform.
  - Scalability and performance:
    - The underlying technology can scale to meet growing user demand without compromising performance.

- Sustainable revenue model:
  - The platform can generate revenue through commissions, subscriptions, or advertising without compromising user experience.
- Investment and funding:
  - There is sufficient funding to support development, maintenance, and expansion efforts

- **Business Layer：** The system has five service providers: Local Transportation Services(LTS), Outdoor Activity Centers(OAC), Tourism Information Center(TIC), Restaurants and Cafes(RC) and Accommodation Provider(AP).
- **Business Processes:** These are the operational workflows that handle user requests and process the necessary information:
  - Receive User Request: The process starts with receiving requests from users.
  - Process Request: The platform processes these requests, which may involve payment, confirmation, and receipt generation.
  - Display Information: After processing, relevant information is displayed to the user.
  - Receive Enquiry Request and Search Enquiry Details: These processes handle specific enquiries made by users, searching and displaying relevant details.
  - Feedback Submission and Response Notification: This process handles user feedback and sends notifications in response.
  - Receive Request, Check Permissions, and Send Notification: This set of processes handles incoming requests, verifies permissions, and sends out the appropriate notifications.

- **Application Layer：** The application layer serves as the foundational infrastructure that supports and enables the various business processes and services. It consists of several systems that handle specific functionalities required for the smooth operation of the platform.
  - There are eight systems — Data Processing System, CRM System (Customer Relationship Management), Authentication System, Customer Support System, Payment Gateway System, Receipt System, Web Crawl System, Notification System.Each of these systems in the application layer plays a crucial role in supporting the overall functionality of the platform. They work together to provide a seamless user experience, ensuring that business services are delivered efficiently and securely.

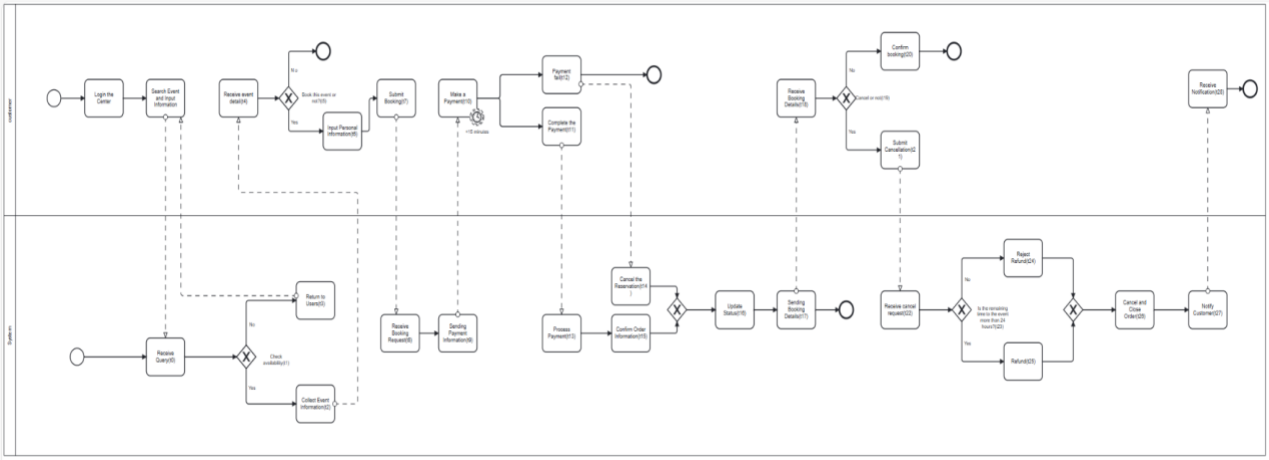
- **Technology Layer:** The Technology Layer in this tourism ecosystem's Enterprise Architecture represents the foundational infrastructure that supports all other layers of the system. This layer is responsible for ensuring the secure, reliable, and efficient

operation of the ecosystem's services. It consists of Database Management, Application Server, Web & Cloud Server and Firewall.

TASK C. Business Process Design and Analysis

Service 1: Tour Booking

I. Business Process Model and Notation



II. Annotation

Task Name	Action Description	Related Rules
Login the Center	Performs(Customer, Login, System)	
Search Event and Input Information	Performs(Customer, Search Event)	
Receive Query	Performs(System, Receive query)	
Collect Event Information	Performs(System, collect information)	
Receive event detail	Performs(Customer, receive)	
Input Personal Information	Performs(Customer, submit personal information)	
Submit Booking	Performs(Customer, Submit)	
Receive Booking Request	Performs(System, Receive Request)	
Sending Payment	Performs(System, Payment	

Information	Information)	
Make a Payment	Performs(Customer, Make a Payment)	
Complete the Payment	Performs(Customer, Payment Success)	
Payment fail	Performs(Customer, Payment Failure)	∧Knows(System, Payment Timeout, 15Minutes)
Process Payment	Performs(System, Process Payment)	
Confirm Order Information	Performs(System, Confirm)	
Update Status	Performs(System, Update Status)	
Sending Booking Details	Performs(System, Sending Details)	
Receive Booking Details	Performs(Customer, Receive Details)	
Confirm booking	Performs(Customer, Confirm)	
Submit Cancellation	Performs(Customer, Cancellation)	
Receive cancel request	Performs(System, Receive request)	
Reject Refund	Performs(System, Reject Refund)	∧Knows(System, Time, More than 24 hours)
Refund	Performs(System, Refund)	
Cancel and Close Order	Performs(System, Close Order)	
Notify Customer	Performs(System, Notification)	

III. Object of Interest

Receive Query(rq)	Return to Users(ru)	Collect Event Information(cei)
Receive Booking Request(rbr)	Sending Payment Information(spi)	Process Payment(pp)

Cancel the Reservation		

IV. Immediate Effects

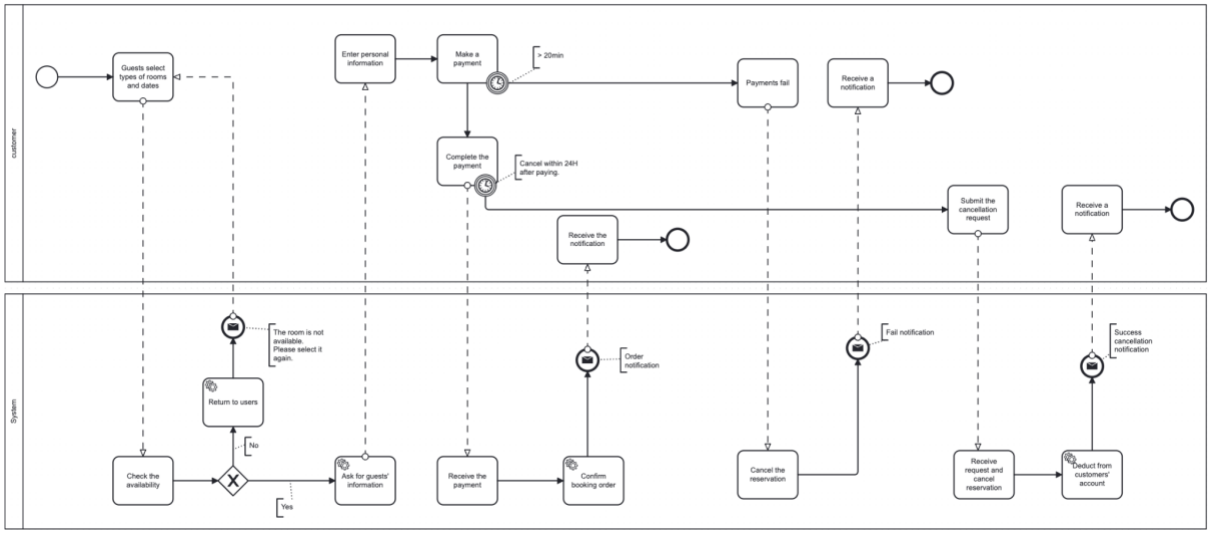
Activity Name	Immediate effect (plain English)	Immediate effect (FOL)
t1: Check availability	After receiving the request, the system checks whether the selected event is available.	TriggerError(tr)∧ SendErrorMessage(sem)
t29:Check If Search Again	The consumer decides whether to search again and if "YES", then search again; If "NO", the session ends, after reviewing the error message.	SearchAgain(sr) v (¬SearchAgain(sr)∧EndSession(es))
t7:Send Order Details	Order details include a unique confirmation number, order summary, price, and estimated pickup or delivery time after the system enters the checkout process.	ProcessCheckout(pc)∧ SendOrderDetails(sod)
t21:Submit Cancellation	The customer decides "YES" to cancel after reviewing the order details and therefore submits a cancellation request.	ReviewOrderDetails(rod)∧CheckCancel(cc)∧ SubmitCancellation(sc)
t10:Submit Payment	The customer decides "NO" to cancel after reviewing the order details and therefore submits a payment request.	ReviewOrderDetails(rod)∧¬CheckCancel(cc)∧SubmitPayment(sp)
t14:PrepareOrder	The system enters the process of preparing the food on the order after the payment is successful.	PaymentSuccessful(ps)∧ PrepareOrder(po)
t19:PaymentFailure(pf)	The system takes more than 10 minutes (> 10 minutes) to process the payment, and the payment fails.	ProcessPayment(pp)∧ PaymentTimeout(pt)∧ PaymentFailure(pf)
t15:UpdateStatus(us)-Post-ProcessPayment	The system updates the status after an order has entered the preparation process or payment has been closed due to a payment failure.	(PrepareOrder(po) vClosePayment(cp))∧UpdateStatus(us)
t24:UpdateStatus(us)-Post-Cancellation	The system updates the status after cancelling and closing order	CancelCloseOrder(cco)∧UpdateStatus(us)

V. Cumulative Effects

No.	Activity Name	Scenarios	Cumulative effect (FOL)
1	Booking Event	t20:<<t0,t1,t2,t4,t5,t6,t7,t8,t9,t10,t11,t13,t15,t16,t17,t18,t19>,t20,{<t1,t3>,<t18,t19>}>	Receive Query(rq)^ Collect Event Information(cei)^ Receive event detail(red)^ Input Personal Information(ipi)^ Submit Booking(sb)^ Receive Booking Request(rbr)^ Sending Payment Information(spi)^ Make a Payment(mp)^ Complete the Payment(cp)^ Process Payment(pp)^ Confirm Order Information(coi)^ Update Status(us)^ Sending Booking Details(sbd)^ Receive Booking Details(rbd)^ Confirm booking(cb)
2	Cancel and Close Order	t26:<<t0,t1,t2,t4,t5,t6,t7,t8,t9,t10,t11,t13,t15,t16,t17,t18,t19,t21,t22,t23,t25>,t26,{<t1,t3>,<t18,t19>}>	Receive Query(rq)^ Collect Event Information(cei)^ Receive event detail(red)^ Input Personal Information(ipi)^ Submit Booking(sb)^ Receive Booking Request(rbr)^ Sending Payment Information(spi)^ Make a Payment(mp)^ Complete the Payment(cp)^ Process Payment(pp)^ Confirm Order Information(coi)^ Update Status(us)^ Sending Booking Details(sbd)^ Receive Booking Details(rbd)^ Submit Cancellation(sc)^ Receive cancel request(rcr)^ Refund(r)^ Cancel and Close Order(cc)
3	Check Information	t5:<<t0,t1,t2,t4>,t5,{<t0,t1>}>	Receive Query(rq)^ Collect Event Information(ced)^ Receive event detail(red)

Service 2: Room Booking

I. Business Process Model and Notation



II. Annotation

Task	Perform(Roles, Task)	Conditions
Select Room	Performs(Customer, Guests select types of rooms and dates)	
Check Availability	Performs(System, Check the availability)	
Return to users	Performs(System, Return to users)	If (Room is not available)
Ask for guests' information	Performs(System, Ask for guests' information)	If (Room is available)
Enter personal information	Performs(Customer, Enter personal information)	
Make a Payment	Performs(Customer, Make a payment)	
Complete the payment	Performs(Customer, Complete the payment)	
Receive the payment	Performs(System, Receive the payment)	
Confirm booking order	Performs(System, Confirm booking order)	

Task	Perform(Roles, Task)	Conditions
Receive the notification	Performs(Customer, Receive the notification)	
Payment fails	Performs(System, Payment fails)	If (Payment is not completed within 20 min)
Cancel the reservation	Performs(Customer, Cancel the reservation)	
Receive requests and cancel a reservation	Performs(System, Receive request and cancel reservation)	
Deduct from customers' account	Performs(System, Deduct from customers' account)	
Submit the cancellation request	Performs(Customer, Submit the cancellation request)	If (Cancel within 24H after paying)
Receive a notification	Performs(Customer, Receive a notification)	
Success cancellation notification	Performs(System, Success cancellation notification)	If (Cancellation request approved)
Fail notification	Performs(System, Fail notification)	If (Room is failed to be booked)

III. Object of Interest

Available(a)	CheckAvailability(ca)
ReturnUser(ru)	Available(rt, d)
SelectDate(d)	

IV. Immediate Effects

Activity Name	Immediate Effect (Plain English)	Immediate Effect (FOL)
t1: Guests select types of rooms and dates	Guests select the type of room and dates they want to book. The system immediately checks the availability of the room chosen	SelectRoom(r) ∧ SelectDate(d) ∧ CheckAvailability(ca)

t2: Check the availability	After receiving the guest's request, the system checks whether the selected room is available.	CheckAvailability(ca) ∧ (Available(a) ∨ ¬Available(a))
t3: Return to users	If the selected room is unavailable, the system immediately prompts the user to select another one.	¬Available(a) ∧ ReturnUser(ru)
t4: Ask for guests' information	Once the room is confirmed available, the system requests the guest to enter their personal information.	Available(rt, d) ∧ RequestInfo(PersonalInfo)
t5: Enter personal information	The guest provides personal information, which is received by the system.	Provide(PersonalInfo) ∧ Received(System, PersonalInfo)
t6: Make a payment	The guest makes a payment, and the system starts a 20-minute timer for completing the payment process.	PaymentInitiated(p) ∧ StartTimer(20min)
t7: Complete the payment	The system processes and completes the payment.	PaymentProcessed(p) ∧ PaymentCompleted
t8: Payments fail	If the payment is not completed within 20 minutes, the system fails the payment.	¬PaymentCompleted ∧ TimeElapsed(20min) ∧ PaymentFailed
t9: Receive the notification	The system sends a notification of booking status (confirmation or failure) to the guest.	SendNotification(System, BookingStatus) ∧ ReceivedNotification(Guest, BookingStatus)
t10: Confirm booking order	The system confirms the booking order and sends an order confirmation to the guest.	ConfirmBookingOrder(rt, d) ∧ SendConfirmation(System, Guest)
t11: Cancel the reservation	The guest submits a request to cancel the reservation within the allowed period.	CancelRequest(Guest, Booking)
t12: Receive request and cancel a reservation	The system processes the cancellation request and cancels the reservation.	CancellationProcessed(System, Booking)
t13: Deduct from	The system deducts the	DeductFee(System,

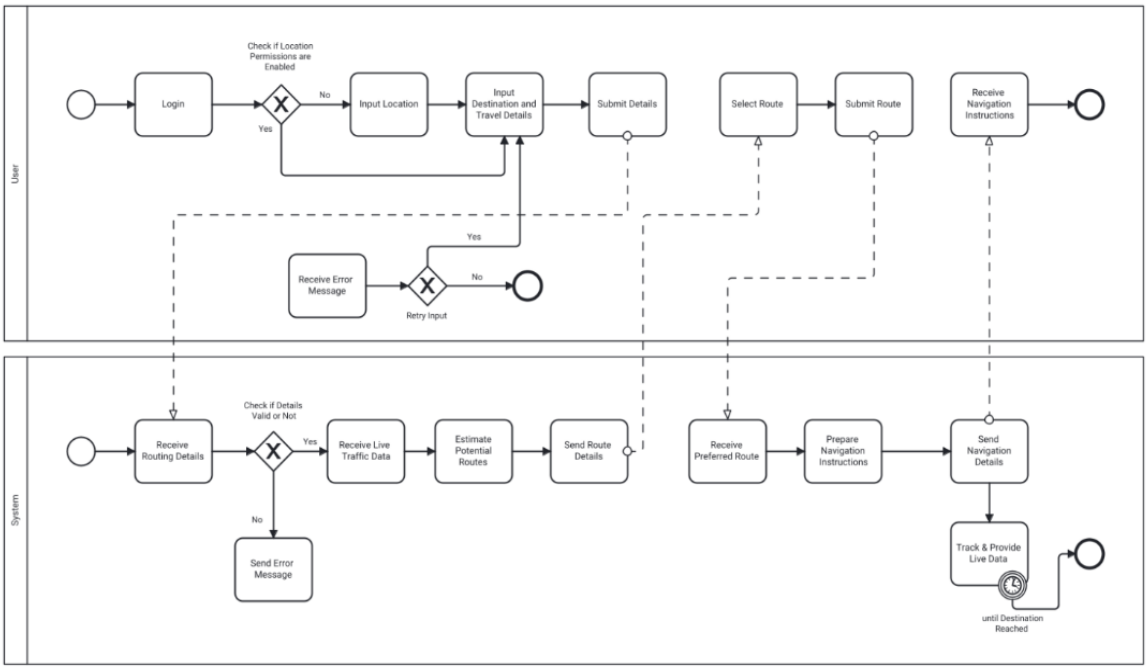
customers' account	cancellation fee from the guest's account.	GuestAccount, Fee)
t14: Submit the cancellation request	The guest submits a cancellation request.	SubmitCancellationRequest(Guest)
t15: Receive a notification	The guest receives a notification about the status of their cancellation request.	ReceivedNotification(Guest, CancellationStatus)

V. Cumulative Effects

No.	Activity Name	Scenarios	Effect (FOL)
1	t3: Return to users	t1->t2->t3	¬Available(a) ∧ReturnUser(ru)
2	t4: Ask for guests' information	t1->t2->t4	Available(rt, d) ∧ RequestInfo(PersonalInfo)
3	t7: Complete the payment	t1->t2->t4->t5->t6->t7	PaymentProcessed(p within 20 mins) ∧ PaymentCompleted
4	t8: Payments fail	t1->t2->t4->t5->t6->t8	¬PaymentCompleted ∧ TimeElapsed(20min) ∧ PaymentFailed
5	t9: Receive the notification	t1->t2->t4->t5->t6->t7->t9	PaymentProcessed(p within 20 mins) ∧ PaymentCompleted ∧ ReceivePayment∧ ConfirmOrder
6	t14: Submit the cancellation request	t1->t2->t4->t5->t6->t7->t14	PaymentProcessed(p within 20 mins) ∧ PaymentCompleted ∧ WantCancelReservation(within 24H after paying)

Service 3: Route Planning

I. Business Process Model and Notation



II. Annotation

Task Name	Action Description	Related Rules
Login	Performs(Customer, Login System)	
Input Location	Performs(Customer, Input source/current location)	if(Location permissions not enabled)
Input Destination and Travel Details	Performs(Customer, Input destination and pit stop details)	if(Location permissions enabled) if(User retries after receiving error message)
Submit Details	Performs(Customer, submit travel details to system)	
Receive Error Message	Performs(Customer, Error for input details)	
Receive Routing Details	Performs(System, Travel details from user)	
Send Error Message	Performs(System, Send error	If(Travel details entered are

	message)	not valid)
Receive Live Traffic Data	Performs(System, Live traffic data from GPS)	If(Travel details entered are valid)
Estimate Potential Routes	Performs(System, Estimate route details for user)	
Send Route Details	Performs(System, Sends finalised route details)	
Select Route	Performs(User, Selects route from available options)	
Submit Route	Performs(User, Submit the selected option)	
Receive Preferred route	Performs(System, Receives route preference from user)	
Prepare Navigation Instructions	Performs(System, Prepare Navigation instructions for user)	
Send Navigation Details	Performs(System, Send the instructions to user)	
Track and Provide Live Data	Performs(System, Continuously track user location until destination)	
Receive Navigation Instructions	Performs(User, Receives instructions)	

CurrentLocation(cur)	Destination(des)	SendErrorMessage(sem)
SubmitDetails(sd)	ReceivePreferredRoute(rpr)	RouteSelected(rs)
EndNavigationSession(ens)	RecieveNavigationInsruction(rni)	SendNavigationInstruction(sni)
RecieveErrorMessage(rem)	TrackLocation(tl)	TravelDetails(td)
PrepareInstructions(pi)	PermissionEnabled(pe)	RecieveRouteDetail(rrd)
CheckDetailsValid(cd)	ValidDetails(vd)	TravelPreferences(tp)
LiveTrafficData(ltd)	EstimateRoutes(er)	SendRouteDetails(srd)
ProvideLiveData(pld)	DestinationReached(dr)	ReceiveNavigationRequest(rnr)
ReceiveDestinationRequest(rdr)	StartNavigation(sn)	SubmitNewRoute(snr)
ReceiveCancelConfirmation(rcc)	ContinueNavigation(cn)	ProvideSessionSummary(pss)
ConfirmSessionEnd(cse)	ValidateCompletion(vc)	

III. Object of Interest

IV. Immediate Effects

No.	Activity name	Immediate effect (plain English)	Immediate effect (FOL)
1.	t1:User Login	The user logs in to the system to input source and destination details	PermissionEnabled(pe) ∨ CurrentLocation(cur) ∧ Destination(des)
2.	t2: Input Source Location	The user inputs the source location	Destination(des) ∧ SubmitDetails(sd) ∧ ReceiveRouteDetail(rrd)
3.	t3: Input destination and travel details	The user inputs the destination and travel preferences	SubmitDetails(sd) ∧ Destination(des) ∧ TravelPreferences(tp)

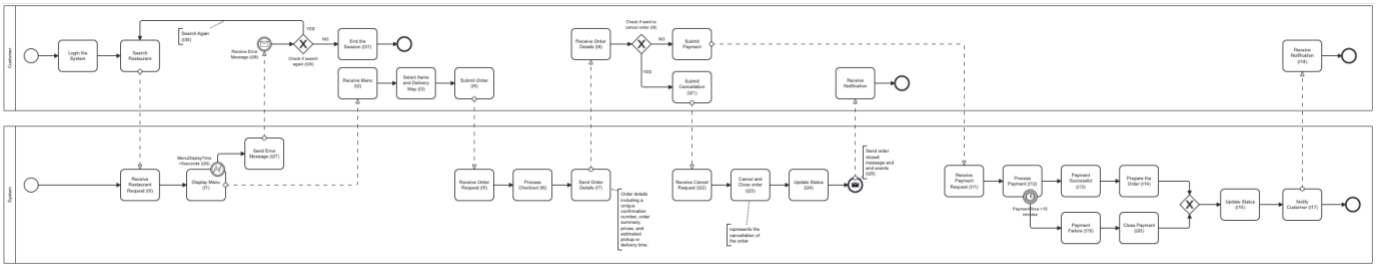
4.	t4: Submit travel details	The user submits the travel details to the system	TravelDetails(td) → CheckDetailsValid(cd)
5.	t5: Receive error message	The user receives an error message if the details are invalid	¬ValidDetails(vd) → SendErrorMessage(sem)
6.	t6: Input valid details	The user inputs valid details after receiving an error	ValidDetails(vd) ∧SubmitDetails(sd)
7.	t7: Receive routing details	The system receives routing details based on valid input	ValidDetails(vd) → ReceiveRoute(rr)
8.	t8: Estimate potential routes	The system estimates potential routes based on live traffic data	LiveTrafficData(ltd) → EstimateRoutes(er)
9.	t9: Send route details	The system sends the user the estimated routes	EstimateRoutes(er) → SendRouteDetails(srd)
10.	t10: Select route	The user selects a preferred route from available options	RouteSelected(rs)
11.	t11: Submit route	The user submits the selected route	RouteSelected(rs) → SubmitRoute(sr)
12.	t12: Receive preferred route	The system receives the preferred route from the user	SubmitRoute(sr) → ReceivePreferredRoute(rpr)
13.	t13: Prepare navigation instructions	The system prepares navigation instructions based on the selected	ReceivePreferredRoute(rpr) → PrepareInstructions(pi)
14.	t14: Send navigation details	The system sends navigation details to the user	PrepareInstructions(pi) → SendNavigationInstructions(sni)
15.	t15: Receive navigation instructions	The user receives navigation instructions from the system	SendNavigationDetails(sni) → ReceiveNavigationInstruction(rni)
16.	t16: Track and provide live data	The system tracks the user's location and provides live updates until the destination is reached	TrackLocation(tl) ∧ ProvideLiveData(pld) until DestinationReached(dr)

V. Cumulative Effects

No.	Activity name	Scenarios	Cumulative effect (FOL)
1	Navigate to Destination	t20: <<t0, t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16>, t20, {<t1, t6>, <t9, t14>}>	ReceiveDestinationRequest(rdr) ∧ ValidateDetails(vd) ∧ SelectRoute(sr) ∧ SubmitRoute(sr) ∧ ReceiveRouteDetails(rrd) ∧ PrepareNavigationInstructions(pni) ∧ SendNavigationInstructions(sni) ∧ ReceiveNavigationInstructions(rni) ∧ StartNavigation(sn) ∧ ProvideLiveData(pld) ∧ ReachDestination(rd)
2	Cancel Navigation	t25: <<t0, t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17, t18, t19>, t25, {<t1, t6>, <t10, t18>}>	ReceiveNavigationRequest(rnr) ∧ ValidateDetails(vd) ∧ SelectRoute(sr) ∧ SubmitRoute(sr) ∧ ReceiveRouteDetails(rrd) ∧ StartNavigation(sn) ∧ CheckCancelRequest(ccr) ∧ SubmitCancelRequest(scr) ∧ ReceiveCancelConfirmation(rcc) ∧ EndNavigationSession(ens)
3	Receive Traffic Data and Reroute	t30: <<t0, t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17, t18, t19, t20>, t30, {<t1, t10>, <t14, t20>}>	ReceiveLiveTrafficData(rlt) ∧ ValidateRoute(vr) ∧ PrepareAlternateRoute(par) ∧ SendAlternateRoute(sar) ∧ ReceiveRoute(rr) ∧ SubmitNewRoute(snr) ∧ PrepareNavigationInstructions(pni) ∧ ContinueNavigation(cn)
4	End Navigation Session	t40: <<t0, t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17, t18, t19, t20, t21, t22, t23, t24>, t40, {<t1, t6>, <t10, t24>}>	ReachDestination(rd) ∧ ValidateCompletion(vc) ∧ EndNavigationSession(ens) ∧ ProvideSessionSummary(pss) ∧ ConfirmSessionEnd(cse)
5	Handle Error and Retry Navigation	t35: <<t0, t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17, t18>, t35, {<t1, t6>, <t10, t18>}>	ReceiveErrorMessage(rem) ∧ CheckInputError(cie) ∧ ValidateInput(vi) ∧ ResubmitDetails(rd) ∧ ReceiveRouteDetails(rrd) ∧ PrepareNavigationInstructions(pni) ∧ ContinueNavigation(cn)

Service 4: Online Ordering

I. Business Process Model and Notation



II. Annotation

Task Name	Action Description	Related Rules
Login the system	Performs(Customer,Login,System)	
Search Restaurant	Performs(Customer,Search Restaurant)	
Receive Restaurant Request	Performs(System, Receive, RestaurantRequest)	
Display Menu	Performs(System, Display, Menu)	∧Knows(System, Menu, DisplayTime)
Select Items and Delivery Way	Performs(Customer, Select, ItemsAndDeliveryWay)	
Review Order Details	Performs(Customer, Review, OrderDetails)	
Submit Payment	Performs(Customer, Submit, Payment)	
Process Payment	Performs(System, Process, Payment)	∧Knows(System, Receive, PaymentRequest)
Payment Successful	Performs(System, Confirm, PaymentSuccess)	
Payment Failure	Performs(System, Process, PaymentFailure)	∧Knows(System, PaymentTimeout, 15Minutes)
Prepare the Order	Performs(System, Prepare, Order)	

Update Status	Performs(System, Update, OrderStatus)	∧Knows(System, Current, OrderStatus)
Notify Customer	Performs(System, Notify, Customer)	∧Knows(System, EventOccurred, NotifyCustomer)
Cancel and Close Order	Performs(System, Cancel, Order)	∧Knows(System, Receive, CancelRequest)
Receive Notification	Performs(Customer, Receive, Notification)	
Send Error Message	Performs(System, Send, ErrorMessage)	∧Knows(System, MenuDisplay, Timeout)
End the Session	Performs(System,End,Session)	∧Knows(System, UserCompleteActions ∧ NotRestartSearch)

III. Object of Interest

ReceiveRestaurantRequest (rrr)	DisplayMenu(dm)	TriggerError(tr)
SendErrorMessage(sem)	SearchAgain(sr)	EndSession(es)
Receive Payment Request(rpr)	ReceiveMessage(rm)	ReceiveMenu(m)
SelectItemsDelivery(s)	SubmitOrder(so)	ReceiveOrderRequest (ror)
ProcessCheckout(pc)	SendOrderDetails(sod)	ReviewOrderDetails(rod)
CheckCancel(cc)	SubmitCancellation(sc)	ReceiveCancelRequest (rcr)
CancelCloseOrder(cco)	UpdateStatus(us)	SubmitPayment(sp)
ProcessPayment(pp)	PaymentSuccessful(ps)	PrepareOrder(po)
PaymentFailure(pf)	ClosePayment(cp)	PaymentTimeout(pt)

IV. Immediate Effects

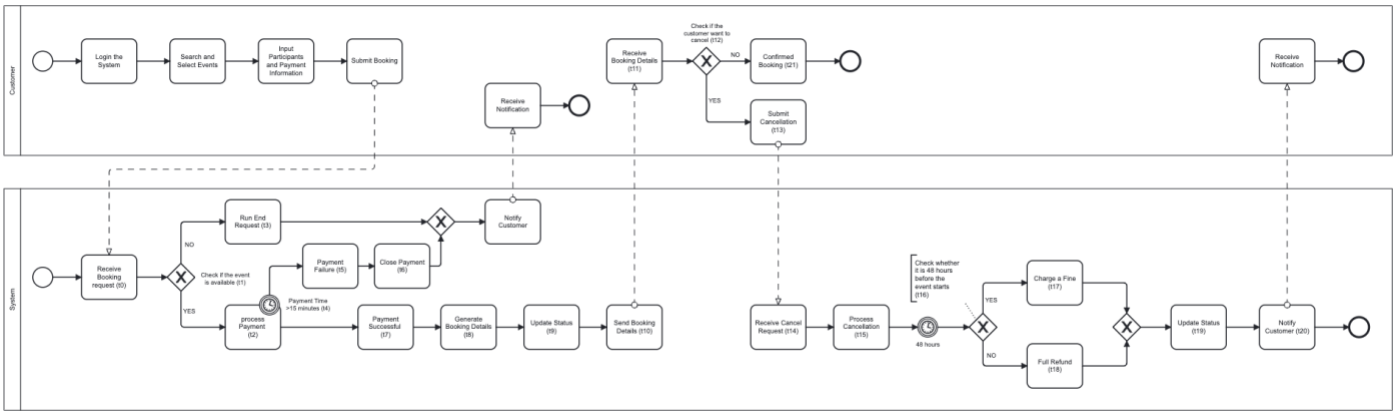
No.	Activity name	Immediate effect (plain English)	Immediate effect (FOL)
1	t27:Send Error Message	The display menu timeout (>5 seconds) triggered error, causing an error message to be sent to the user	TriggerError(tr)∧ SendErrorMessage(sem)
2	t29:Check If Search Again	The consumer decides whether to search again, and if "YES", then search again; If "NO", the session ends, after reviewing the error message.	SearchAgain(sr) v (¬SearchAgain(sr)∧EndSession(es))
3	t7:Send Order Details	Order details include a unique confirmation number, order summary, price, and estimated pickup or delivery time, after the system enters the checkout process.	ProcessCheckout(pc)∧ SendOrderDetails(sod)
4	t21:Submit Cancellation	The customer decides "YES" to cancel after reviewing the order details and therefore submits a cancellation request.	ReviewOrderDetails(rod)∧CheckCancel(cc)∧ SubmitCancellation(sc)
5	t10:Submit Payment	The customer decides "NO" to cancel after reviewing the order details and therefore submits a payment request.	ReviewOrderDetails(rod)∧¬CheckCancel(cc)∧ SubmitPayment(sp)
6	t14:PrepareOrder	The system enters the process of preparing the food on the order after the payment is successful.	PaymentSuccessful(ps) ∧ PrepareOrder(po)
7	t19:PaymentFailure(pf)	The system takes more than 10 minutes (> 10 minutes) to process the payment, the payment fails.	ProcessPayment(pp)∧ PaymentTimeout(pt)∧ PaymentFailure(pf)
8	t15:UpdateStatus(us)-Post-ProcessPayment	The system updates the status after an order has entered the preparation process or payment has been closed due to a payment failure.	(PrepareOrder(po) vClosePayment(cp))∧ UpdateStatus(us)
9	t24:UpdateStatus(us)-Post-Cancellation	The system updates the status after cancel and close order	CancelCloseOrder(cco) ∧UpdateStatus(us)

V. Cumulative Effects

No.	Activity name	Scenarios	Cumulative effect (FOL)
1	End the Session	t31:<<t0,t1,t26,t27,t28,t29>, t31,{<t1,t2>}>	ReceiveRestaurantRequest (rrr)∧DisplayMenu(dm)∧TriggerError(tr)∧SendErrorMessage(sem)∧ReceiveMessage(rm)∧¬SearchAgain(sr)∧EndSession(es)
2	Cancel and Close Order	t23:<<t0,t1,t2,t3,t4,t5,t6,t7,t8,,t9,t21,t22>, t23, {<t1,t26>,<t9,t10>}>	ReceiveRestaurantRequest(rrr)∧DisplayMenu(dm)∧¬TriggerError(tr)∧ReceiveMenu(m)∧SelectItemsDelivery(s)∧SubmitOrder(so)∧ReceiveOrderRequest (ror)∧ProcessCheckout(pc)∧SendOrderDetails(sod)∧ReviewOrderDetails(rod)∧CheckCancel(cc)∧SubmitCancellation(sc)∧ReceiveCancelRequest (rcr)∧CancelCloseOrder(cco)
3	Prepare the Order	t14:<<t0,t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13>,t14,{<t1,t26>,<t9,t21>}>	ReceiveRestaurantRequest(rrr)∧DisplayMenu(dm)∧¬TriggerError(tr)∧ReceiveMenu(m)∧SelectItemsDelivery(s)∧SubmitOrder(so)∧ReceiveOrderRequest (ror)∧ProcessCheckout(pc)∧SendOrderDetails(sod)∧ReviewOrderDetails(rod)∧¬CheckCancel(cc)∧SubmitPayment (sp)∧Receive Payment Request(rpr)∧ProcessPayment(pp)∧¬PaymentTimeout(pt)∧PaymentSuccessful(ps)∧PrepareOrder(po)

Service 5: Booking Activity

I. Business Process Model and Notation



II. Annotation

Task Name	Action Description	Related Rules
Login the System	Performs(Customer, Login, System)	
Search and Select Events	Performs(Customer, SearchAndSelect, Events)	
Input Participants and Payment Information	Performs(Customer, Input, ParticipantsAndPaymentInformation)	
Submit Booking	Performs(Customer, Submit, Booking)	
Receive Booking Request	Performs(System, Receive, BookingRequest)	
Check if the event is available	Performs(System, Check, EventAvailability)	∧Knows(System, Event, Availability)
Run End Request	Performs(System, End, Request)	∧Knows(System, Event, Unavailability)
Process Payment	Performs(System, Process, Payment)	∧Knows(System, Receive, PaymentRequest)
Payment Time 15 minutes	Performs(System, Check, PaymentTime)	∧Knows(System, PaymentTimeout, Exceeded 15 Minutes)

Payment Failure	Performs(System, Handle, PaymentFailure)	∧Knows(System, PaymentTimeout ∨ PaymentError)
Close Payment	Performs(System, Close, Payment)	
Generate Booking Details	Performs(System, Generate, BookingDetails)	∧Knows(System, PaymentSuccessful)
UpdateStatus(us)-Post-PaymentSuccessful	Performs(System, Update, Status)	∧Knows(System, Genera, OrderStatus)
Send Booking Details	Performs(System, Send, BookingDetails)	∧Knows(System, BookingDetailsGenerated)
Check if customers want to cancel	Performs(Customer, Decide, CancelOrConfirm)	∧Knows(Customer, BookingDetails)
Submit Cancellation	Performs(Customer, Submit, Cancellation)	
Receive Cancel Request	Performs(System, Receive, CancelRequest)	∧Knows(System, CancellationRequested)
Check whether it is 48 hours before the event starts	Performs(System, Check, TimeBeforeEvent)	∧Knows(System, TimeToEventStart)
Charge a Fine	Performs(System, Charge, Fine)	∧Knows(System, CancellationWithin48Hours)
Full Refund	Performs(System, Handle FullRefund)	∧Knows(System, CancellationOutside48Hours)
UpdateStatus(us)-Post-PorocessCancel	Performs(System, Update, Status)	∧Knows(System, CancellationCompleted)
Notify Customer	Performs(System, Notify, Customer)	∧Knows(System, CancellationProcessedStatus)

Available(a)	Payment(p)	RunEndRequest(en)
PaymentTime(pt)	PaymentFailure(pf)	PaymentSuccessful(ps)
ClosePayment(cp)	GenerateBookingDetails (gbd)	SendBookingDetails(sbd)
Receive Booking Details(rbd)	ConfirmedBooking(cb)	CheckCancel(cc)
SubmitCancellation(sc)	ChargeFine(cf)	FullRefund(fr)
ReceiveCancelRequest (rcr)	CheckTimeout(ct)	ReceiveRequest(rr)
UpdateStatus(us)-Post-PaymentSuccessful	UpdateStatus(us)-Post-PorocessCancel	ProcessCancellation(pc)

III. Object of Interest

IV. Immediate Effects

No.	Activity name	Immediate effect (plain English)	Immediate effect (FOL)
1	t1:Check available for booking the event	Check the availability of the event reservation after receiving the booking request.	ReceiveRequest(rr) ∧ (Available(a) v ¬Available(a))
2	t2:Process payment	The payment of the booking event is processed by this system if the booking event is available.	Available(a) ∧Payment(p)
3	t3:Run end request	The system automatically ends the booking request after checks that the activity booking is not available.	¬Available(a) ∧ RunEndRequest(en)
4	t4:PaymentTime 15 minutes	The system default payment processing time is 15 minutes. If it exceeds this time, the payment will time out.	Payment(p)∧ (PaymentTime(pt)v ¬PaymentTime(pt))
5	t5: Payment Failure	If the timer is overrun 15 minutes and the payment is not completed, the WithinTime (wt) condition is	¬PaymentTime(pt)∧PaymentFailure(pf)

		not met and the process causes the payment to fail.	
6	t6: Close Payment	The system will automatically close the payment process after a payment failure.	PaymentFailure(pf)∧ ClosePayment(cp)
7	t7:Payment Successful	The system completes the payment within payment time, the payment is automatically marked as successful	PaymentTime(pt)∧PaymentSuccessful(ps)
8	t9:Update status	The booking status is updated in the system to reflect the final state after payment is successful.	PaymentSuccessful(ps) ∧ UpdateStatus(us)
9	t13:Submit Cancellation	The customer decides "YES" to cancel after reviewing the order details and therefore submits a cancellation request.	CheckCancel(cc)∧ SubmitCancellation(sc)
10	t21:Confirmed Booking	The customer confirmed booking after deciding "NO" to cancel.	¬CheckCancel(cc)∧ ConfirmedBooking(cb)
11	t17:Charge a Fine	The system check time is less than 48 hours before the event starts, the cancellation penalty will be deducted automatically	CheckTimeout(ct)∧ ChargeFine(cf)
12	t19:UpdateStatus (us)-Post-PorocessCancel	The system updates the status after completing the collection of fines or a full refund.	(ChargeFine(cf)vFullRefund(fr))∧UpdateStatus(us)-Post-PorocessCancel

V. Cumulative Effects

No.	Activity name	Scenarios	Cumulative effect (FOL)
1	Payment Successful	t7:<<t0,t1,t2>, t7,{<t1,t3>,<t2,t4>}>	ReceiveRequest(rr) ∧ Available(a) ∧ Payment(p) ∧PaymentTime(pt) ∧ PaymentSuccessful(ps)
2	Close Payment	t6:<<t0,t1,t2,¬t4,t5>,t6,{<t1,t3>,<t2,t4>}>	ReceiveRequest(rr) ∧ Available(a) ∧ Payment(p) ∧ ¬PaymentTime(pt)∧ PaymentFailure(pf) ∧ ClosePayment(cp)

3	Confirmed Booking	t21:<<t0,t1,t2,t7,t8,t9,t10,t11,t12>,t21,{<t1,t3>,<t2,t4>,<t12,t13>}>	ReceiveRequest(rr) ∧ Available(a) ∧ Payment(p) ∧ PaymentTime(pt) ∧ PaymentSuccessful(ps)∧ GenerateBookingDetails (gbd)∧UpdateStatus(us)∧ SendBookingDetails(sbd)∧ Receive Booking Details(rbd)∧ConfirmedBooking(cb)
4	Charge a Fine	t17:<t0,t1,t2,t7,t8,t9,t10,t11,t12, t13,t14,t15,t16>,t17,{<t1,t3>,<t2,t4>,<t12,t21>,<t16,t18>}	ReceiveRequest(rr) ∧ Available(a) ∧ Payment(p) ∧ PaymentTime(pt) ∧ PaymentSuccessful(ps)∧ GenerateBookingDetails (gbd)∧UpdateStatus(us)∧ SendBookingDetails(sbd)∧ Receive Booking Details(rbd)∧CheckCancel(cc)∧SubmitCancellation(sc)∧ ReceiveCancelRequest(rcr)∧ ProcessCancellation(pc)∧CheckTimout(ct)∧ChargeFine(cf)

TASK D. Service Design and Analysis using SoaML

Service Architecture 1 -Tourism Information Centre(TIC)

Service 1: Visitor Information

Service Participants:

- **Visitor Information System (Service Providers)**: Service provider is responsible for providing Information to the visitor based on the destination
- **Customers(Service Consumers)**: Service Consumers are the one who will request for the information

Service Architecture: High level structure of how components interact

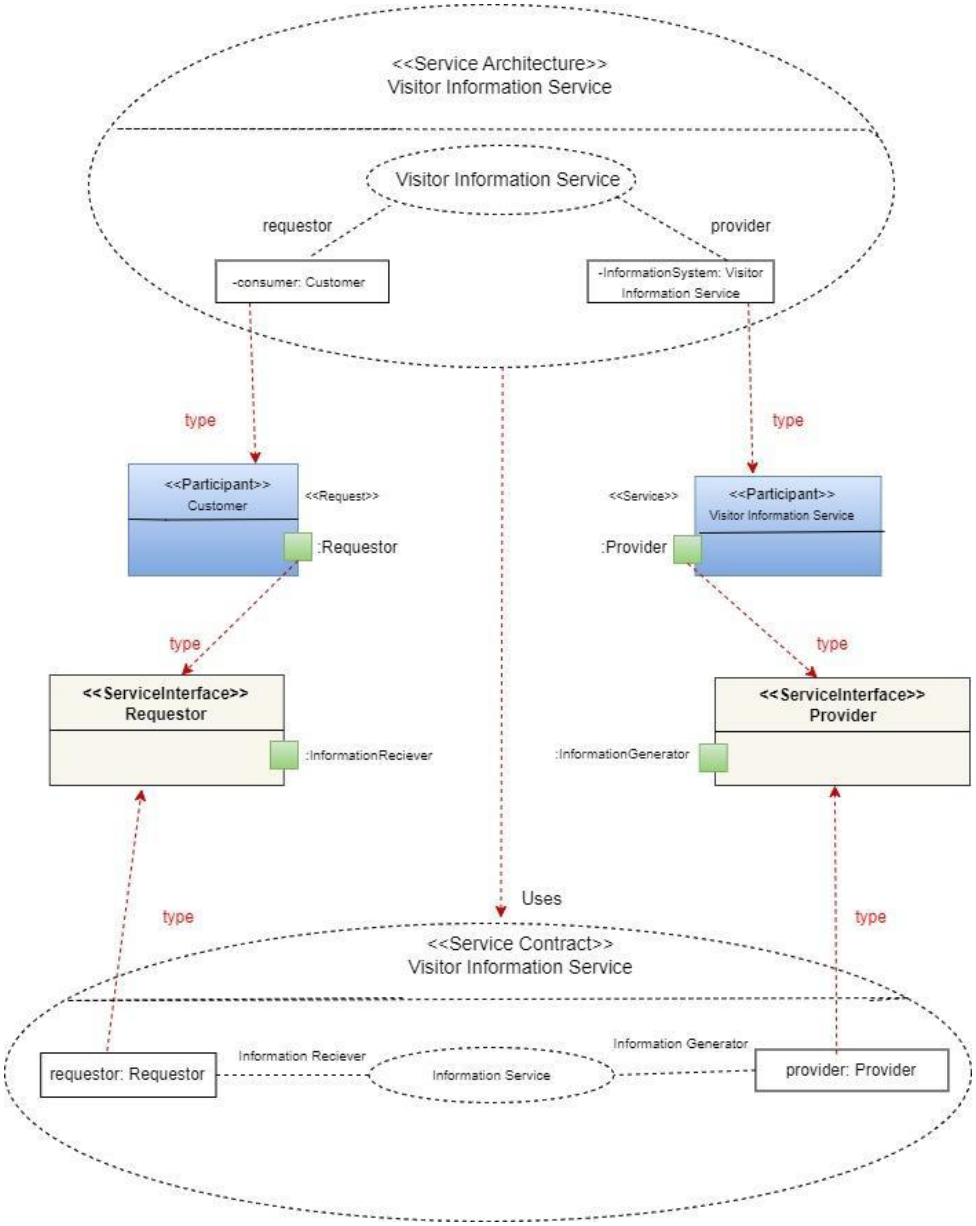
- Illustrate the interaction between components:
    - Visitor Information System

Service Interface: Operations provided by the participants

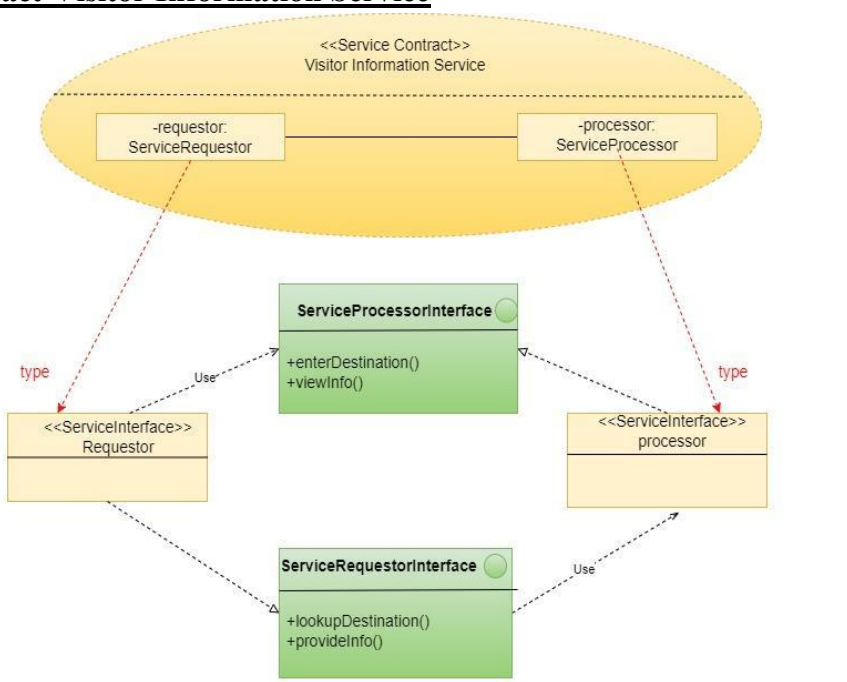
- **EnterDestination():** Allows users to select location. Hotel, date of arrival, date of departure
- **AddDetails():** Allows users to add details about their trip for the destination
- **SendDetails()**: Sends details to the user once all information has been retrieved

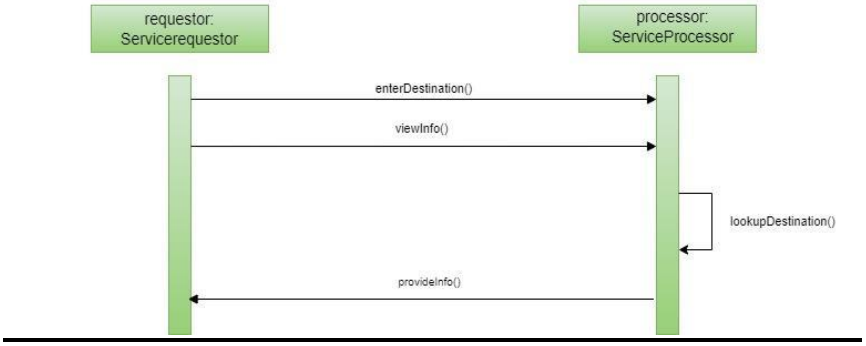
Service Contracts: Agreements between service providers and consumers

Service Architecture


<<Service Architecture>>
Visitor Information Service
Visitor Information Service
requestor
provider
-consumer: Customer
-InformationSystem: Visitor Information Service
type
type
<<Participant>>
Customer
<<Request>>
:Requestor
<<Service>>
<<Participant>>
Visitor Information Service
:Provider
type
type
<<ServiceInterface>>
Requestor
:InformationReciever
<<ServiceInterface>>
Provider
:InformationGenerator
Uses
type
type
<<Service Contract>>
Visitor Information Service
requestor: Requestor
Information Reciever
Information Service
Information Generator
provider: Provider

Service Contract-Visitor Information Service





Service 2: Tour Booking

Service Participants:

- **Tour Booking System(Service Providers)**: Service provider is responsible for providing the best deals to users for their planned tour
- **Customers(Service Consumers)**: Service Consumers are the one who will use the Tour Booking System for booking their tours

Service Architecture: High level structure of how components interact

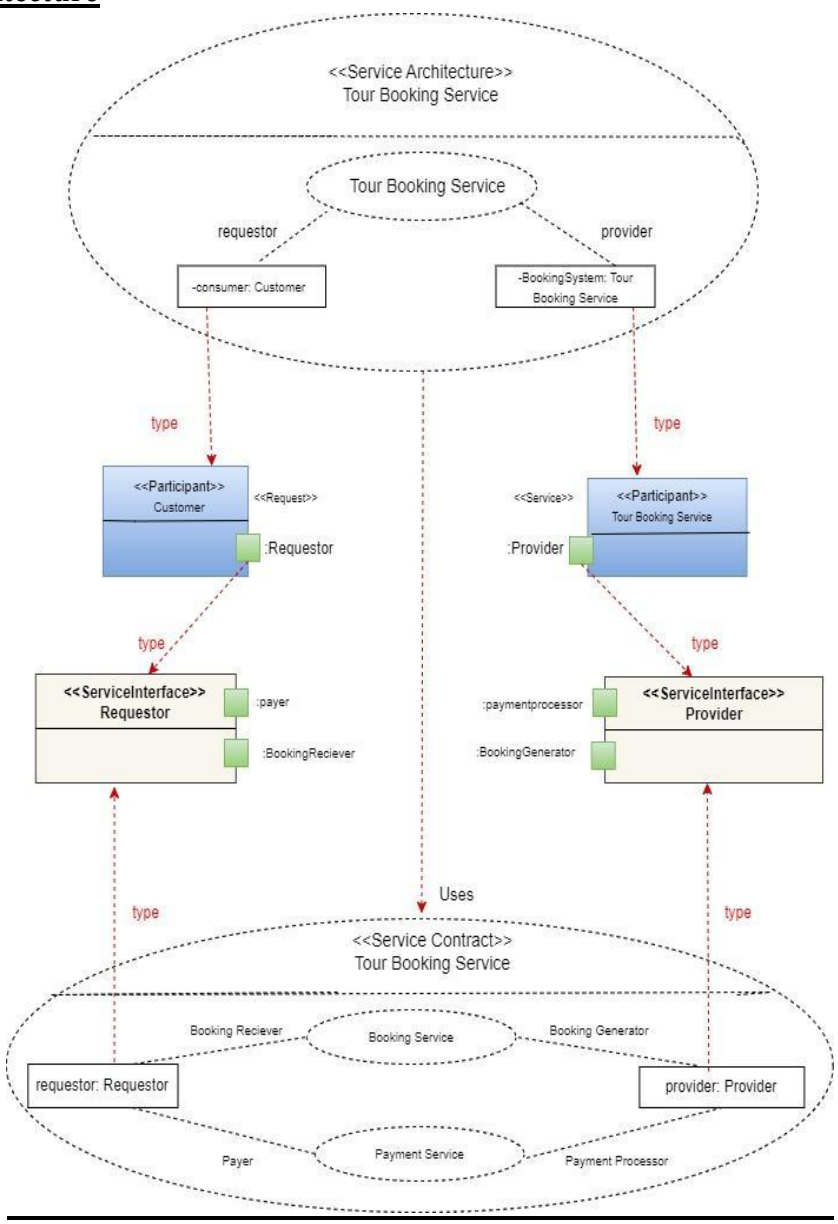
- Illustrate the interaction between components:
    - Tour Booking System
    - Notification System
    - Customer database
    - Payment System

Service Interface: Operations provided by the participants

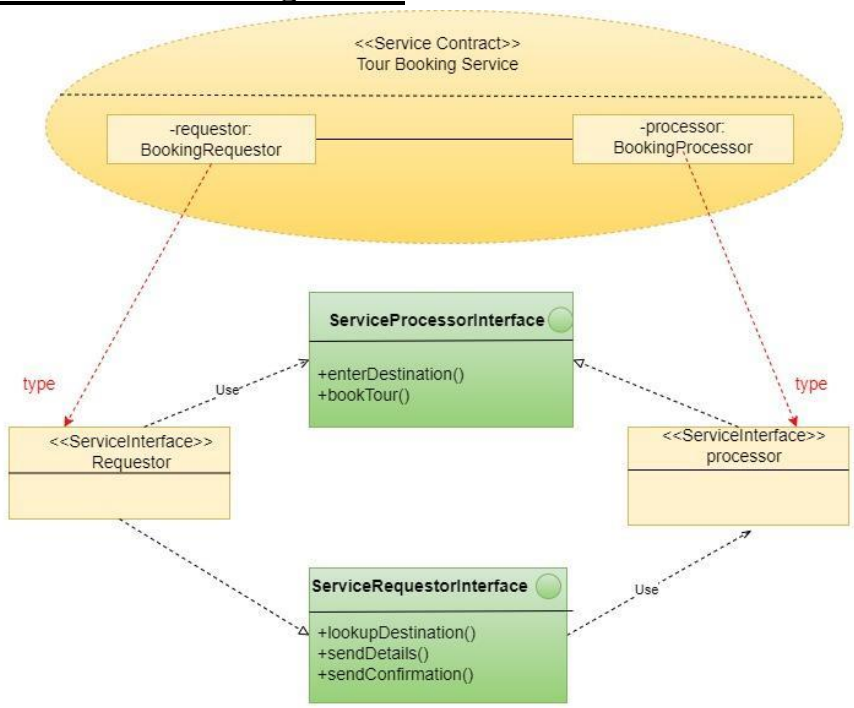
- **BookTour():** Allows users to select location. Hotel, date of arrival, date of departure
- **CheckAvailibility():** Allows users to add details about their tour including budget, destination, number of people and so on based on chosen dates
- **SendConfirmation()**: Sends details to the user once all information has been retrieved and payment is done

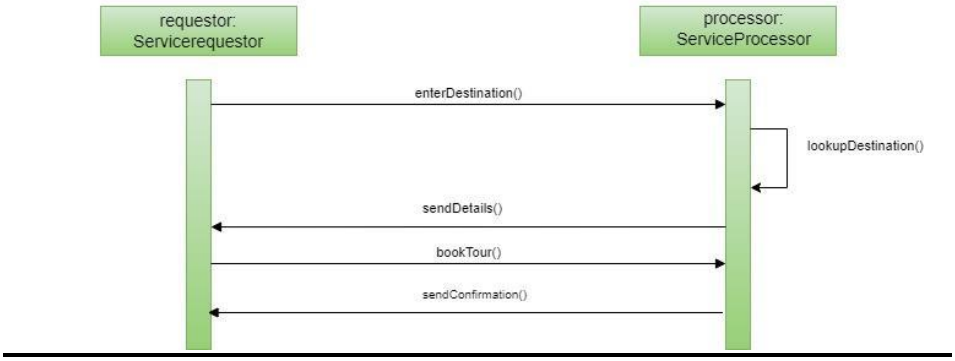
Service Contracts: Agreements between service providers and consumers

Service Architecture

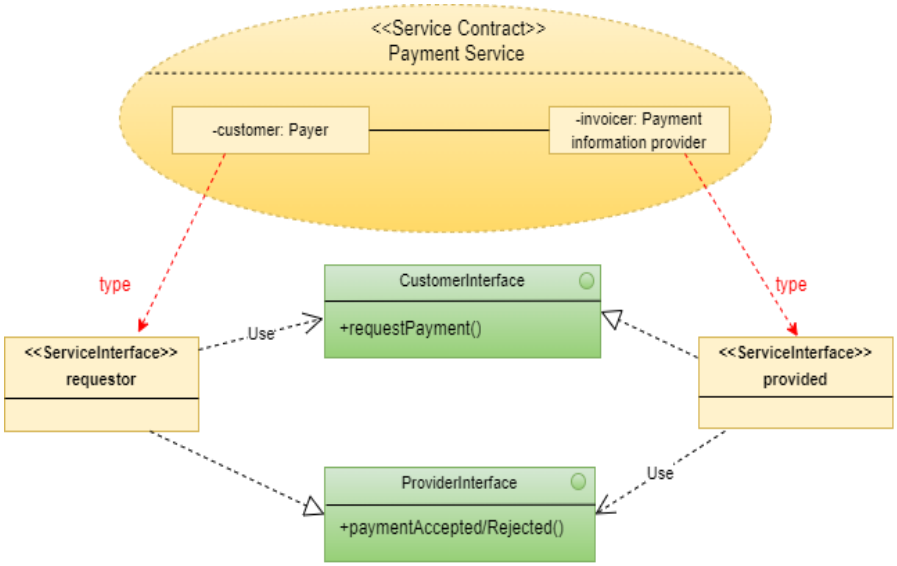


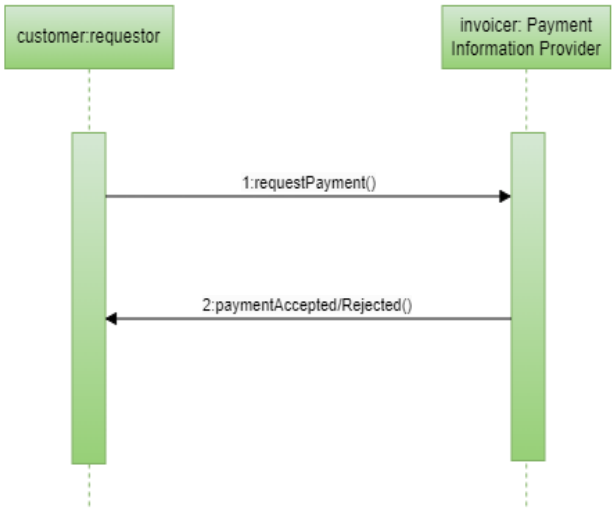
Service Contract- Tour Booking Service





Service Contract- Payment Service





Service 3: Event Notification

Service Participants:

- **Event Notification System(Service Providers)**: Send notification for events to users who have booked a tour
- **Customers(Service Consumers)**: Service Consumers are the one who will use the Event Notification System for receiving notification for the booking

Service Architecture: High level structure of how components interact

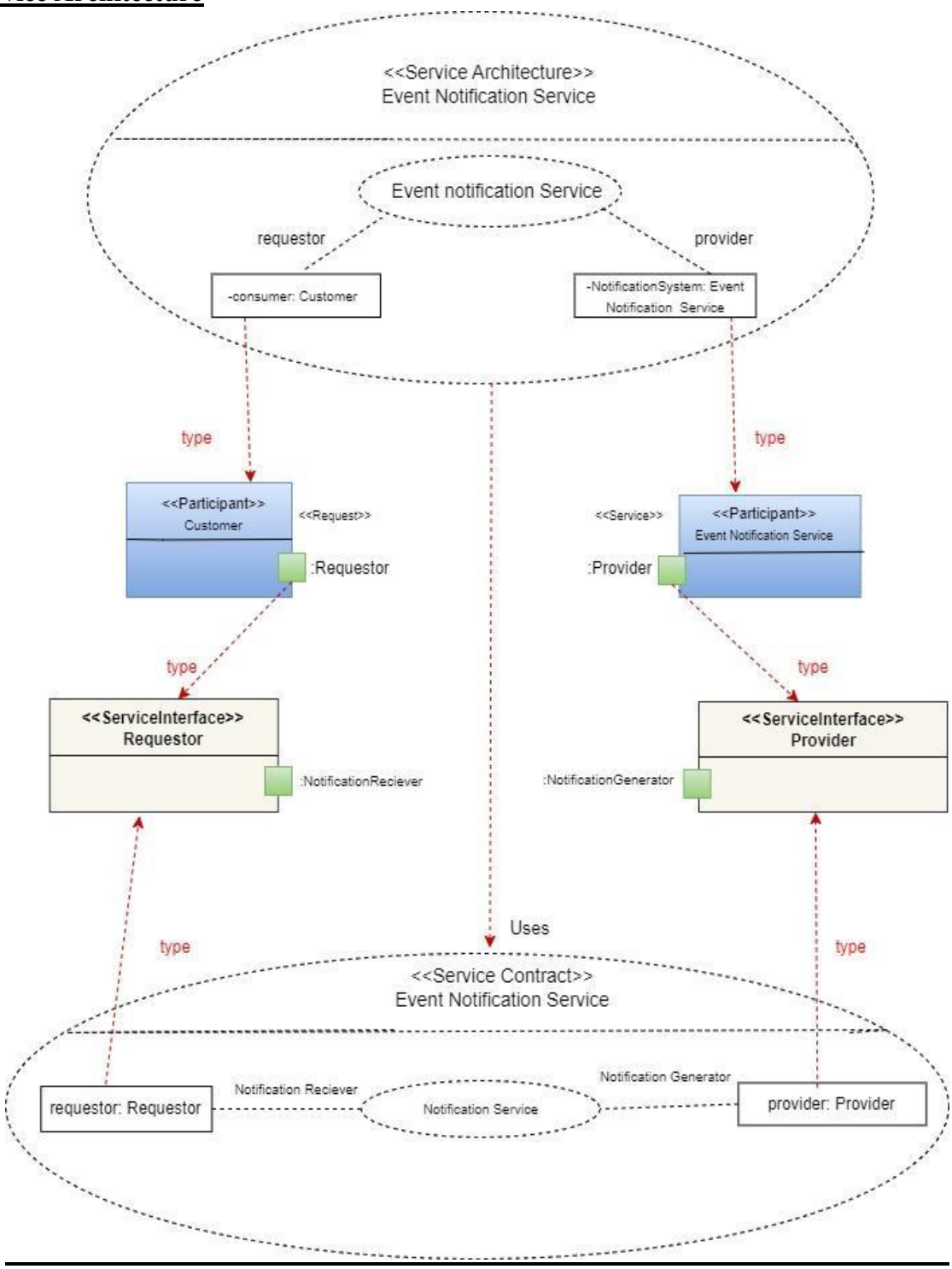
- Illustrate the interaction between components:
    - Event Notification System
    - Customer database

Service Interface: Operations provided by the participants

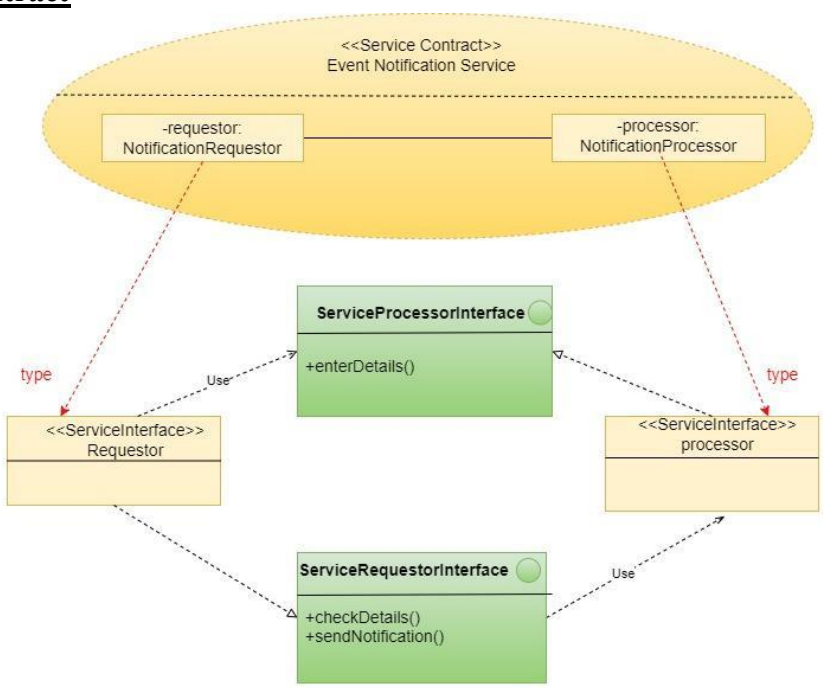
- **SendNotification()**: Sends details to the user once all information has been retrieved and payment is done
- **CheckDetails():** Checks if the user details are present in the database.

Service Contracts: Agreements between service providers and consumers

Service Architecture



Service Contract



Service Architecture 2 - Local Transportation Service(LTS)

Service 1: Ride Booking

Service Participants:

- **Ride booking System (Service Providers)**: Service provider is responsible for providing ride booking service
- **Customers(Service Consumers)**: Service Consumers are the one availing the local transportation service to book rides..

Service Architecture: High level structure of how components interact

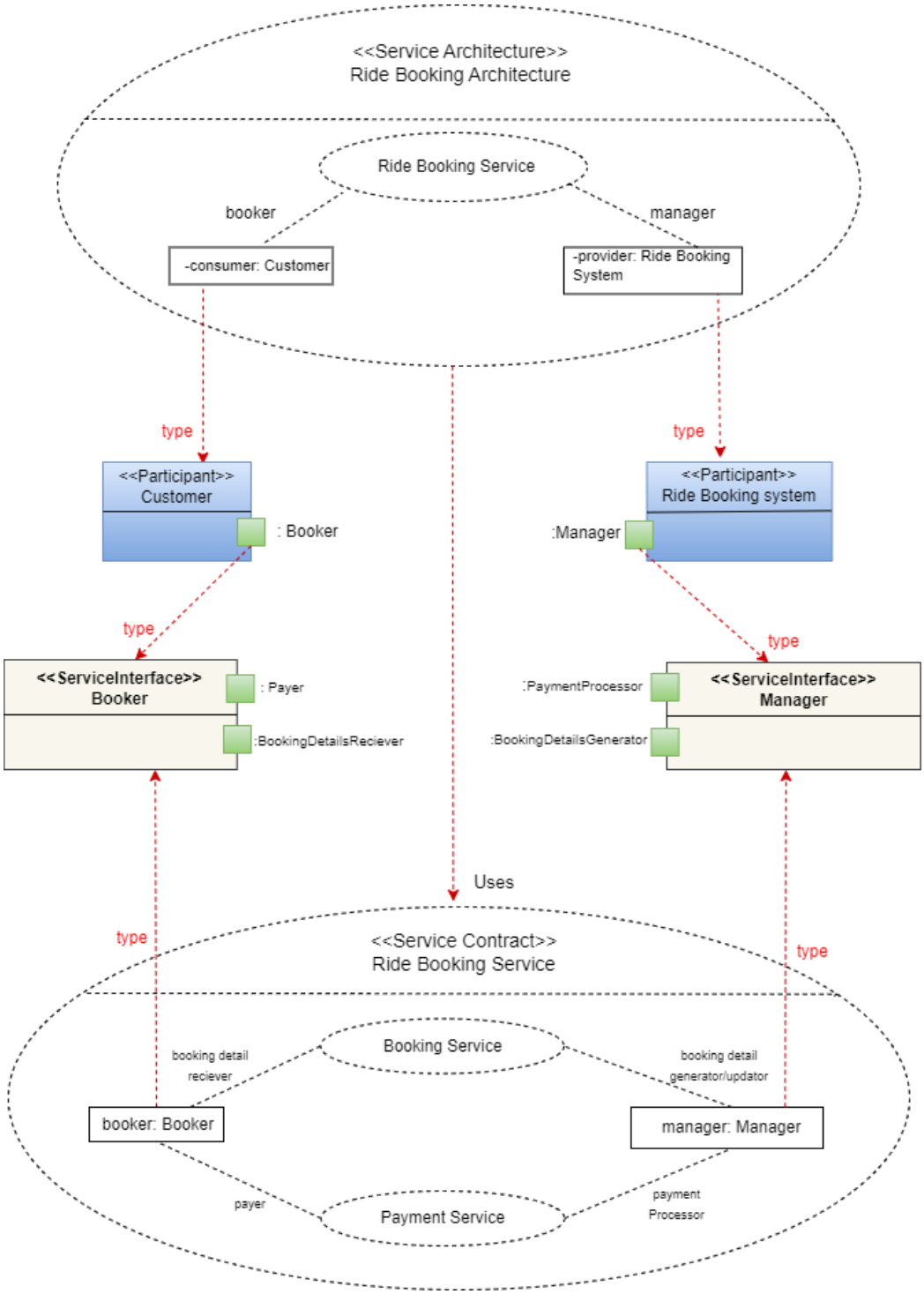
- Illustrate the interaction between components:
    - Ride Booking System
    - Payment Gateway
    - Notification System
    - Customer Database

Service Interface: Operations provided by the participants

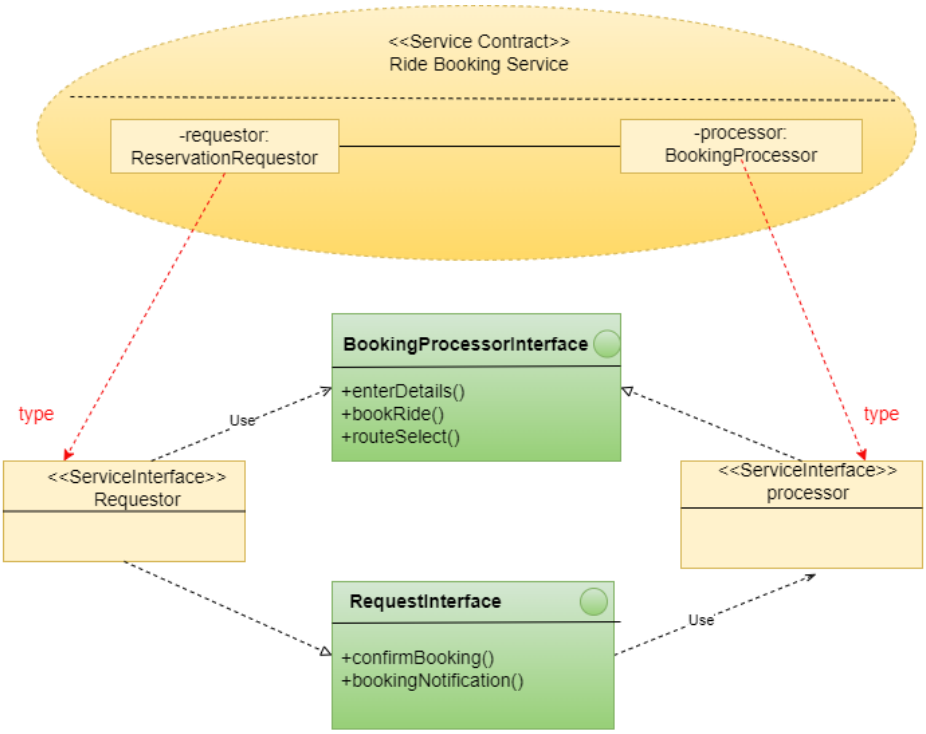
- **RouteSelection():** Allows to choose a navigation route and calculates duration based on the current location and destination.
- **BookRide():** Takes input as current location, destination, preferred mode of transport and payment details and provides Payment Receipt, vehicle number and ETA as output.
- **SendConfirmation():** Sends a confirmation for a ride booked to the customer.

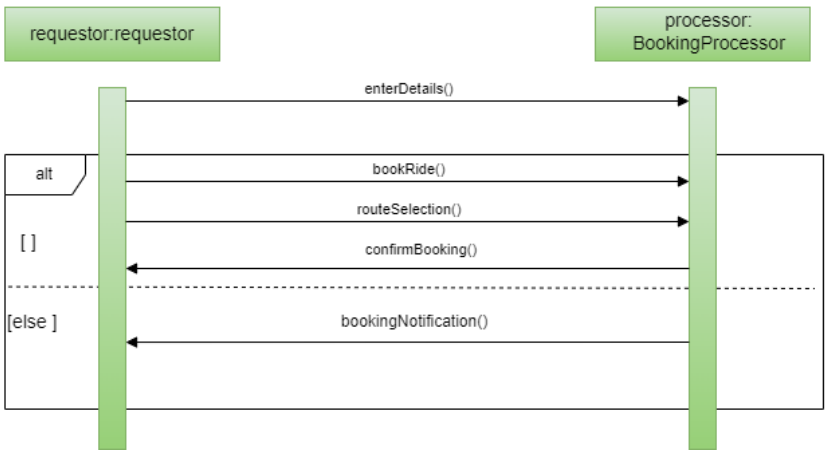
Service Contracts: Agreements between service providers and consumers

SERVICE ARCHITECTURE

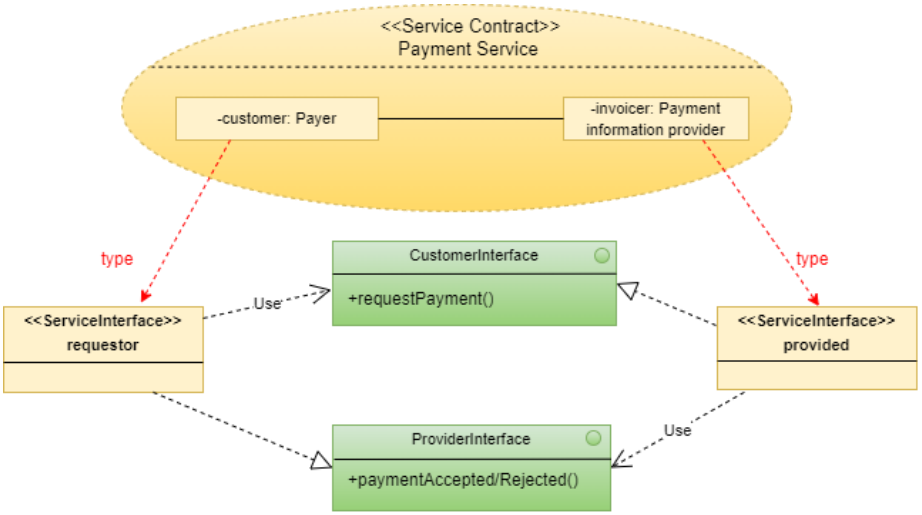


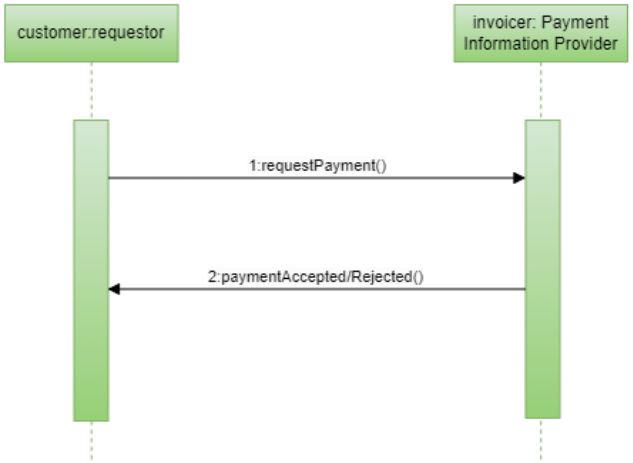
SERVICE CONTRACT - BOOKING SERVICE





SERVICE CONTRACT - PAYMENT SERVICE





Service 2: Real-Time Traffic Monitoring

Service Participants:

- **Real-Time Monitoring service (Service Providers)**: Service provider is responsible for providing live traffic information.
- **Customers(Service Consumers)**: Service Consumers are the one availing the local transportation service to get real-time traffic information.

Service Architecture: High level structure of how components interact

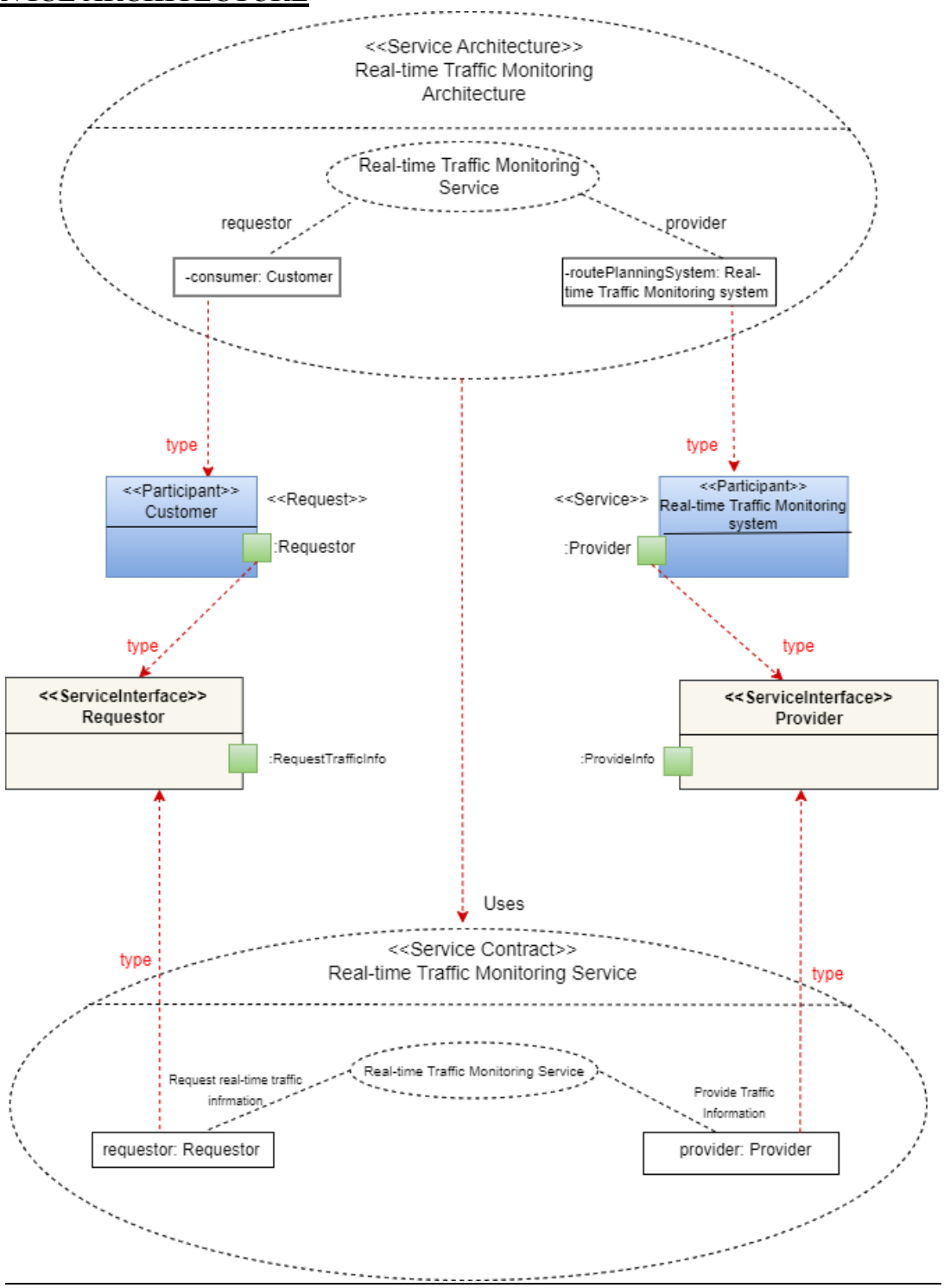
- Illustrate the interaction between components:
    - Real-time traffic monitoring system

Service Interface: Operations provided by the participants

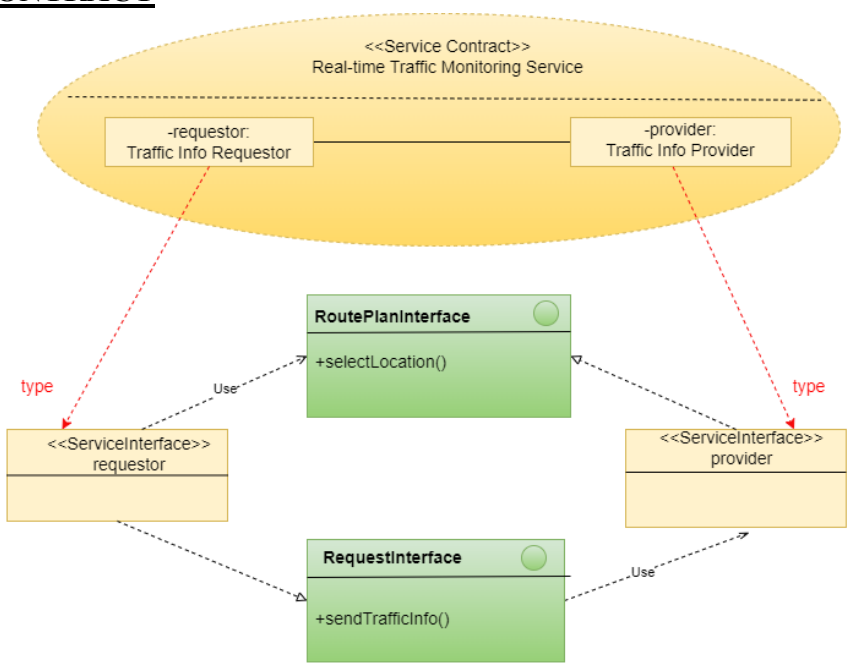
- **TrafficInfo():** Takes input as current location, time of the day, destination and provides information on traffic congestion, delays and operating routes.

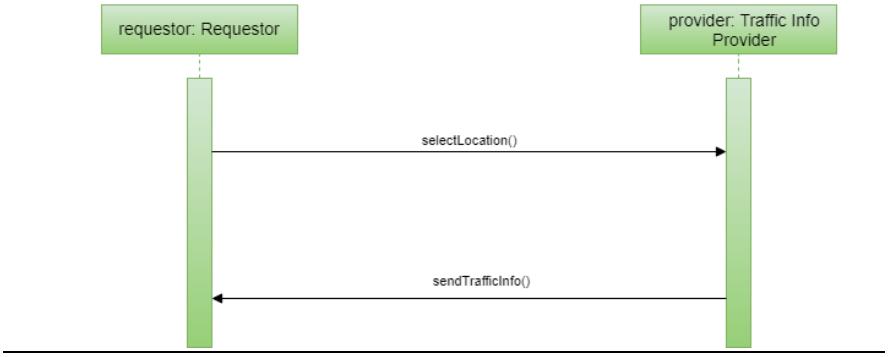
Service Contracts: Agreements between service providers and consumers

SERVICE ARCHITECTURE



SERVICE CONTRACT





Service 3: Route Planning

Service Participants:

- **Route Planning service (Service Providers)**: Service provider is responsible for providing live traffic information, best route, time and other travel related information
- **Customers (Service Consumers)**: Service Consumers are the one availing the local transportation service for route planning based on travel destination.

Service Architecture: High level structure of how components interact

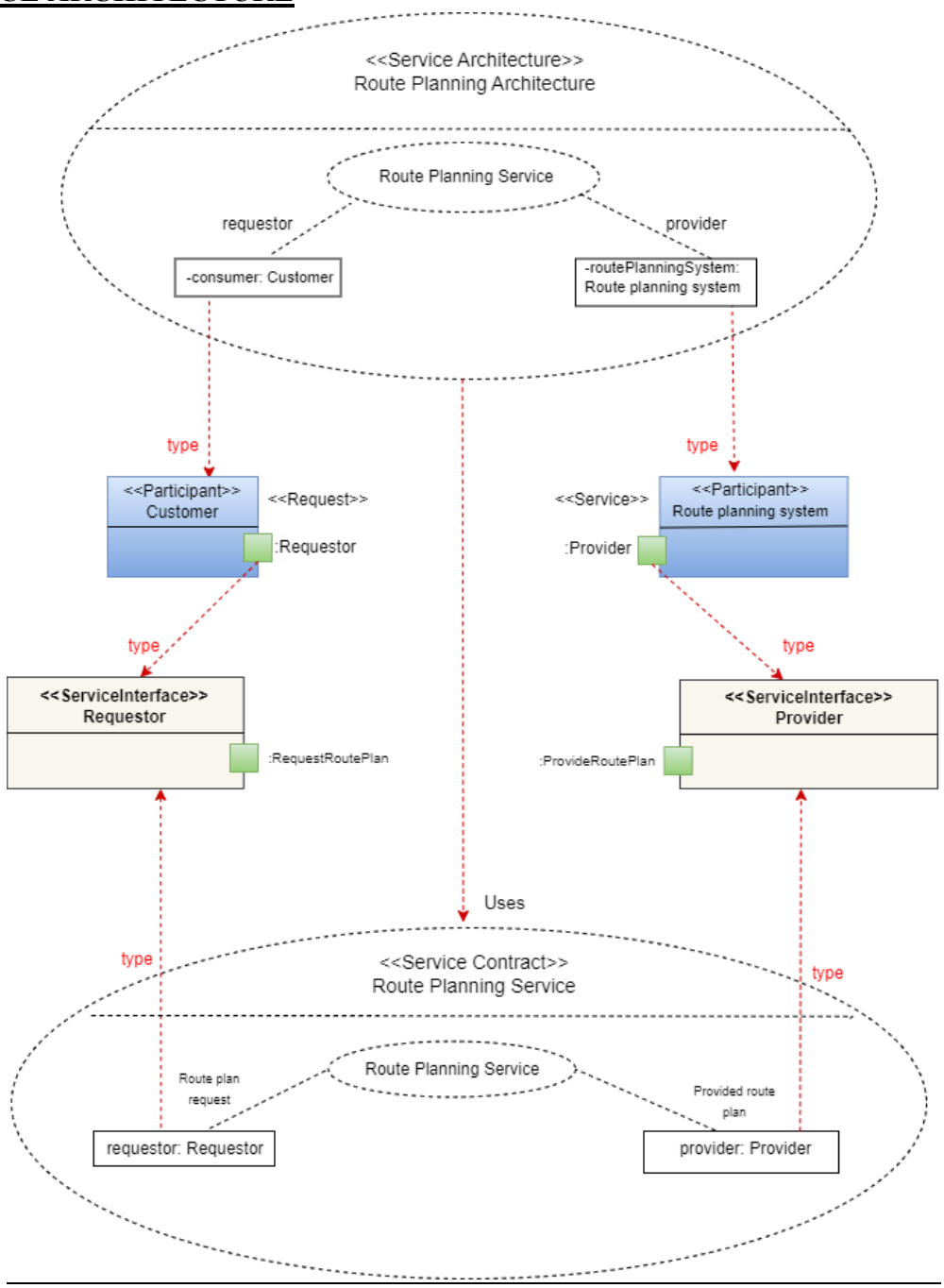
- Illustrate the interaction between components:
    - Navigation System

Service Interface: Operations provided by the participants

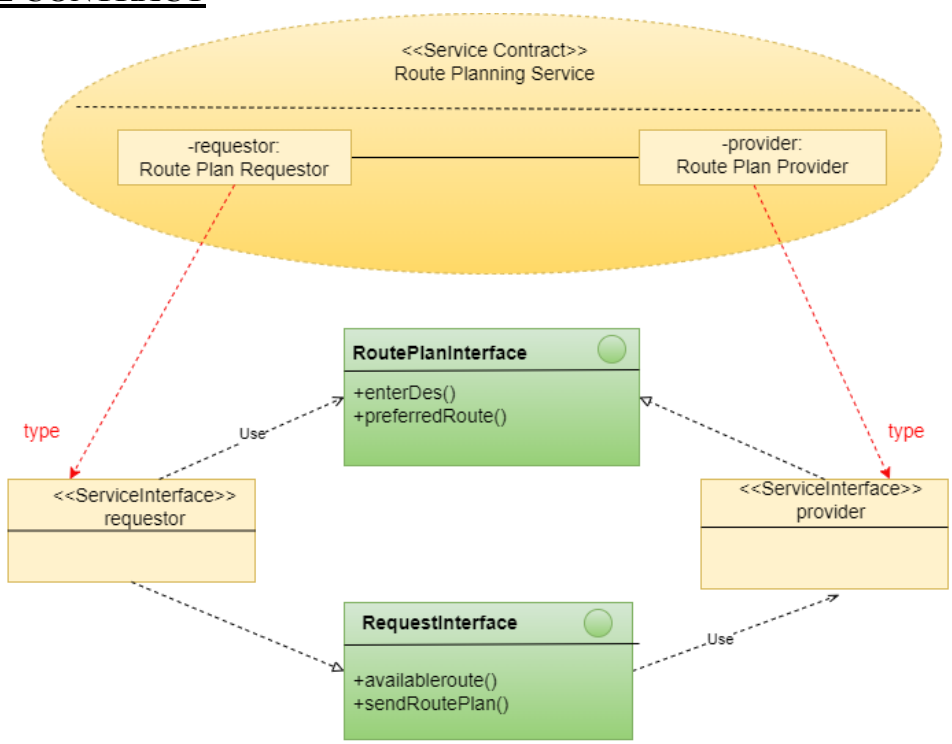
- **RoutePlanning():** Takes input as current location, time of the day, destination and provides information on traffic congestion, delays and best operating route.

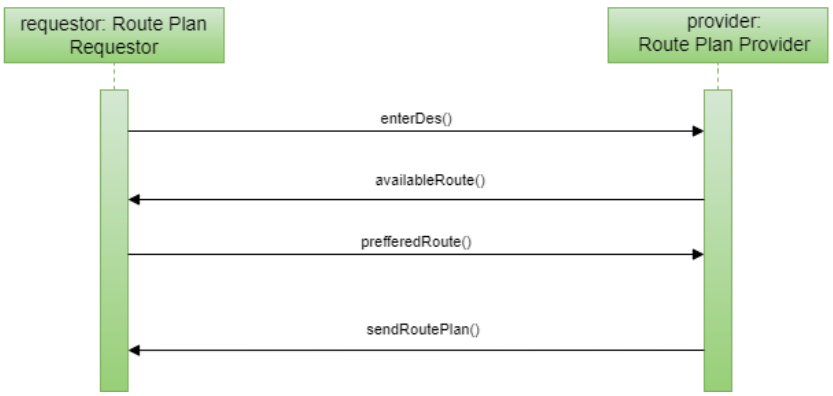
Service Contracts: Agreements between service providers and consumers

SERVICE ARCHITECTURE



SERVICE CONTRACT





Service Architecture 3 - Accommodation Provider (AP)

Service 1: Room Booking

Service Participants:

- **Room booking System (Service Providers)**: Service provider is responsible for providing room booking service
- **Customers(Service Consumers)**: Service Consumers are the one availing the service to book rooms.

Service Architecture: High level structure of how components interact

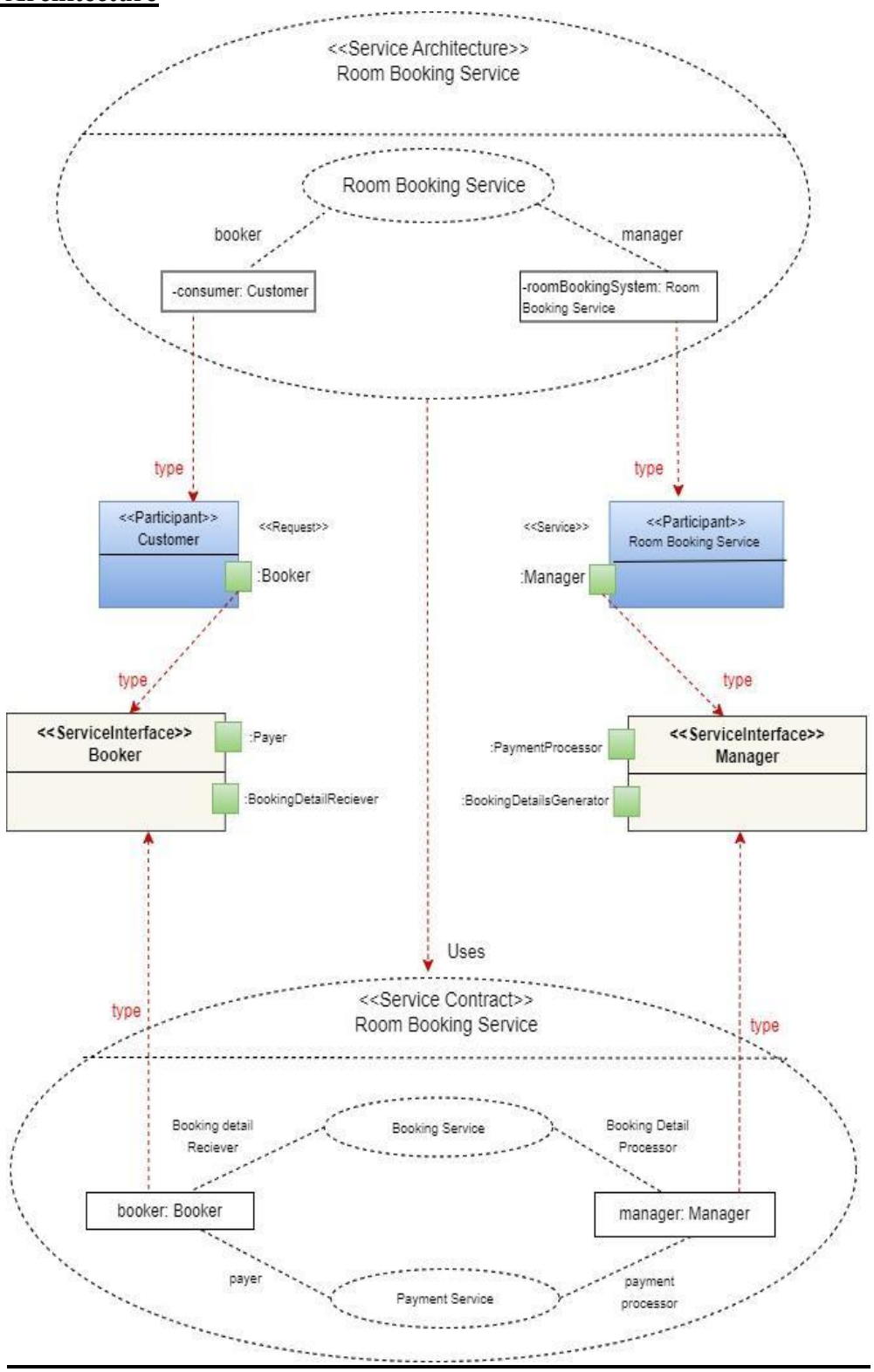
- Illustrate the interaction between components:
    - Room Booking System
    - Payment Gateway
    - Notification System
    - Customer Database

Service Interface: Operations provided by the participants

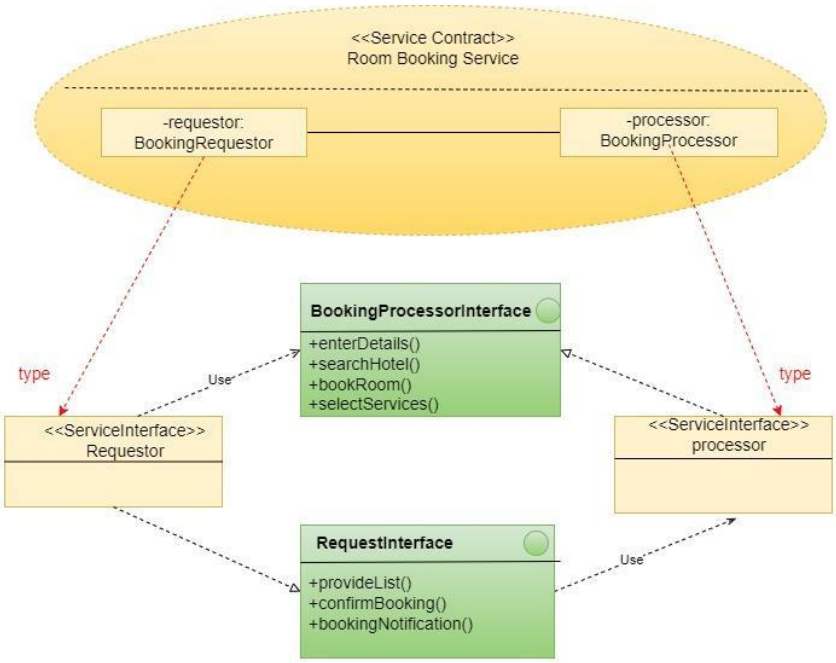
- **HotelSelection():** Allows users to select location. Hotel, date of arrival, date of departure
- **AddServices():** Allows users to add or modify services for the duration of the stay
- **SendConfirmation()**: Sends confirmation to the user once booking is done and payment is made.

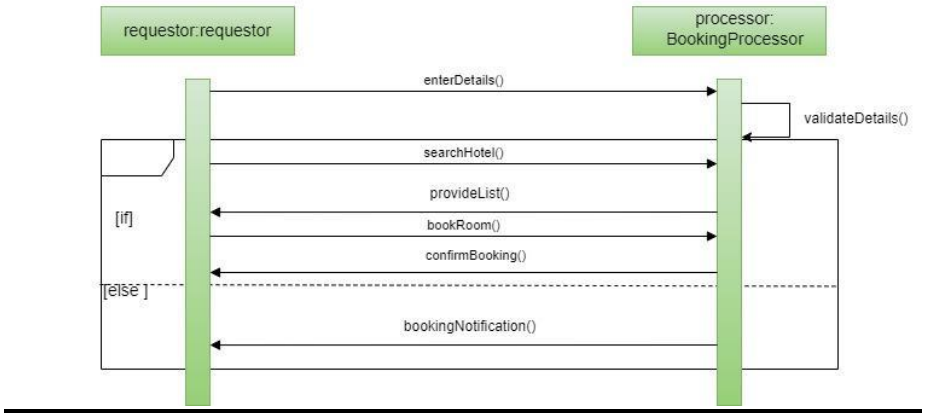
Service Contracts: Agreements between service providers and consumers

Service Architecture

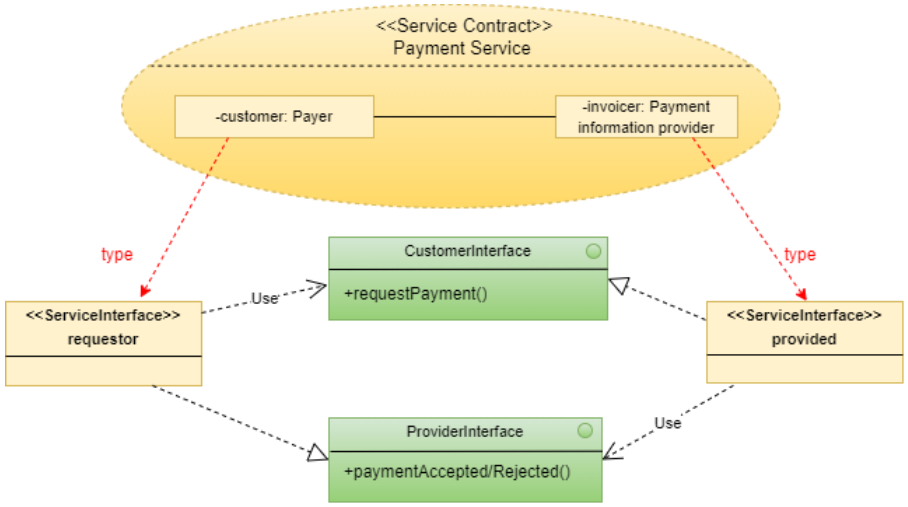


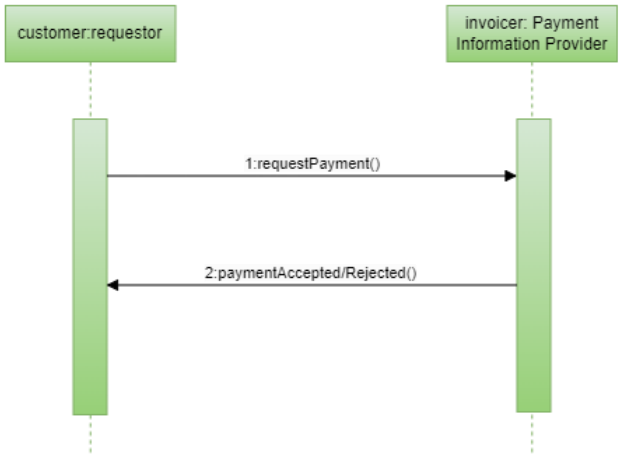
Service Contract-Booking Service





Service Contract-Payment Service





Service 2: Guest Services

Service Participants:

- **Guest service System (Service Providers)**:
- **Customers(Service Consumers)**

Service Architecture: High level structure of how components interact

- Illustrate the interaction between components:
    - Guest service System
    - Payment Gateway
    - Notification System
    - Customer database

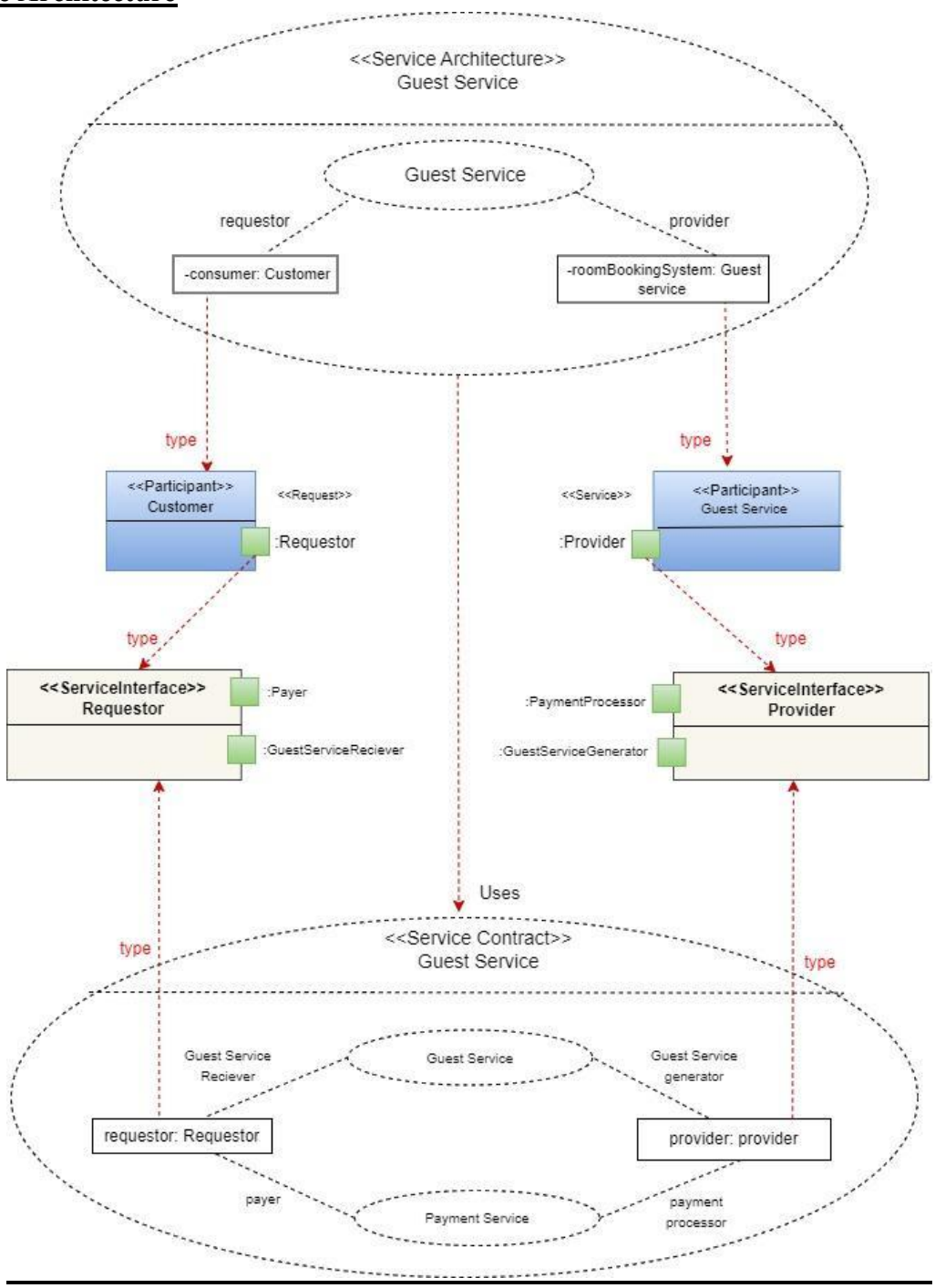
Service Interface: Operations provided by the participants

SelectService(): Allows the user to choose between room service, concierge services, housekeeping requests, morning calls, etc.

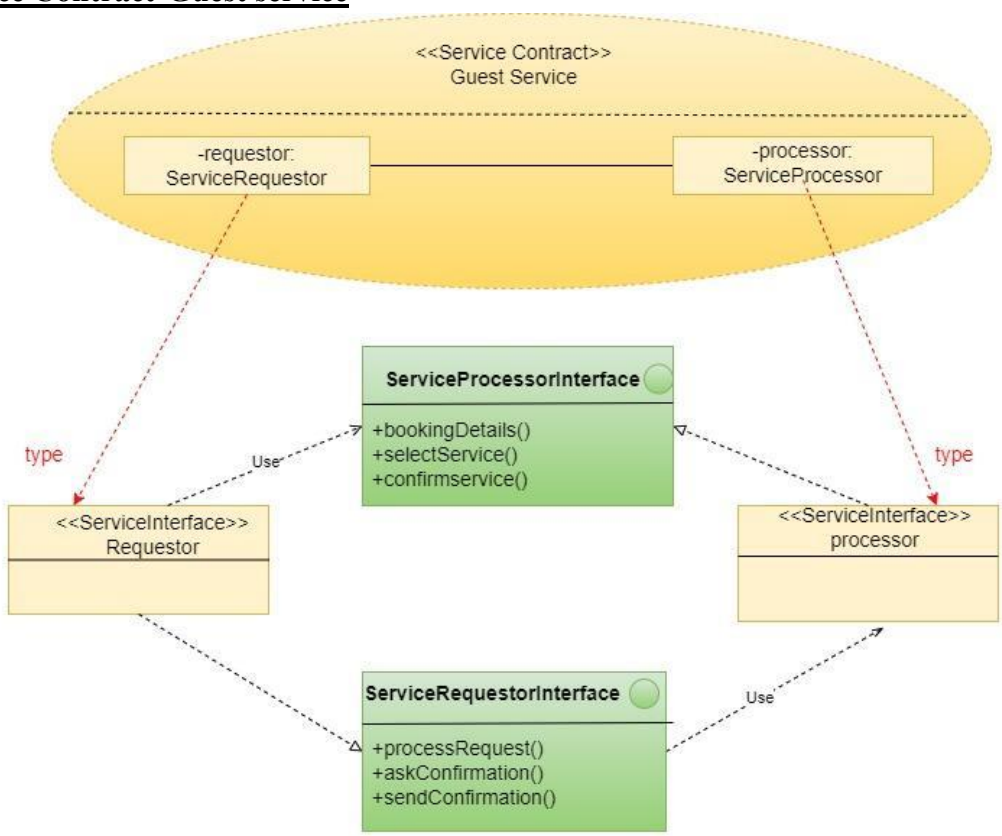
SendConfirmation(): Sends confirmation to the user once booking is done and payment is made.

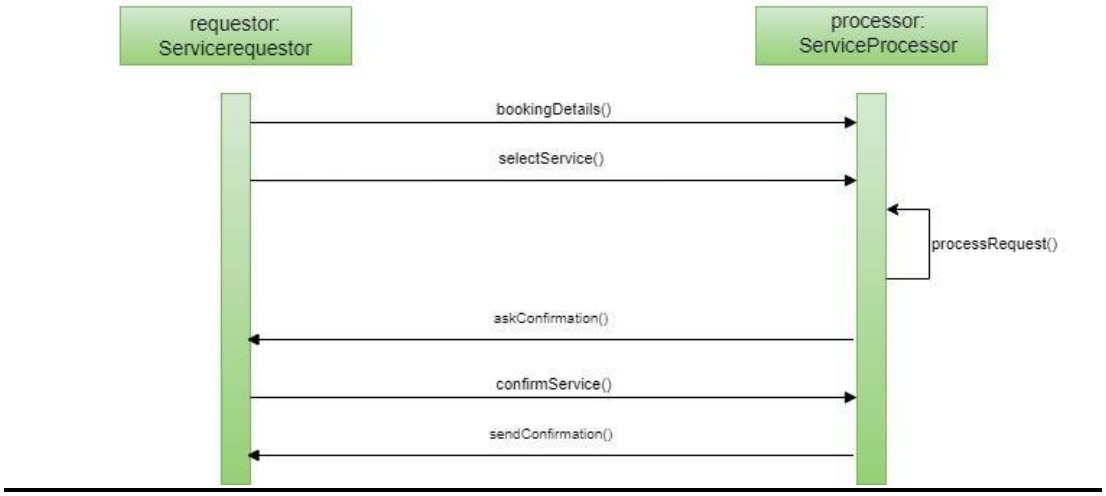
Service Contracts: Agreements between service providers and consumers

Service Architecture

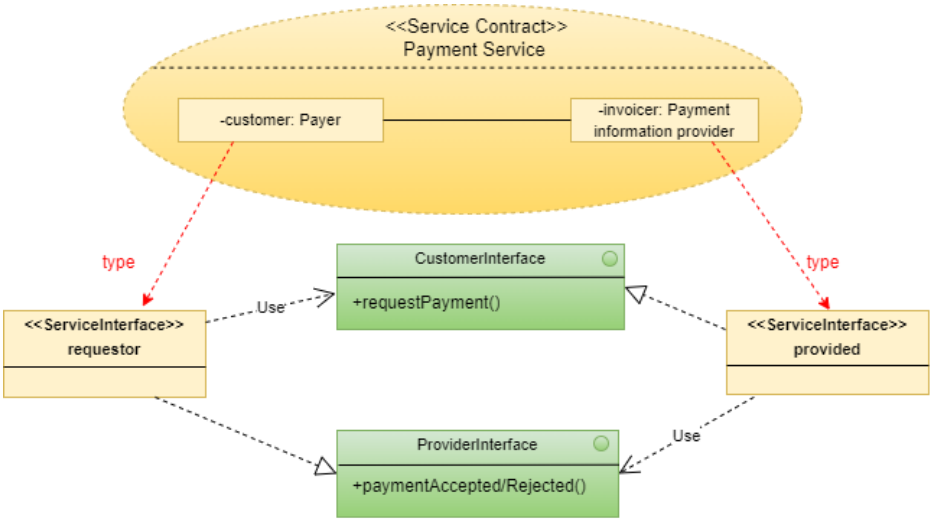


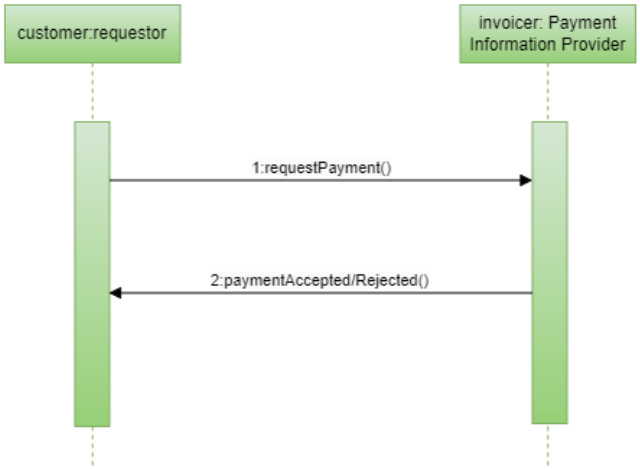
Service Contract-Guest service





Service Contract-Payment Service





Service 3: Feedback and reviews

Service Participants:

- **Feedback and Review System (Service Providers)**: Ask the user to review the availed services and their experience
- **Customers(Service Consumers):** Provide the feedback and the review

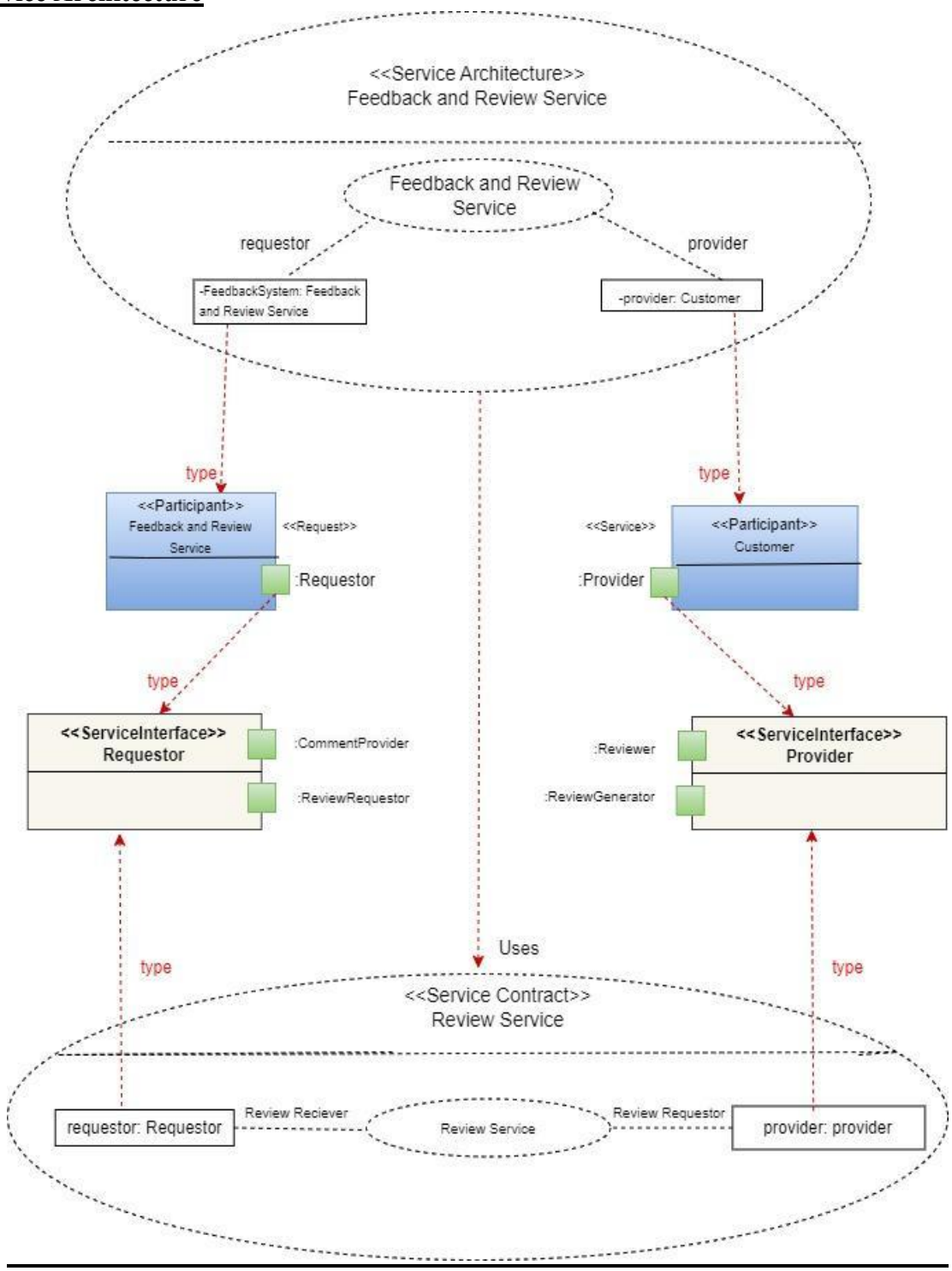
Service Architecture: High level structure of how components interact

- Illustrate the interaction between components:
    - Feedback and Review system
    - Customer database

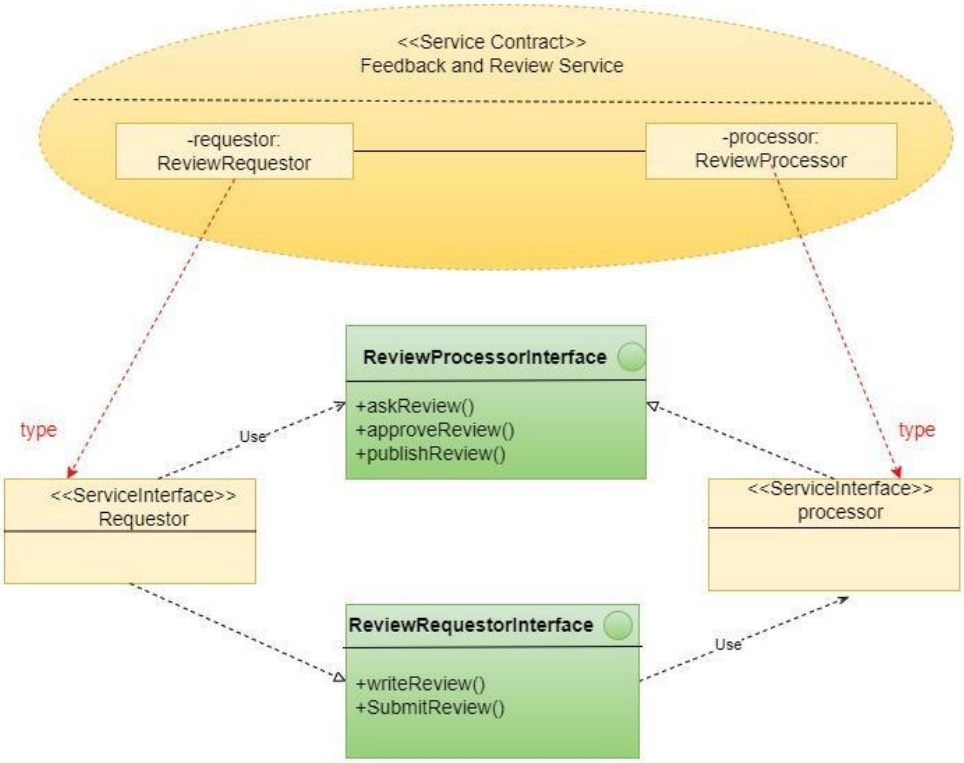
Service Interface: Operations provided by the participants
AskReview(): Allows user to send reviews for being published
PublishReview(): Published the review to the website

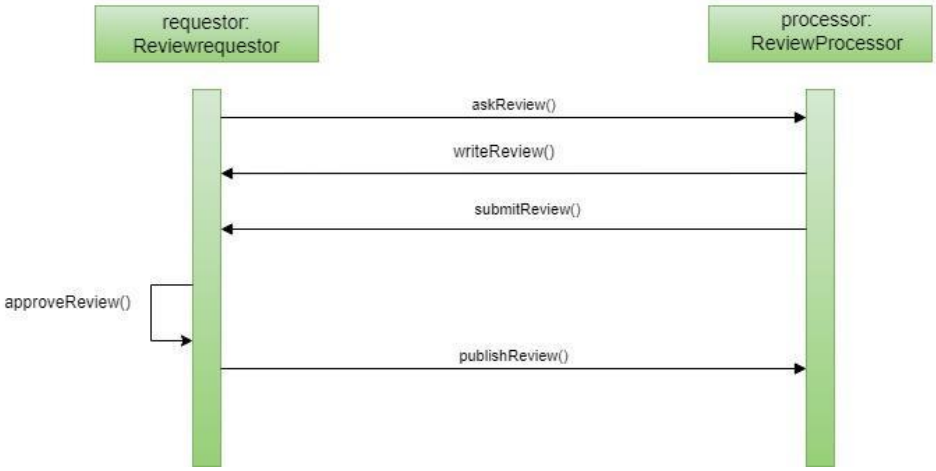
Service Contracts: Agreements between service providers and consumers

Service Architecture



Service Contract





Service Architecture 4 - Restaurant and Cafe (RC)

Service 1: Table Reservation

Service Participant:

- **Customers (Service Consumer)**: Service consumers who use the reservation system to book tables.
- **Reservation system (Service Provider)**: The service provider is responsible for offering the table reservation service.

Service Architecture: High-level structure of how the components interact

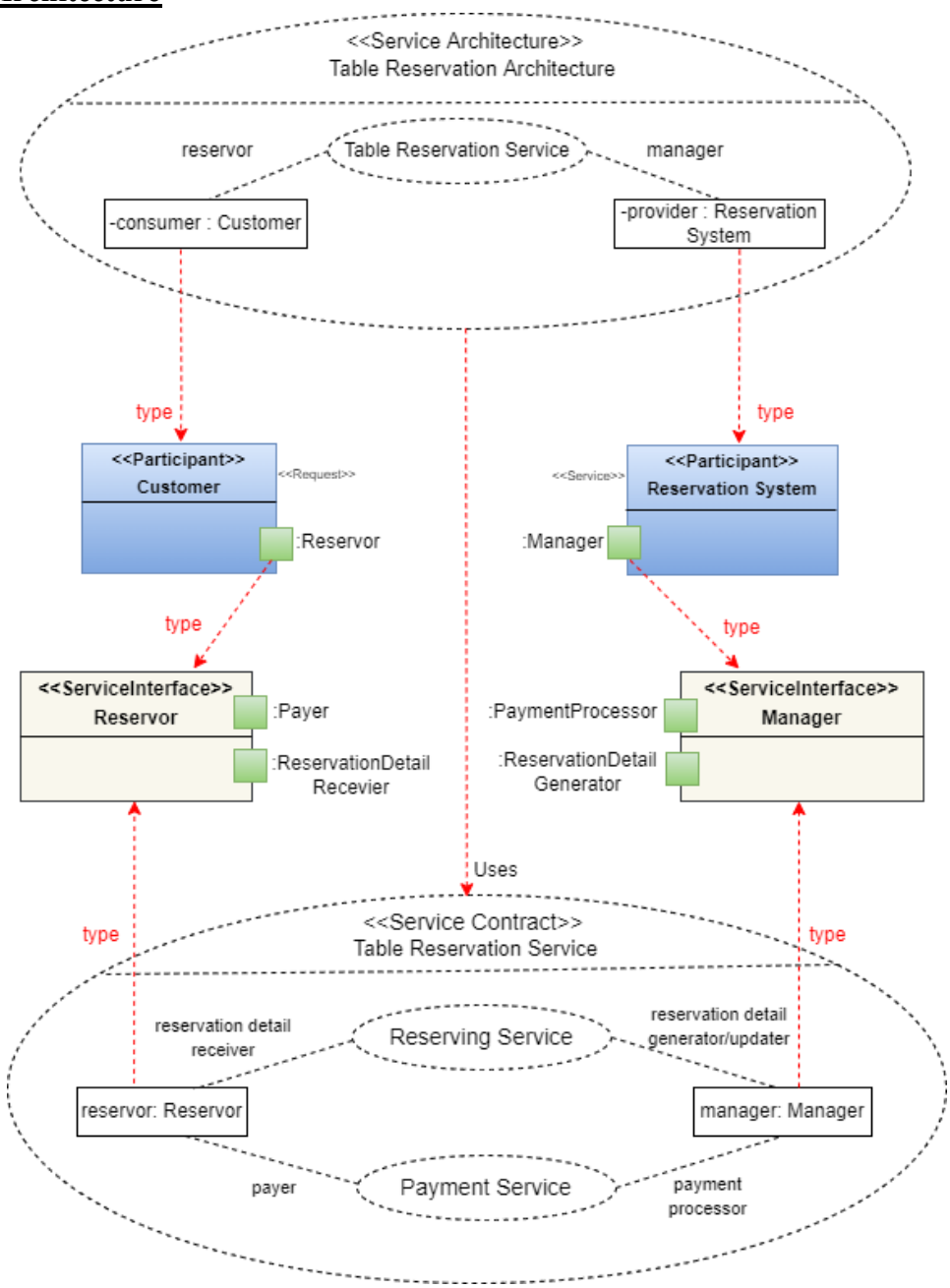
- Illustrate the interaction between components:
    - Reservation system
    - Payment Gateway
    - Notification System
    - Customer Database

Service Interface: Operation provided by the participants

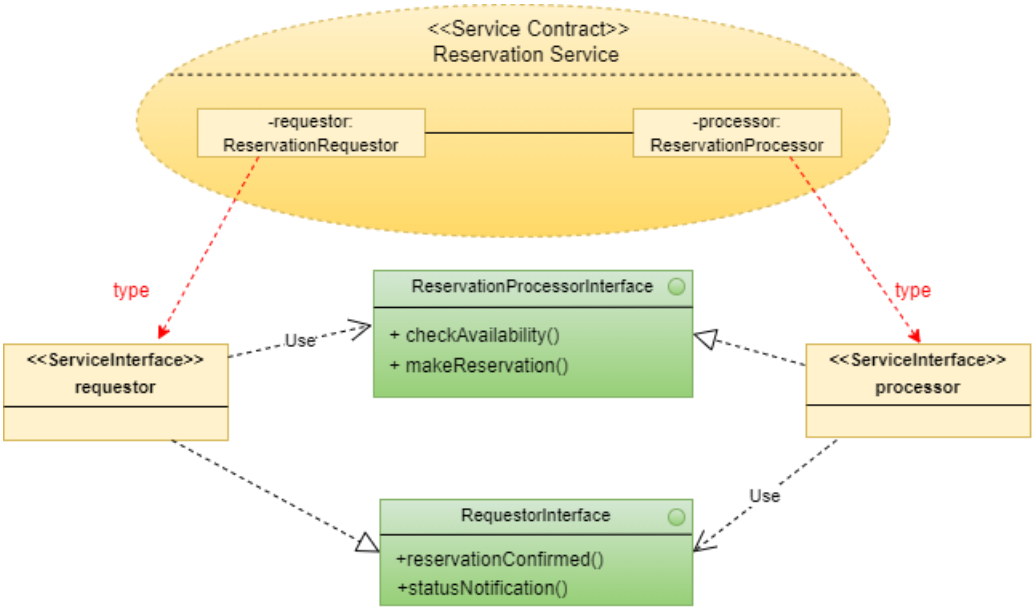
- **MakeReservation():** Takes inputs like restaurant ID, date, time, number of guests, customer details, payment info, etc., and returns outputs like ReservationID, Status, and PaymentReceipt.
- **CheckAvailability():** Check whether the table is available at the given date and time
- **SendConfirmation():** send confirmation details to the customer

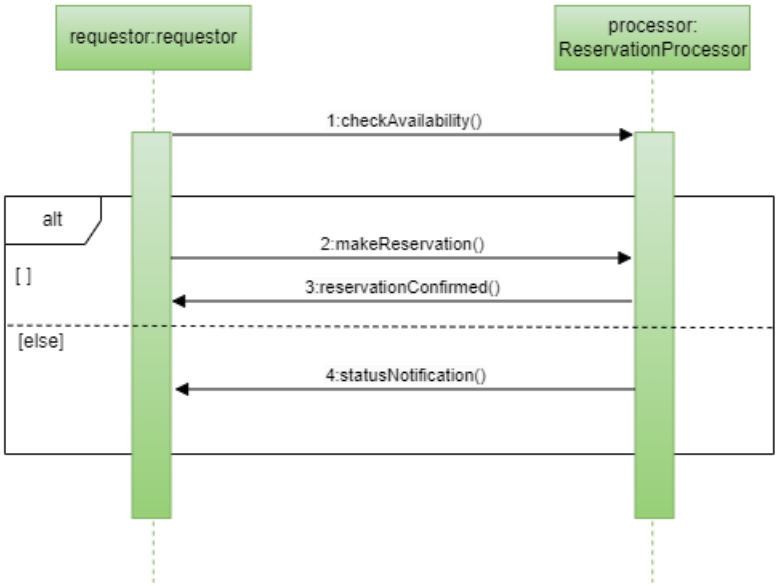
Service Contracts: Agreements between service providers and consumers

Service Architecture

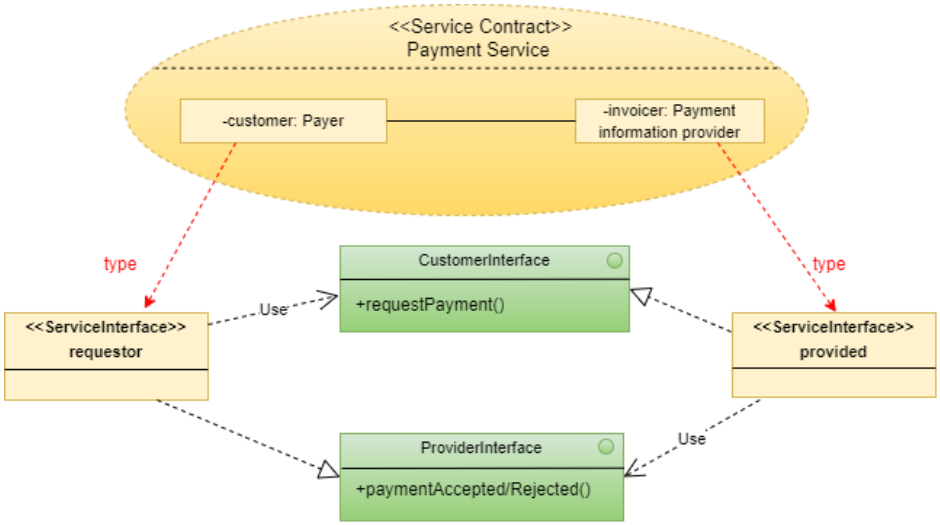


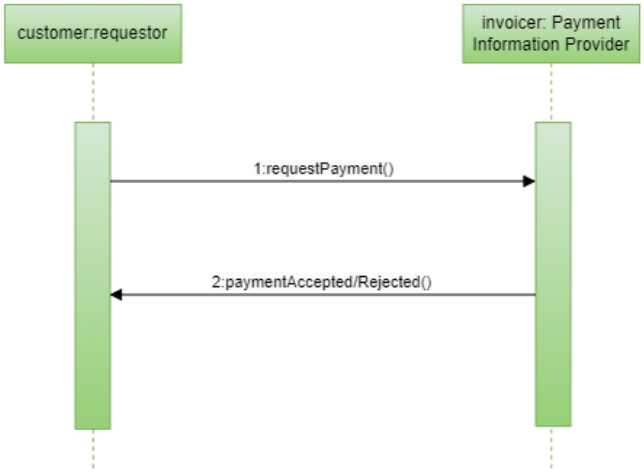
SERVICE CONTRACT - Reservation Service





SERVICE CONTRACT - Payment Service





Service 2: Menu Recommendation

Service Participant:

- **Customers (Service Consumer)**: Service consumers who use the menu recommendation system to view menu lists.
- **Menu Recommendation System (Service Provider)**: The service provider recommends menu list items.

Service Architecture: High-level structure of how the components interact

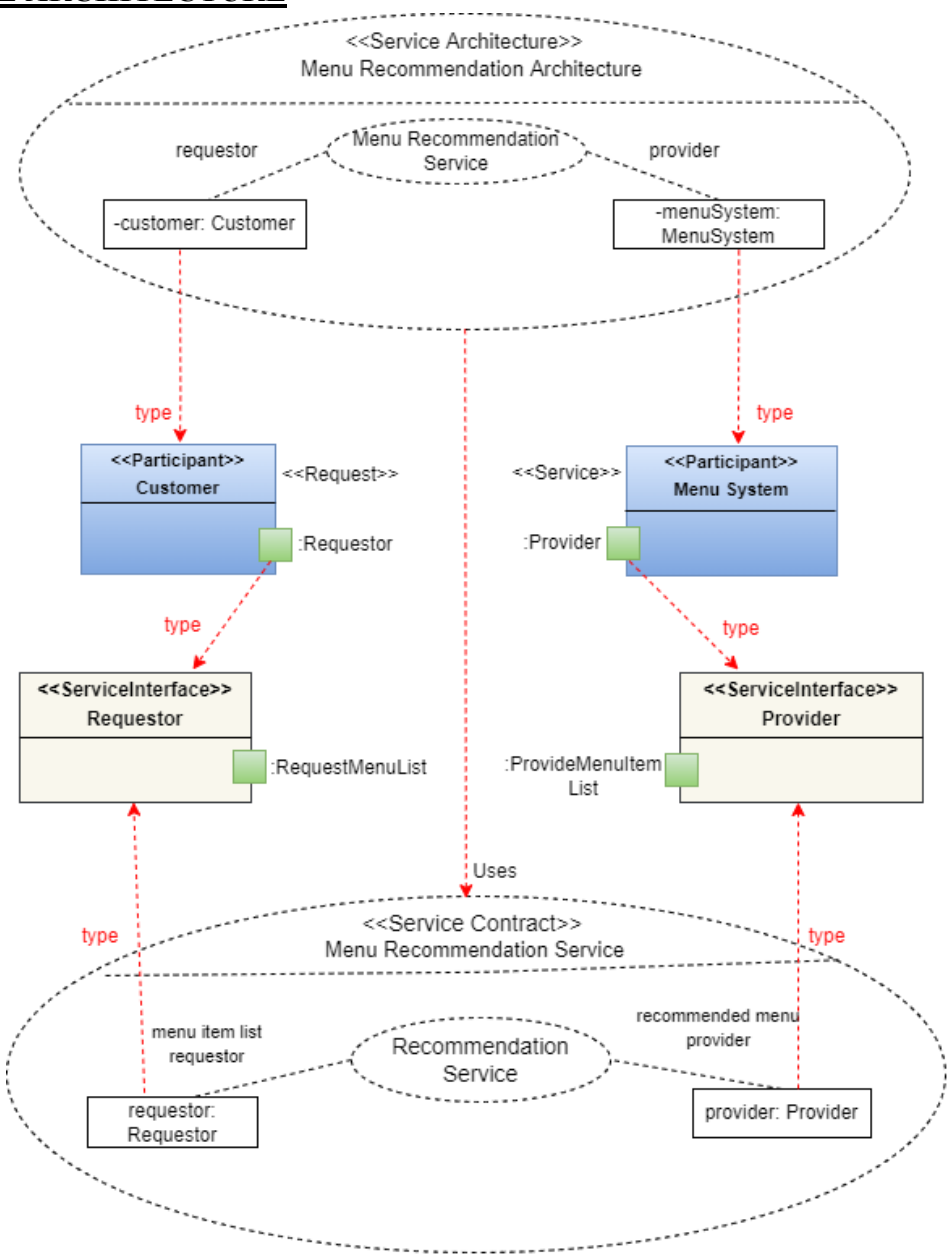
- Illustrate the interaction between components:
    - Recommendation system
    - MenuItem Database

Service Interface: Operation provided by the participants

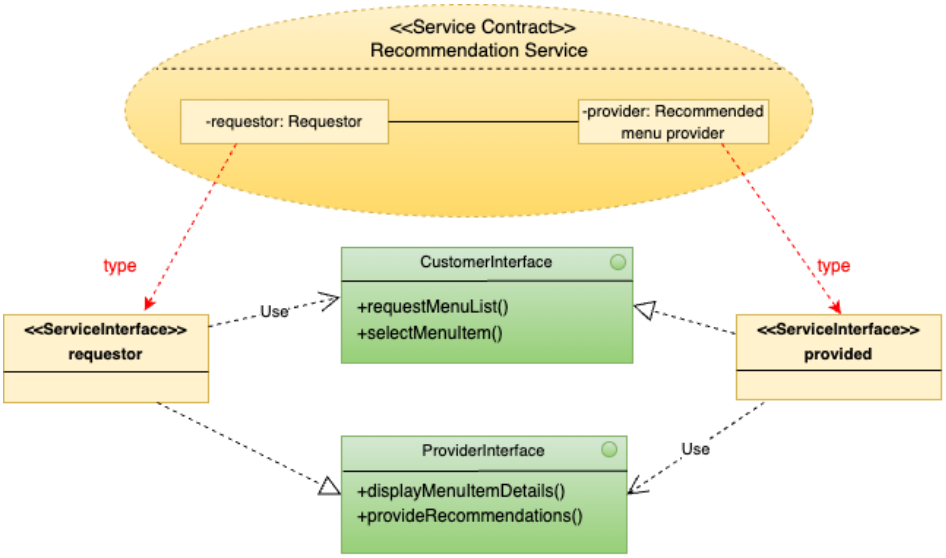
- requestMenuList()
- displayMenuItemDetails()
- selectMenuItems()
- provideRecommendation()

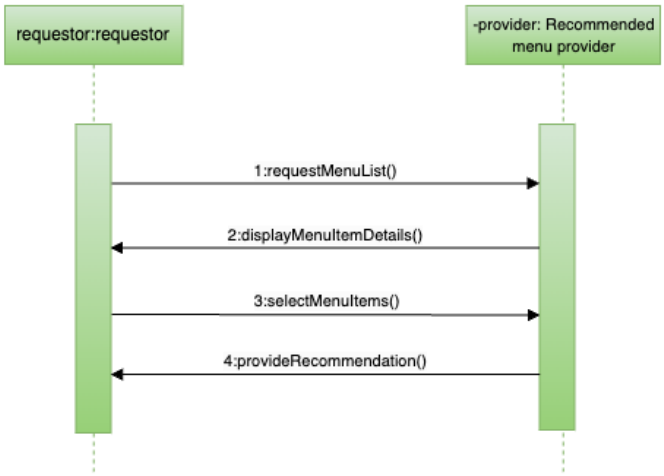
Service Contracts: Agreements between service providers and consumers

SERVICE ARCHITECTURE



SERVICE CONTRACT - Recommendation Service





Service 3: Online ordering

Service Participant:

- **Customers (Service Consumer)**: Service consumers who use the online ordering system to order online
- **Online Ordering System (Service Provider)**: The service provider is responsible for offering the online ordering service.

Service Architecture: High-level structure of how the components interact

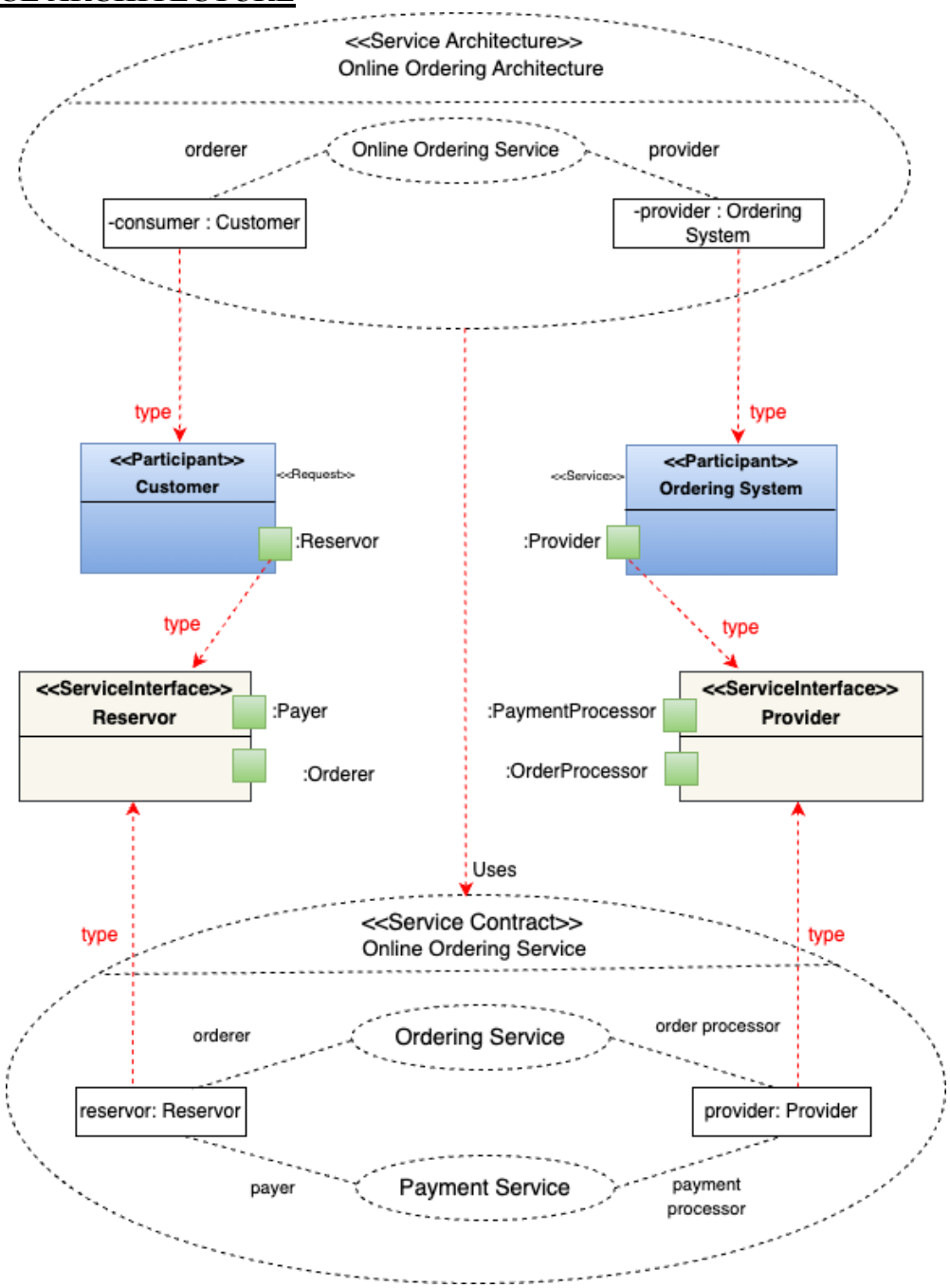
- Illustrate the interaction between components:
    - Ordering system
    - Payment Gateway
    - Customer Database

Service Interface: Operation provided by the participants

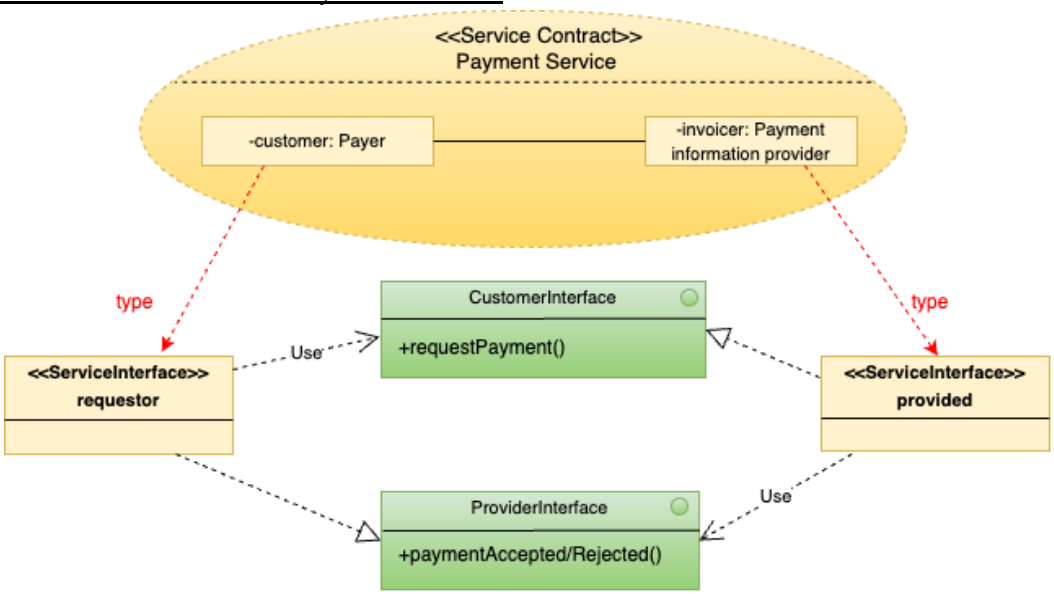
- **fullfillOnlineOrder()**
- **orderScheduled()**
- **orderRejected()**
- **requestPayment()**
- **requestAccept/Rejected()**

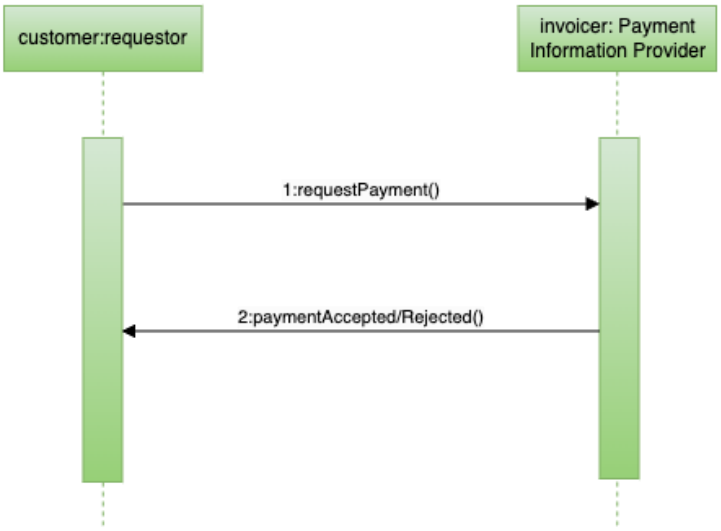
Service Contracts: Agreements between service providers and consumers

SERVICE ARCHITECTURE

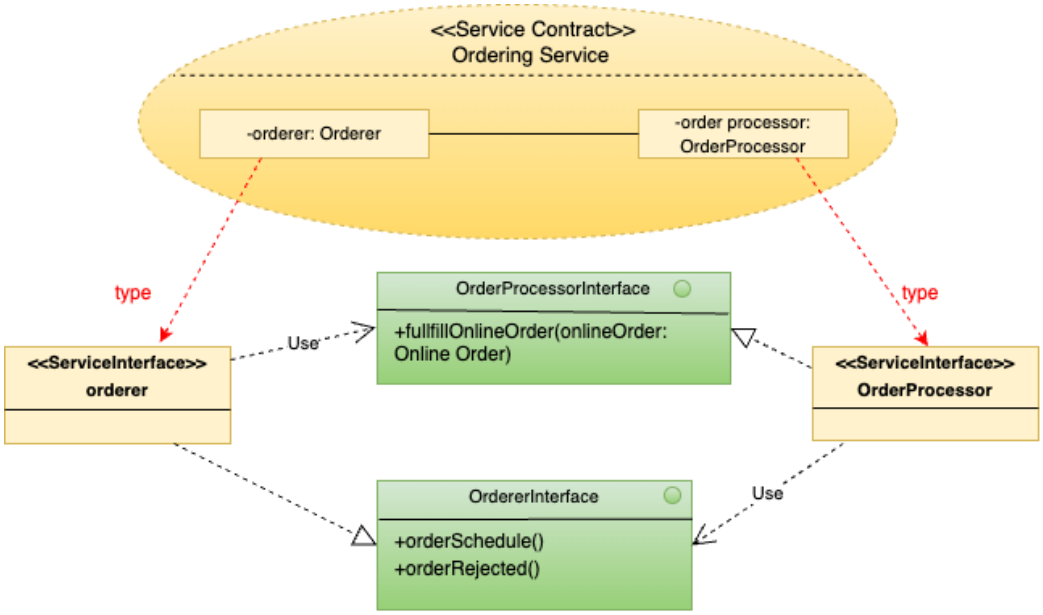


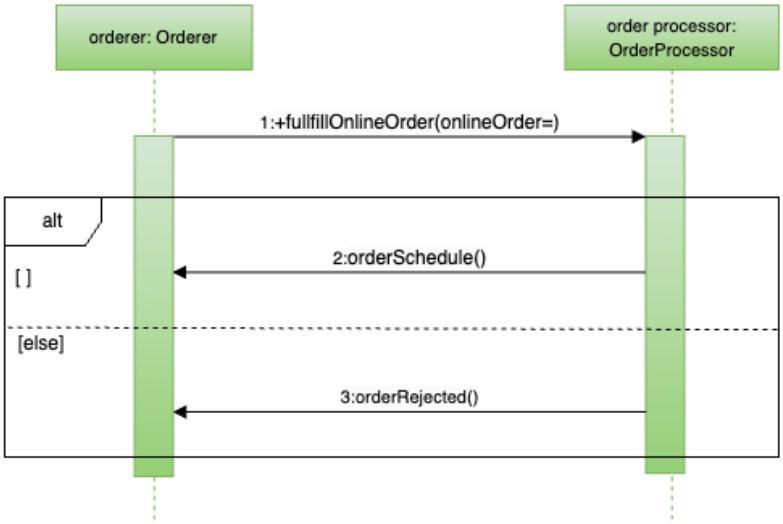
SERVICE CONTRACT - Payment Service





SERVICE CONTRACT - Ordering Service


<<Service Contract>>
Ordering Service
-orderer: Orderer
-order processor: OrderProcessor
type
type
OrderProcessorInterface
+fullfillOnlineOrder(onlineOrder: Online Order)
Use
<<ServiceInterface>>
orderer
<<ServiceInterface>>
OrderProcessor
Use
OrdererInterface
+orderSchedule()
+orderRejected()


orderer: Orderer
order processor: OrderProcessor
1:+fullfillOnlineOrder(onlineOrder=)
alt
[ ]
2:orderSchedule()
[else]
3:orderRejected()

Service Architecture 5 - Outdoor Activity Centers(OAC)

Service 1: Booking Activity

Service Participant:

- **Customer (Service Consumer)**: Service consumers who use the booking system to book events.
- **Booking system (Service Provider)**: The service provider is responsible for offering the activity booking service.

Service Architecture: High-level structure of how the components interact

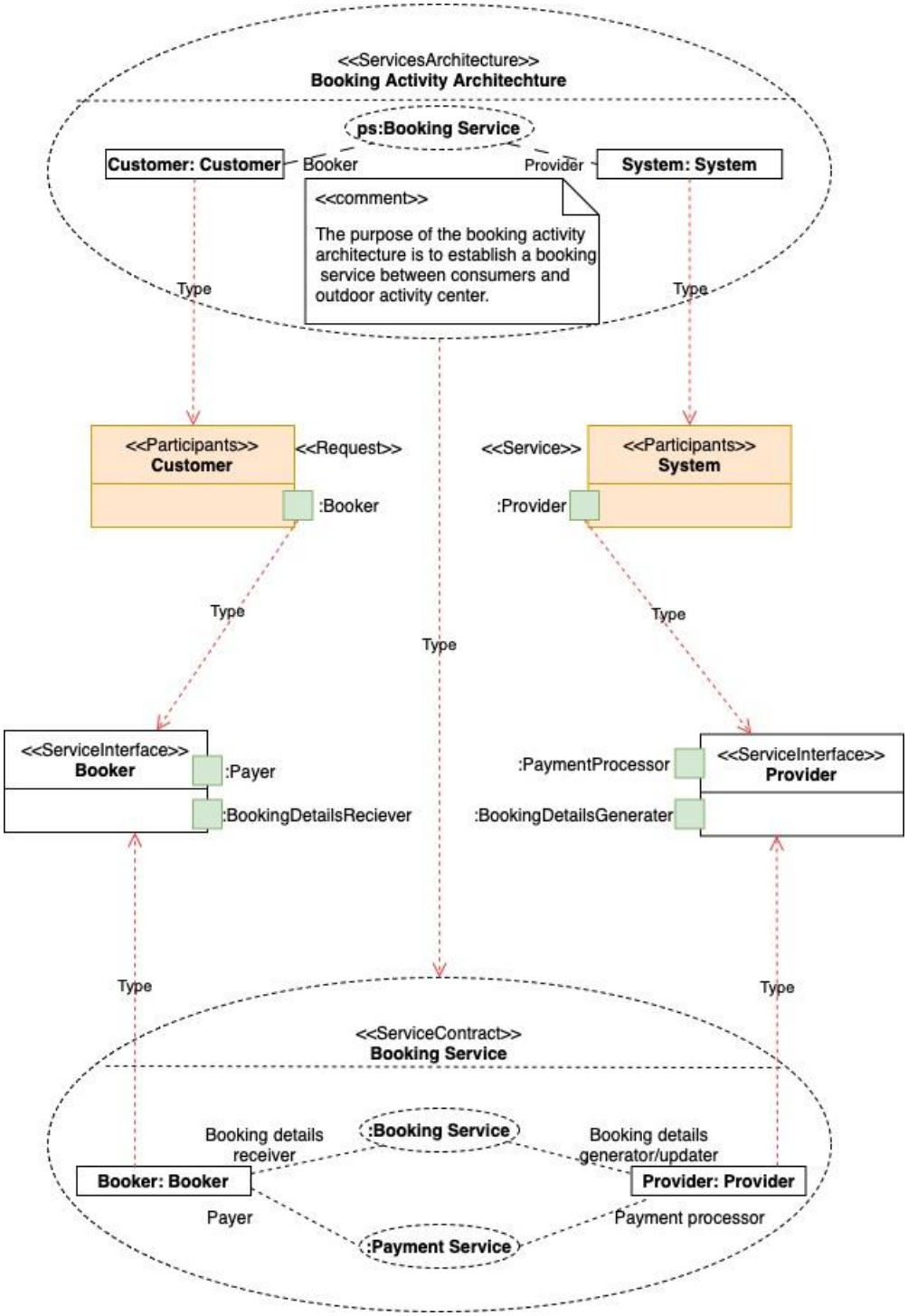
- Illustrate the interaction between components:
    - Payment Gateway
    - Database System (DBMS)
    - Notification System

Service Interface: Operation provided by the participants

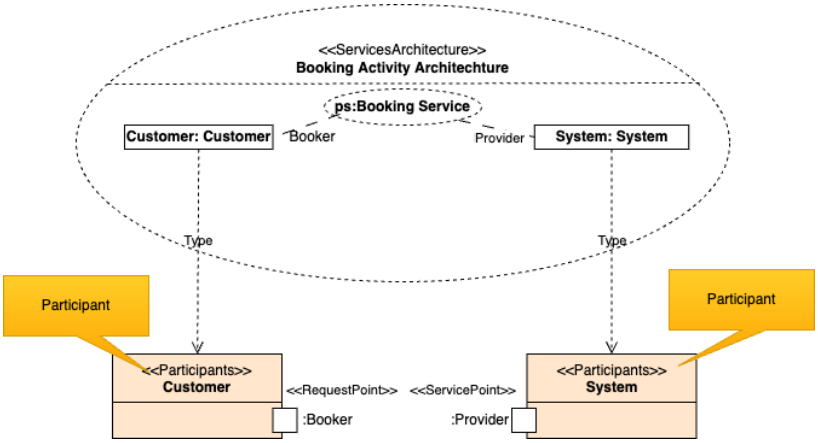
- **submitBooking():** After entering the event participant and payment information, the booker sends a booking request to the booking service.
- **checkAvailability():** the system checks whether the event booking is available at the given information.
- **sendBookingDetails():** the system sends the booking details to the customer.
- **processPayment():** the system processes user payment requests.
- **updateStatus():** the system updates the status of the booking request after the payment is successful.

Service Contracts: Agreements between service providers and consumers

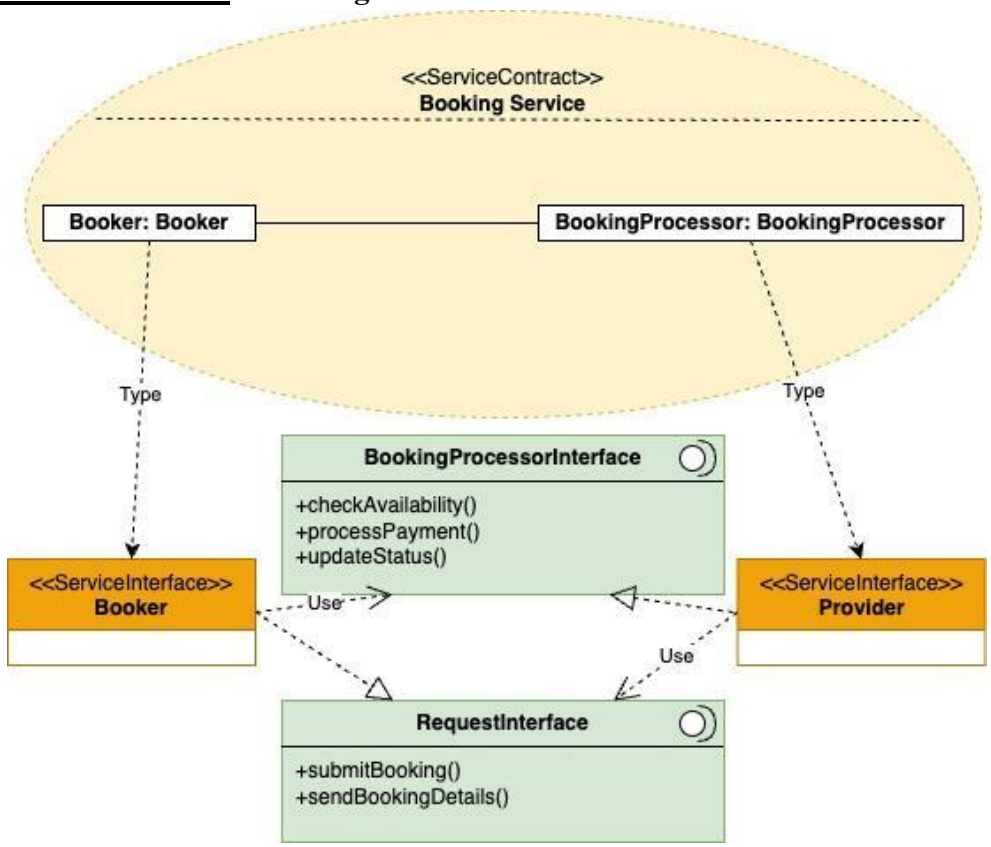
SERVICE ARCHITECTURE


<<ServicesArchitecture>>
Booking Activity Architechture
ps:Booking Service
Customer: Customer
Booker
Provider
System: System
<<comment>>
The purpose of the booking activity architecture is to establish a booking service between consumers and outdoor activity center.
Type
Type
<<Participants>>
Customer
<<Request>>
:Booker
<<Service>>
<<Participants>>
System
:Provider
Type
Type
Type
<<ServiceInterface>>
Booker
:Payer
:BookingDetailsReciever
:PaymentProcessor
<<ServiceInterface>>
Provider
:BookingDetailsGenerater
Type
Type
<<ServiceContract>>
Booking Service
Booking details receiver
:Booking Service
Booking details generator/updater
Booker: Booker
Provider: Provider
Payer
Payment processor
:Payment Service

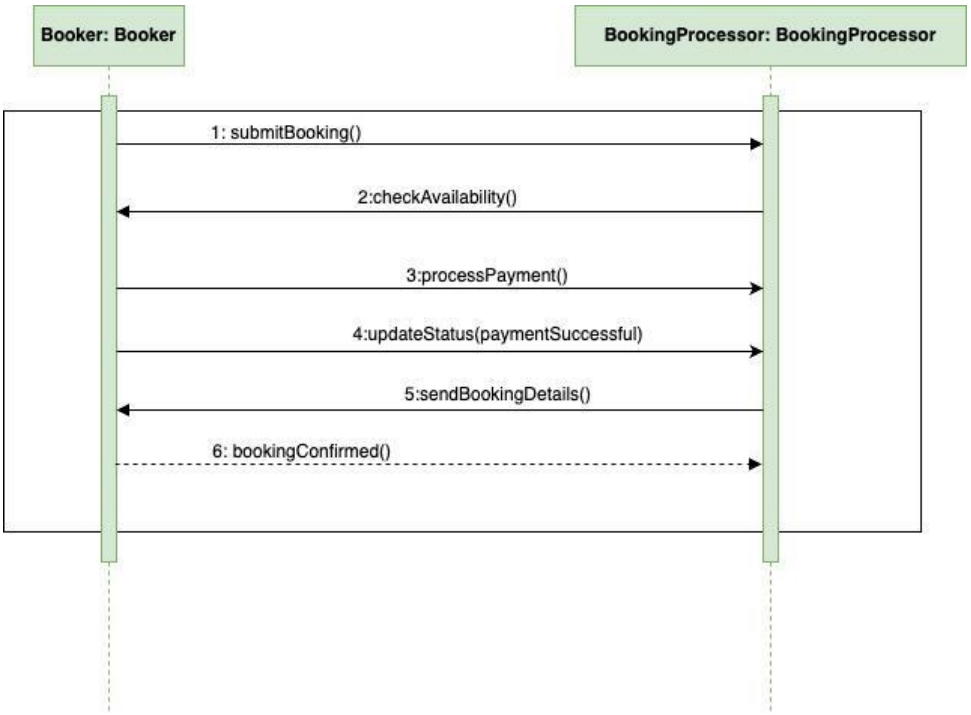
SERVICE PARTICIPANTS

Description	Service Participant
• **Customer as a role for booker** in the process of this service, and the port of this participant is all bookers. It sends requests and receives information; • **This system is another role for service providers** in the process of this service, and the port of this participation is all service providers. It deals with requests which are from the customer.	

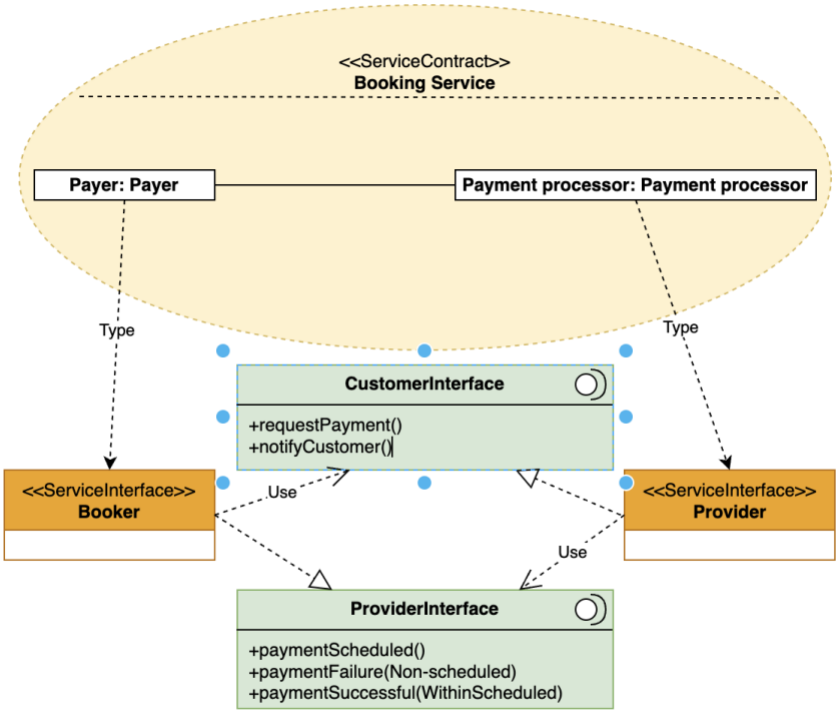
SERVICE CONTRACT - Booking Service



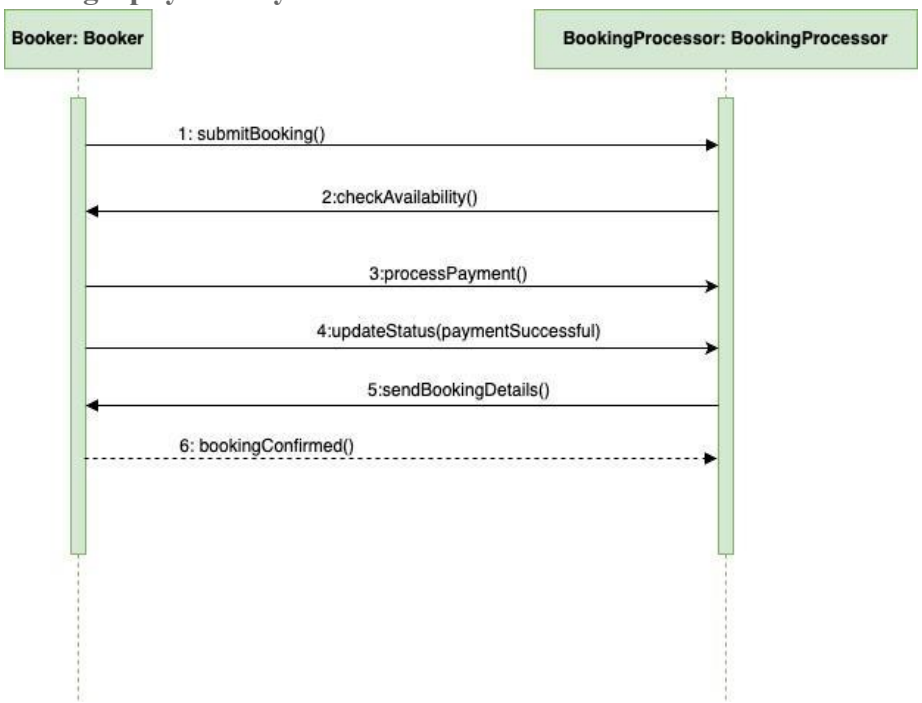
Protocol Choreography for Booking Service Contract



SERVICE CONTRACT - Payment Service



Protocol Choreography for Payment Service Contract



Service 2: Equipment Lease

Service Participant:

- **User (Service Consumer)**: A participant in the tourism ecosystem who rents equipment.
- **Equipment Rental Service (Service Provider)**: The main service that allows users to search for, reserve, and lease equipment. It also handles the rental agreement.

Service Architecture: High-level structure of how the components interact

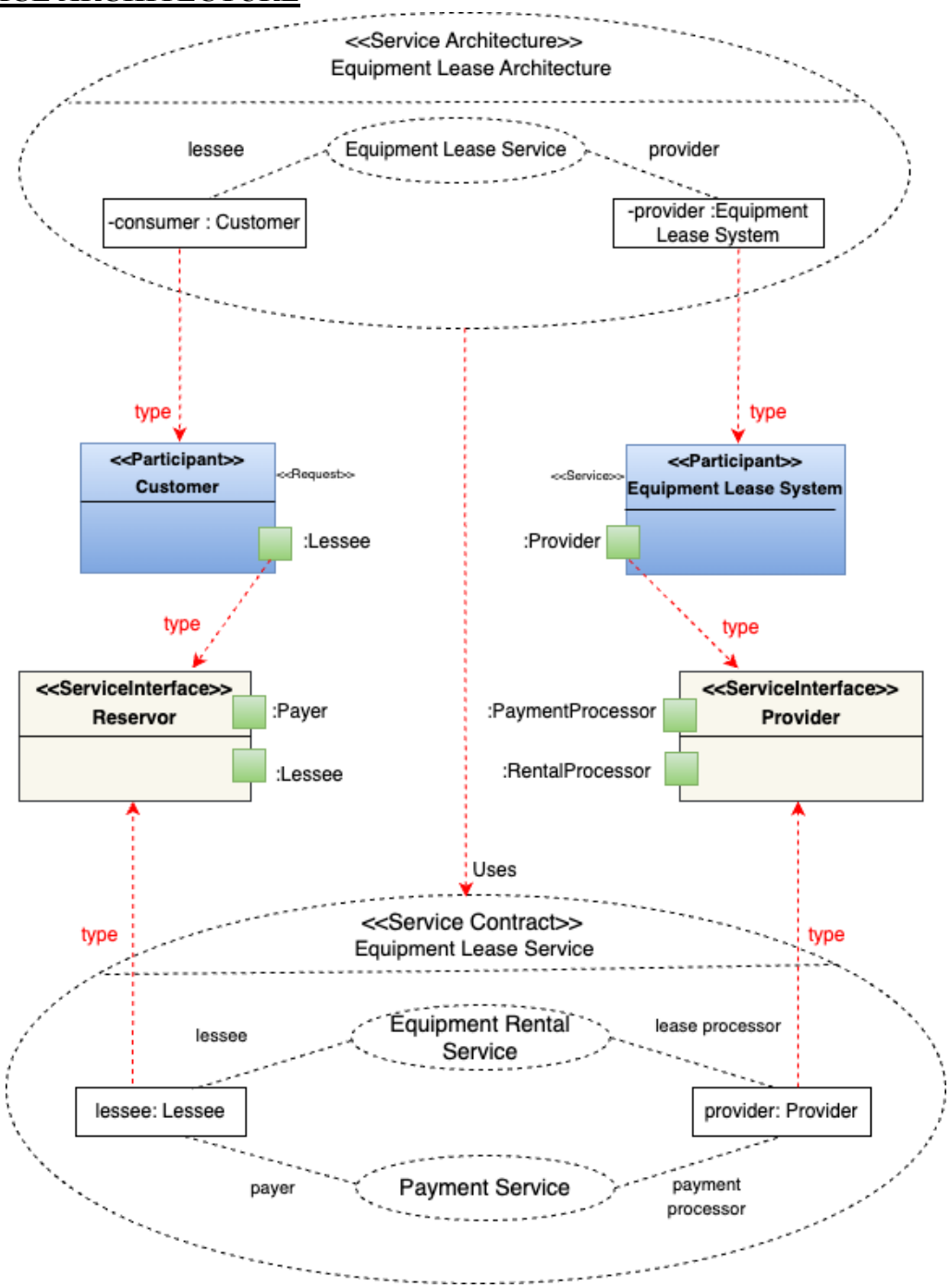
- Illustrate the interaction between components:
    - Equipment Rental system
    - Payment Gateway
    - Notification System

Service Interface: Operation provided by the participants

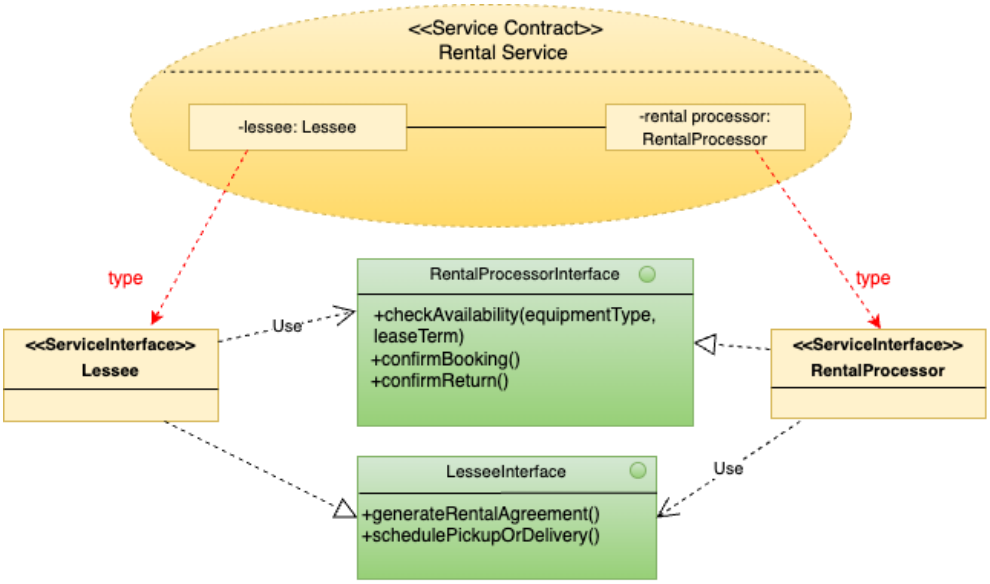
- **check availability()**
- **generateRentalAgreement()**
- **process payment()**

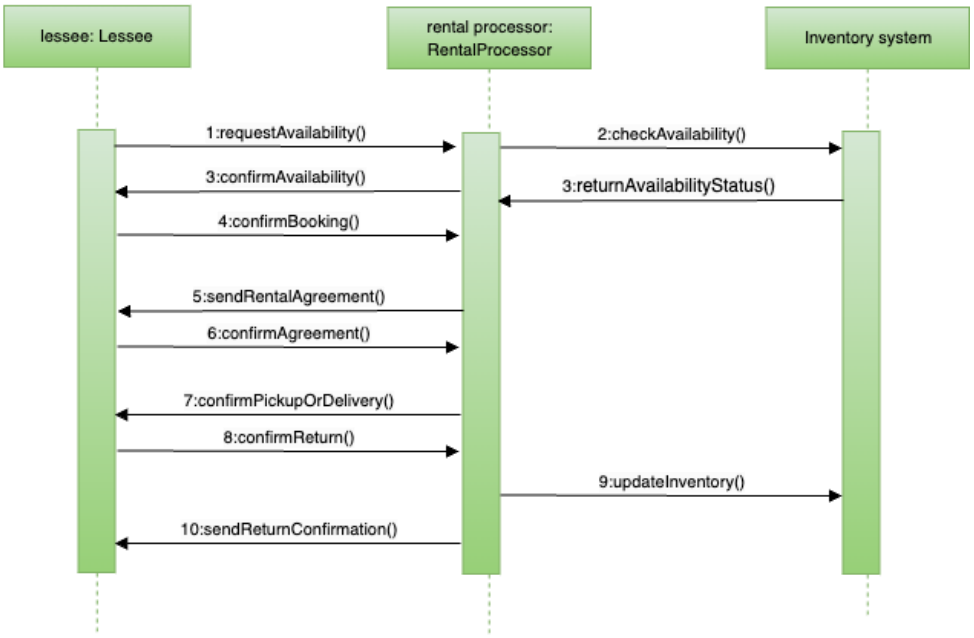
Service Contracts: Agreements between service providers and consumers

SERVICE ARCHITECTURE

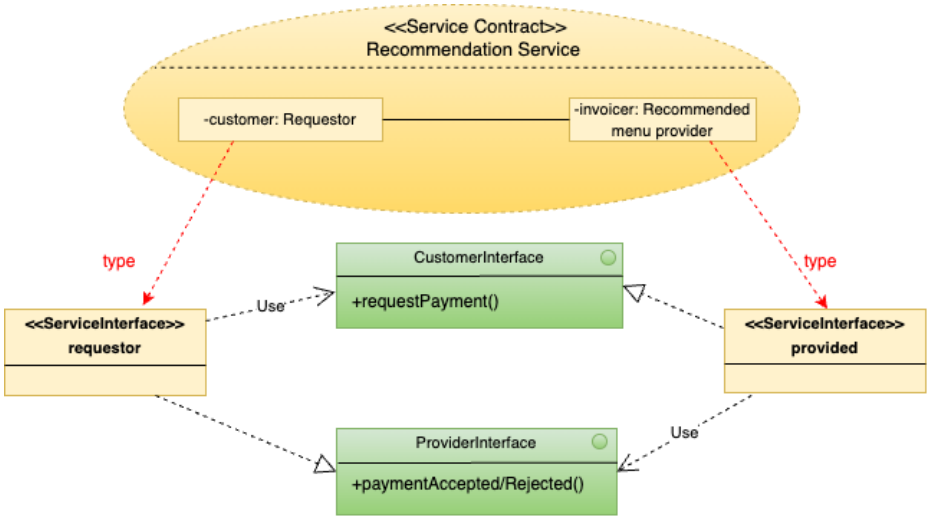


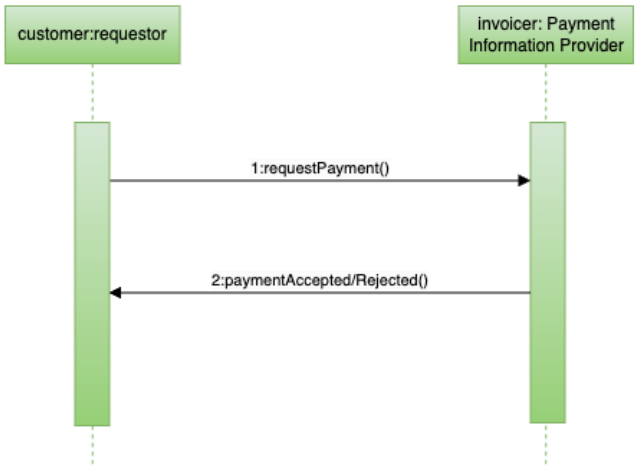
SERVICE CONTRACT - Rental Service





SERVICE CONTRACT - Payment Service


<<Service Contract>>
Recommendation Service
-customer: Requestor
-invoicer: Recommended menu provider
type
type
CustomerInterface
+requestPayment()
Use
<<ServiceInterface>>
requestor
<<ServiceInterface>>
provided
ProviderInterface
+paymentAccepted/Rejected()
Use


customer:requestor
invoicer: Payment Information Provider
1:requestPayment()
2:paymentAccepted/Rejected()

Service 3: Safety Guide

Service Participant:

- **Customer (Service Consumer)**: A participant in the tourism ecosystem requesting safety guidance.
- **Safety Guide Service (Service Provider)**: The main service for processing requests, getting data, and providing safety guidelines, emergency contracts, etc.

Service Architecture: High-level structure of how the components interact

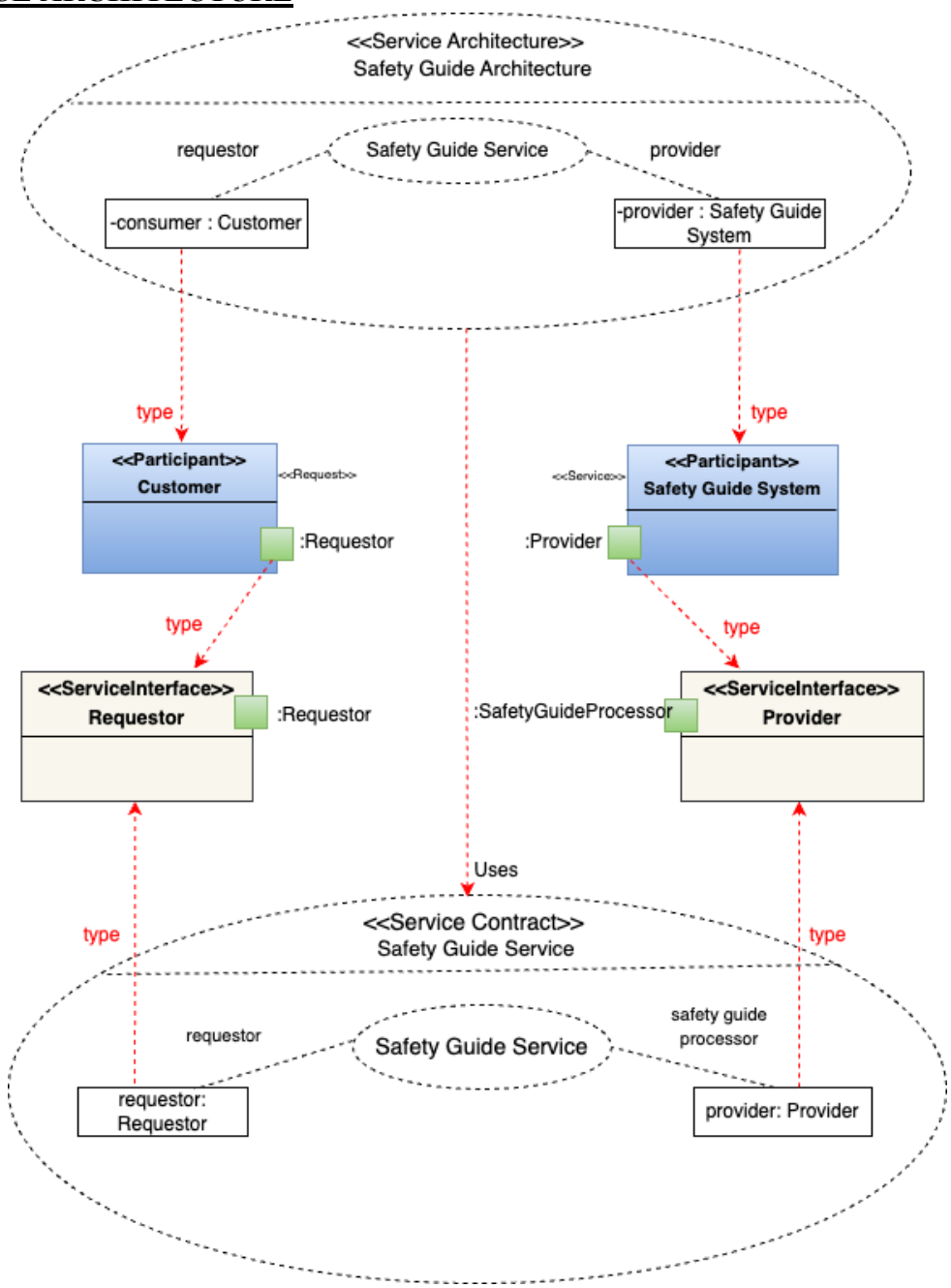
- Illustrate the interaction between components:
    - **Safety Guide** system
    - Database System

Service Interface: Operation provided by the participants

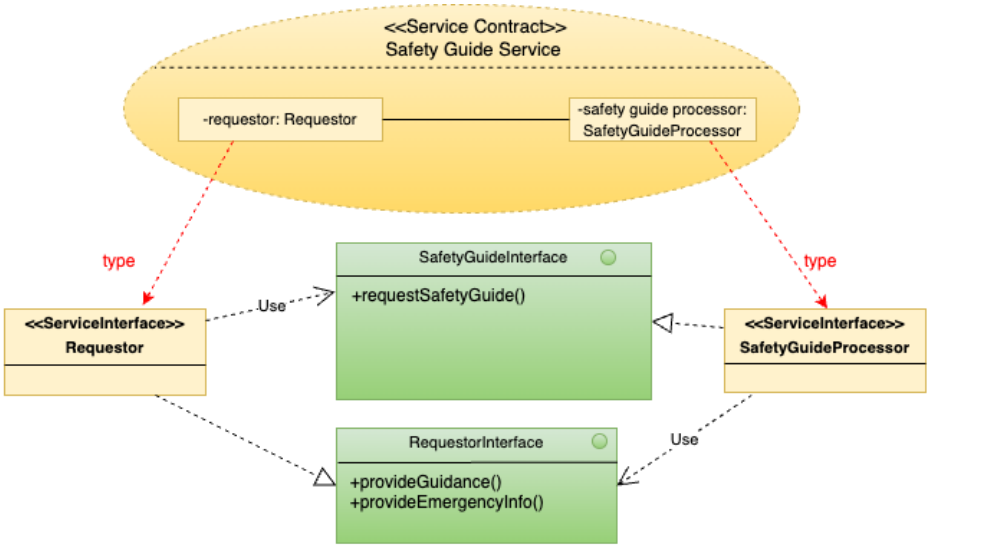
- **requestSafetyGuide()**
- **safetyInfo()**
- **provideGuidance()**

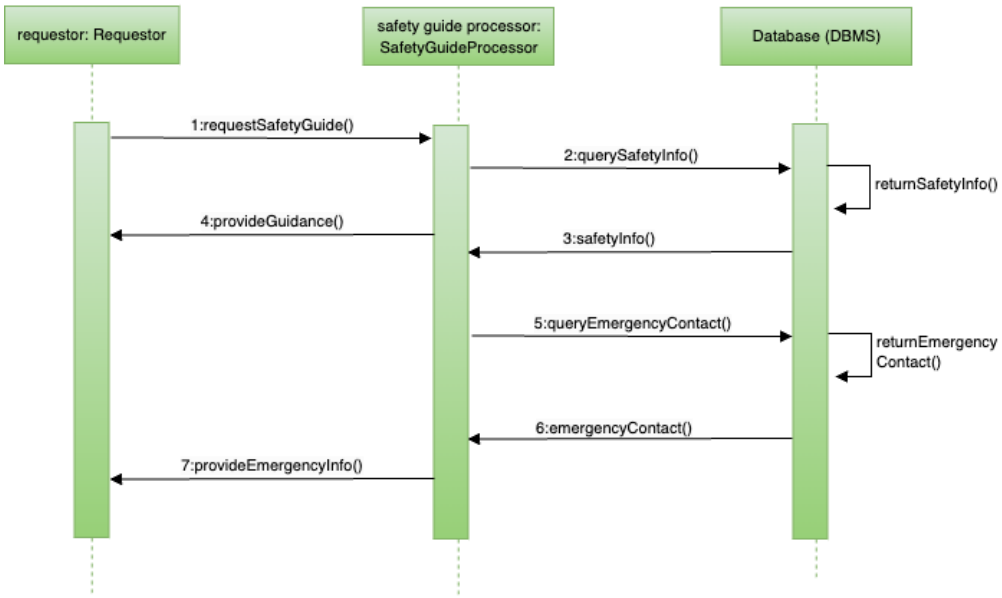
Service Contracts: Agreements between service providers and consumers

SERVICE ARCHITECTURE



SERVICE CONTRACT - SAFETY GUIDE SERVICE

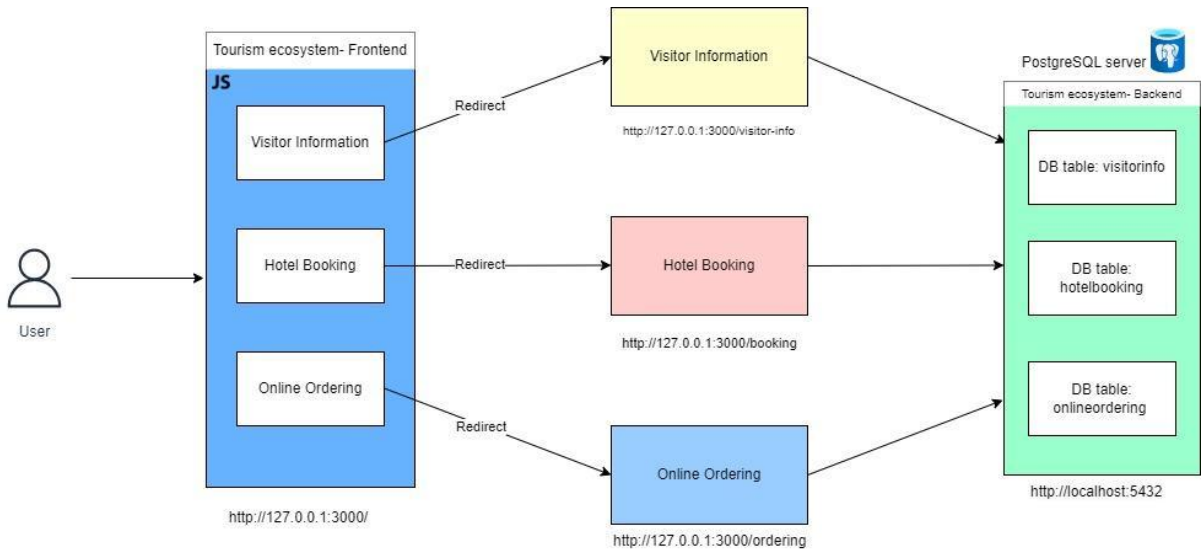




TASK E. Microservices Design and Implementation

In this section, we detail the design and implementation of three critical microservices within the Restaurant and Cafe segment of our Tourism Ecosystem: **Room Booking**, **Online Ordering**, and **Visitor information**. This comprehensive approach ensures scalability, maintainability, and resilience, aligning with service-oriented software engineering principles.

Microservices Overview



The Tourism Ecosystem leverages a microservices architecture to manage distinct functionalities independently. Focusing on the Restaurant and Cafe services, the following microservices are identified:

1. Visitor Information Service
2. Room(Hotel) Booking service
3. Online Ordering service

Each microservice is designed to handle specific business capabilities, allowing for independent development, deployment, and scaling.

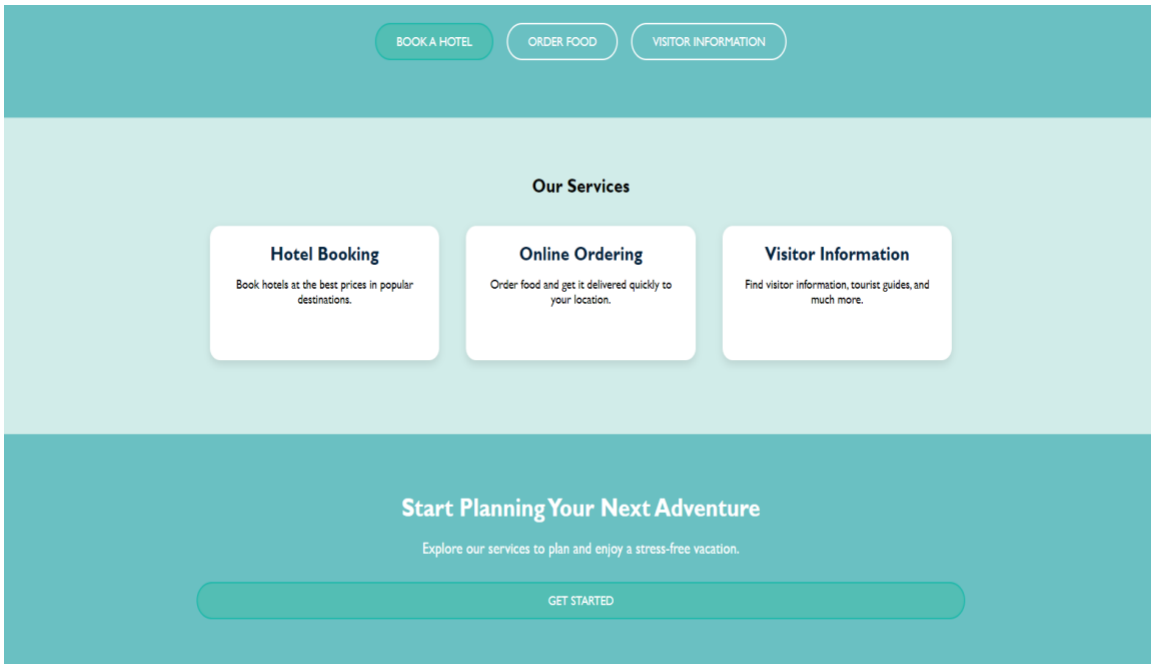
Programming languages and frameworks used in designing the microservices architecture:

1. Frontend: HTML, CSS & React
3. Microservices Hosting and deployment: Docker Containers and Kubernetes
4. API: Flask, Python
4. Database: PostgreSql (The tables are created using postgre SQL)
5. Code review: Github and VS Cod

Home Page

This is the home page for our website which takes the user to the three services, that is, click **BOOK A HOTEL, ORDER FOOD** and **VISITOR INFORMATION** to navigate to hotel booking, online ordering and visitor information page.

I. Frontend



II. Code

```
// src/components/Home/Home.js

import React from 'react';
import { Link } from 'react-router-dom';
import './home.css'; // Optional, for any specific styles

const Home = () => {
  return (
    <div className='home-css'>
      <div className="home">
        {/* Hero Section */}
        <section className="home-hero">
          <div className="container">
            <h1 className="hero-title">Welcome to the Tourism Ecosystem</h1>
            <p className="hero-text">
              Discover amazing places, book hotels, order delicious food, and explore visitor informat
            </p>
            <div className="btn-group">
              <Link to="/booking" className="btn btn-primary">Book a Hotel</Link>
              <Link to="/ordering" className="btn btn-secondary">Order Food</Link>
              <Link to="/visitor-info" className="btn btn-secondary">Visitor Information</Link>
            </div>
          </div>
        </section>

        {/* Services Section */}
        <section className="home-services">
          <div className="container">
            <h2 className="section-title">Our Services</h2>
            <ul className="service-list">
              <li>
                <h3>Hotel Booking</h3>
```

```
                <h3>Visitor Information</h3>
                <p>Find visitor information, tourist guides, and much more.</p>
                <Link to="/visitor-info" className="btn btn-secondary">View Information</Link>
              </li>
            </ul>
          </div>
        </section>

        {/* Call to Action Section */}
        <section className="home-cta">
          <div className="container">
            <h2 className="cta-title">Start Planning Your Next Adventure</h2>
            <p className="cta-text">Explore our services to plan and enjoy a stress-free vacation.</p>
            <Link to="/booking" className="btn btn-primary">Get Started</Link>
          </div>
        </section>
      </div>
      {/* <h1>Welcome to Our Service</h1> */}
      {/* <p>Choose a service to get started:</p>
      <ul>
        <li><Link to="/visitor-info">Visitor Information</Link></li>
        <li><Link to="/booking">Hotel Booking</Link></li>
        <li><Link to="/ordering">Online Ordering</Link></li>
      </ul> */}
    </div>
  );
};

export default Home;
```

III. Deployment

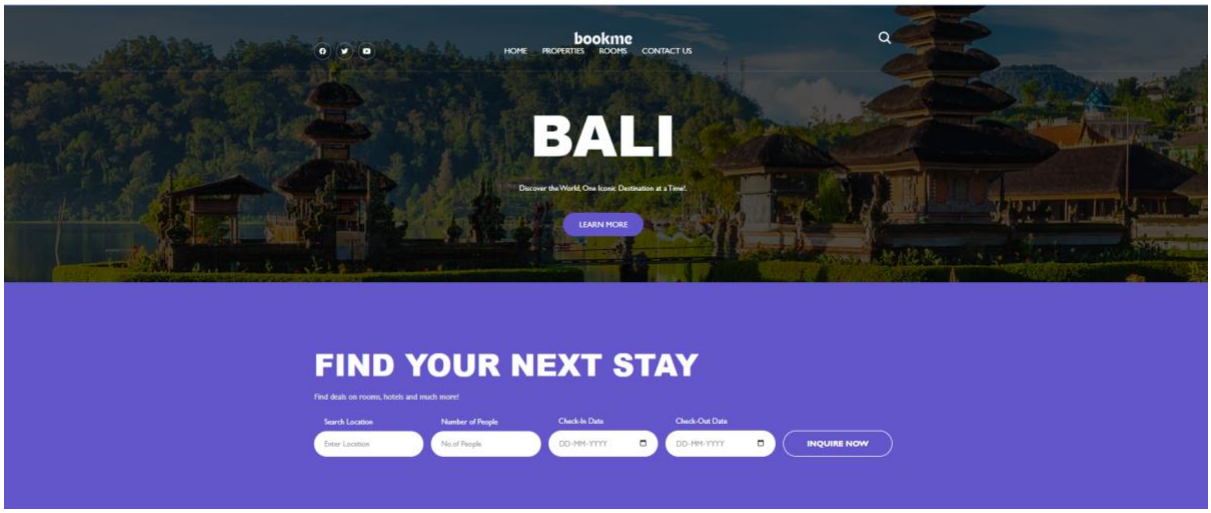
frontend-deployment.yaml

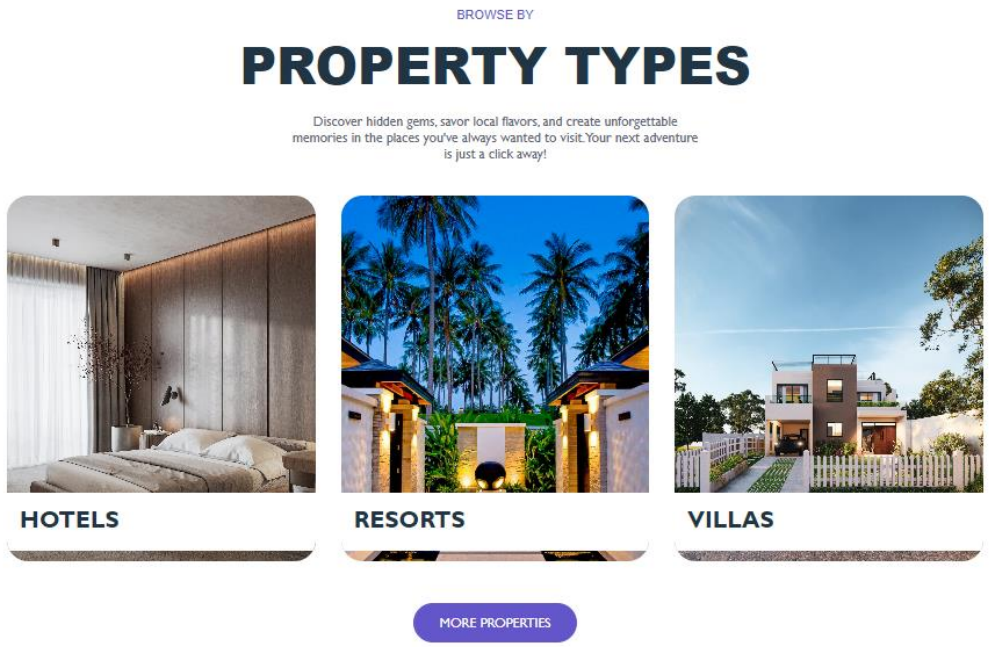
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - name: frontend
        image: varsingh/frontend:latest
        ports:
        - containerPort: 3000

---
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
spec:
  type: NodePort
  ports:
    - port: 3000
      targetPort: 3000
      nodePort: 30000
  selector:
    app: frontend
```

Service 1: Room Booking

This the page for the hotel booking microservice which allows user to choose location and book a hotel accordingly

I. Frontend





POPULAR STAYS

CHECKOUT OUR MOST POPULAR LOCAL PLACES TO STAY

Curated journeys offering all-inclusive experiences, from flights and accommodation to guided tours, making travel hassle-free and unforgettable.

SWAN PARADISE

A Pramana Experience has an outdoor swimming pool, fitness center, and terrace in Gianyar. This 4-star resort offers a bar.

(871 reviews) ★★★★

One-Bedroom Suite with Garden View

$220

/ per person

BOOK NOW

VIEW ALL ROOMS

CONTACT US

HAVE FUN WITH US. REMEMBER US!

We'd love to hear from you! Whether need assistance with bookings, or want to share your hotel experiences, our dedicated team is here to help.

CONTACT US

bookme

Reach out via email, phone, or our convenient contact form, and we'll get back to you promptly. We're eager to assist you every step of the way!

Contact Us

Feel free to contact and reach us !!

+64 045 955 068

info.bookme.com

2500 Wollongong, Australia

II. Code

```
import React, { useState } from 'react';
import './booking.css';


const Booking = () => {
  const [bookingData, setBookingData] = useState({
    destination: '',
    people: '',
    checkin: '',
    checkout: ''
  });

  const handleChange = (e) => {
    setBookingData({ ...bookingData, [e.target.name]: e.target.value });
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    fetch('http://localhost:5000/api/hotel-booking', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(bookingData)
    })
      .then(response => response.json())
      .then(data => {
        console.log("Booking successful:", data);
      })
      .catch(error => {
        console.error("Error making booking:", error);
      });
  };
```

```
return (
  <body id="top">
    <header class="header" data-header>
      <div class="overlay" data-overlay></div>
      <div class="header-top">
        <div class="container">
          <a href="tel:+01123456790" class="helpline-box">
          </a>

          <a href="#" class="logo">
            <img src="/assets/images/logo2.png" alt="Logo" />
          </a>

          <div class="header-btn-group">

            <button class="search-btn" aria-label="Search">
              <ion-icon name="search"></ion-icon>
            </button>

            <button class="nav-open-btn" aria-label="Open Menu" data-nav-open-btn>
              <ion-icon name="menu-outline"></ion-icon>
            </button>

          </div>


        </div>
      </div>
      <div class="header-bottom">
        <div class="container">
```

```
        {/* CTA */}
        <section class="cta" id="contact">
          <div class="container">

            <div class="cta-content">
              <p class="section-subtitle">Contact Us</p>

              <h2 class="h2 section-title">Have Fun With Us. Remember Us!</h2>

              <p class="section-text">
                We'd love to hear from you! Whether need assistance with bookings,
              </p>
            </div>

            <button class="btn btn-secondary">Contact Us</button>

          </div>
        </section>

      </article>
    </main>

    {/* FOOTER */}
    <footer class="footer">

      <div class="footer-top">
        <div class="container">
```

III. Backend

The app.py file enables the website to access the APIs and also database tables.

backend > hotel-booking-service > app.py > ...

```python
from flask import Flask, send_from_directory
from flask_sqlalchemy import SQLAlchemy

from flask_cors import CORS

app = Flask(__name__)
CORS(app)  # This will enable CORS for all routes


# Database configuration
app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://user:password@db:5432/hotelbooking'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

@app.route('/')
def serve():
    return send_from_directory(app.static_folder, 'index.html')

# Move the blueprint import here
def create_app():
    from routes import booking_blueprint
    app.register_blueprint(booking_blueprint)

    return app

if __name__ == '__main__':
    app = create_app()
    app.run(host='0.0.0.0', port=5001)
```

```
> Users > varni > Downloads > hotel-booking-service > models.py > ...
from flask_sqlalchemy import SQLAlchemy
from app import db

db = SQLAlchemy()

class Booking(db.Model):
    __tablename__ = 'bookings'

    id = db.Column(db.Integer, primary_key=True)
    customer_name = db.Column(db.String(100), nullable=False)
    room_type = db.Column(db.String(50), nullable=False)
    check_in_date = db.Column(db.Date, nullable=False)
    check_out_date = db.Column(db.Date, nullable=False)
    total_price = db.Column(db.Float, nullable=False)

    def __init__(self, customer_name, room_type, check_in_date, check_out_date, total_price):
        self.customer_name = customer_name
        self.room_type = room_type
        self.check_in_date = check_in_date
        self.check_out_date = check_out_date
        self.total_price = total_price

    def __repr__(self):
        return f'<Booking {self.id}>'

```

IV. API Routes

Hotel Booking Service

- GET `/api/hotel-booking`: Retrieve all bookings
- POST `/api/hotel-booking`: Create a new booking

```
C: > Users > varni > Downloads > hotel-booking-service > routes.py > ...
from flask import Blueprint, jsonify, request
from models import db, Booking

booking_blueprint = Blueprint('booking', __name__)

# Get all bookings
@booking_blueprint.route('/api/booking', methods=['GET'])
def get_bookings():
    bookings = Booking.query.all()
    return jsonify([booking.to_dict() for booking in bookings])

# Add a new booking
@booking_blueprint.route('/api/hotel-booking', methods=['POST'])
def add_booking():
    data = request.get_json()
    new_booking = Booking(
        customer_name=data['customer_name'],
        hotel_name=data['hotel_name'],
        check_in=data['check_in'],
        check_out=data['check_out']
    )
    db.session.add(new_booking)
    db.session.commit()
    return jsonify(new_booking.to_dict()), 201
```

V. Deployment

Building and Pushing images to Dockerhub

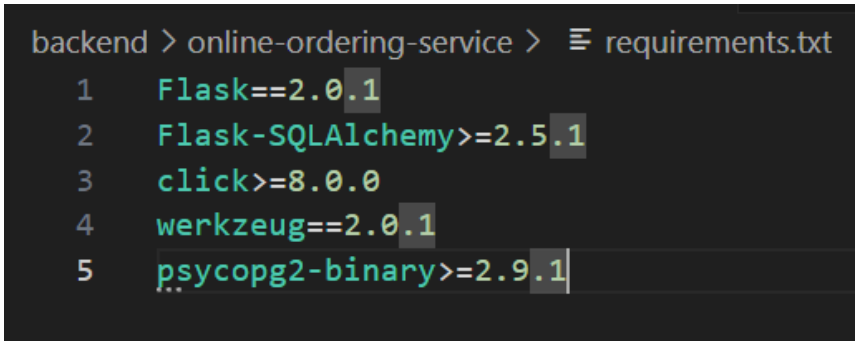
Before deploying the containers to the system, we build and push the images to Docker hub via a personal access token and as we go on, we can keep updating this image.
cd backend/visitor-information-service

Build the Docker image
docker build -t varsingh/visitor-information-service:latest .

Push the image to Docker Hub
docker push varsingh/visitor-information-service:latest

Note that all the three microservices use the same requirements.txt and Dockerfile.

Requirements.txt



Dockerfile

```dockerfile
# Use Python image
FROM python:3.8-slim

# Set the working directory
WORKDIR /app

# Copy requirements.txt
COPY requirements.txt /app/requirements.txt

# Install dependencies
RUN pip install --no-cache-dir -r /app/requirements.txt

# Copy the rest of the app
COPY . /app

# Expose port
EXPOSE 5000

# Command to run the app
CMD ["python", "app.py"]
```

Manifest file

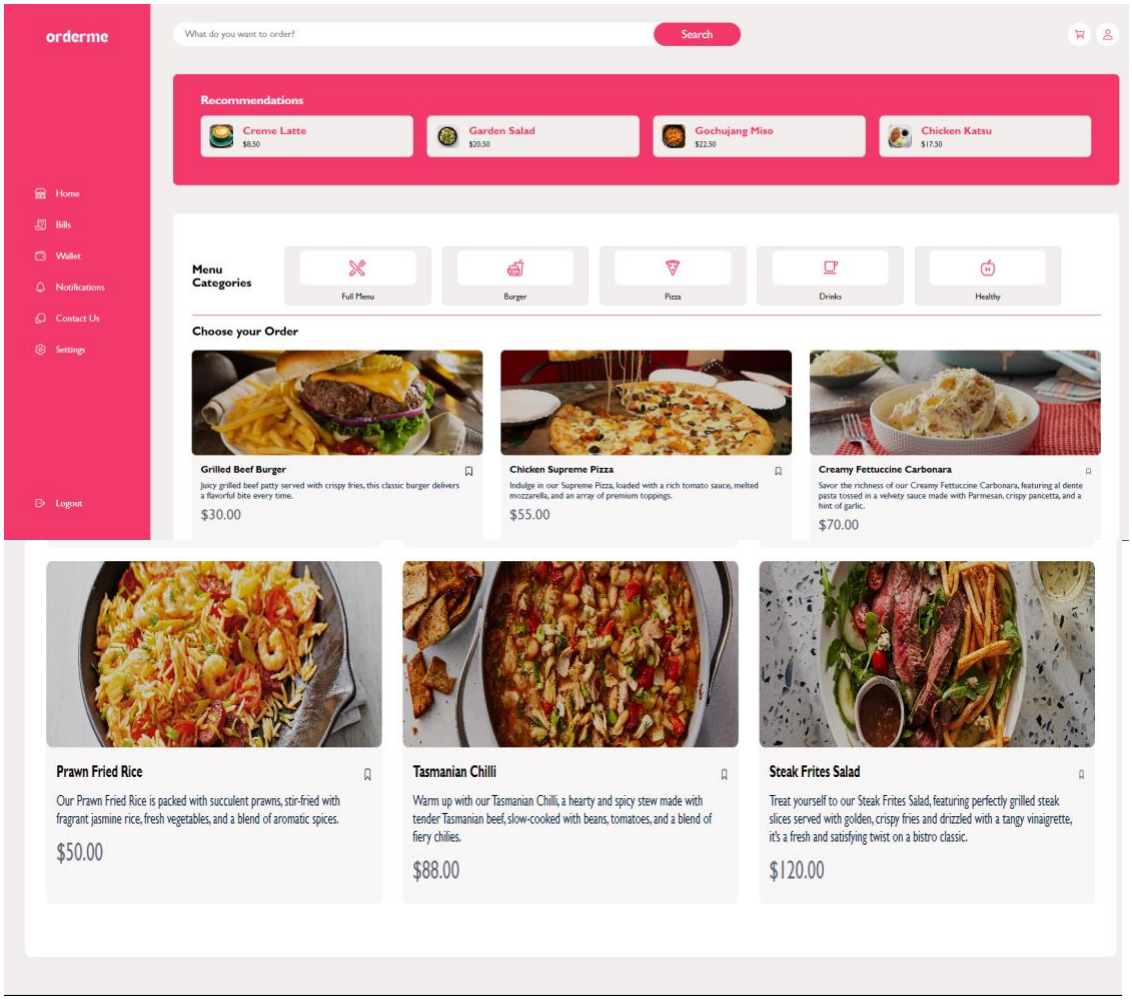
hotel-booking-deployment.yaml

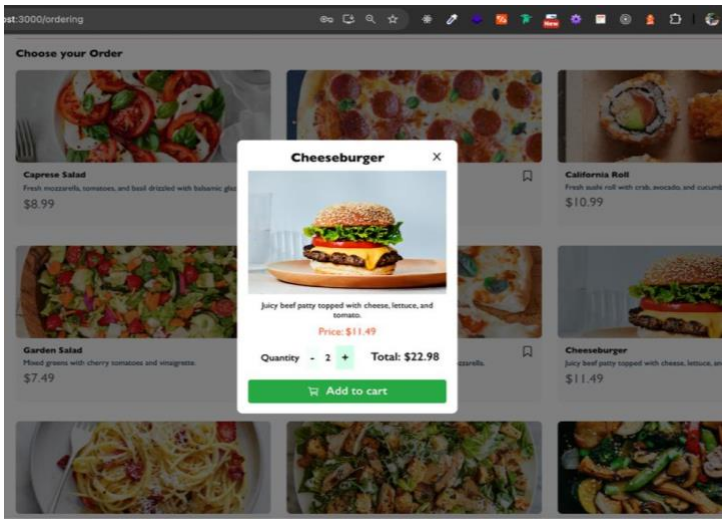
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hotel-booking
spec:
  replicas: 2
  selector:
    matchLabels:
      app: hotel-booking
  template:
    metadata:
      labels:
        app: hotel-booking
    spec:
      containers:
      - name: hotel-booking
        image: varsingh/hotel-booking:latest
        ports:
        - containerPort: 5000
        env:
        - name: POSTGRES_USER
          value: "user"
        - name: POSTGRES_PASSWORD
          value: "password"
        - name: POSTGRES_DB
          value: "hotelbooking"
        - name: DB_HOST
          value: "postgres-service"

---
```

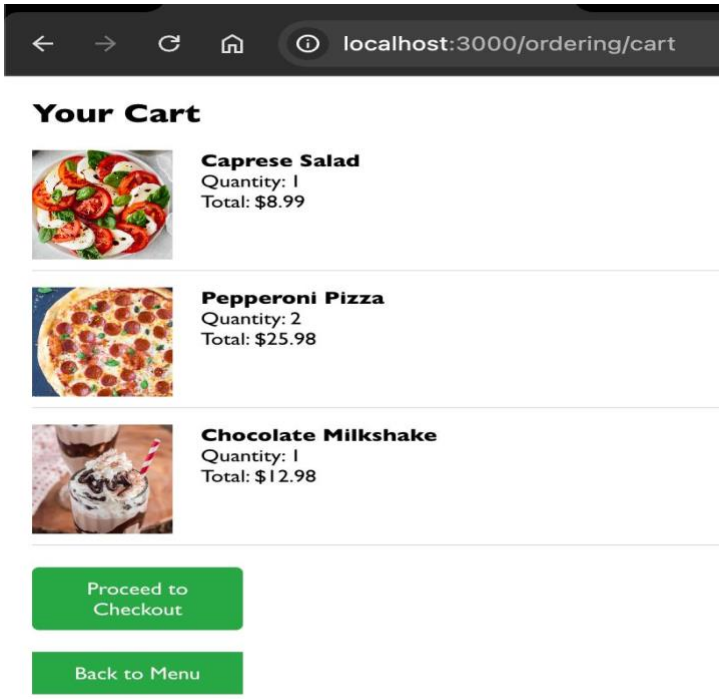
```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: hotel-booking-service
spec:
  type: NodePort
  ports:
    - port: 5000
      targetPort: 5000
      nodePort: 30010
  selector:
    app: hotel-booking
```

Service 2: Online Ordering

I. Frontend:







II. Code

src > components > OnlineOrdering > JS Ordering.js > Ordering

```
1  import React, { useState, useEffect } from 'react';
2  import './ordering.css';
3  import axios from 'axios';
4  import { Link } from 'react-router-dom';
5
6  const Ordering = () => {
7    const [foodItems, setFoodItems] = useState([]);
8    const [selectedFood, setSelectedFood] = useState(null); //
9    const [isModalOpen, setIsModalOpen] = useState(false);  //
10   const [quantity, setQuantity] = useState(1);            //
11   const [totalPrice, setTotalPrice] = useState(0);        //
12
13
14   // Fetch food items from the backend
15   useEffect(() => {
16     axios.get('http://localhost:5001/api/online-ordering')
17       .then(response => {
18         console.log("this is response", response.data);
19         setFoodItems(response && response.data);  // Get food
20       })
21       .catch(error => {
22         console.error('Error fetching food items:', error);
23       });
24   }, []);
25
26   // Handle selecting a food item
27   const handleSelectFood = (food) => {
28     setSelectedFood(food);
29     setQuantity(1);  // Reset quantity
30     setTotalPrice(food.price);  // Set total price based on t
31     setIsModalOpen(true);  // Open modal
32   };
33
```

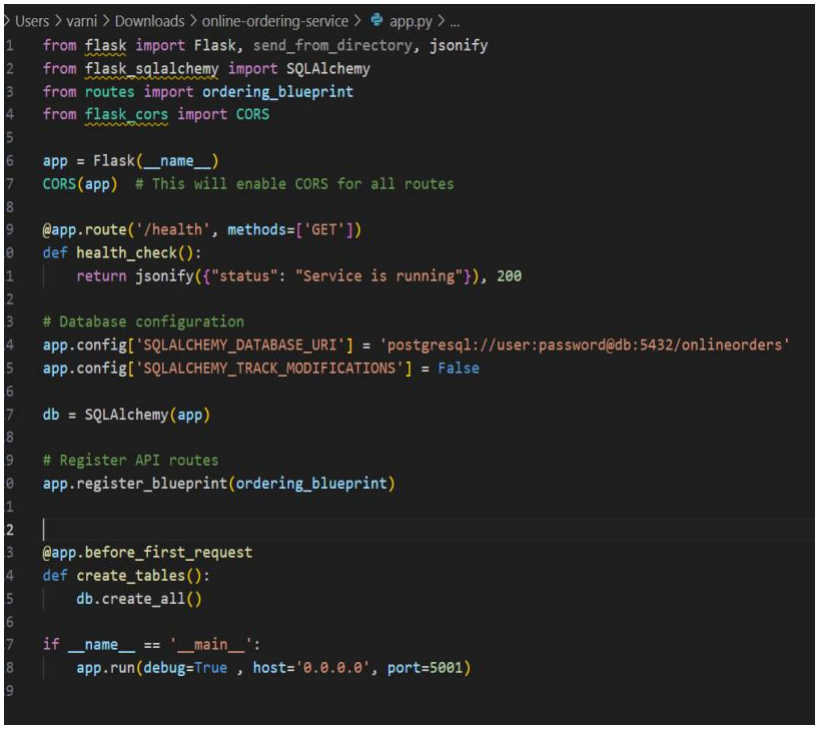
```
// Handle quantity change
const handleQuantityChange = (e) => {
  const newQuantity = e.target.value;
  setQuantity(newQuantity);
  setTotalPrice(newQuantity * selectedFood.price);  // Update total price
};

// Handle closing the modal
const handleCloseModal = () => {
  setIsModalOpen(false);
  setSelectedFood(null);
};

// Handle ordering submission
const handleSubmitOrder = () => {
  // Prepare the data to send to the backend for placing the order
  const orderData = {
    item: selectedFood.name,
    quantity: quantity,
    totalPrice: totalPrice
  };

  // Send order to the backend
  axios.post('http://localhost:5001/api/online-ordering', orderData)
    .then(response => {
      console.log("Order placed successfully:", response.data);
      handleCloseModal();  // Close the modal
    })
    .catch(error => {
      console.error("Error placing order:", error);
    });
};
```

```
const filterByCategory = (category) => {
  // Make API call with filter
  axios.get(`http://localhost:5000/api/ordering?filter=${category}`)
    .then(response => {
      console.log("Filtered response", response.data);
      setFoodItems(response.data); // Update foodItems directly with the f
    })
    .catch(error => {
      console.error('Error fetching filtered food items:', error);
    });
};

// Open modal and set selected food item
const openModal = (item) => {
  console.log("here", item)
  setSelectedFood(item);
  setQuantity(1); // Reset quantity to 1 when a new item is selected
  setTotalPrice(item.price); // Set the initial total price to the item pr
  setIsModalOpen(true);
  console.log("selected food", selectedFood, isModalOpen)
};

// Close modal
const closeModal = () => {
  setIsModalOpen(false);
};

// Update quantity and total price
const updateQuantity = (type) => {
  if (type === 'increase') {
    setQuantity(quantity + 1);
```

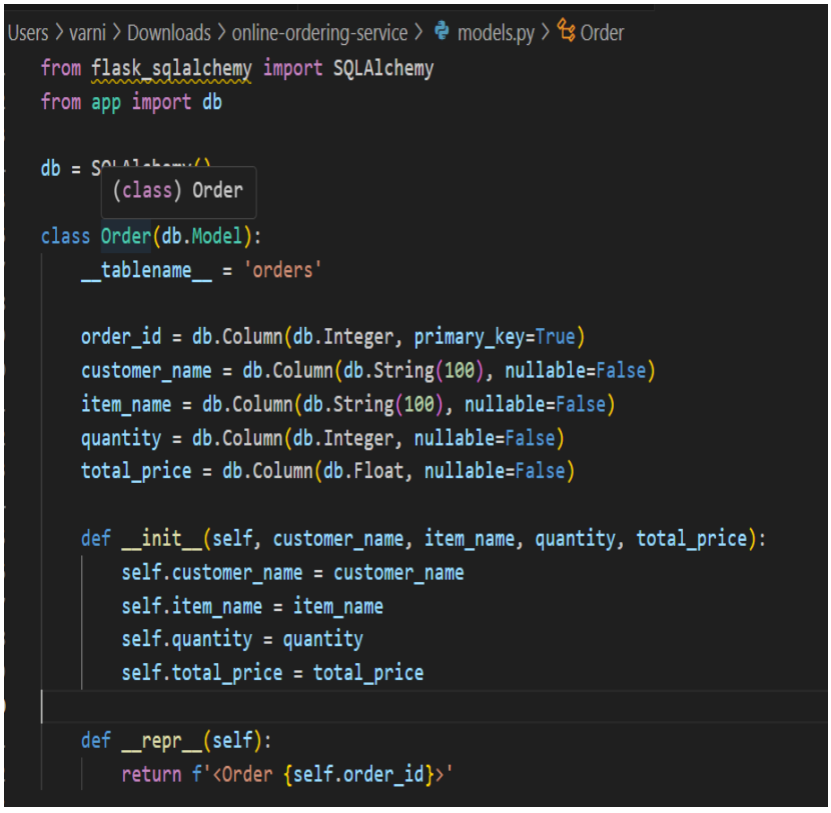
```
    setQuantity(quantity + 1);
    setTotalPrice((quantity + 1) * selectedFood.price);
  } else if (type === 'decrease' && quantity > 1) {
    setQuantity(quantity - 1);
    setTotalPrice((quantity - 1) * selectedFood.price);
  }
};

return (
  <div className='onlineOrderingCSS'>
    <body2>
      <div class="sidebar">
        <img class="logo4" src="/assets/images/logo4.png"></img>

        <div class="sidebar-menus">
          <a href="#"><ion-icon name="storefront-outline"></ion-icon> Home </a>
          <a href="#"><ion-icon name="receipt-outline"></ion-icon> Bills </a>
          <a href="#"><ion-icon name="wallet-outline"></ion-icon> Wallet </a>
          <a href="#"><ion-icon name="notifications-outline"></ion-icon> Notifications </a>
          <a href="#"><ion-icon name="chatbubbles-outline"></ion-icon> Contact Us </a>
          <a href="#"><ion-icon name="settings-outline"></ion-icon> Settings </a>
        </div>

        <div class="sidebar-logout">
          <a href="#"><ion-icon name="log-out-outline"></ion-icon>Logout</a>
        </div>

      </div>
      <div class="main">
        <div class="main-navbar">
```

```
        </div>

      </div>
      <div class="main">
        <div class="main-navbar">
          <ion-icon class="menu-toggle" name="menu-outline"></ion-icon>

          <div class="search">
            <input type="text" placeholder="What do you want to order?"></input>
            <button class="search-btn"> Search </button>
          </div>

          <div class="profile">
            <a class="cart" href="#"><ion-icon name="cart-outline"></ion-icon></a>
            <a class="user" href="#"><ion-icon name="person-outline"></ion-icon></a>
          </div>

        </div>

        <div class="main-highlight">
          <div class="main-header">
            <h2 class="main-title2">Recommendations</h2>
          </div>

          <div class="highlight-wrapper">
            <div class="highlight-card">
              <img class="highlight-img" src="/assets/images/ico-1.jpg"></img>
              <div class="highlight-desc">
                <h4>Creme Latte</h4>
                <p>$8.50</p>
              </div>
            </div>
```

III. Backend



```
Users > varni > Downloads > online-ordering-service > app.py > ...
from flask import Flask, send_from_directory, jsonify
from flask_sqlalchemy import SQLAlchemy
from routes import ordering_blueprint
from flask_cors import CORS

app = Flask(__name__)
CORS(app)  # This will enable CORS for all routes

@app.route('/health', methods=['GET'])
def health_check():
    return jsonify({"status": "Service is running"}), 200

# Database configuration
app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://user:password@db:5432/onlineorders'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

# Register API routes
app.register_blueprint(ordering_blueprint)

@app.before_first_request
def create_tables():
    db.create_all()

if __name__ == '__main__':
    app.run(debug=True , host='0.0.0.0', port=5001)
```



```
Users > varni > Downloads > online-ordering-service > models.py > Order
from flask_sqlalchemy import SQLAlchemy
from app import db

db = SQLAlchemy()
     (class) Order
class Order(db.Model):
    __tablename__ = 'orders'

    order_id = db.Column(db.Integer, primary_key=True)
    customer_name = db.Column(db.String(100), nullable=False)
    item_name = db.Column(db.String(100), nullable=False)
    quantity = db.Column(db.Integer, nullable=False)
    total_price = db.Column(db.Float, nullable=False)

    def __init__(self, customer_name, item_name, quantity, total_price):
        self.customer_name = customer_name
        self.item_name = item_name
        self.quantity = quantity
        self.total_price = total_price

    def __repr__(self):
        return f'<Order {self.order_id}>'
```

IV. API Routes

- GET `/api/online-ordering`
- POST `/api/online-ordering`

C: > Users > varni > Downloads > online-ordering-service > routes.py > ...

```python
from flask import Blueprint, jsonify, request
from models import db, Order

ordering_blueprint = Blueprint('ordering', __name__)

# Get all orders
@ordering_blueprint.route('/api/online-ordering', methods=['GET'])
def get_orders():
    orders = Order.query.all()
    return jsonify([order.to_dict() for order in orders])

# Add a new order
@ordering_blueprint.route('/api/online-ordering', methods=['POST'])
def add_order():
    data = request.get_json()
    new_order = Order(
        customer_name=data['customer_name'],
        item=data['item'],
        quantity=data['quantity'],
        status=data['status'],
        order_date=data['order_date']
    )
    db.session.add(new_order)
    db.session.commit()
    return jsonify(new_order.to_dict()), 201
```

V. Deployment

Building and Pushing images to Dockerhub

cd backend/online-ordering-service

Build the Docker image
docker build -t varsingh/online-ordering-service:latest .

Push the image to Docker Hub
docker push varsingh/online-ordering-service:latest

Requirements.txt

```
Flask==2.0.1
Flask-SQLAlchemy>=2.5.1
click>=8.0.0
werkzeug==2.0.1
psycopg2-binary>=2.9.1
```

Dockerfile

```dockerfile
# Use Python image
FROM python:3.8-slim

# Set the working directory
WORKDIR /app

# Copy requirements.txt
COPY requirements.txt /app/requirements.txt

# Install dependencies
RUN pip install --no-cache-dir -r /app/requirements.txt

# Copy the rest of the app
COPY . /app

# Expose port
EXPOSE 5000

# Command to run the app
CMD ["python", "app.py"]
```

Manifest file

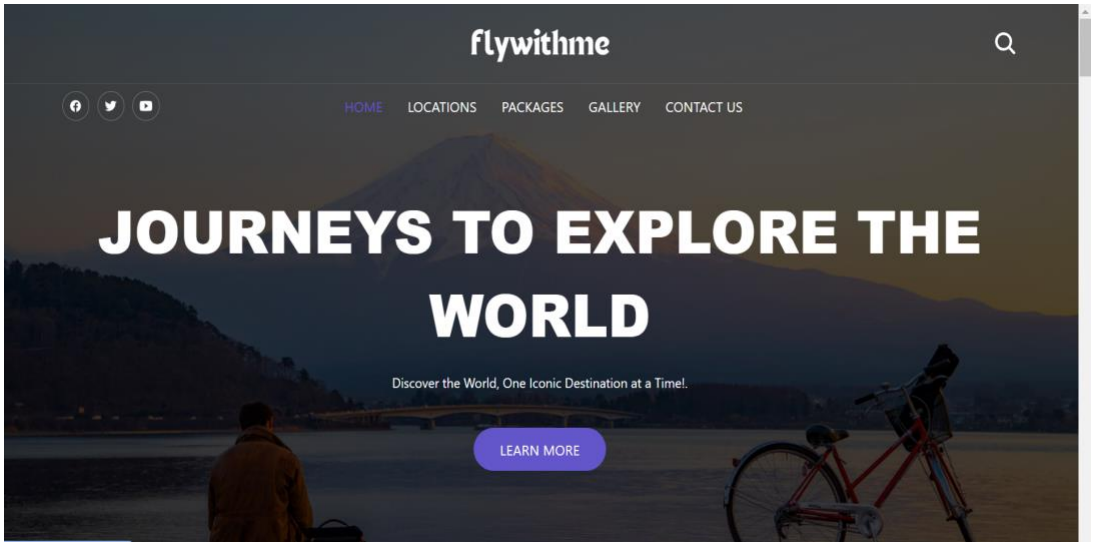
online-ordering-deployment.yaml

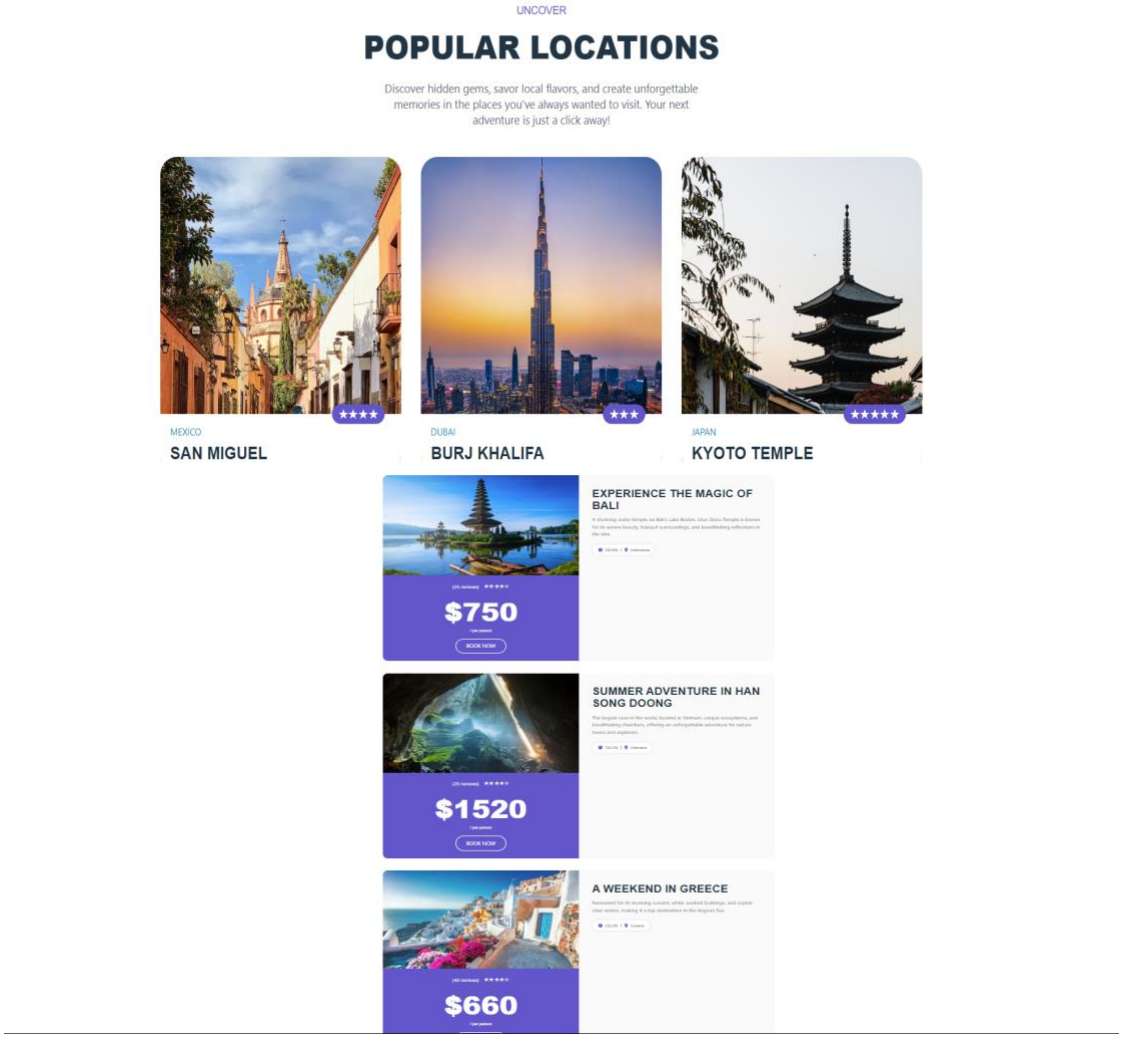
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: online-ordering
spec:
  replicas: 2
  selector:
    matchLabels:
      app: online-ordering
  template:
    metadata:
      labels:
        app: online-ordering
    spec:
      containers:
      - name: online-ordering
        image: varsingh/online-ordering:latest
        ports:
        - containerPort: 5000
        env:
        - name: POSTGRES_USER
          value: "user"
        - name: POSTGRES_PASSWORD
          value: "password"
        - name: POSTGRES_DB
          value: "onlineorders"
        - name: DB_HOST
          value: "postgres-service"

---
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: online-ordering-service
spec:
  type: NodePort
  ports:
    - port: 5000
      targetPort: 5000
      nodePort: 30011
  selector:
    app: online-ordering
```

Service 3: Visitor Information

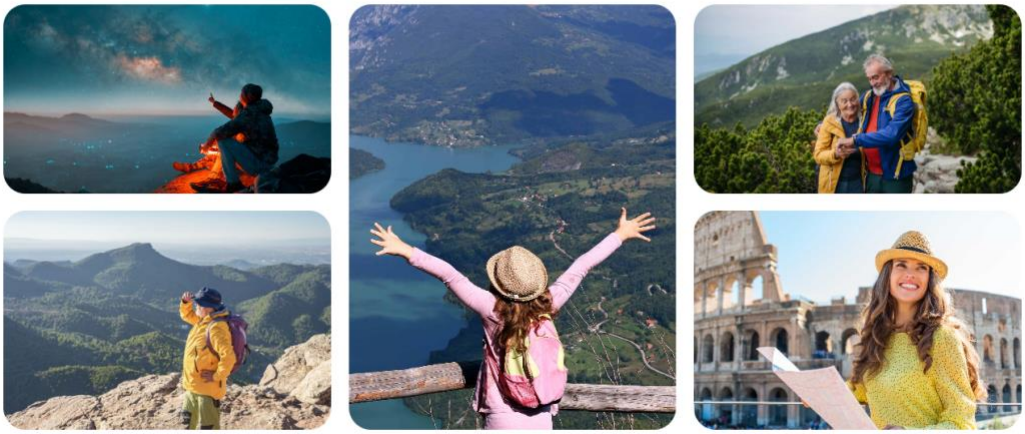
I. Frontend





MEMORIES FROM ALL OVER THE WORLD

Dreaming of your next getaway or reminiscing about past adventures? We invite you to experience diversity of travel through the lens of passionate photographers. Let these images inspire your wanderlust and help you envision your next journey!



CONTACT US

HAVE FUN WITH US. REMEMBER US!

We'd love to hear from you! Whether you have questions about our travel packages, need assistance with bookings, or want to share your travel experiences, our dedicated team is here to help.

CONTACT US

flywithme

Reach out via email, phone, or our convenient contact form, and we'll get back to you promptly. Your journey starts with us, and we're eager to assist you every step of the way!

Contact Us

Feel free to contact and reach us !!

+64 045 955 068

info.flywithme.com

2500 Wollongong, Australia

II. Code

```
omponents > VisitorInfo > JS VisitorInfo.js > VisitorInfo
import React, { useEffect, useState } from 'react';
import './visitorInfo.css';
import axios from 'axios';

const VisitorInfo = () => {
  const [visitors, setVisitors] = useState([]);

  useEffect(() => {
    fetch('http://localhost:5002/api/visitor-info')
      .then(response => response.json())
      .then(data => setVisitors(data))
      .catch(error => console.error('Error fetching visitor data:', error));
  }, []);

  return (
    <body id="top">
      <header class="header" data-header>
        <div class="overlay" data-overlay></div>
        <div class="header-top">
          <div class="container">
            <a href="tel:+01123456790" class="helpline-box"> </a>
            <a href="#" class="logo">
              <img src="/assets/images/logo.png" alt="Logo" />
            </a>
            <div class="header-btn-group">
              <button class="search-btn" aria-label="Search">
                <ion-icon name="search"></ion-icon>
              </button>
              <button class="nav-open-btn" aria-label="Open Menu" data-nav-open-btn>
                <ion-icon name="menu-outline"></ion-icon>
              </button>
            </div>
          </div>
        </div>
        <div class="header-bottom">
          <div class="container">
            <ul class="social-list">
              <li>
                <a href="#" class="social-link">
                  <ion-icon name="logo-facebook"></ion-icon>
                </a>
              </li>
              <li>
                <a href="#" class="social-link">
                  <ion-icon name="logo-twitter"></ion-icon>
                </a>
              </li>
              <li>
                <a href="#" class="social-link">
                  <ion-icon name="logo-youtube"></ion-icon>
                </a>
              </li>
            </ul>
            <nav class="navbar" data-navbar>

              <div class="navbar-top">

                <a href="#" class="logo">

                </a>

                <button class="nav-close-btn" aria-label="Close Menu" data-nav-close-btn>
                  <ion-icon name="close-outline"></ion-icon>
                </button>
```

```
{/* POPULAR */}
<section class="popular" id="location">
  <div class="container">

    <p class="section-subtitle">Uncover</p>

    <h2 class="h2 section-title">Popular Locations</h2>

    <p class="section-text">
      Discover hidden gems, savor local flavors, and create unforgettable memories in the plac
    </p>

    <ul class="popular-list">

      <li>
        <div class="popular-card">

          <figure class="card-img">
            <img src="./assets/images/popular-1.png" alt="San miguel, Mexico" loading="lazy" /
          </figure>

          <div class="card-content">

            <div class="card-rating">
              <ion-icon name="star"></ion-icon>
              <ion-icon name="star"></ion-icon>
              <ion-icon name="star"></ion-icon>
              <ion-icon name="star"></ion-icon>
            </div>
          </p>

          <h3 class="h3 card-title">
            <a href="#">San miguel</a>
          </h3>

          <p class="card-text">
            A picturesque colonial town in Mexico, known for its vibrant culture, colorful streets
          </p>

        </div>

      </div>
    </li>

    <li>
      <div class="popular-card">

        <figure class="card-img">
          <img src="/assets/images/popular-2.jpg" alt="Burj khalifa, dubai" loading="lazy" />
        </figure>

        <div class="card-content">

          <div class="card-rating">
            <ion-icon name="star"></ion-icon>
            <ion-icon name="star"></ion-icon>
            <ion-icon name="star"></ion-icon>
          </div>

          <p class="card-subtitle">
            <a href="#">Dubai</a>
```

III. Backend

```python
from flask import Flask, send_from_directory,jsonify
from flask_sqlalchemy import SQLAlchemy
from routes import visitor_blueprint
from flask_cors import CORS

app = Flask(__name__)
CORS(app)  # This will enable CORS for all routes


# Database configuration
app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://user:password@db:5432/visitorinfo'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

# Register API routes
app.register_blueprint(visitor_blueprint)

@app.route('/')
def serve():
    return send_from_directory(app.static_folder, 'index.html')

@app.before_first_request
def create_tables():
    db.create_all()

if __name__ == '__main__':
    app.run(debug=True , host='0.0.0.0', port=5002)
```

```
rs > varni > Downloads > visitor-info-service > models.py > ...
from flask_sqlalchemy import SQLAlchemy
from app import db
db = SQLAlchemy()

class VisitorInfo(db.Model):
    __tablename__ = 'visitor_info'

    id = db.Column(db.Integer, primary_key=True)
    visitor_name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(100), nullable=False)
    visit_date = db.Column(db.Date, nullable=False)

    def __init__(self, visitor_name, email, visit_date):
        self.visitor_name = visitor_name
        self.email = email
        self.visit_date = visit_date

    def __repr__(self):
        return f'<Visitor {self.visitor_name}>'
```

IV. API Routes

Visitor Info Service

-GET `/api/visitor-info`
-POST ‘/api/visitor-info’’

C: > Users > varni > Downloads > visitor-info-service > routes.py > ...

```python
from flask import Blueprint, jsonify, request
from models import db, VisitorInfo

visitor_blueprint = Blueprint('visitor', __name__)

# Get all visitor info
@visitor_blueprint.route('/api/visitor-info', methods=['GET'])
def get_visitor_info():
    visitors = VisitorInfo.query.all()
    return jsonify([visitor.to_dict() for visitor in visitors])

# Add new visitor info
@visitor_blueprint.route('/api/visitor-info', methods=['POST'])
def add_visitor_info():
    data = request.get_json()
    new_visitor = VisitorInfo(
        visitor_name=data['visitor_name'],
        visit_date=data['visit_date'],
        info=data['info']
    )
    db.session.add(new_visitor)
    db.session.commit()
    return jsonify(new_visitor.to_dict()), 201
```

V. Deployment

Building and Pushing images to Dockerhub

cd backend/online-ordering-service

Build the Docker image
docker build -t varsingh/online-ordering-service:latest .

Push the image to Docker Hub
docker push varsingh/online-ordering-service:latest

Requirements.txt

```
Flask==2.0.1
Flask-SQLAlchemy>=2.5.1
click>=8.0.0
werkzeug==2.0.1
psycopg2-binary>=2.9.1
```

Dockerfile

```dockerfile
# Use Python image
FROM python:3.8-slim

# Set the working directory
WORKDIR /app

# Copy requirements.txt
COPY requirements.txt /app/requirements.txt

# Install dependencies
RUN pip install --no-cache-dir -r /app/requirements.txt

# Copy the rest of the app
COPY . /app

# Expose port
EXPOSE 5000

# Command to run the app
CMD ["python", "app.py"]
```

Manifest file

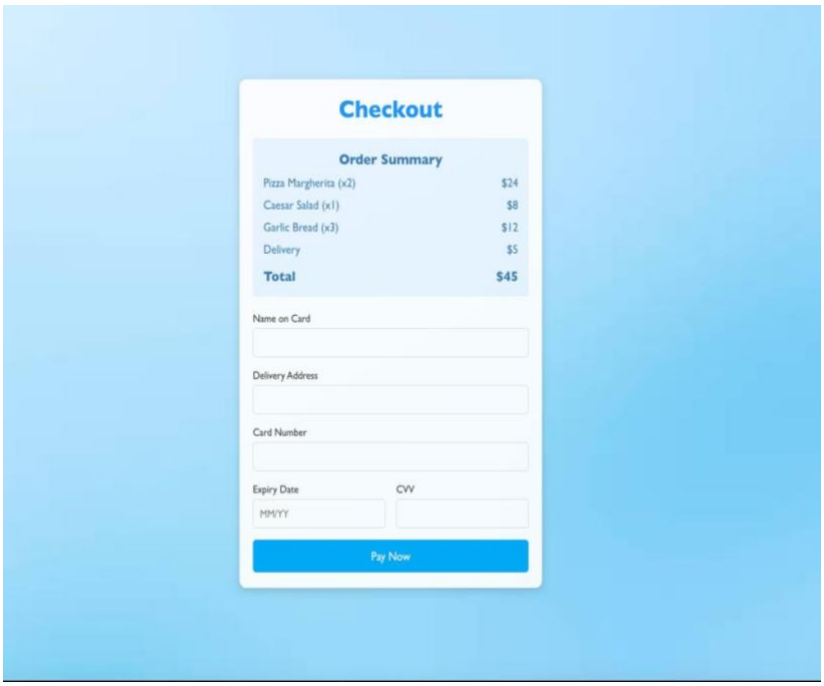
visitor-info-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: visitor-info
spec:
  replicas: 2
  selector:
    matchLabels:
      app: visitor-info
  template:
    metadata:
      labels:
        app: visitor-info
    spec:
      containers:
      - name: visitor-info
        image: <your-dockerhub-user>/visitor-info:latest
        ports:
        - containerPort: 5000
        env:
        - name: POSTGRES_USER
          value: "user"
        - name: POSTGRES_PASSWORD
          value: "password"
        - name: POSTGRES_DB
          value: "visitorinfo"
        - name: DB_HOST
          value: "postgres-service"
```

```yaml
---

apiVersion: v1
kind: Service
metadata:
  name: visitor-info-service
spec:
  type: NodePort
  ports:
    - port: 5000
      targetPort: 5000
      nodePort: 30012
  selector:
    app: visitor-info
```

Checkout page

I. Frontend



II. Code

checkout.js

src > components > CheckoutPage > checkout.js > CheckoutPage

```
import React, { useState } from "react";
import "./checkout.css";
import { Link } from "react-router-dom";

const CheckoutPage = () => {
  const [formData, setFormData] = useState({
    name: "",
    address: "",
    cardNumber: "",
    expiryDate: "",
    cvv: ""
  });
  //for sucess message
  const [successMessage, setSuccessMessage] = useState('');
  const [showOrder, setShowOrder] = useState(true);

  const handleInputChange = (e) => {
    const { name, value } = e.target;
    setFormData({ ...formData, [name]: value });
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    // setSuccessMessage("Payment Successful!");
    setFormData(
      {
        name: "",
        address: "",
        cardNumber: "",
        expiryDate: "",
        cvv: ""
      }
    );
    setShowOrder(false)
  };

```

Backend Manifest file

```
        env:
        - name: POSTGRES_USER
          value: "user"
        - name: POSTGRES_PASSWORD
          value: "password"
        - name: POSTGRES_DB
          value: "tourism"
        volumeMounts:
        - name: postgres-storage
          mountPath: /var/lib/postgresql/data
      volumes:
      - name: postgres-storage
        persistentVolumeClaim:
          claimName: postgres-pvc
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Microservices Deployment

docker compose up –build

docker-compose.yml

```yaml
version: '3.8'

services:
  db:
    image: postgres:13
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: hotelbooking
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

  hotel-booking:
    build: ./backend/hotel-booking-service
    ports:
      - "5001:5001"
    environment:
      - SQLALCHEMY_DATABASE_URI=postgresql://user:password@db:5432/hotelbooking
    depends_on:
      - db
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:5001/health"]
      interval: 30s
      timeout: 10s
      retries: 3

  visitor-info:
    build: ./backend/visitor-info-service
    ports:
      - "5002:5002"
    environment:
      - SQLALCHEMY_DATABASE_URI=postgresql://user:password@db:5432/visitorinfo
    depends_on:
      - db
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:5002/health"]
      interval: 30s
      timeout: 10s
      retries: 3

  online-ordering:
    build: ./backend/online-ordering-service
    ports:
      - "5003:5003"
    environment:
      - SQLALCHEMY_DATABASE_URI=postgresql://user:password@db:5432/ordering
    depends_on:
      - db
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:5003/health"]
      interval: 30s
      timeout: 10s
      retries: 3

  frontend:
    build: .
    ports:
      - "3000:3000"
    depends_on:
      - hotel-booking
```

```yaml
  frontend:
    build: .
    ports:
      - "3000:3000"
    depends_on:
      - hotel-booking
      - visitor-info
      - online-ordering
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:3000"]
      interval: 30s
      timeout: 10s
      retries: 3

volumes:
  postgres_data:
```

docker ps

```
@singhvar24 ➜/workspaces/tourism-ecosystem-fe (features/sakshamta) $ docker ps
CONTAINER ID   IMAGE                                  COMMAND                  CREATED         STATUS
RTS                                  NAMES
114cb6d28f65   tourism-ecosystem-fe-frontend          "docker-entrypoint.s…"   2 minutes ago   Up 2 minutes
0.0.0:3000->3000/tcp, :::3000->3000/tcp   tourism-ecosystem-fe-frontend-1
8123f47d36ce   tourism-ecosystem-fe-visitor-info      "python app.py"          2 minutes ago   Up 2 minutes
0.0.0:5002->5002/tcp, :::5002->5002/tcp   tourism-ecosystem-fe-visitor-info-1
84d7d3f453b5   tourism-ecosystem-fe-hotel-booking     "python app.py"          2 minutes ago   Up 2 minutes
0.0.0:5001->5001/tcp, :::5001->5001/tcp   tourism-ecosystem-fe-hotel-booking-1
4a428e633321   tourism-ecosystem-fe-online-ordering   "python app.py"          2 minutes ago   Up 2 minutes
0.0.0:5003->5003/tcp, :::5003->5003/tcp   tourism-ecosystem-fe-online-ordering-1
b9747452a768   postgres:13                            "docker-entrypoint.s…"   2 minutes ago   Up 2 minutes
0.0.0:5432->5432/tcp, :::5432->5432/tcp   tourism-ecosystem-fe-db-1
```

TASK F. Service/Process Analytics

Service1: Tour Booking

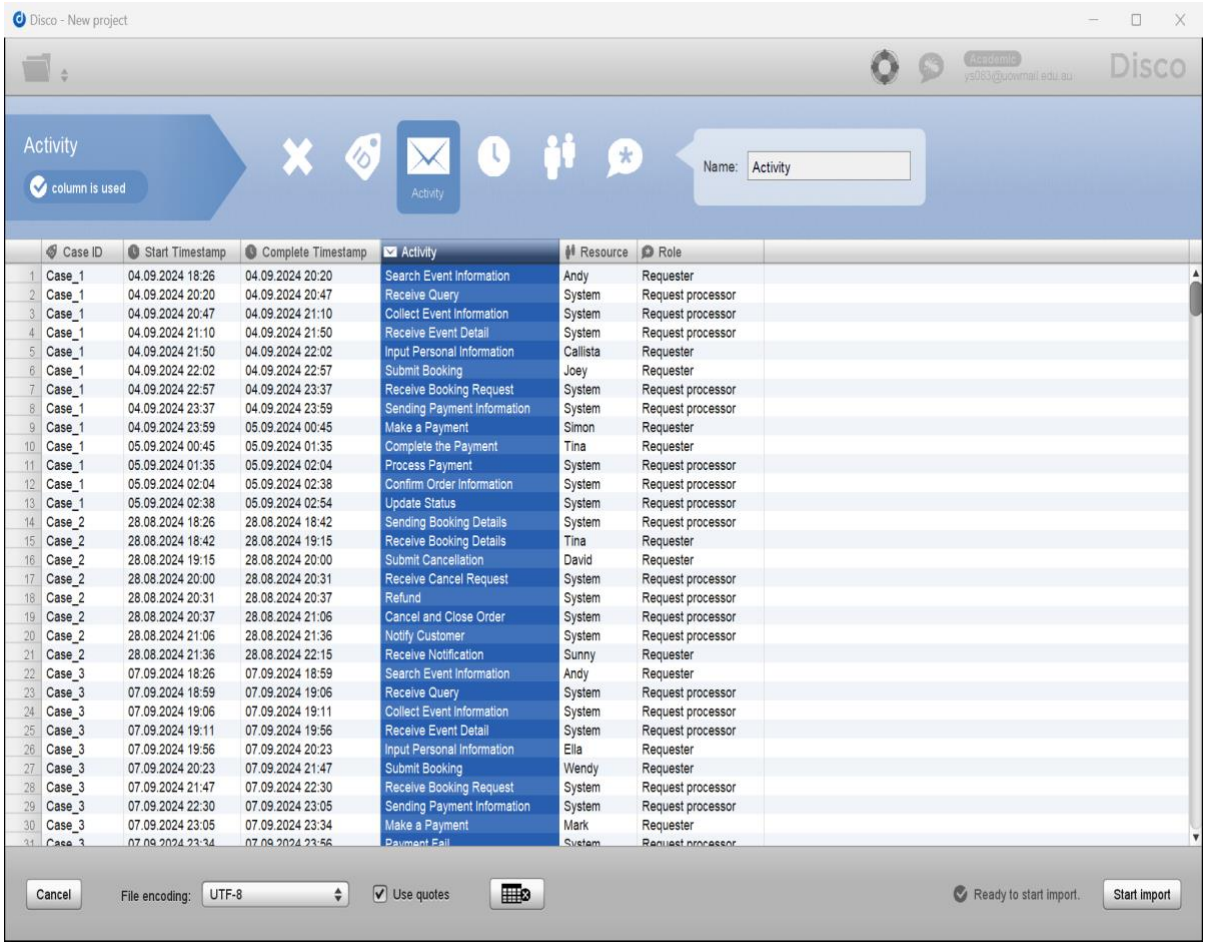
1. Generate Event Logs

The purpose of using python to generate event logs to assume periods randomly and automatically, increasing confidence that they fit real-world cases. Main elements of a log: Case ID, Start Timestamp, Complete Timestamp, Activity, Resource, Role.

After executing the implementation code and simulation conditions, the obtained log is displayed：

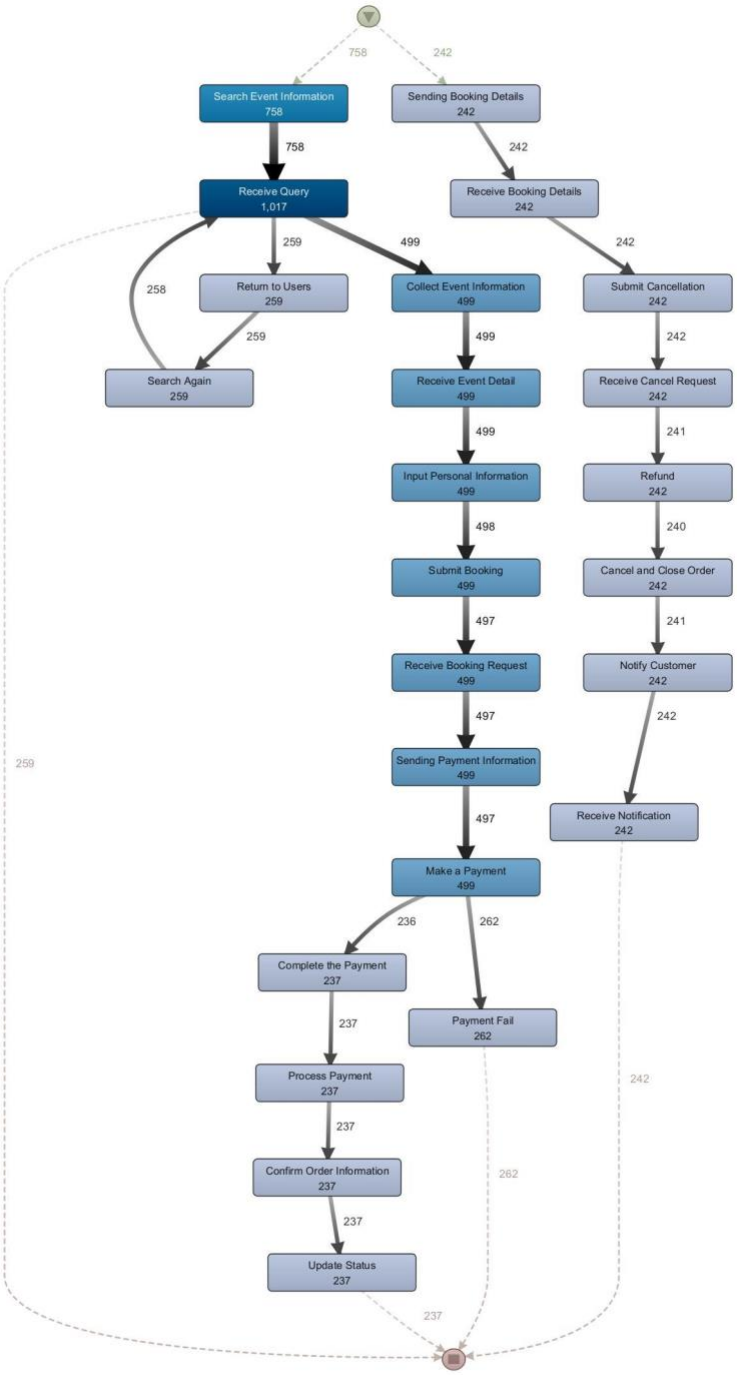
	A	B	C	D	E	F
1	Case ID	Start Timestamp	Complete Timestamp	Activity	Resource	Role
2	Case_1	23.09.2024 23:11	24.09.2024 01:08	Search Event Information	Rachel	Requester
3	Case_1	24.09.2024 01:08	24.09.2024 01:51	Receive Query	System	Request processor
4	Case_1	24.09.2024 01:51	24.09.2024 02:15	Collect Event Information	System	Request processor
5	Case_1	24.09.2024 02:15	24.09.2024 03:04	Receive Event Detail	System	Request processor
6	Case_1	24.09.2024 03:04	24.09.2024 03:24	Input Personal Information	Jack	Requester
7	Case_1	24.09.2024 03:24	24.09.2024 04:07	Submit Booking	Rachel	Requester
8	Case_1	24.09.2024 04:07	24.09.2024 05:04	Receive Booking Request	System	Request processor
9	Case_1	24.09.2024 05:04	24.09.2024 05:15	Sending Payment Informatio	System	Request processor
10	Case_1	24.09.2024 05:15	24.09.2024 06:00	Make a Payment	David	Requester
11	Case_1	24.09.2024 06:00	24.09.2024 06:41	Complete the Payment	Mark	Requester
12	Case_1	24.09.2024 06:41	24.09.2024 07:14	Process Payment	System	Request processor
13	Case_1	24.09.2024 07:14	24.09.2024 07:52	Confirm Order Information	System	Request processor
14	Case_1	24.09.2024 07:52	24.09.2024 08:19	Update Status	System	Request processor
15	Case_2	29.08.2024 23:11	29.08.2024 23:57	Sending Booking Details	System	Request processor
16	Case_2	29.08.2024 23:57	30.08.2024 00:08	Receive Booking Details	Eithan	Requester
17	Case_2	30.08.2024 00:08	30.08.2024 00:52	Submit Cancellation	Chris Lee	Requester
18	Case_2	30.08.2024 00:52	30.08.2024 01:43	Receive Cancel Request	System	Request processor
19	Case_2	30.08.2024 01:43	30.08.2024 02:14	Refund	System	Request processor
20	Case_2	30.08.2024 02:14	30.08.2024 03:11	Cancel and Close Order	System	Request processor
21	Case_2	30.08.2024 03:11	30.08.2024 03:24	Notify Customer	System	Request processor
22	Case_2	30.08.2024 03:24	30.08.2024 04:11	Receive Notification	Ella	Requester
23	Case_3	15.08.2024 23:11	16.08.2024 00:20	Search Event Information	Steven	Requester
24	Case_3	16.08.2024 00:20	16.08.2024 00:41	Receive Query	System	Request processor
25	Case_3	16.08.2024 00:41	16.08.2024 01:22	Collect Event Information	System	Request processor
26	Case_3	16.08.2024 01:22	16.08.2024 01:34	Receive Event Detail	System	Request processor
27	Case_3	16.08.2024 01:34	16.08.2024 02:04	Input Personal Information	Ella	Requester
28	Case_3	16.08.2024 02:04	16.08.2024 02:49	Submit Booking	Wendy	Requester
29	Case_3	16.08.2024 02:49	16.08.2024 03:37	Receive Booking Request	System	Request processor
30	Case_3	16.08.2024 03:37	16.08.2024 03:57	Sending Payment Informatio	System	Request processor
31	Case_3	16.08.2024 03:57	16.08.2024 04:38	Make a Payment	Andy	Requester
32	Case_3	16.08.2024 04:38	16.08.2024 05:10	Complete the Payment	Wendy	Requester
33	Case_3	16.08.2024 05:10	16.08.2024 06:02	Process Payment	System	Request processor

Importing this file by using the Disco application for working on process analytics.



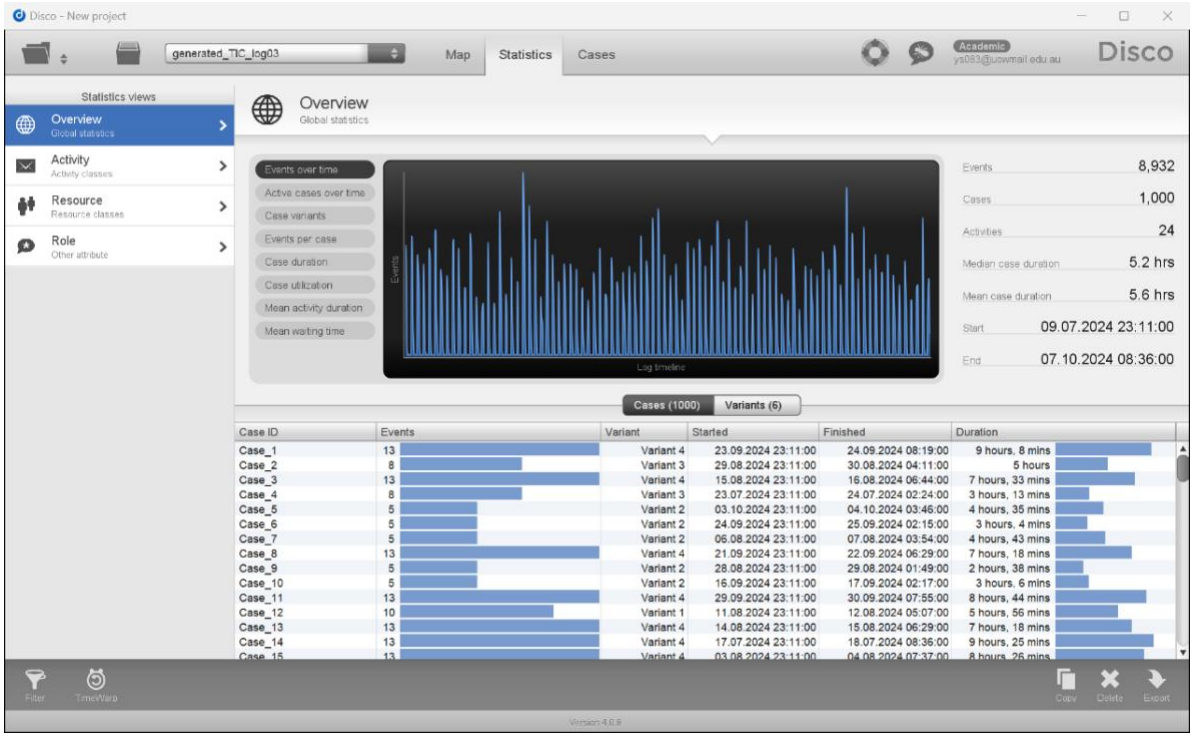
	Case ID	Start Timestamp	Complete Timestamp	Activity	Resource	Role
1	Case_1	04.09.2024 18:26	04.09.2024 20:20	Search Event Information	Andy	Requester
2	Case_1	04.09.2024 20:20	04.09.2024 20:47	Receive Query	System	Request processor
3	Case_1	04.09.2024 20:47	04.09.2024 21:10	Collect Event Information	System	Request processor
4	Case_1	04.09.2024 21:10	04.09.2024 21:50	Receive Event Detail	System	Request processor
5	Case_1	04.09.2024 21:50	04.09.2024 22:02	Input Personal Information	Callista	Requester
6	Case_1	04.09.2024 22:02	04.09.2024 22:57	Submit Booking	Joey	Requester
7	Case_1	04.09.2024 22:57	04.09.2024 23:37	Receive Booking Request	System	Request processor
8	Case_1	04.09.2024 23:37	04.09.2024 23:59	Sending Payment Information	System	Request processor
9	Case_1	04.09.2024 23:59	05.09.2024 00:45	Make a Payment	Simon	Requester
10	Case_1	05.09.2024 00:45	05.09.2024 01:35	Complete the Payment	Tina	Requester
11	Case_1	05.09.2024 01:35	05.09.2024 02:04	Process Payment	System	Request processor
12	Case_1	05.09.2024 02:04	05.09.2024 02:38	Confirm Order Information	System	Request processor
13	Case_1	05.09.2024 02:38	05.09.2024 02:54	Update Status	System	Request processor
14	Case_2	28.08.2024 18:26	28.08.2024 18:42	Sending Booking Details	System	Request processor
15	Case_2	28.08.2024 18:42	28.08.2024 19:15	Receive Booking Details	Tina	Requester
16	Case_2	28.08.2024 19:15	28.08.2024 20:00	Submit Cancellation	David	Requester
17	Case_2	28.08.2024 20:00	28.08.2024 20:31	Receive Cancel Request	System	Request processor
18	Case_2	28.08.2024 20:31	28.08.2024 20:37	Refund	System	Request processor
19	Case_2	28.08.2024 20:37	28.08.2024 21:06	Cancel and Close Order	System	Request processor
20	Case_2	28.08.2024 21:06	28.08.2024 21:36	Notify Customer	System	Request processor
21	Case_2	28.08.2024 21:36	28.08.2024 22:15	Receive Notification	Sunny	Requester
22	Case_3	07.09.2024 18:26	07.09.2024 18:59	Search Event Information	Andy	Requester
23	Case_3	07.09.2024 18:59	07.09.2024 19:06	Receive Query	System	Request processor
24	Case_3	07.09.2024 19:06	07.09.2024 19:11	Collect Event Information	System	Request processor
25	Case_3	07.09.2024 19:11	07.09.2024 19:56	Receive Event Detail	System	Request processor
26	Case_3	07.09.2024 19:56	07.09.2024 20:23	Input Personal Information	Ella	Requester
27	Case_3	07.09.2024 20:23	07.09.2024 21:47	Submit Booking	Wendy	Requester
28	Case_3	07.09.2024 21:47	07.09.2024 22:30	Receive Booking Request	System	Request processor
29	Case_3	07.09.2024 22:30	07.09.2024 23:05	Sending Payment Information	System	Request processor
30	Case_3	07.09.2024 23:05	07.09.2024 23:34	Make a Payment	Mark	Requester
31	Case_3	07.09.2024 23:34	07.09.2024 23:56	Payment Fail	System	Request processor

Online Ordering Process Map



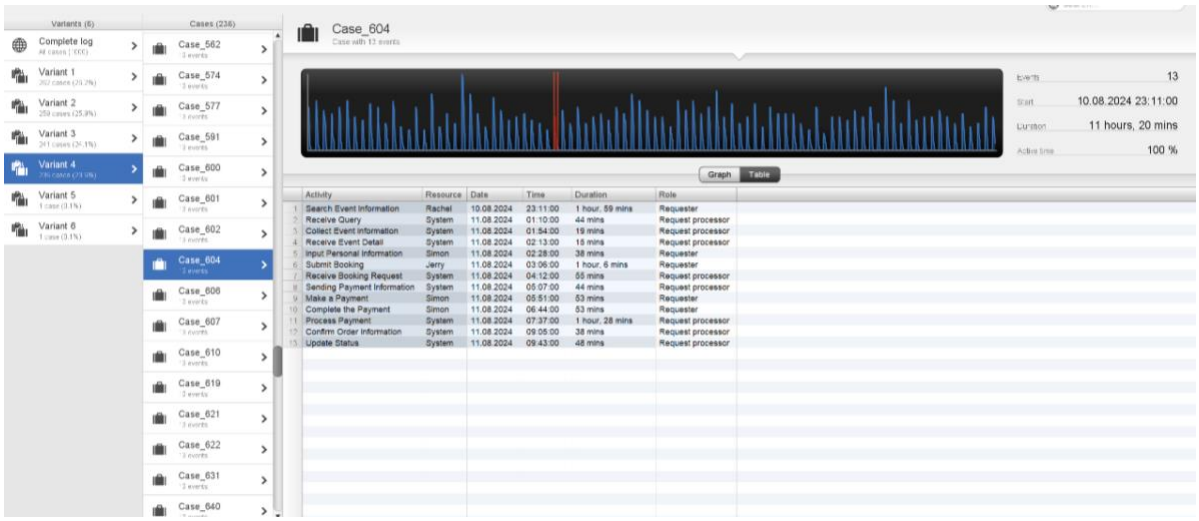
2. Performance Analysis

2.1 Check Statistic



Overview within the duration from 9th July, 2024 to 7th October, 2024, this process analysis contains 8932 events across 1000 cases, and the number of activities on the log is 24. In addition, there are 6 distinct process variants.

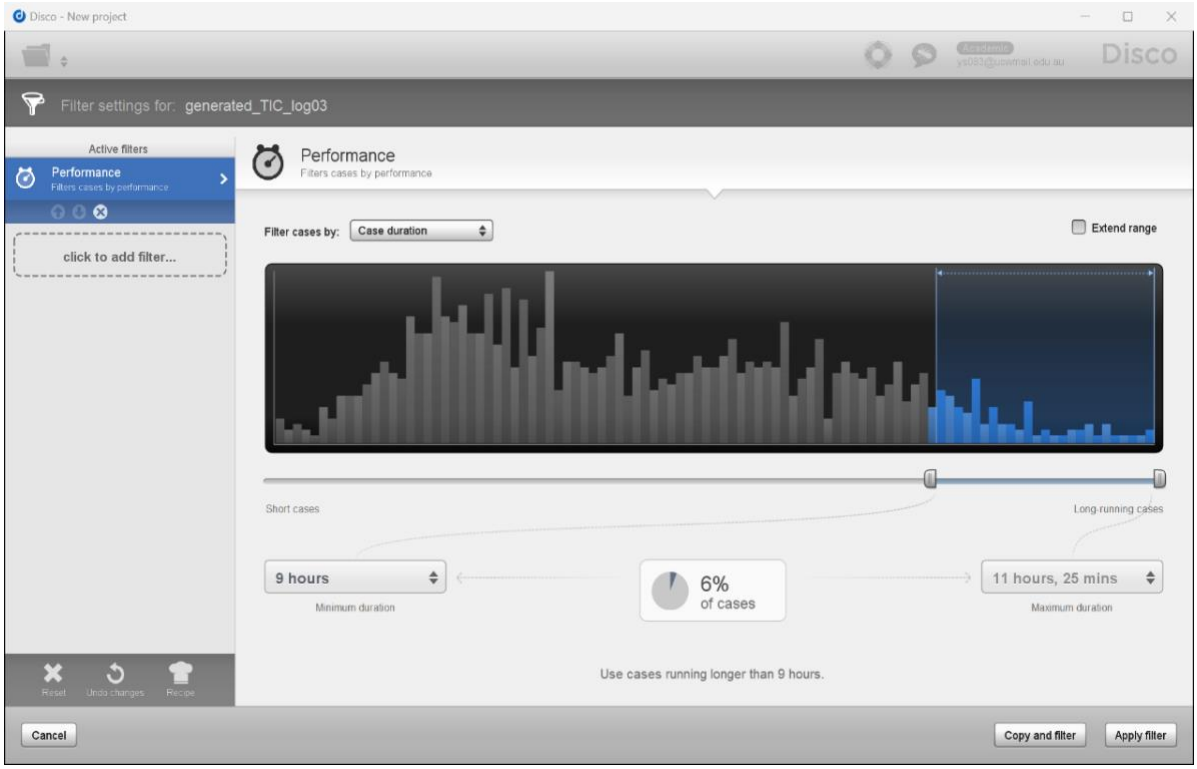
2.2 Check Cases



Variant 4 is selected, and Case_604 is one of the 236 cases (23.6%) that follow this specific variant of the process. The maximum duration is 11 hours, 20 mins.

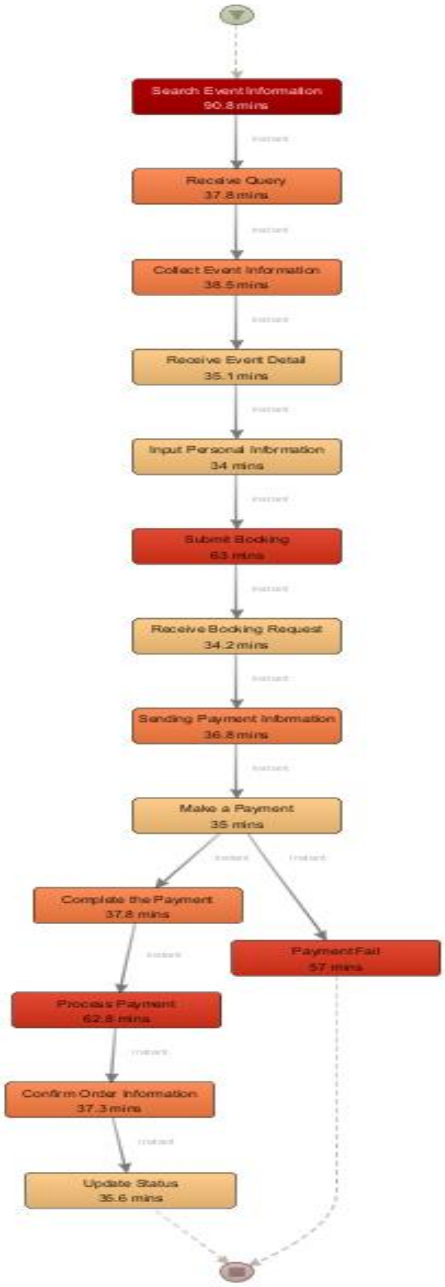
Analysis based on the selection:
Search Event Information, Process Payment, Submit Booking are the longest activities in this case, taking 1 hour, 59 mins and 1 hours, 28 mins and 1 hour 6mins. These steps could potentially be bottlenecks in the process.
In other variants, there are still many cases that have the similar conditions. That's the next thing we need to investigate.

2.3 Performance Filter



Using a performance filter, let analysis focus on a duration between 9 hours and 11 hours, 25 mins. This filter highlights 6% of all cases which are the long-running cases.

2.4 Visual Bottleneck

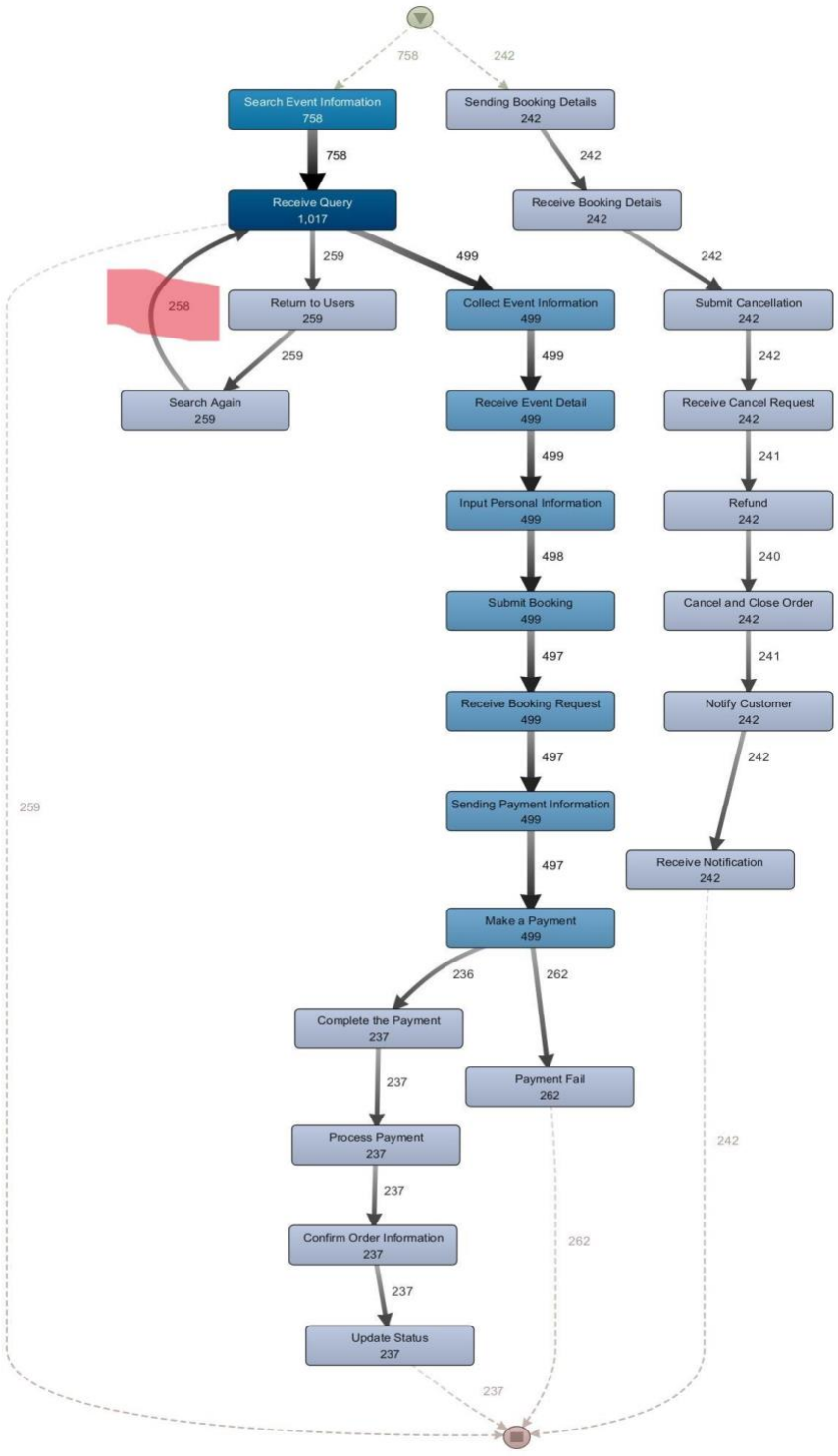


Switch to **Performance** to display **Mean Duration**, this visual process map has been highlighted to show where the most time is being spent. Here are the key points:

- **Search Event Information** takes an average of 90.8 minutes.
- **Submit Booking** takes 63 minutes.
- **Process Payment** takes an average of 62.8 minutes.
- Other steps take between 34-37 minutes on average, suggesting that these steps are more effective than delayed above.

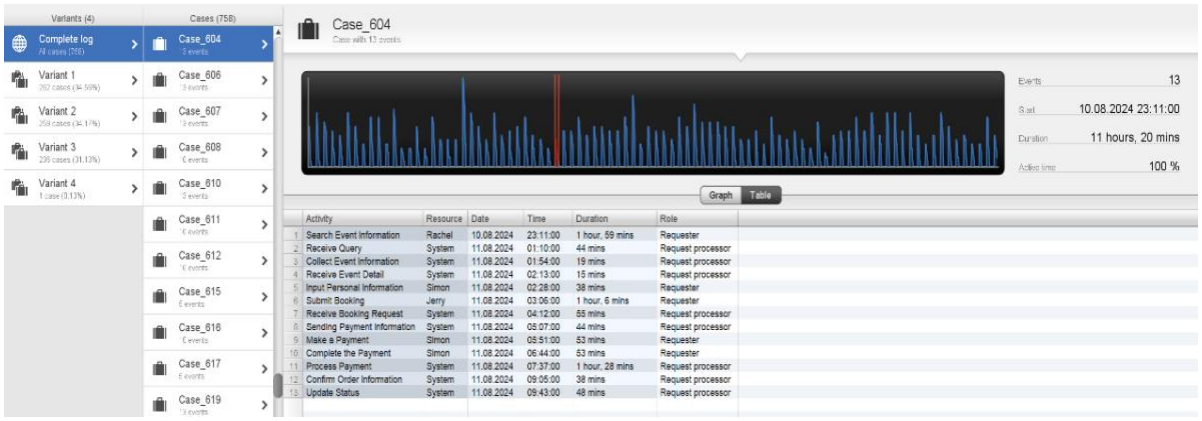
The delayed activity has too many tasks, the whole process will be stuck and need to wait until the final sequence is completed.

2.5 Compliance Check



Switching back to the original (full) data and checking the absolute frequency, case frequency, max. The maximum repetition is the loop one which is Search Again.

2.5.1 Filter All Cases Related to Top3 Low Latency



The Filter-Follower preserves only all cases in the process that exactly follow the specific path "Search Event Information -> Receive Query”
We can analyse the causes of issues and available solutions by obtaining these cases.

3. Recommendations

3.1 Search Event Information

The extended duration for searching event information (90.8 minutes on average) indicates this could be a significant bottleneck in the overall process. It is important to analyze whether the delay stems from user-side issues, such as frequent searches or indecision, or from system inefficiencies. For instance, if the system's response time to a search query surpasses 5 seconds, it may trigger errors, increasing the overall process time and error likelihood.

- **Solution:**
  - **System Side:**
    - **Optimize database performance:** Implement caching strategies and improve indexing to accelerate search query results, reducing the server load and response time.
    - **Reduce response time**: Streamline backend processes to minimize the delay in querying event information and aim for a response time under 5 seconds to avoid error generation.
  - **Customer Side:**
    - **Enhance filter options**: Provide users with more refined search filters (e.g., event type, date range, location) to speed up decision-making and reduce search frequency.
    - **Introduce event recommendations**: Implement a recommendation engine based on user preferences or history to minimize the need for repeated searches.

3.2 Submit Booking

This step averages 63 minutes, potentially due to complex booking confirmation processes or backend delays.

- **Solution:**
    - **Improve booking workflows**: Automate the booking confirmation process and incorporate real-time tracking to quickly identify and address delays.
    - **Load balancing during peak times**: Implement strategies to balance system load during high-traffic periods, reducing the likelihood of slowdowns.

3.3 Process Payment

At 62.8 minutes, processing payments is a considerable delay, which could be due to interactions with external payment systems or database processing issues.

- **Solution:**
    - **Payment status caching**: Cache recent payment information to reduce reliance on external systems and speed up payment processing.
    - **Optimize payment interfaces:** Streamline interactions with third-party payment processors using multithreading to improve communication speed.
    - **Database optimization**: Ensure that order and customer IDs are indexed properly, and use preprocessed data storage to enhance database query performance during payment processing.

Service2: Room Booking

1.Generate Event Logs

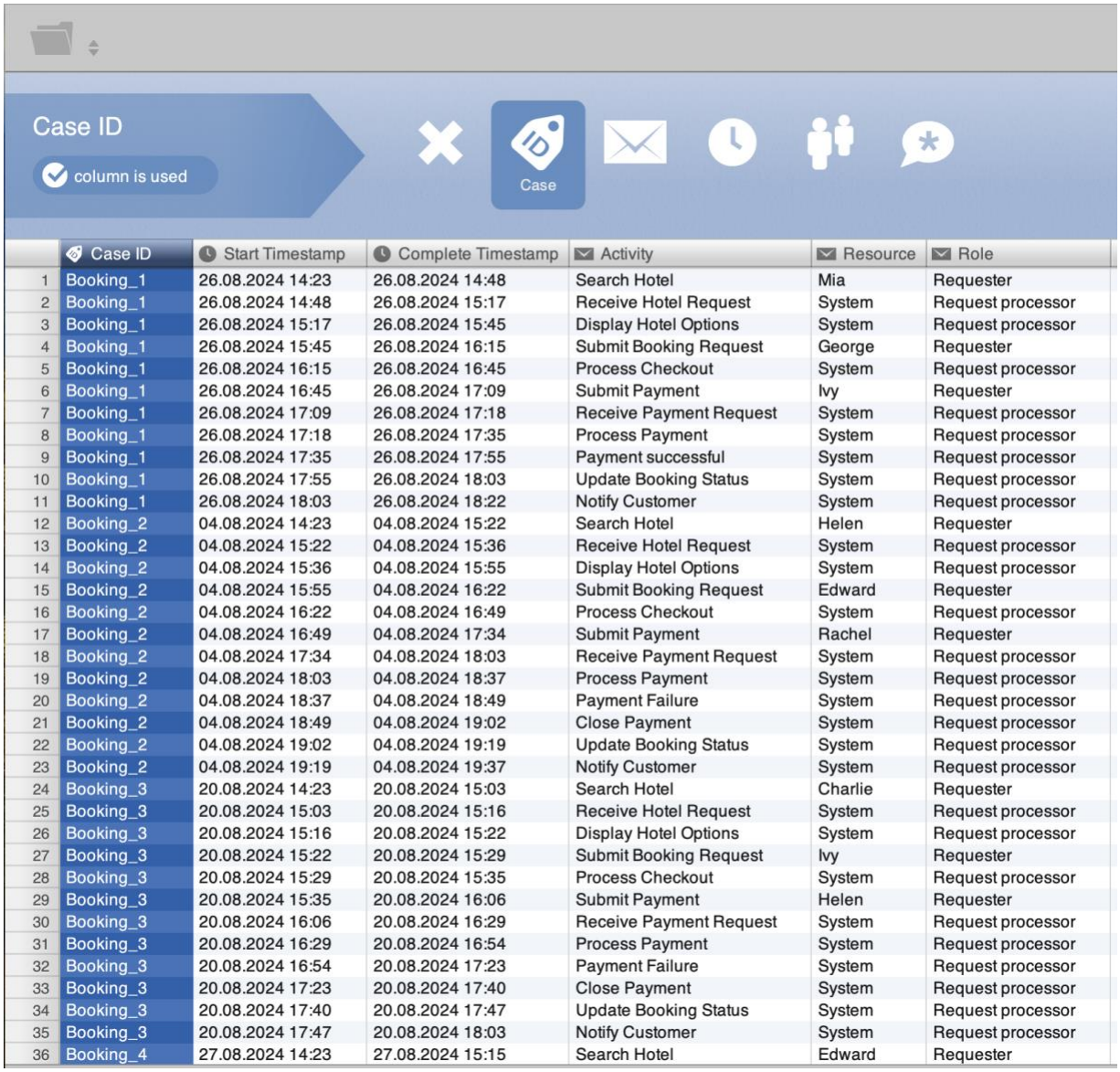
For the creation of the event log we use Python for generating the CSV file.

They implement the code by defining the function for each action by assuming the period randomly which is reliable in reality.

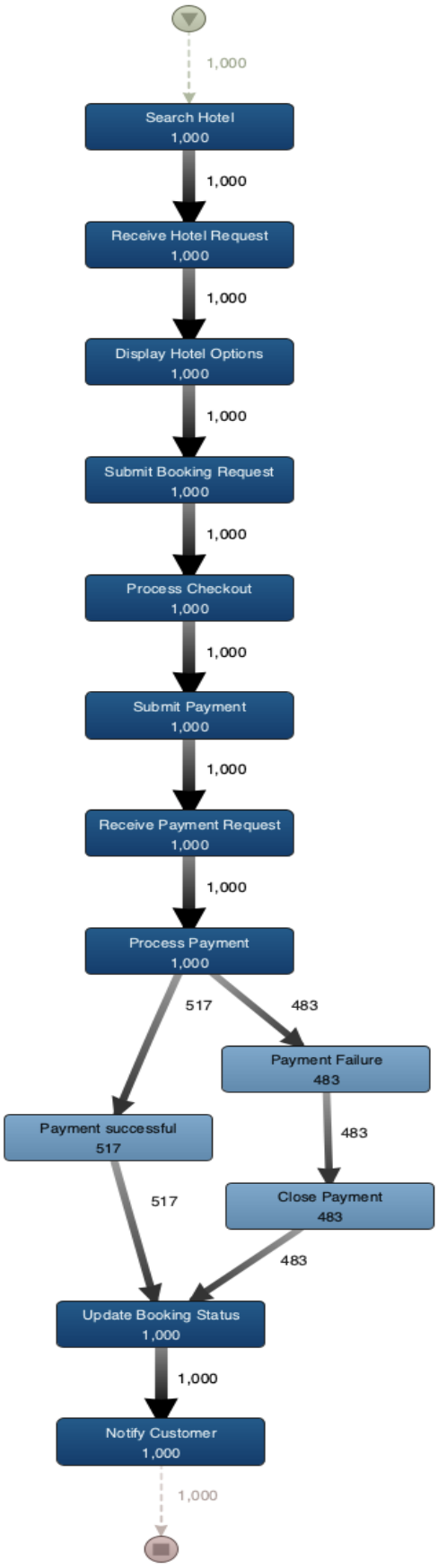
Once a python file is executed, we get the event log in csv format demonstrated below.

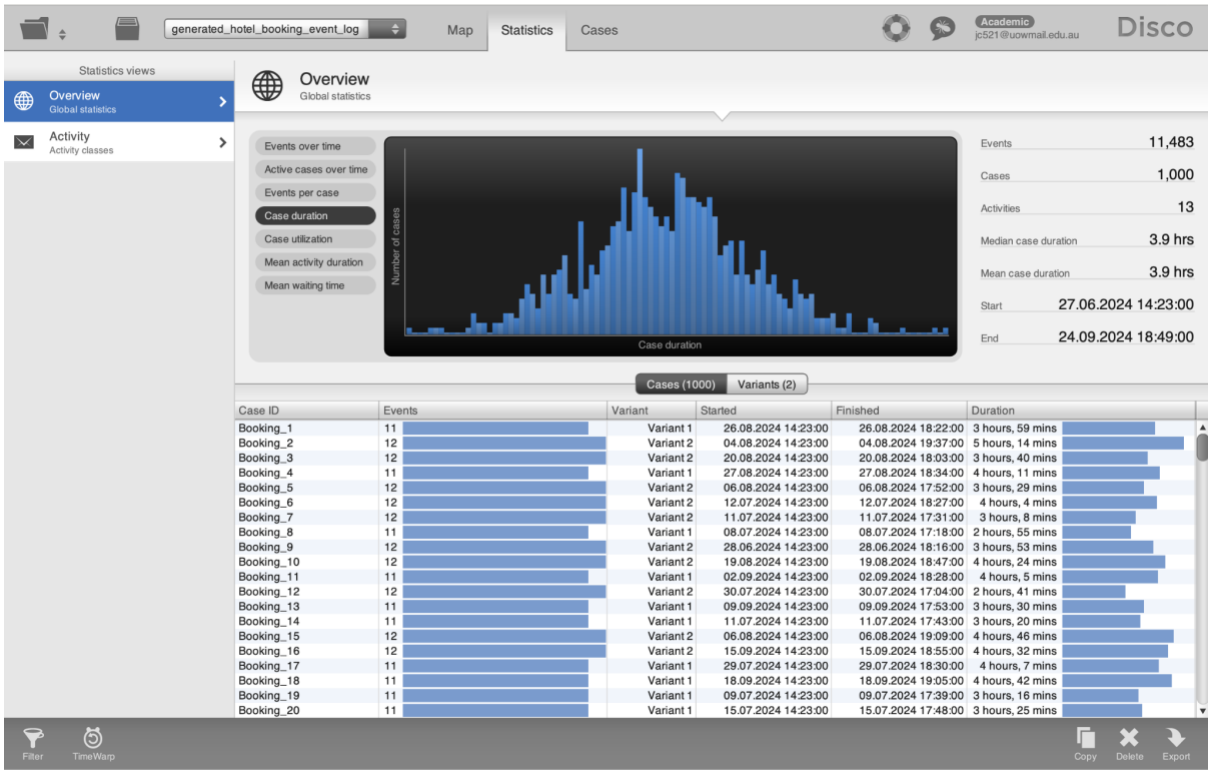
	A	B	C	D	E	F
1	Case ID	Start Timestamp	Complete Timestamp	Activity	Resource	Role
2	Booking_1	26.08.2024 14:23	26.08.2024 14:48	Search Hotel	Mia	Requester
3	Booking_1	26.08.2024 14:48	26.08.2024 15:17	Receive Hotel Request	System	Request processor
4	Booking_1	26.08.2024 15:17	26.08.2024 15:45	Display Hotel Options	System	Request processor
5	Booking_1	26.08.2024 15:45	26.08.2024 16:15	Submit Booking Request	George	Requester
6	Booking_1	26.08.2024 16:15	26.08.2024 16:45	Process Checkout	System	Request processor
7	Booking_1	26.08.2024 16:45	26.08.2024 17:09	Submit Payment	Ivy	Requester
8	Booking_1	26.08.2024 17:09	26.08.2024 17:18	Receive Payment Request	System	Request processor
9	Booking_1	26.08.2024 17:18	26.08.2024 17:35	Process Payment	System	Request processor
10	Booking_1	26.08.2024 17:35	26.08.2024 17:55	Payment successful	System	Request processor
11	Booking_1	26.08.2024 17:55	26.08.2024 18:03	Update Booking Status	System	Request processor
12	Booking_1	26.08.2024 18:03	26.08.2024 18:22	Notify Customer	System	Request processor
13	Booking_2	04.08.2024 14:23	04.08.2024 15:22	Search Hotel	Helen	Requester
14	Booking_2	04.08.2024 15:22	04.08.2024 15:36	Receive Hotel Request	System	Request processor
15	Booking_2	04.08.2024 15:36	04.08.2024 15:55	Display Hotel Options	System	Request processor
16	Booking_2	04.08.2024 15:55	04.08.2024 16:22	Submit Booking Request	Edward	Requester
17	Booking_2	04.08.2024 16:22	04.08.2024 16:49	Process Checkout	System	Request processor
18	Booking_2	04.08.2024 16:49	04.08.2024 17:34	Submit Payment	Rachel	Requester
19	Booking_2	04.08.2024 17:34	04.08.2024 18:03	Receive Payment Request	System	Request processor
20	Booking_2	04.08.2024 18:03	04.08.2024 18:37	Process Payment	System	Request processor
21	Booking_2	04.08.2024 18:37	04.08.2024 18:49	Payment Failure	System	Request processor
22	Booking_2	04.08.2024 18:49	04.08.2024 19:02	Close Payment	System	Request processor
23	Booking_2	04.08.2024 19:02	04.08.2024 19:19	Update Booking Status	System	Request processor
24	Booking_2	04.08.2024 19:19	04.08.2024 19:37	Notify Customer	System	Request processor
25	Booking_3	20.08.2024 14:23	20.08.2024 15:03	Search Hotel	Charlie	Requester
26	Booking_3	20.08.2024 15:03	20.08.2024 15:16	Receive Hotel Request	System	Request processor
27	Booking_3	20.08.2024 15:16	20.08.2024 15:22	Display Hotel Options	System	Request processor
28	Booking_3	20.08.2024 15:22	20.08.2024 15:29	Submit Booking Request	Ivy	Requester
29	Booking_3	20.08.2024 15:29	20.08.2024 15:35	Process Checkout	System	Request processor
30	Booking_3	20.08.2024 15:35	20.08.2024 16:06	Submit Payment	Helen	Requester
31	Booking_3	20.08.2024 16:06	20.08.2024 16:29	Receive Payment Request	System	Request processor
32	Booking_3	20.08.2024 16:29	20.08.2024 16:54	Process Payment	System	Request processor
33	Booking_3	20.08.2024 16:54	20.08.2024 17:23	Payment Failure	System	Request processor
34	Booking_3	20.08.2024 17:23	20.08.2024 17:40	Close Payment	System	Request processor
35	Booking_3	20.08.2024 17:40	20.08.2024 17:47	Update Booking Status	System	Request processor
36	Booking_3	20.08.2024 17:47	20.08.2024 18:03	Notify Customer	System	Request processor

Since the event log file has already exited, we import the file by using the Disco application for working on process analytics



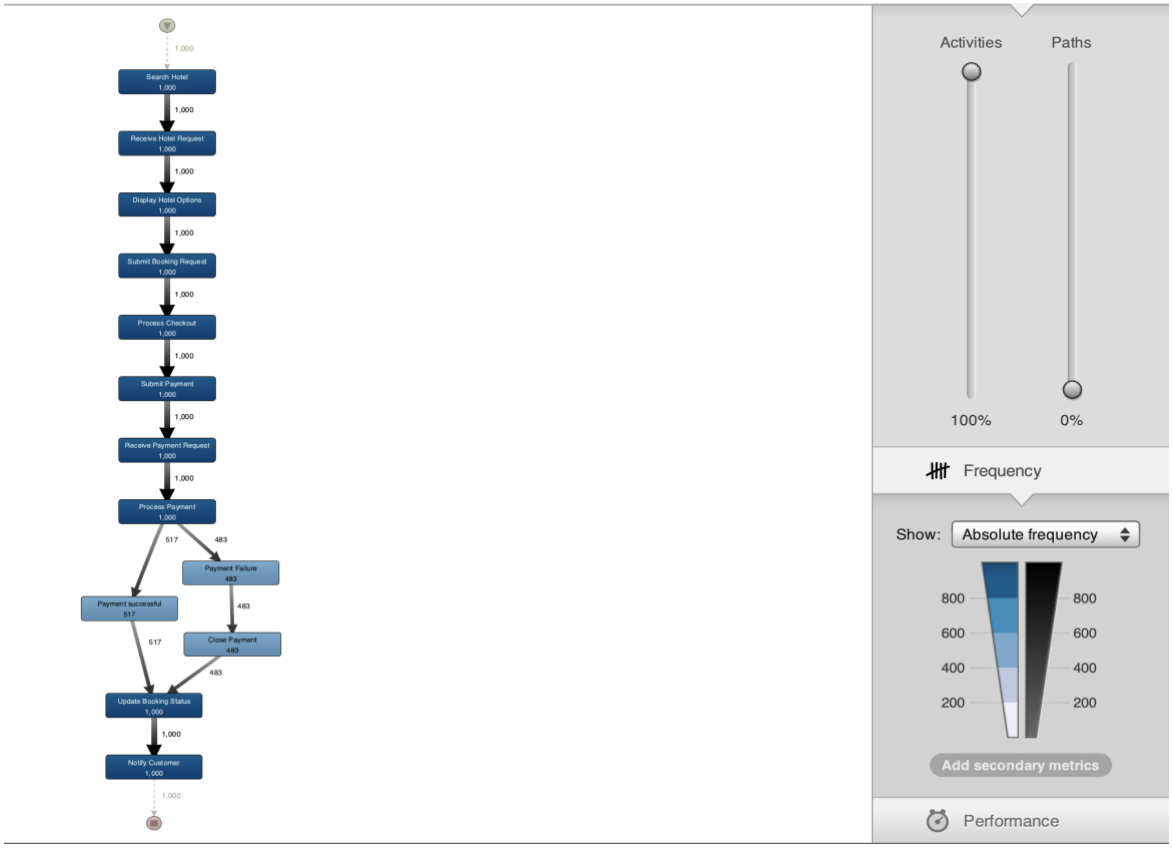
	Case ID	Start Timestamp	Complete Timestamp	Activity	Resource	Role
1	Booking_1	26.08.2024 14:23	26.08.2024 14:48	Search Hotel	Mia	Requester
2	Booking_1	26.08.2024 14:48	26.08.2024 15:17	Receive Hotel Request	System	Request processor
3	Booking_1	26.08.2024 15:17	26.08.2024 15:45	Display Hotel Options	System	Request processor
4	Booking_1	26.08.2024 15:45	26.08.2024 16:15	Submit Booking Request	George	Requester
5	Booking_1	26.08.2024 16:15	26.08.2024 16:45	Process Checkout	System	Request processor
6	Booking_1	26.08.2024 16:45	26.08.2024 17:09	Submit Payment	Ivy	Requester
7	Booking_1	26.08.2024 17:09	26.08.2024 17:18	Receive Payment Request	System	Request processor
8	Booking_1	26.08.2024 17:18	26.08.2024 17:35	Process Payment	System	Request processor
9	Booking_1	26.08.2024 17:35	26.08.2024 17:55	Payment successful	System	Request processor
10	Booking_1	26.08.2024 17:55	26.08.2024 18:03	Update Booking Status	System	Request processor
11	Booking_1	26.08.2024 18:03	26.08.2024 18:22	Notify Customer	System	Request processor
12	Booking_2	04.08.2024 14:23	04.08.2024 15:22	Search Hotel	Helen	Requester
13	Booking_2	04.08.2024 15:22	04.08.2024 15:36	Receive Hotel Request	System	Request processor
14	Booking_2	04.08.2024 15:36	04.08.2024 15:55	Display Hotel Options	System	Request processor
15	Booking_2	04.08.2024 15:55	04.08.2024 16:22	Submit Booking Request	Edward	Requester
16	Booking_2	04.08.2024 16:22	04.08.2024 16:49	Process Checkout	System	Request processor
17	Booking_2	04.08.2024 16:49	04.08.2024 17:34	Submit Payment	Rachel	Requester
18	Booking_2	04.08.2024 17:34	04.08.2024 18:03	Receive Payment Request	System	Request processor
19	Booking_2	04.08.2024 18:03	04.08.2024 18:37	Process Payment	System	Request processor
20	Booking_2	04.08.2024 18:37	04.08.2024 18:49	Payment Failure	System	Request processor
21	Booking_2	04.08.2024 18:49	04.08.2024 19:02	Close Payment	System	Request processor
22	Booking_2	04.08.2024 19:02	04.08.2024 19:19	Update Booking Status	System	Request processor
23	Booking_2	04.08.2024 19:19	04.08.2024 19:37	Notify Customer	System	Request processor
24	Booking_3	20.08.2024 14:23	20.08.2024 15:03	Search Hotel	Charlie	Requester
25	Booking_3	20.08.2024 15:03	20.08.2024 15:16	Receive Hotel Request	System	Request processor
26	Booking_3	20.08.2024 15:16	20.08.2024 15:22	Display Hotel Options	System	Request processor
27	Booking_3	20.08.2024 15:22	20.08.2024 15:29	Submit Booking Request	Ivy	Requester
28	Booking_3	20.08.2024 15:29	20.08.2024 15:35	Process Checkout	System	Request processor
29	Booking_3	20.08.2024 15:35	20.08.2024 16:06	Submit Payment	Helen	Requester
30	Booking_3	20.08.2024 16:06	20.08.2024 16:29	Receive Payment Request	System	Request processor
31	Booking_3	20.08.2024 16:29	20.08.2024 16:54	Process Payment	System	Request processor
32	Booking_3	20.08.2024 16:54	20.08.2024 17:23	Payment Failure	System	Request processor
33	Booking_3	20.08.2024 17:23	20.08.2024 17:40	Close Payment	System	Request processor
34	Booking_3	20.08.2024 17:40	20.08.2024 17:47	Update Booking Status	System	Request processor
35	Booking_3	20.08.2024 17:47	20.08.2024 18:03	Notify Customer	System	Request processor
36	Booking_4	27.08.2024 14:23	27.08.2024 15:15	Search Hotel	Edward	Requester


1,000
Search Hotel
1,000
1,000
Receive Hotel Request
1,000
1,000
Display Hotel Options
1,000
1,000
Submit Booking Request
1,000
1,000
Process Checkout
1,000
1,000
Submit Payment
1,000
1,000
Receive Payment Request
1,000
1,000
Process Payment
1,000
517
483
Payment Failure
483
Payment successful
517
483
517
Close Payment
483
483
Update Booking Status
1,000
1,000
Notify Customer
1,000
1,000



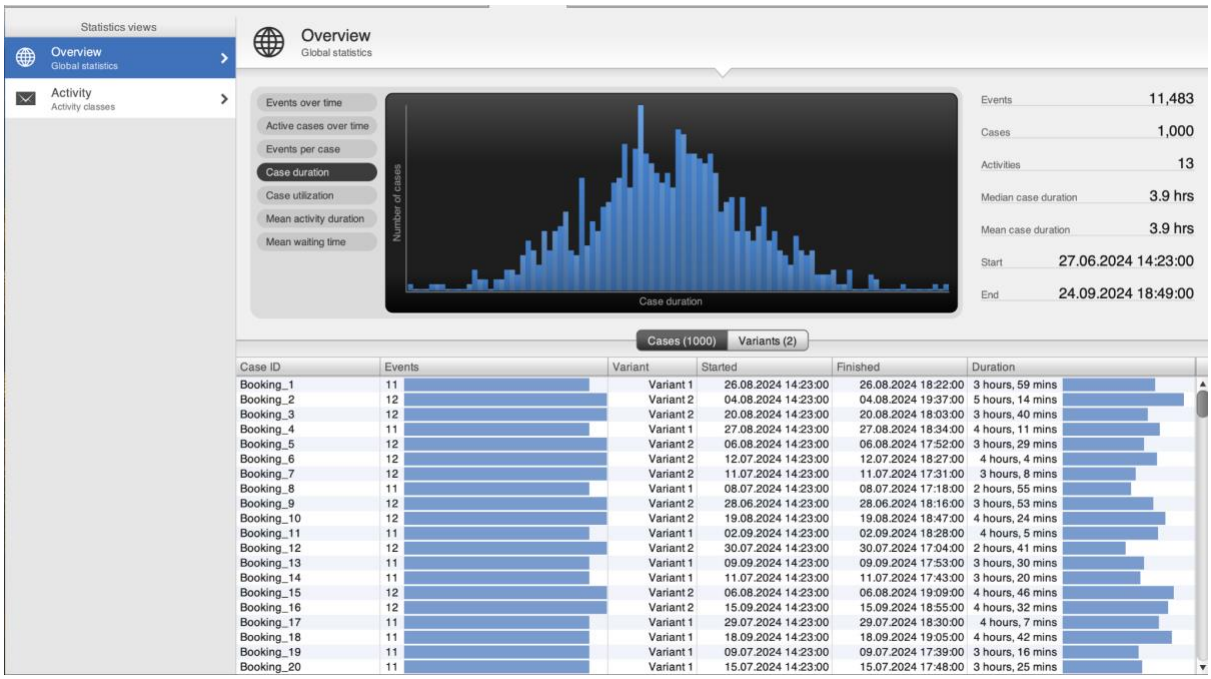
Process Delivery

Absolute frequency



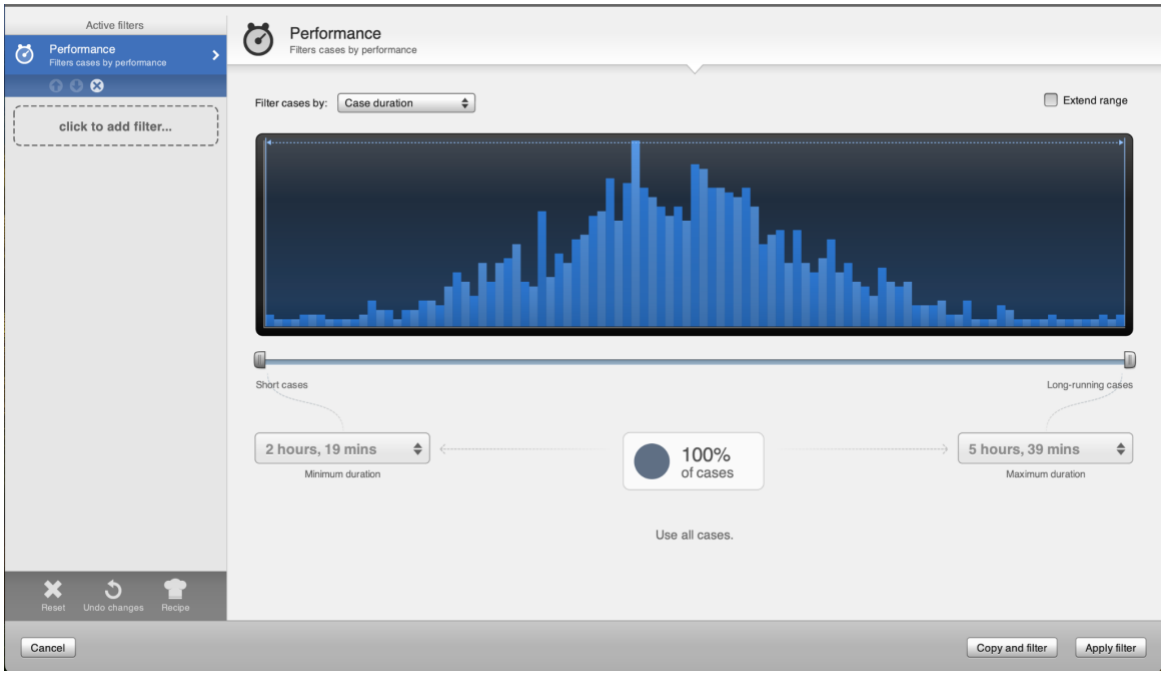
The process of delivery for hotel room booking starts with searching for a hotel, receiving the request, and displaying available options. The customer submits a booking request, processing the checkout, and submitting the payment. Once the payment request is received, it is processed. If successful, the booking status is updated, and the customer is notified. In cases where payment fails, the system closes the payment and informs the customer. The model tracks the absolute frequency of each step, highlighting key activities such as payment success or failure, which influence the subsequent flow of the process.

2.Performance Analysis

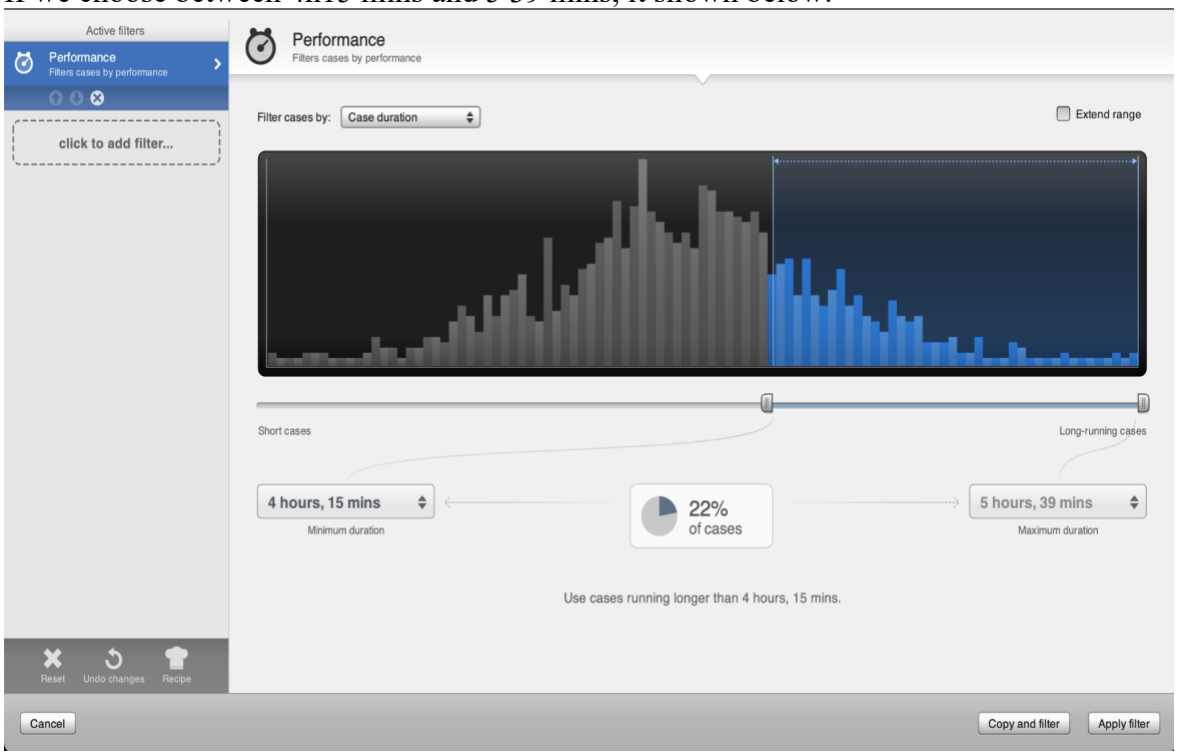


In overview, a total number of 11483 events are generated by 1000 cases of 13 activities.

We could filter the data:

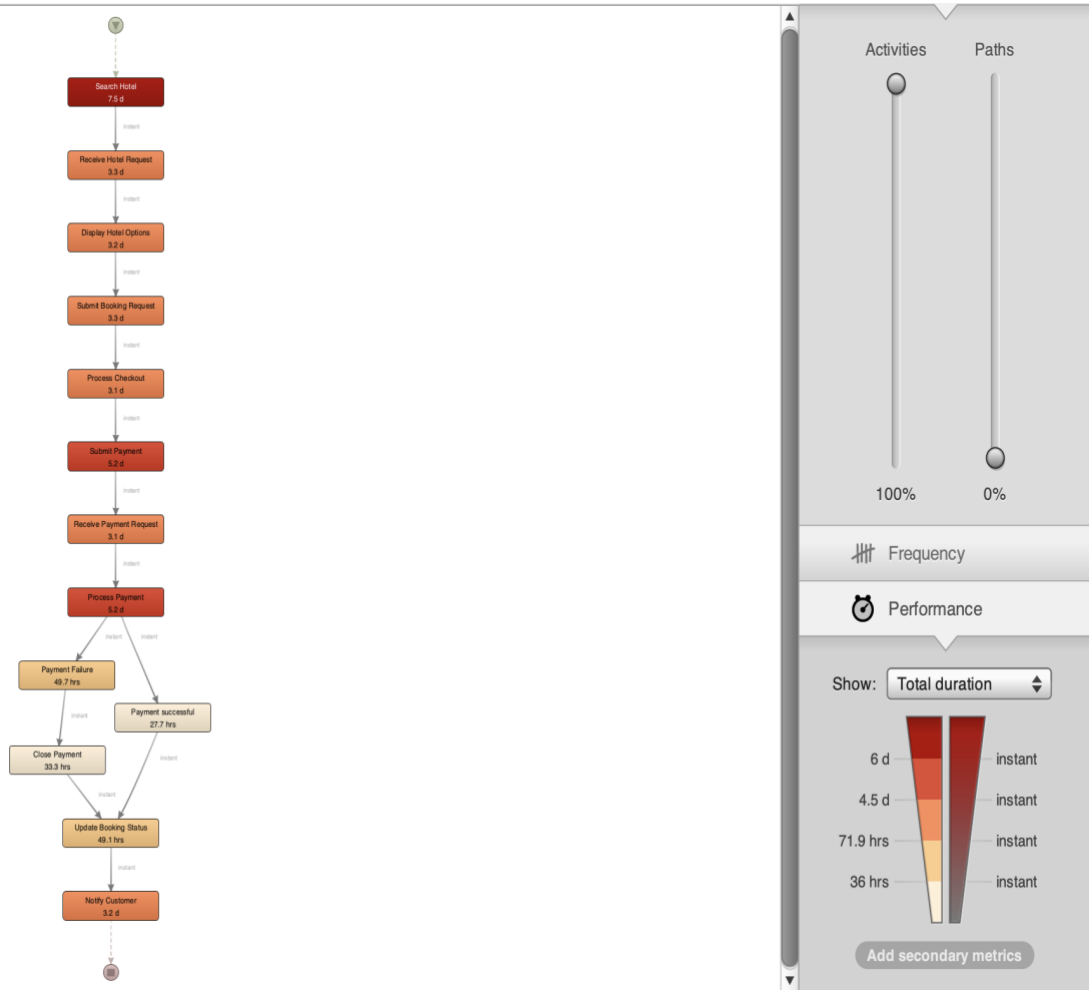


If we choose between 4h15 mins and 5 39 mins, it shown below:

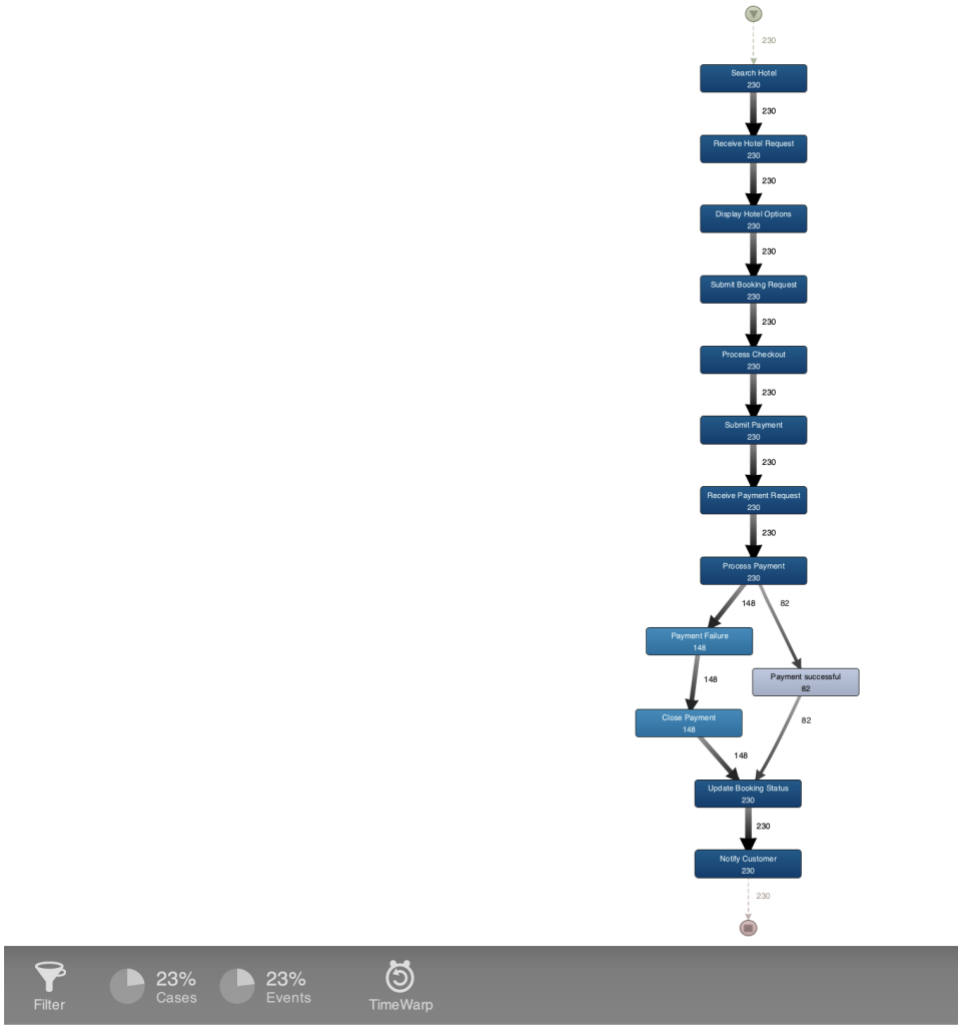


You could see 22% of cases in the long term.

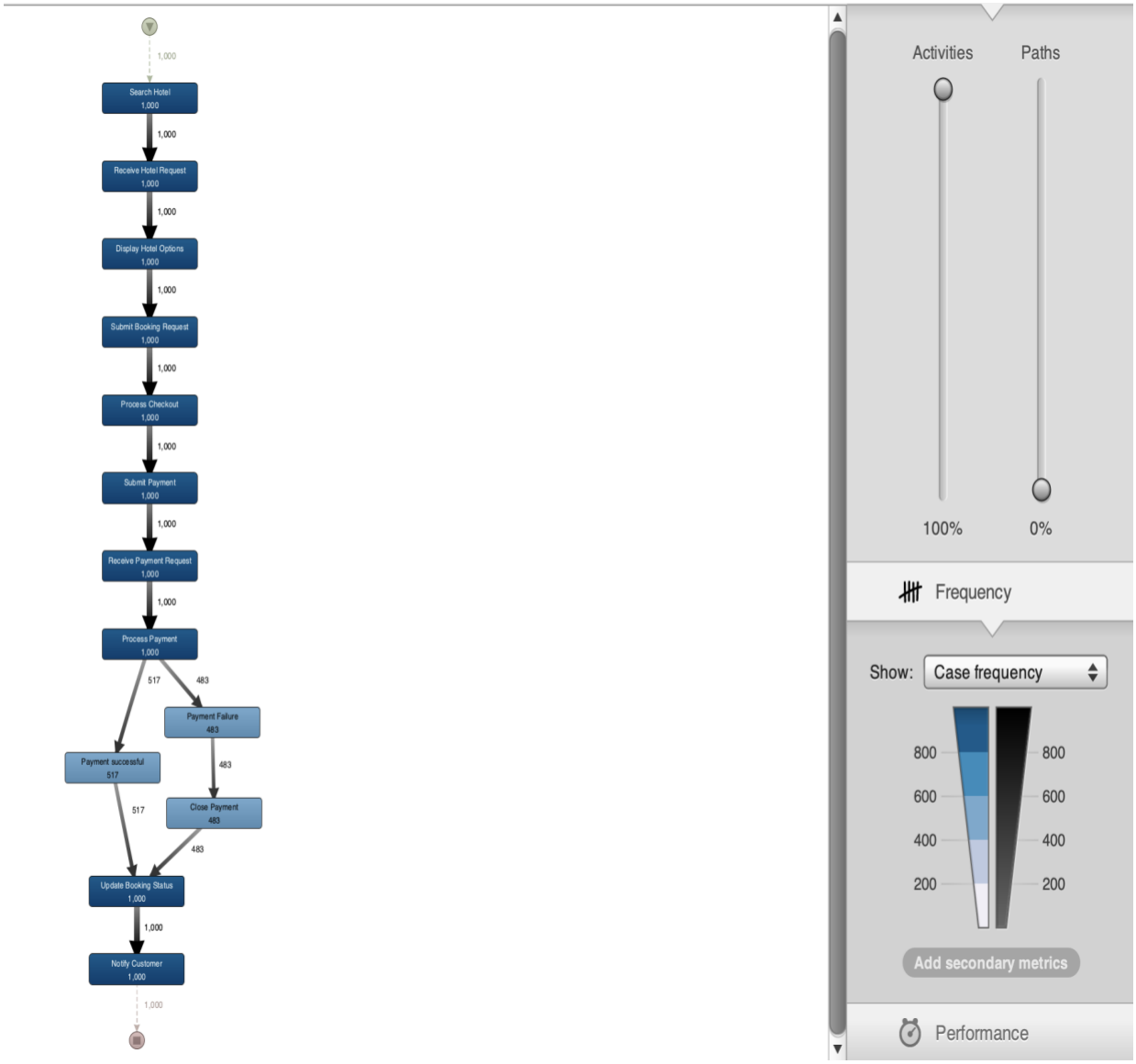
Visual Bottleneck



The absolute frequency diagram shows below:

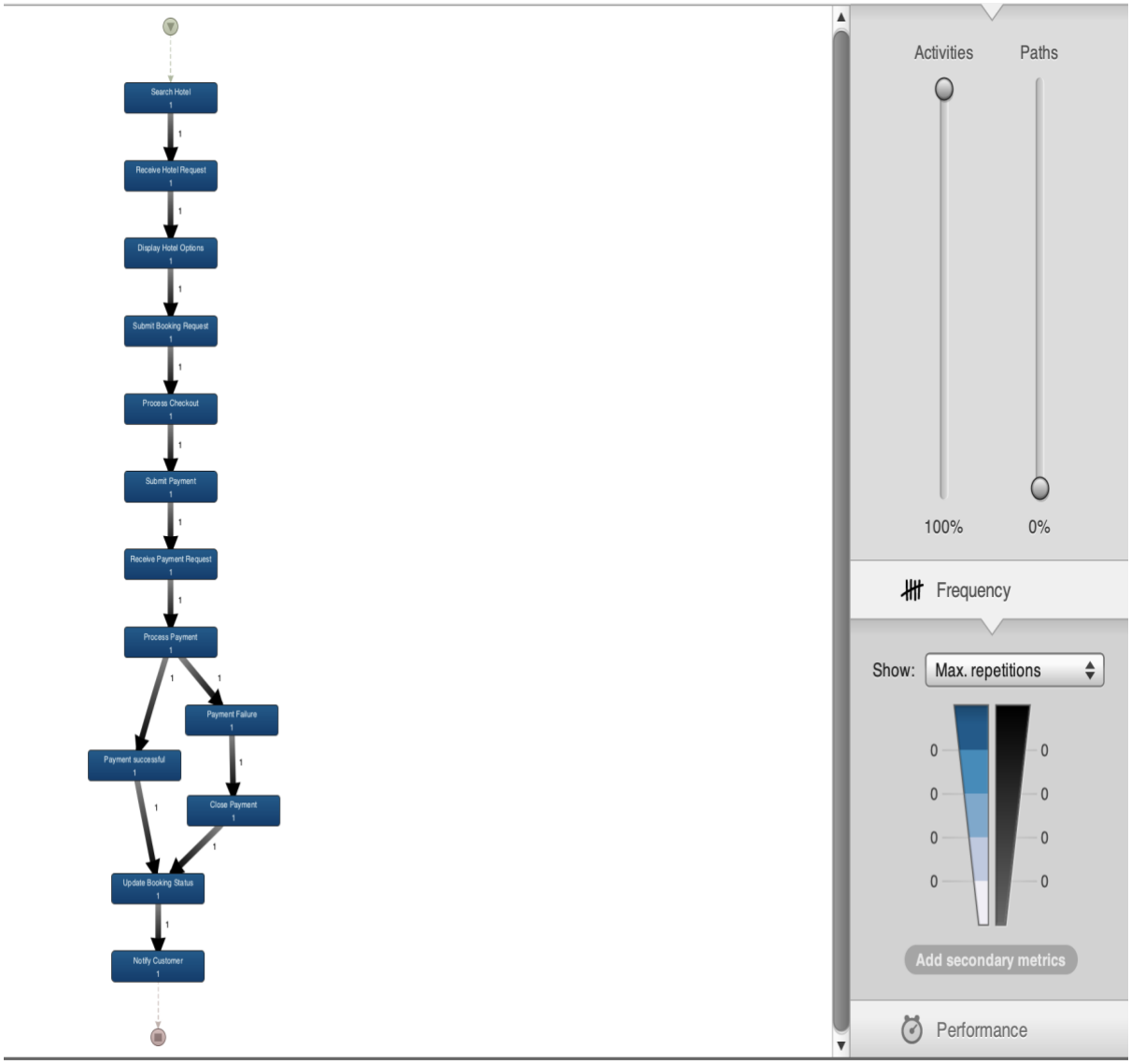


1. Case frequency



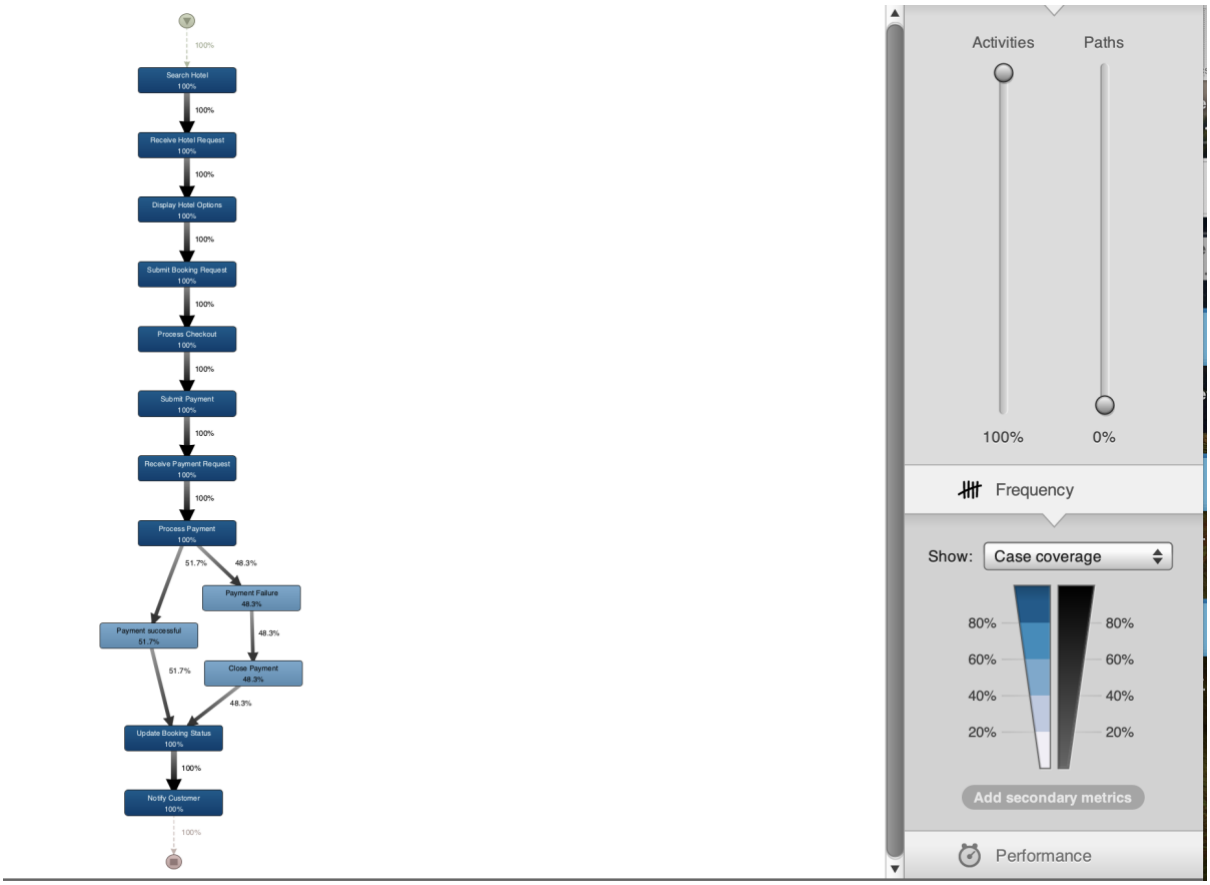
For case frequency we have 10 activities which are above 800 and 3 are below 800.

2. Max repetitions



The maximum repetition is shown in the above figure.

3. Case coverage



For case coverage, we have 9 activities which are giving 100 percent coverage.

3. Recommendations

1. **Search Hotel**
   The duration for the "Search Hotel" activity shows a significantly long average time (7.5 days), suggesting a potential bottleneck. This delay may be caused by either system-side inefficiencies or customer indecision during the hotel search process.
   Solution:
   - **System-Side Improvements:** Optimize database queries, apply caching techniques, and enhance server processing speed to reduce the time it takes to retrieve and display hotel options. This could decrease response time and make the search process more seamless.
   - **Customer-Side Improvements:** Implement personalized hotel recommendations based on user preferences or past bookings. Provide filtering options like location, price, and amenities to assist users in making quicker decisions.
2. **Payment Processing**
   The payment processing activities (submit payment, process payment) exhibit delays of over 5 days. Payment failures add an additional delay, extending the process

unnecessarily.

Solution:

- **Payment Gateway Optimization:** Collaborate with payment processors to ensure faster transaction processing times. Use multithreading to handle multiple payments simultaneously and reduce overall time.
- **Automated Retry Mechanism:** Implement an automated retry for failed payments, reducing the time between retries and notifying customers promptly when a failure occurs.

3. **Update Booking Status and Notify Customer**
   Updating the booking status and notifying the customer takes around 49 hours and 3.2 days, respectively. These durations seem longer than necessary and may result from system lags or poor notification handling.
   Solution:
   - **Real-Time Updates:** Ensure that once payment is confirmed, booking status is updated immediately using an event-driven architecture to minimize delays.
   - **Streamlined Notification System:** Use SMS or push notifications for real-time customer updates, reducing the lag time in notifying customers about their booking.

Service3: Online Ordering

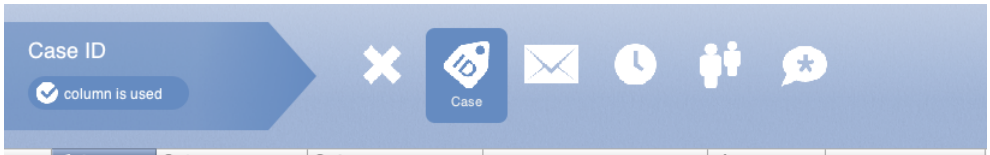
1. Generate Event Logs

The purpose of using python to generate event logs to assume periods randomly and automatically, increasing confidence that they fit real-world cases. Main elements of a log: Case ID, Start Timestamp, Complete Timestamp, Activity, Resource, Role.

After executing the implementation code and simulation conditions, the obtained log is displayed：

	A	B	C	D	E	F
1	Case ID	Start Timestamp	Complete Timestamp	Activity	Resource	Role
2	Case_1	16.09.2024 14:48	16.09.2024 15:56	Search Restaurant	Yao	Requester
3	Case_1	16.09.2024 15:56	16.09.2024 16:48	Receive Restaurant Request	System	Request processor
4	Case_1	16.09.2024 16:48	16.09.2024 17:31	Display Menu	System	Request processor
5	Case_1	16.09.2024 17:31	16.09.2024 18:03	Submit Order	Zhizhi Wang	Requester
6	Case_1	16.09.2024 18:03	16.09.2024 18:21	Process Checkout	System	Request processor
7	Case_1	16.09.2024 18:21	16.09.2024 18:54	Submit Payment	Amanda	Requester
8	Case_1	16.09.2024 18:54	16.09.2024 19:45	Receive Payment Request	System	Request processor
9	Case_1	16.09.2024 19:45	16.09.2024 20:08	Process Payment	System	Request processor
10	Case_1	16.09.2024 20:08	16.09.2024 21:07	Payment Failure	System	Request processor
11	Case_1	16.09.2024 21:07	16.09.2024 21:44	Close Payment	System	Request processor
12	Case_1	16.09.2024 21:44	16.09.2024 21:57	Update status	System	Request processor
13	Case_1	16.09.2024 21:57	16.09.2024 22:44	notify customer	System	Request processor
14	Case_2	12.07.2024 14:48	12.07.2024 16:09	Search Restaurant	Jingjing	Requester
15	Case_2	12.07.2024 16:09	12.07.2024 16:51	Receive Restaurant Request	System	Request processor
16	Case_2	12.07.2024 16:51	12.07.2024 17:07	Display Menu	System	Request processor
17	Case_2	12.07.2024 17:07	12.07.2024 17:23	Submit Order	Alice	Requester
18	Case_2	12.07.2024 17:23	12.07.2024 17:47	Process Checkout	System	Request processor
19	Case_2	12.07.2024 17:47	12.07.2024 18:04	Submit Payment	Jianlian Yi	Requester
20	Case_2	12.07.2024 18:04	12.07.2024 18:45	Receive Payment Request	System	Request processor
21	Case_2	12.07.2024 18:45	12.07.2024 20:11	Process Payment	System	Request processor
22	Case_2	12.07.2024 20:11	12.07.2024 20:35	Payment successful	System	Request processor
23	Case_2	12.07.2024 20:35	12.07.2024 23:18	Prepare the order	System	Request processor
24	Case_2	12.07.2024 23:18	12.07.2024 23:59	Update status	System	Request processor
25	Case_2	12.07.2024 23:59	13.07.2024 00:08	notify customer	System	Request processor
26	Case_3	20.07.2024 14:48	20.07.2024 16:30	Search Restaurant	Han	Requester
27	Case_3	20.07.2024 16:30	20.07.2024 17:01	Receive Restaurant Request	System	Request processor
28	Case_3	20.07.2024 17:01	20.07.2024 17:33	Display Menu	System	Request processor
29	Case_3	20.07.2024 17:33	20.07.2024 18:16	Submit Order	Jianlian Yi	Requester
30	Case_3	20.07.2024 18:16	20.07.2024 19:08	Process Checkout	System	Request processor
31	Case_3	20.07.2024 19:08	20.07.2024 19:39	Submit Cancellation	Sarah	Requester
32	Case_3	20.07.2024 19:39	20.07.2024 19:54	Receive Cancel Request	System	Request processor
33	Case_3	20.07.2024 19:54	20.07.2024 21:50	Cancel and Close order	System	Request processor
34	Case_3	20.07.2024 21:50	20.07.2024 22:15	Update status	System	Request processor
35	Case_3	20.07.2024 22:15	20.07.2024 22:31	notify customer	System	Request processor
36	Case_4	02.08.2024 14:48	02.08.2024 15:29	Search Restaurant	John	Requester

Importing this file by using the Disco application for working on process analytics.

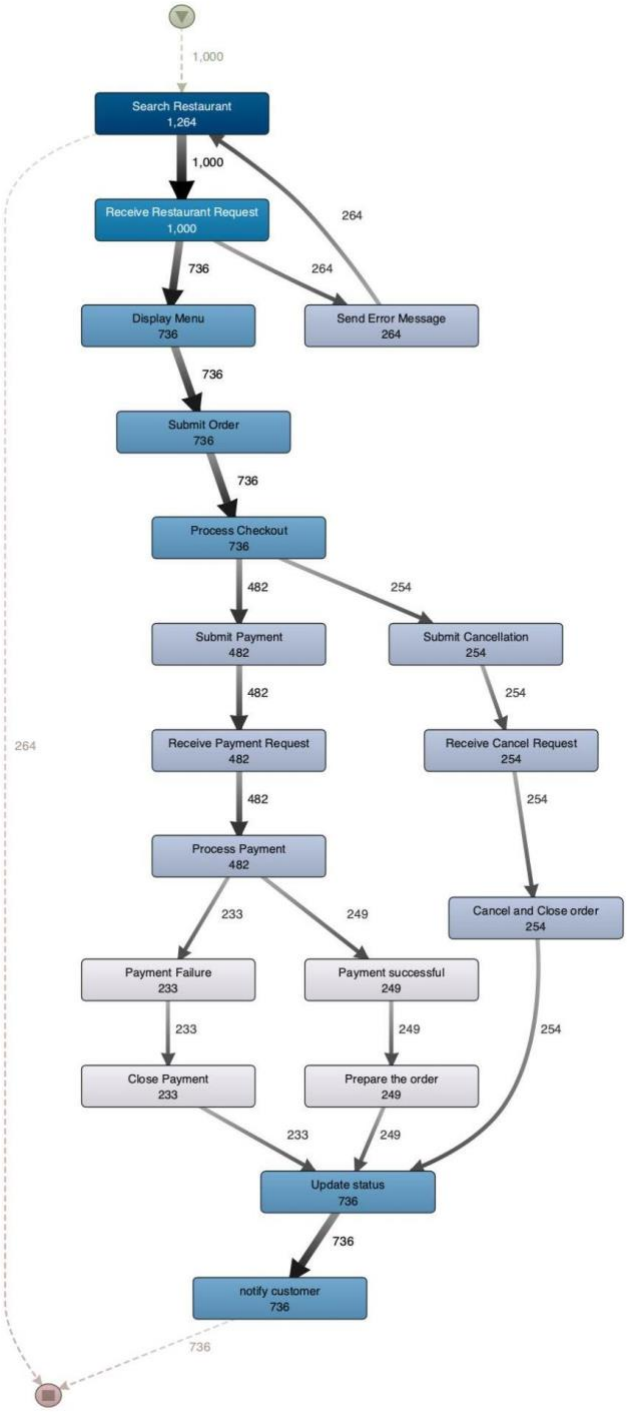


	Case ID	Start Timestamp	Complete Timestamp	Activity	Resource	Role
1	Case_1	16.09.2024 14:48	16.09.2024 15:56	Search Restaurant	Yao	Requester
2	Case_1	16.09.2024 15:56	16.09.2024 16:48	Receive Restaurant Request	System	Request processor
3	Case_1	16.09.2024 16:48	16.09.2024 17:31	Display Menu	System	Request processor
4	Case_1	16.09.2024 17:31	16.09.2024 18:03	Submit Order	Zhizhi Wang	Requester
5	Case_1	16.09.2024 18:03	16.09.2024 18:21	Process Checkout	System	Request processor
6	Case_1	16.09.2024 18:21	16.09.2024 18:54	Submit Payment	Amanda	Requester
7	Case_1	16.09.2024 18:54	16.09.2024 19:45	Receive Payment Request	System	Request processor
8	Case_1	16.09.2024 19:45	16.09.2024 20:08	Process Payment	System	Request processor
9	Case_1	16.09.2024 20:08	16.09.2024 21:07	Payment Failure	System	Request processor
10	Case_1	16.09.2024 21:07	16.09.2024 21:44	Close Payment	System	Request processor
11	Case_1	16.09.2024 21:44	16.09.2024 21:57	Update status	System	Request processor
12	Case_1	16.09.2024 21:57	16.09.2024 22:44	notify customer	System	Request processor
13	Case_2	12.07.2024 14:48	12.07.2024 16:09	Search Restaurant	Jingjing	Requester
14	Case_2	12.07.2024 16:09	12.07.2024 16:51	Receive Restaurant Request	System	Request processor
15	Case_2	12.07.2024 16:51	12.07.2024 17:07	Display Menu	System	Request processor
16	Case_2	12.07.2024 17:07	12.07.2024 17:23	Submit Order	Alice	Requester
17	Case_2	12.07.2024 17:23	12.07.2024 17:47	Process Checkout	System	Request processor
18	Case_2	12.07.2024 17:47	12.07.2024 18:04	Submit Payment	Jianlian Yi	Requester
19	Case_2	12.07.2024 18:04	12.07.2024 18:45	Receive Payment Request	System	Request processor
20	Case_2	12.07.2024 18:45	12.07.2024 20:11	Process Payment	System	Request processor
21	Case_2	12.07.2024 20:11	12.07.2024 20:35	Payment successful	System	Request processor
22	Case_2	12.07.2024 20:35	12.07.2024 23:18	Prepare the order	System	Request processor
23	Case_2	12.07.2024 23:18	12.07.2024 23:59	Update status	System	Request processor
24	Case_2	12.07.2024 23:59	13.07.2024 00:08	notify customer	System	Request processor
25	Case_3	20.07.2024 14:48	20.07.2024 16:30	Search Restaurant	Han	Requester
26	Case_3	20.07.2024 16:30	20.07.2024 17:01	Receive Restaurant Request	System	Request processor
27	Case_3	20.07.2024 17:01	20.07.2024 17:33	Display Menu	System	Request processor
28	Case_3	20.07.2024 17:33	20.07.2024 18:16	Submit Order	Jianlian Yi	Requester
29	Case_3	20.07.2024 18:16	20.07.2024 19:08	Process Checkout	System	Request processor
30	Case_3	20.07.2024 19:08	20.07.2024 19:39	Submit Cancellation	Sarah	Requester
31	Case_3	20.07.2024 19:39	20.07.2024 19:54	Receive Cancel Request	System	Request processor
32	Case_3	20.07.2024 19:54	20.07.2024 21:50	Cancel and Close order	System	Request processor
33	Case_3	20.07.2024 21:50	20.07.2024 22:15	Update status	System	Request processor
34	Case_3	20.07.2024 22:15	20.07.2024 22:31	notify customer	System	Request processor
35	Case_4	02.08.2024 14:48	02.08.2024 15:29	Search Restaurant	John	Requester
36	Case_4	02.08.2024 15:29	02.08.2024 16:21	Receive Restaurant Request	System	Request processor
37	Case_4	02.08.2024 16:21	02.08.2024 17:05	Send Error Message	System	Request processor
38	Case_4	02.08.2024 17:05	02.08.2024 18:17	Search Restaurant	Marry	Requester
39	Case_5	27.06.2024 14:48	27.06.2024 15:36	Search Restaurant	Ben	Requester

Online Ordering Process Map

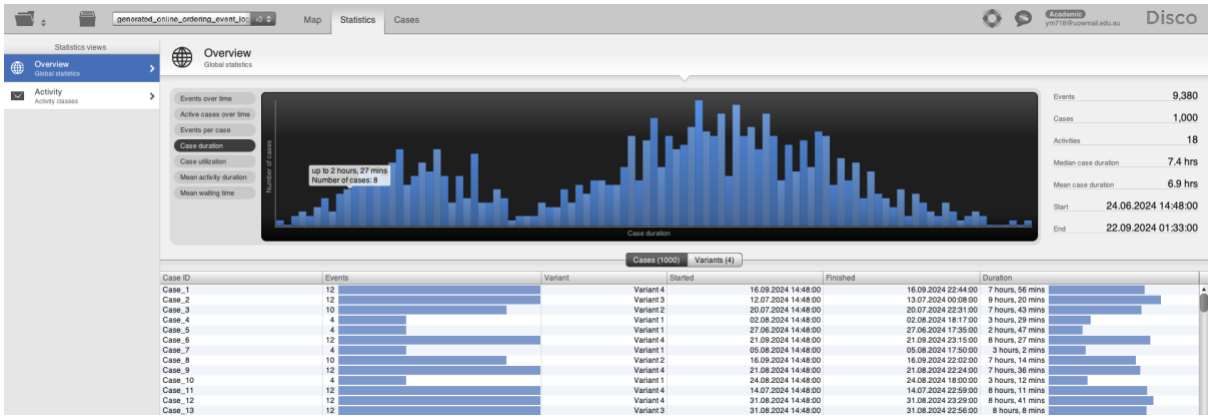
At the beginning, sliding the “Activity” and “Paths” to 100% in order to display the reality of how detailed and complex the process changes in the real-world case. It shows a whole process

of Online Ordering, including Search Restaurant, Send Error Message(if trigger error), Display Menu, Process Checkout and Submit Cancellation(if the customer submitted) etc.



2.Performance Analysis

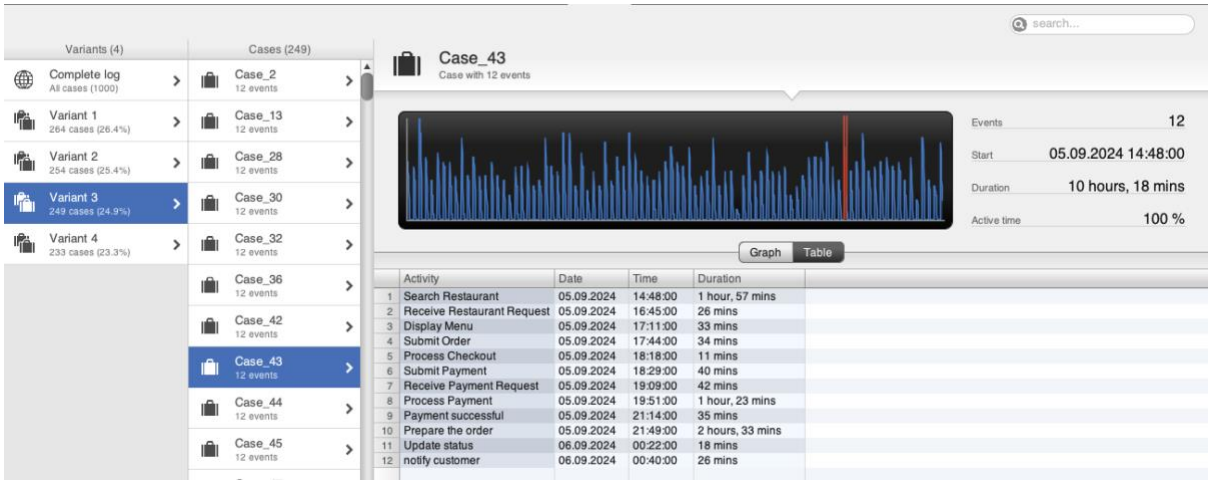
2.1 Check Statistic



Overview within the duration from 24 June, 2024 to 22 September, 2024, this process analysis contains 9380 events across 1000 cases, and the number of activities on the log is 18. In addition, there are 4 distinct process variants.

The duration histogram shows, there are more cases that take around 5 to 7 hours, a few cases are complete under 2 hours. However, there are also cases that stretch beyond 10 hours, this seems to be something seriously wrong with the whole process.

2.2 Check Cases

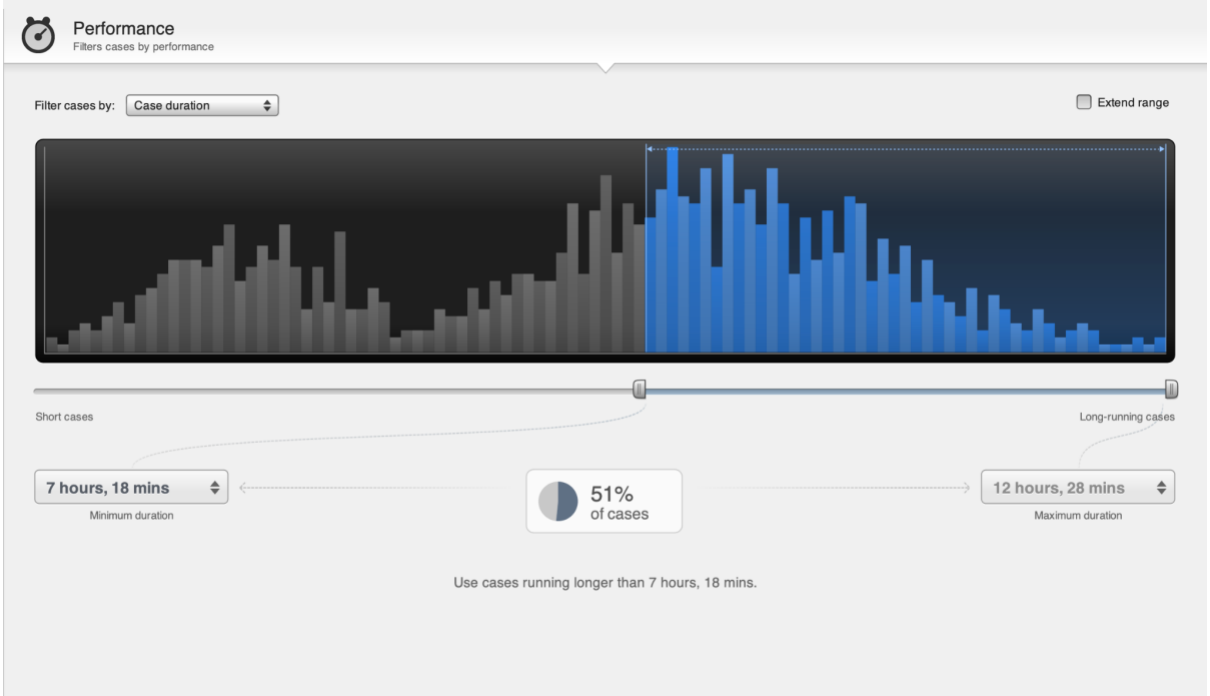


	Activity	Date	Time	Duration
1	Search Restaurant	05.09.2024	14:48:00	1 hour, 57 mins
2	Receive Restaurant Request	05.09.2024	16:45:00	26 mins
3	Display Menu	05.09.2024	17:11:00	33 mins
4	Submit Order	05.09.2024	17:44:00	34 mins
5	Process Checkout	05.09.2024	18:18:00	11 mins
6	Submit Payment	05.09.2024	18:29:00	40 mins
7	Receive Payment Request	05.09.2024	19:09:00	42 mins
8	Process Payment	05.09.2024	19:51:00	1 hour, 23 mins
9	Payment successful	05.09.2024	21:14:00	35 mins
10	Prepare the order	05.09.2024	21:49:00	2 hours, 33 mins
11	Update status	06.09.2024	00:22:00	18 mins
12	notify customer	06.09.2024	00:40:00	26 mins

Variant 3 is selected, and Case_43 is one of the 249 cases (24.9%) that follow this specific variant of the process. The maximum duration is 10 hours, 18 mins.

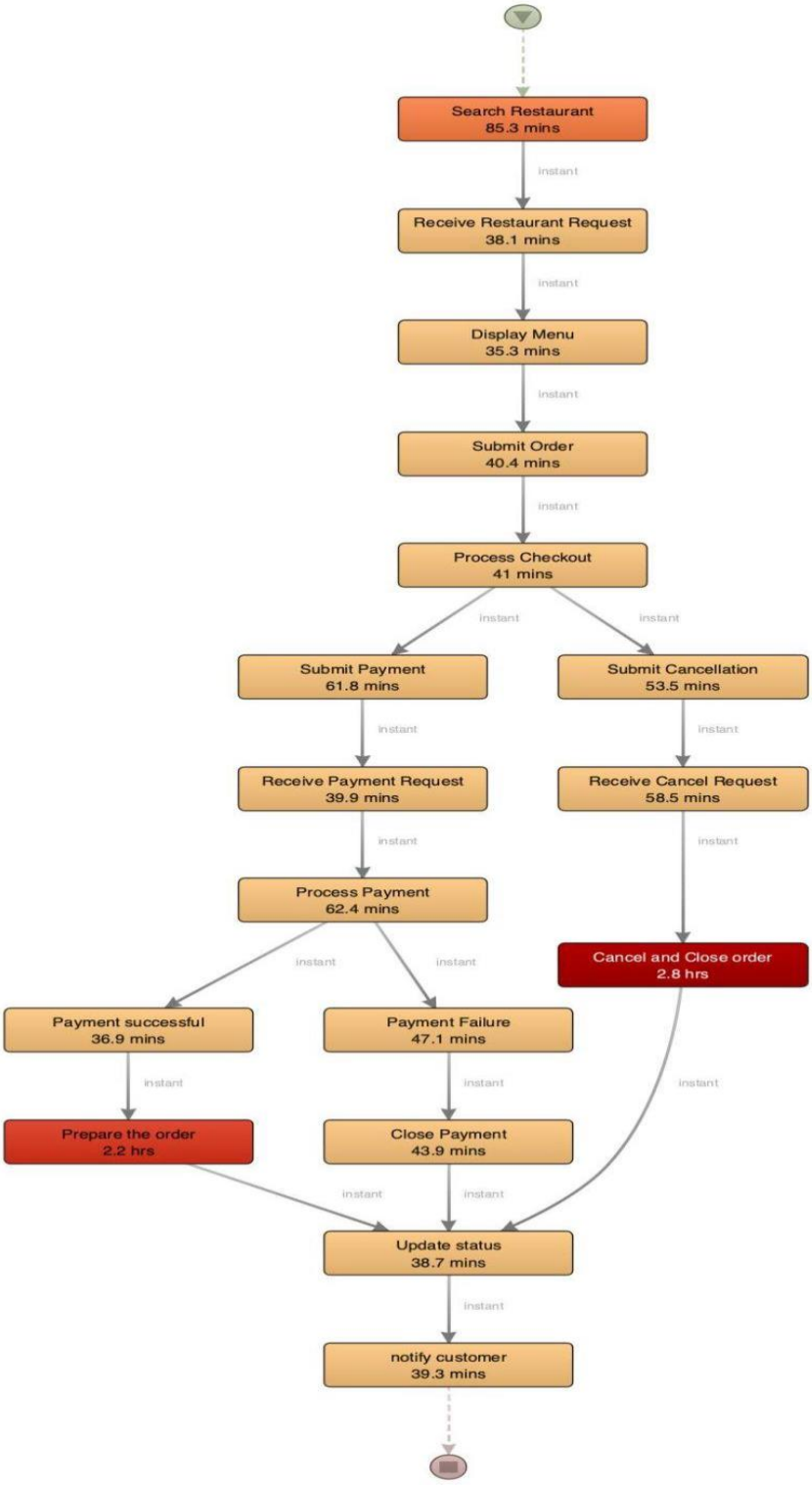
Analysis based on the selection:
Search Restaurant, Prepare the order and Process Payment are the longest activities in this case, taking 1 hour, 57 mins and 2 hours, 33 mins and 1 hour, 33 mins. These steps could potentially be bottlenecks in the process.
In other variants, there are still many cases that have the similar conditions. That's the next thing we need to investigate.

2.3 Performance Filter



Using a performance filter, let analysis focus on a duration between 7 hours and 12 hours. This filter highlights 51% of all cases which are the long-running cases.

2.4 Visual Bottleneck

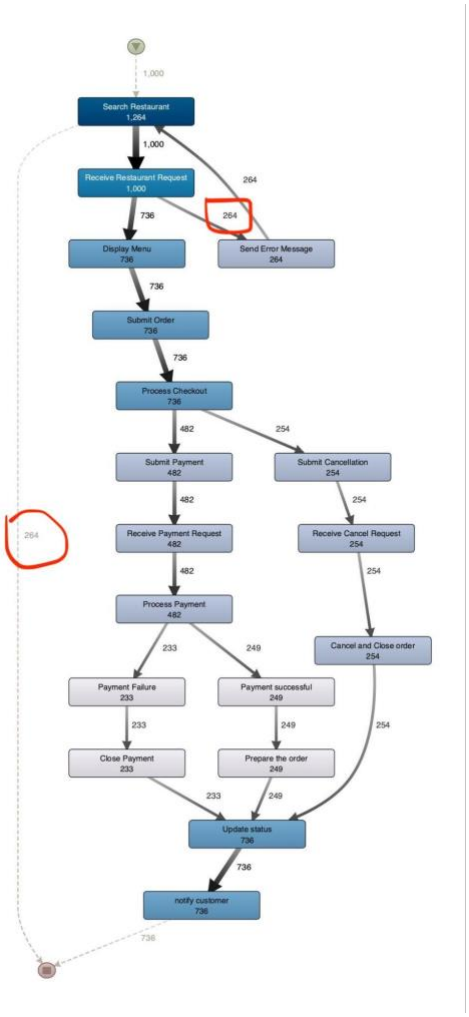


Switch to **Performance** to display **Mean Duration**, this visual process map has been highlighted to show where the most time is being spent. Here are the key points:

- **Search Restaurant** takes an average of 85.3 minutes.
- **Prepare the Order** takes 2.2 hours.
- **Cancel and Close Order** takes an average of 2.8 hours.
- Payment processing indicates that the payment process is another area where delays occur:
    - Submit Payment takes 61.8 minus
    - Process Payment takes 62.4 minutes
    - Payment Failure takes 47.1 minutes
- Other steps take between 35-41 minutes on average, suggesting that these steps are more effective than delayed above.

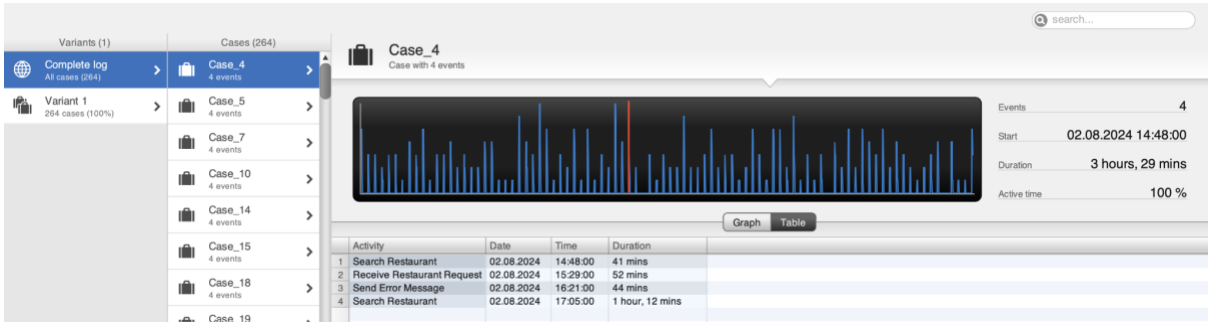
The delayed activity has too many tasks, the whole process will be stuck and need to wait until the final sequence is completed.

2.5 Compliance Check

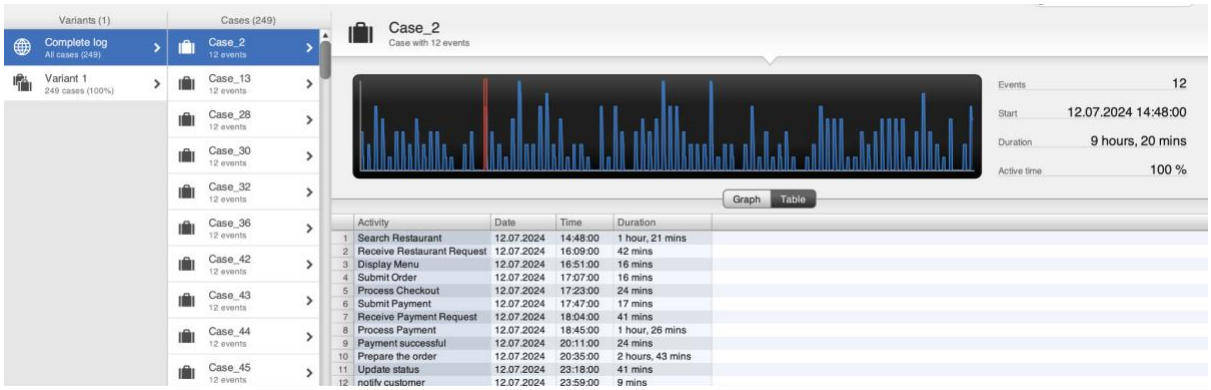


Switching back to the original (full) data and checking the absolute frequency, case frequency, max. repetitions, and case coverage, the 264 cases show how many times the process encountered an error after trying to process a restaurant request. This represents a 26.4% error rate for the 1000 cases that processed the restaurant request, and this rate fits its case coverage.

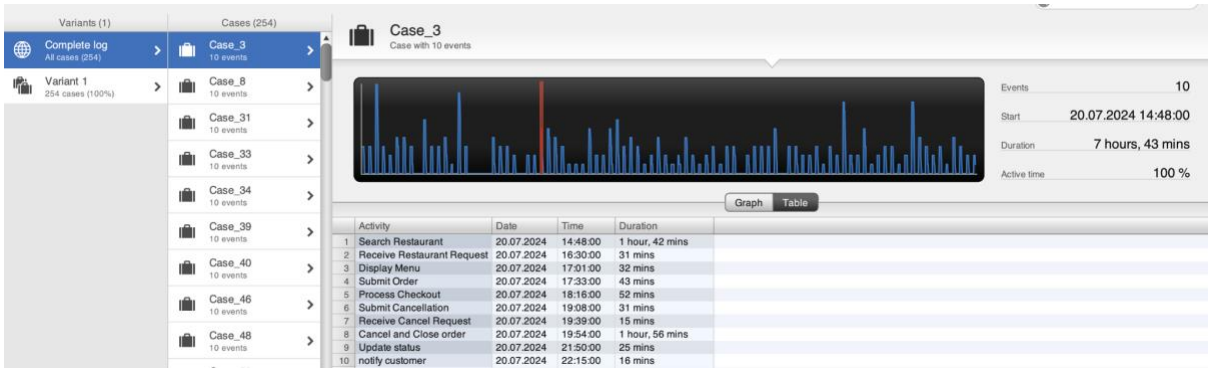
2.5.1 Filter All Cases Related to Top3 Low Latency



The Filter-Follower preserves only all cases in the process that exactly follow the specific path "Search Restaurant -> Receive Restaurant Request -> Send Error Message".



The Filter-Follower preserves only all cases in the process that exactly follow the specific path "Process Payment -> Payment Successful-> Prepare the order".



The Filter-Follower preserves only all cases in the process that exactly follow the specific path "Receive Cancel Request -> Cancel and Close order".

We can analyse the causes of issues and available solutions by obtaining these cases.

3. Recommendations

3.1 Search Restaurant

This significantly long duration for searching a restaurant (85.2 minutes) suggests this is a bottleneck. Investigating whether this is due to user-side delays, such as indecision, or system-side inefficient, such as after receiving the restaurant search request, the Response Time

exceeds 5 seconds, triggering an error. These could help reduce overall process duration and trigger error probability.

Solution:

- **System Side:**
  - **Improve system response speed**. Optimize database queries, such as using caching techniques to speed up responses to search results. Ensure efficient indexing and query of restaurant data, reducing server processing and return of search results.
  - **Shorten response time**: Streamline the back-end service processing logic, reduce the time to query and filter restaurants, and aim to reduce response time to less than 5 seconds to avoid triggering errors.
- **Customer Side:**
  - **Introduce an automated recommendation system**. For example, provide quick access to popular restaurants and provide personalized recommendations based on user history.

3.2 Prepare the order

This step is quite long at 2.2 hours. It could be due to delays in the kitchen or logistical issues.

Solution:

- **Improving order preparation workflows** or tracking delays in real-time might help reduce this time. And introduce load balancing, if the system often slows down during peak hours.

3.3 **Cancel and Close order**

This step is the longest at 2.8 hours. There are two potentially reasons, 1)Third-party payment systems, processing cancellations might rely on the third-party payment systems, which might have their own response times and delays; 2)Database delays, the system needs to retrieve large amounts of data related to the order, the database might be slow in processing the cancel request.

Solution:

- **Caching payment status**: Payment status data caches recent payment status information in the system, reducing the number of frequent communications with third-party payment systems, thereby reducing latency.
- **Payment system optimization**: When working with third-party payment systems, optimize the interface between them and use multithreading technology to speed up communication with payment systems.
- **Database query optimization and using preprocessing methods**. Optimise index techniques, such as ensuring that the order ID and customer ID are well indexed, and at the same time, the required order information can be cached in a faster storage system in advance when the order is cancelled.

Conclusion

This report has fulfilled our original goal of developing a comprehensive tourism ecosystem by completing six main tasks using the knowledge and skills acquired in the lecture. The business process design and analysis are closely linked to microservice design and implementation, as well as service process analysis. We have faced several challenges, such as performance analysis, which goes from the initial service definition to the compiled implementation, which requires analysis and identification of bottlenecks from the user interaction layer to the technical implementation layer, and ultimately requires a solution that integrates these levels to achieve service optimization. In the future, in order to continuously optimise and ensure the user experience, we also need some mechanisms to undertake functions such as A/B testing and incremental iteration.