Task 2:

(b) *CloudTables-Service*: a web interface for running on desktop or tablet computer for restaurant service staffs to help them providing services to the restaurant customers.

Task 2:

· Security and Privacy:

- **Data Encryption:** All communication between the **CloudTables – Service** subsystem, as well as the cloud server will be encrypted. Which is essential within such a system dealing with such personal customer information, such as booking details for tables, food orders, and dynamic environments where multiple restaurants and users are being involved, e.g. phone numbers, names, allergies, timings, card details. The way in which this will be secured by making use of **SSL/TLS** encryption assures all data transferred between the service staff's devices (tablets or desktop interfaces) as well as cloud servers is secured from any potential interception or potential tampering that may have taken place. Especially due to service staff dealing with data, (e.g., customer names, booking times, allergens, birthdates).

- **Why SSL/TLS:** SSL/TLS is commonly used amongst many networks, and ensures the security of communication through networks, this is highly essential within a **multi-tenant** environment described in the case study. The system must protect data as data moves across networks, via many different restaurants and users, whilst also assuring the privacy regulations are being applied, especially if customer data is shared across different countries, or even globally, where **GDPR** or **CCPA** requirements may come into play.

- **Authentication and Authorisation: OAuth2** will be implemented to handle user authentication for service staff (waiters, food runners, bar staff, etc). OAuth2 is chosen as it is a highly secure and widely used standard that can handle the access of data amongst several users at once, by depending on the user's roles. For example, service staff will have the ability to access bookings, and to see table numbers as well as tables available, as well as the ability to place orders. However, a manager would have access to sales, table layouts, and void functions, which would be only accessible for managers.

- **Why OAuth2?** OAuth2 has high levels of security, as well as at the same time supporting multi-device access, as required of service staff making use of any tablets/desktops. This system is fundamentally cloud based, and serves several restaurants (multi- tenancy), therefore having a secure/scalable authentication system is essential. Especially with **CloudTables**, OAuth2 makes use of easy token-based authentication, preventing session hijacking or any attacks in cloud environments.

- **Data Privacy:** As the system deals with bookings from customers, as well as personal details. **Role-Based Access Control (RBAC)** is essential. This allows service staff to have access that is only needed for them to take out their job role correctly, e.g booking name, table number, amount of people. Service staff 's data is logged, which enables management to check on who, and when staff had access to customer related data, therefore preventing unauthorised access.

- **Why RBAC?** : This is a cloud based system serving several restaurants, which assists restaurant management in assigning access to staff based on their given roles, whilst also simultaneously maintaining a high level of efficiency.

2. **Performance**:

- **Real-Time Data Handling:** The system will use **WebSockets**, this assures that table availability across restaurants are pushed and updated based on real time changes, with minimal delay of a few seconds. This will prevent staff from making double reservations, which will cause the restaurant to change the booking for the customer, and ensure waiting staff have access to accurate real time information. This technology is fundamental for the functionality of the system as the **dynamic update of information** is highlighted in the case study.

- **Response Time:** The system runs on a **2-3 second response time** for retrieving booking information and updating table occupancy. This is ensured by making use of **data caching** for frequently accessed information (such as table availability). By making use of this system, it optimises the efficiency of restaurant operations during peak service hours, as this reduces the need for repeated database queries, such as lunch/dinner times, where quick access is essential for restaurants.

- **Load Handling**: The **cloud-based architecture** allows for dynamic scaling during peak times, using **load balancing** in order to distribute incoming traffic across multiple servers. **Autoscaling** will instantly assign more resources as traffic increases , ensuring user performance is as functional as possible during peak times, such during holidays, weekends, bank holidays. Which is also highlighted in the case study where scalability is essential across multiple restaurants.

- **3. Reliability:**

- **Fault Tolerance:** The system will make use of a highly reliable **failover mechanism** in order to assure that if a server becomes unavailable, an alternative server will instantaneously take over. This guarantees that service staff can continue to fulfil their roles adequately regardless of any interruption. **Real-time** updates between devices, in order to ensure that customer service is not affected due to technical issues, which correlates directly with what's mentioned in the case study regarding reliability concerns.

- **Error Handling:** The system handles errors in a way in which providing staff clear and concise feedback based on user input (e.g., "Booking time unavailable. Please try again".) Whilst also logging errors for the restaurant's technical teams to review and resolve. **Retry mechanisms** will assist the user in resolving temporary issues such as network failures, in order to prevent minor issues from disrupting service.

- **Data Consistency:** Using **ACID-compliant transactions,** the system will assure updates are simultaneously updated and reflected across all devices, across all restaurants. This will ensure that possibilities of different devices showing a table to be available while another shows it as occupied. This would result in conflicts between bookings, and result in an effect in customer service, which is a concern specifically mentioned in the case study where there is a need for a seamless integration within the system.

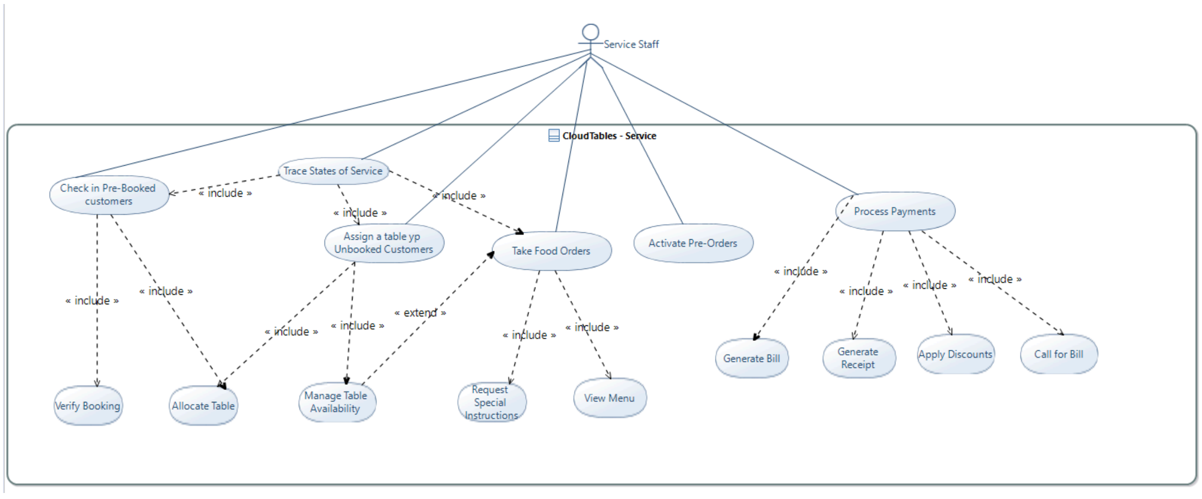
- 4. Scalability :

- **Vertical Scalability**: The system will support the addition of new features. For example, an expanded table management tool, which is integrated with external booking systems. A real time example would be booking systems such as Dojo, which has direct access to table layouts, and based on customer bookings allocates customers a table, and at the same time provides customers with alternative times if table/time isn't available. Therefore, by upgrading the server capacity when necessary, the system will have the ability to handle growing restaurant chains without re-architecting the entirety of the infrastructure, aligning with the multi-tenant SaaS structure described in the case study.

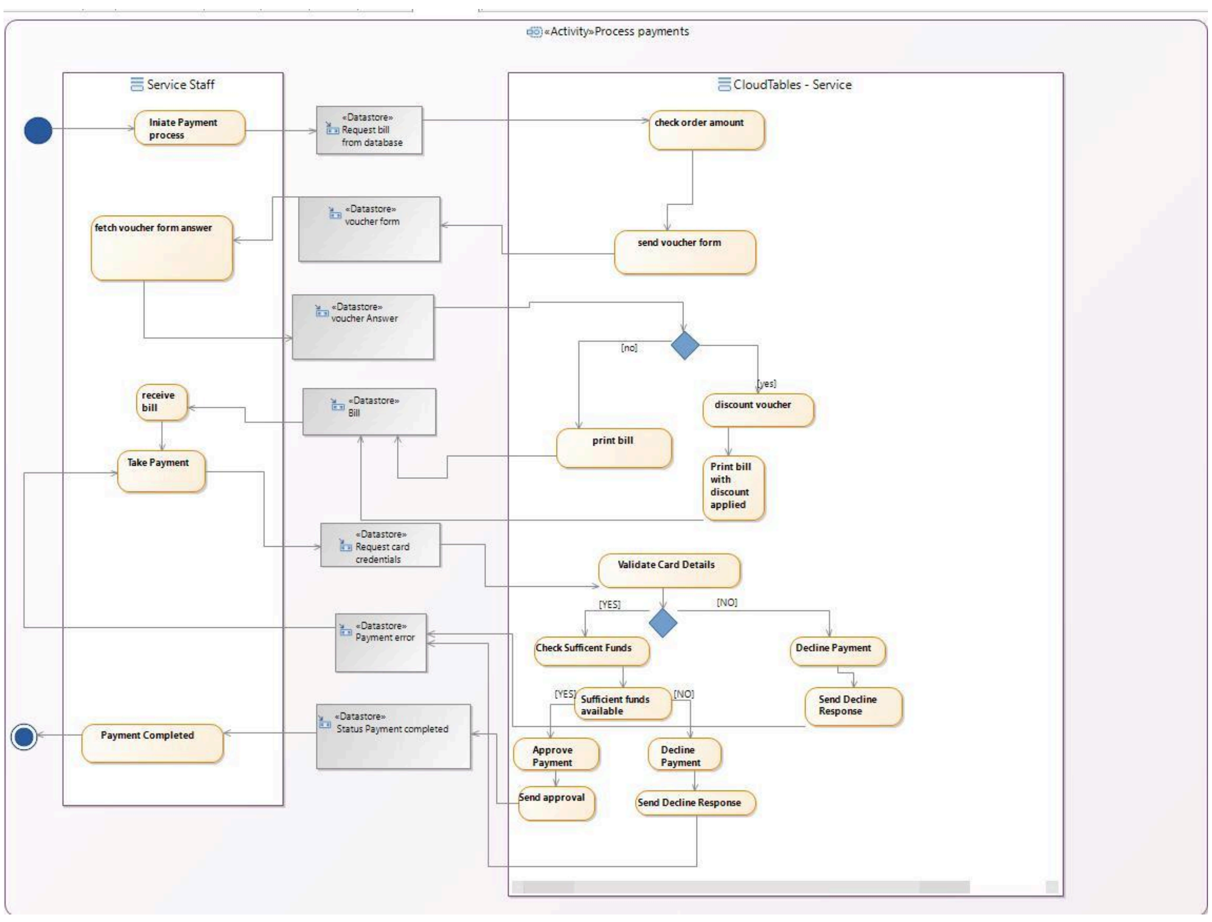
- **Horizontal Scalability**: By making use of **microservices architecture,** the system has the ability to scale individual services (such as booking management or table updates) Which then allows the system to handle an increase of the number of restaurants, as well as the increase of restaurant staff. Therefore, resulting in the system distributing traffic across several more servers, ensuring that performance remains consistent regardless of the load received.

- **Geographical Distribution**: It will deploy cloud resources in multiple geographical locations to reduce latency for restaurants from operating in different regions. When restaurants are in several countries, this guarantees that the service staff will always have access to booking information, which is needed for the cross-restaurant aspects of the case study

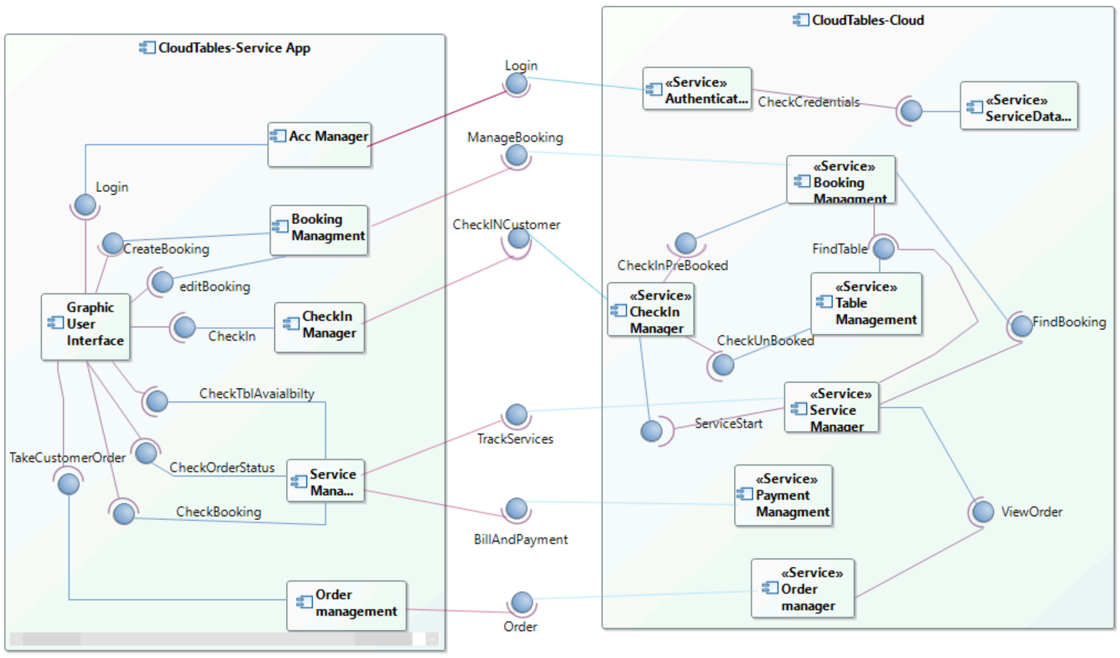
- 3A:


Service Staff
CloudTables - Service
Check in Pre-Booked customers
Trace States of Service
« include »
Assign a table yp Unbooked Customers
Take Food Orders
Activate Pre-Orders
Process Payments
« extend »
Verify Booking
Allocate Table
Manage Table Availability
Request Special Instructions
View Menu
Generate Bill
Generate Receipt
Apply Discounts
Call for Bill

3b:


«Activity»Process payments
Service Staff
CloudTables - Service
Iniate Payment process
«Datastore» Request bill from database
check order amount
«Datastore» voucher form
fetch voucher form answer
send voucher form
«Datastore» voucher Answer
[no]
[yes]
discount voucher
receive bill
«Datastore» Bill
print bill
Take Payment
Print bill with discount applied
«Datastore» Request card credentials
Validate Card Details
[YES]
[NO]
«Datastore» Payment error
Check Sufficent Funds
Decline Payment
[YES]
Sufficient funds available
[NO]
Send Decline Response
«Datastore» Status Payment completed
Payment Completed
Approve Payment
Decline Payment
Send approval
Send Decline Response

4A:



Components Table:

Component Name	**Description**	**Stereo Type**	**Required Interfaces**	**Provided Interfaces**
Booking Management	Handling of processes: creating, modifying, and viewing bookings.	Microservice	CheckPermission, TableManagement	CreateBooking(), EditBooking(), FindBooking()
Table Management	Management of: tables, availability, allocation, and status updates.	Microservice	ServiceManager	FindTable(), SetOccupied(), GetTableStatus()

Service Management	Management of: restaurant services, service requests, status change, and customer interaction with service staff.	Microservice	BookingManagement, TableManagement	StartService(), TrackService(), ServiceStatus()
Order Management	Handling of: food orders, adding items, order status updates, and order completion.	Microservice	ServiceManager, PaymentManagement	AddOrder(), ViewOrder(), CompleteOrder()
Payment Management	Processing of: customer payments for services and orders, including billing systems.	Microservice	Authentication, OrderManagement	BillAndPayment(), TakePayment()
CheckIn Manager	Management of the check-in process for booked and unbooked customers.	Microservice	BookingManagement, TableManagement	CheckInBooked(), CheckInUnBooked()
Account Manager	Management of restaurant accounts, login and registration	Microservice	Authentication	Login(), CreateAccount(), ManageAccount()

	functionality.			
ServiceDataStore	Stores data related to services, bookings, and customer interactions for updating.	Microservice	None	getServiceData(), updateServiceData()

<u>Operations Table:</u>

Name	**Provider**	**Operation**	**Signature**	**Function Description**
Login	Account Manager	operatorLogin()	operatorLogin(op_id: String, password: String, ip_Address: String): accessCode: String	Checks credentials are acceptable, returns an access code if successful ,and an error if unacceptable.

Create Booking	Booking Management	createBooking()	createBooking(customerID: String, tableID: String, time: String): Boolean	Creates a booking with the requested details and returns true if successful.
Edit Booking	Booking Management	editBooking()	editBooking(bookingID: String, newDetails: String): Boolean	Modifying existing bookings with new details and returns true if successful.
Find Table	Table Management	findTable()	findTable(criteria: String): List	Finds availability of tables based on specific customer requests and returns available tables.
Start Service	Service Management	startService()	startService(serviceID: String, tableID: String): Boolean	Begins service for table and service ID and returns true if successful.

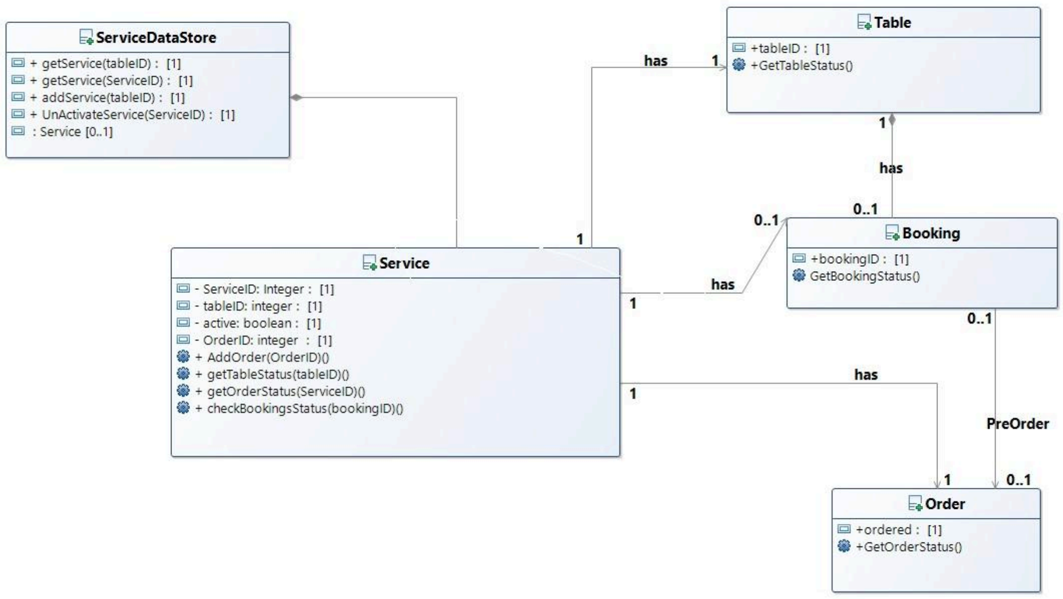
Take Payment	Payment Management	takePayment()	takePayment(paymentDetails: String): Boolean	Process a payment and return true if successful, or false.
Check In Customer	CheckIn Manager	checkInCustomer()	checkInCustomer(bookingID: String): Boolean	Manage check in process for customers with a valid booking ID.
Add Order	Order Management	addOrder()	addOrder(orderID: String, items: List): Boolean	Adds items to order and returns true if successful.
Complete Order	Order Management	completeOrder()	completeOrder(orderID: String): Boolean	Marks order as completed then returns true if successful.
Manage Account	Account Manager	manageAccount()	manageAccount(accountID: String, changes: String): Boolean	Manages and updates account details, returning true if successful.

Database Table:

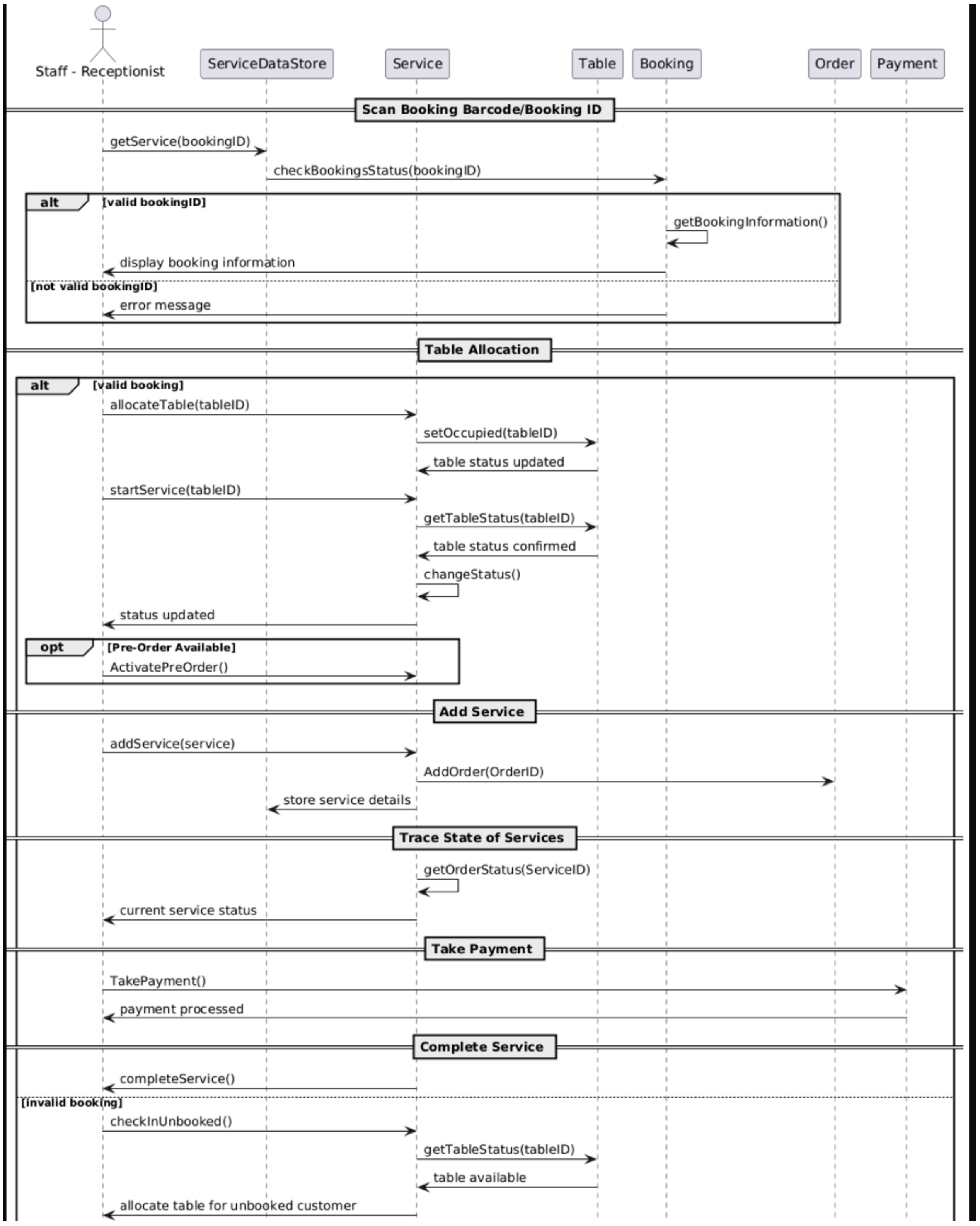
Database Name	Description	Type	Design (tables)	Table Fields
Booking Database	stores details on bookings, including table, customer, and time information.	Relational database	- Bookings	bookingID, customerID, tableID, time, status
Table Database	Stores information on restaurant tables, e.g, as availability and status.	Relational database	- Tables	tableID, location, status
Service Database	Stores service data, including customer service requests and their status.	Relational database	- Services	serviceID, tableID, customerID, status
Order Database	Store order details, items ordered, status, and associated table or booking.	Relational database	- Orders	orderID, items, tableID, status, totalAmount
Payment Database	Store payment transactions, customer payment details and order association.	Relational database	- Payments	paymentID, orderID, amount, status, paymentMethod
Account Database	Store account details for operators, customers, and managers, including login information.	Relational database	- Accounts	accountID, username, password, role, status

Event Database	Stores information on promotional events and activities held by the restaurant.	Relational database	- Events	eventID, eventName, eventDate, description
Analytics Database	Stores statistical data about restaurant performance, customer preferences, and service efficiency.	Relational database	- Analytics	analyticsID, metric, value, date

5A:



5B:



Staff - Receptionist
ServiceDataStore
Service
Table
Booking
Order
Payment
Scan Booking Barcode/Booking ID
getService(bookingID)
checkBookingsStatus(bookingID)
alt
[valid bookingID]
getBookingInformation()
display booking information
[not valid bookingID]
error message
Table Allocation
alt
[valid booking]
allocateTable(tableID)
setOccupied(tableID)
table status updated
startService(tableID)
getTableStatus(tableID)
table status confirmed
changeStatus()
status updated
opt
[Pre-Order Available]
ActivatePreOrder()
Add Service
addService(service)
AddOrder(OrderID)
store service details
Trace State of Services
getOrderStatus(ServiceID)
current service status
Take Payment
TakePayment()
payment processed
Complete Service
completeService()
[invalid booking]
checkInUnbooked()
getTableStatus(tableID)
table available
allocate table for unbooked customer