Introduction

In modern software development, architectural design is essential to guaranteeing the efficiency, scalability, and adaptability of a system. A strong and scalable architecture is required for the CloudTables project, which is a multi-tenant Software-as-a-Service (SaaS) solution for managing and booking restaurant tables. Several subsystems, including CloudTables-Customer, CloudTables-Service, CloudTables-Manager, and CloudTables-Operation, are integrated into the system to meet the needs of different users.

One of the most important of them is the CloudTables-Service subsystem, which helps restaurant service employees by handling payments, meal orders, table assignments, and client check-ins. This subsystem uses a microservices design to guarantee smooth operations within the restaurant. In order to meet the demands of dynamic restaurant environments, this architectural approach permits the autonomous creation, deployment, and scaling of services.

Task 2: Analysis and Specification of Software Quality Requirements

Subsystem: CloudTables-Service (Web Interface for Restaurant Service Staff)

1. Security and Privacy Protection

Since customer and order data are handled by the CloudTables-Service subsystem, security is essential to avoiding unauthorised access and protecting data privacy.

- **Authentication**:
  Use of strong authentication methods, such OAuth 2.0 “provides consented access and restricts actions of what the client app can perform on resources on behalf of the user, without ever sharing the user's credentials.”[1], to guarantee safe access to the subsystem. Every user needs to be verified according to their role (restaurant management, operator, or service personnel).
- **Data Encryption**:
  Encrypt sensitive information, such as payment information and customer orders, using TLS “a widely adopted security protocol designed to facilitate privacy and data security for communications over the Internet.”[2] is used for data in transit and AES which is “a symmetric block cipher chosen by the U.S. government to protect

classified information.” [3], used for data at rest. By doing this, any possible breaches during storage or communication are prevented.

- **Access Control**:
  Use role-based access control to limit the service employees' access to only the information necessary for their position (e.g., food orders, table reservations). Extended permissions can be granted to managers and owners.
- **Data Anonymization**:
  Data from order histories and customer reviews should be anonymised before being shared for study or public access to ensure the compliance with data privacy laws like the GDPR which “is a legal framework that sets guidelines for the collection and processing of personal information from individuals who live in and outside of the European Union (EU).”[4].

2. Performance

For service employees to process customer orders and requests quickly, particularly during peak hours, the CloudTables-Service subsystem needs to function reliably.

- **Real-Time Updates**:
  The subsystem must guarantee that meal order statuses and table availability are updated instantly, with a response time of less than two seconds for every interaction.
- **High Throughput**:
  During peak hours, the service should be able to manage up to 500 concurrent requests (such as simultaneous bookings and order submissions) without experiencing any performance issues.
- **Database Optimization**:
  To speed up query execution times for obtaining restaurant layouts, menus, and availability information, use indexing and caching techniques .
- **Resource Efficiency**: Given that it may be used on a variety of devices, including tablets and desktop PCs, the web interface should be designed for low resource utilisation.
- **Concurrent Processing**: Allow many service professionals to use the system simultaneously without any delays by enabling the subsystem to handle multiple queries concurrently without latency.

3. Reliability

The CloudTables-Service subsystem must be reliable because any outage or malfunction could have a direct effect on how well the restaurant serves its customers.

- **Fault Tolerance**:
  To guarantee continuous operations in the event of server failures, integrate failover techniques, which are methods that switch operations to a backup system when a primary system fails, for cloud-based services.
- **Data Integrity**:
  To guarantee data consistency in the event of unforeseen delays, keep transaction logs for crucial processes like reservations, food orders, and payments.
- **Error Handling**:
  When operations go wrong, give service employees concise, actionable error notifications (such as "Booking unavailable" or "Payment processing failed") to help with recovery.
- **Availability**: In order to guarantee reliable access for service personnel, particularly during business hours, the system should attain a minimum uptime of 99.9%.
- **Data Consistency**: Make sure that order data is synchronised in real time across all devices to prevent inconsistencies, such as staff members viewing differing order status information.

4. Scalability

The CloudTables-Service subsystem needs to scale effectively to meet growth in demand without compromising performance as the system grows to accommodate more restaurants and a bigger user base.

- **Horizontal Scaling**:
  Create a subsystem that allows for horizontal scaling by dynamically adding extra service instances to meet demand.
- **Load Balancing**:
  Use load balancers to avoid bottlenecks by dividing incoming requests equally among several service instances.
- **Support for Future Expansion**:
  Potential modules (such as delivery services or reward programs) and new restaurants must be able to be seamlessly integrated with the design.
- **Flexible Load Management**: Distribute requests among several servers using dynamic load balancing to guarantee peak performance at busy times (e.g., peak dining hours).

These quality requirements help in guaranteeing that the CloudTables-Service subsystem satisfies the exacting standards required to provide clients with prompt, effective, and secure service. By taking care of these features, the subsystem will serve the CloudTables system's overarching objectives and help the restaurant environment run smoothly.

Task 3: Specification and Modelling Software Functional Requirements

Part A: Use Case Diagram

Purpose:

The CloudTables-Service subsystem's functional scope is summarised in the Use Case Diagram. It shows the main features (use cases) that the subsystem provides and indicates the interactions between the subsystem and its users (actors). The subsystem's responsibilities and expected behaviours are made clearer by this diagram.

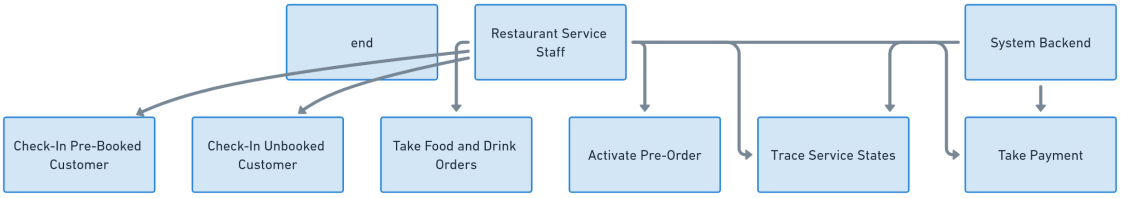
Actors:

1. **Restaurant Service Staff** (Primary Actor):
   In charge of overseeing all client contacts, including payments, table assignments, food orders, and check-ins.
2. **System Backend** (Secondary Actor):
   Gives the wait staff up-to-date information on food and table availability, along with safe payment processing.

Use Cases:

1. **Check-In Pre-Booked Customer:**
   Verifies customer bookings and assigns tables.
2. **Check-In Unbooked Customer:**
   Finds and assigns tables to walk-in customers.
3. **Take Food and Drink Orders:**
   Records customer orders and forwards them to the kitchen.
4. **Activate Pre-Order:**
   Activates and optionally modifies pre-orders upon customer arrival.
5. **Trace Service States:**
   Monitors table status and service progress (e.g., order fulfilment).
6. **Take Payment:**
   Generates bills and processes payments.

Diagram: Use Case Diagram



Explanation of the Use Case Diagram:

System Boundary:

- The subsystem's boundaries are shown by the rectangle with the name CloudTables-Service Subsystem.
- It makes sure all interactions within the rectangle are specific to this subsystem by isolating all of the functionalities that it handles.

Actors:

- **Primary Actor - Restaurant Service Staff:**
  - The primary user of the subsystem is the service personnel. They are in charge of handling important duties like processing payments, organising orders, and checking in guests.
  - Shown on the left, outside the system border, as a stick figure with the label "Restaurant Service Staff."
- **Secondary Actor - System Backend:**
  - The subsystem is supported by the backend, which manages secure payment processing, regulates table and order statuses, and provides real-time data.
  - Represented by a second stick figure on the right that is labelled System Backend.

Connections:

- Solid lines show the interactions between actors and use cases:
  - **Restaurant Service Staff → All Use Cases:** Shows that the service staff directly interacts with every functionality in the subsystem.
  - **System Backend → Trace Service States & Take Payment:** Shows that the backend provides data and processing support for these functionalities.
- The relationships highlight the cooperation and dependence needed to complete tasks.

Part B: Activity Diagram

Use Case Modelled: Check-In Pre-Booked Customer

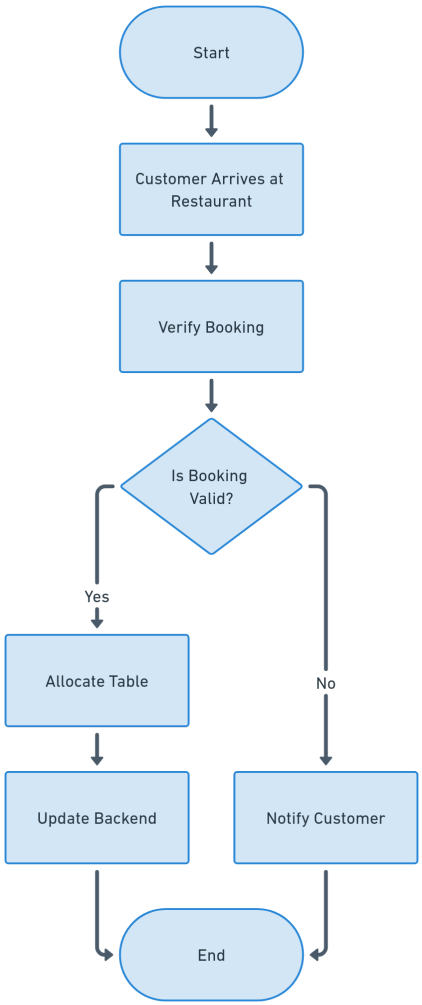
Purpose:

The "Check-In Pre-Booked Customer" use case's step-by-step workflow is shown in detail in the Activity Diagram. It documents the series of choices, actions, and results, showing how the subsystem communicates with users and backend elements to carry out this specific job.

Workflow Steps:

1. **Start:**
   A customer arrives at the restaurant with a booking.
2. **Verify Booking:**
   Service staff searches for the customer's booking in the system and verifies its valid.
3. **Decision Point:**
   - If the booking is valid, customers proceed to allocate a table.
   - If the booking is invalid, notify the customer and terminate the process.
4. **Allocate Table:**
   Assign the reserved table to the customer and update the table's status to "occupied."
5. **Update Backend:**
   Mark the booking as "fulfilled" in the system to ensure it reflects the completed check-in.
6. **End:**
   The customer is guided to the allocated table, completing the process.

Diagram: Activity Diagram



Explanation of the Activity Diagram:

- **Start Node:**
  - This node, which is represented by the blue oval with the label Start, shows the process starts when a customer walks into the restaurant.
- **First Action - Customer Arrival:**
  - A rectangle with the description "Customer Arrives at Restaurant" designates the first stage.
  - This action represents the workflow's initial triggering event.
- **Verify Booking:**
  - The support staff uses the subsystem to verify the customer's booking details in the system in this stage, which is represented by another rectangle with the caption Verify Booking.
- **Decision Point - Valid Booking?:**
  - A diamond-shaped node represents a critical decision: whether the booking is valid.

    - Two possible outcomes are connected to this decision:
        - **Yes (Valid Booking):**
            - Proceeds to the next action: Allocate Table.
        - **No (Invalid Booking):**
            - Causes the activity to occur The employee notifies the client that the reservation is invalid and ends the transaction.
- **Allocate Table:**
    - This step gives the customer a reserved table, which is shown as a rectangle with the name Allocate Table.
    - The table's status is updated to "occupied" in the system.
- **Update Backend:**
    - This step makes sure the backend displays the new booking status as "fulfilled." It is represented by another rectangle with the label Update Backend.
- **Customer Seated:**
    - After completing the check-in procedure, the customer is seated at the designated table.
    - Represented by a rectangle labeled Customer Seated.
- **End Node:**
    - Represented by an oval labeled End, this node shows that workflow has been completed successfully.
- **Workflow Flow:**
    - All steps are connected by arrows, guaranteeing an accurate representation of the activities' progression from beginning to end.
    - The flow changes at the point of choosing, showing different possible outcomes (booking that is legitimate or not).
- **Clarity and Realism:**
    - The system's handling of consumers who have made reservations is depicted realistically in the diagram.
    - It highlights the durability of the system by handling errors via the "Invalid Booking" path.

Conclusion:

The CloudTables-Service subsystem's Use Case Diagram and Activity Diagram clearly show its features and procedures, guaranteeing compliance with the functional requirements. The Use Case Diagram emphasises simplicity and scope while capturing the essential elements of the subsystem, including interactions between the system backend, use cases, and restaurant service personnel. The "Check-In Pre-Booked Customer" use case's thorough process is the main emphasis of the Activity Diagram, which also includes decision points, error handling, and sequential operations to guarantee robustness and usability. When combined, these diagrams improve stakeholder communication, remove uncertainty during implementation, and provide a framework for creating an accurate and scalable subsystem that satisfies project objectives.

Task 4: Software Architectural Design

Overview:

A key component of the CloudTables system, the CloudTables-Service subsystem uses modular microservices to help with efficient restaurant operations. The subsystem guarantees smooth operation even in the face of high demand by separating functions such as payment processing, food ordering, and booking administration. To ensure reliability, each service can be independently launched and communicated using REST APIs. All services are supported by a common database that serves as a centralised location for up to date information about tables, order, and payments. By adhering to industry standards, this design facilitates safe interactions, effective scaling, and fault tolerance.

Subsystem Microservices:

Booking Service:

- **Responsibilities:**
  - Handles the table reservations, updates the table statuses(e.g., “booked,” “occupied”), and manages any cancellations
- **APIs:**
  - bookTable(customerID, tableID, timeSlot): This reserves a table for the specific time slot.
  - cancelBooking(bookingID): Cancels the existing reservation.
- **Use Case:** Supports “Check-In Pre-Booked Customer” and “Check-In Unbooked Customer” workflows.

Order Service:

- **Responsibilities:**
  - Manages food and drink orders and sends them to the kitchen.
- **APIs:**
  - placeOrder(orderDetails): Records the customers' food order.
  - updateOrder(orderID, changes): Updates an existing order.
- **Use Case:** Supports “Take Food and Drink Orders.”

Payment Service:

- **Responsibilities:**
  - Process payments, generate bills, and ensures that the transactions are secure.
- **APIs:**
  - generateBill(orderID): Creates a bill for the given order.

    - processPayment(orderID, paymentDetails): Completes the payments process.
- **Use Case:** Supports “Take Payment”.

State Tracker Service:

- **Responsibilities:**
    - Tracks the status of the tables, orders, and booking while they happen in real time.
- **APIs:**
    - updateStatus(entityType, entityID, newState): Updates the start of the specified entry.
    - getStatus(entityType, entityID): Brings back the current state of an entity
- **Use Case:** Supports “Trace Service States.”

Advantages of the Architectural Design for CloudTables-Service Subsystem

The CloudTables-Service subsystems microservices-based architectural design offers a number of benefits that are consistent with contemporary software engineering concepts. These consist of:

Scalability

- **Horizontal Scaling:**

  Since each microservice (such as Booking, Order, and Payment) may scale independently in response to demand, the subsystem can effectively manage a range of workloads.

- **Targeted Resource Allocation:**

  It is possible to prioritise growing high-traffic services, like the Order Service during peak hours, while maintaining less active services, like the State Tracker.

Modularity

- **Separation of Concerns:**

  Every microservice concentrates on specific tasks, such as collecting the payments or handling reservations. This improves the subsystems understanding, development, and upkeep.

- **Independent Updates:**

  There is less chance of the entire subsystem malfunctioning when one microservice is updated or changed (for example, by introducing new payment methods) without requiring changes to other services.

Fault Tolerance

- **Isolated Failures:**

  The subsystem as a whole can continue to operate with little interruption even if one microservice(such as the Notification Service) fails.

- **Retry Mechanisms:**

  Failed requests can be retired without causing the system to become inoperable thanks to asynchronous communication between the services.

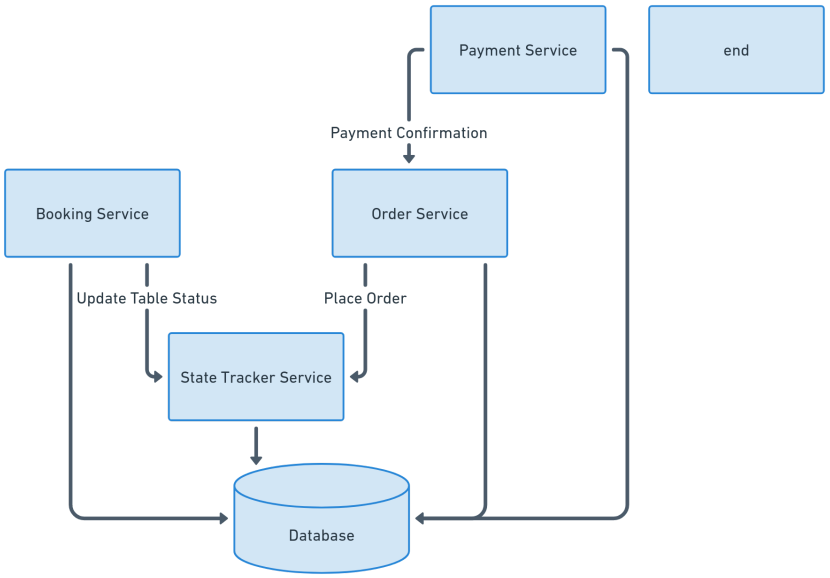
Ease of Maintenance

Independent Development:

- Teams can operate on separate microservices at the same time without affecting one another, which speeds up the development and lowers the complexity.

Simplified Disbudding:

- Because each microservice has its own log and a well defined scope, problems are simpler to find and fix.



The relationships and interactions between its microservices are shown in the CloudTables-Service Subsystem Architecture Diagram. While each service (shown as a rectangle) functions separately, RESTful APIs are used to communicate with other services. For example, the Payment Service interacts with the Order Service to verify order details prior to processing payments, and the Booking Service communicates with the State Tracker Service to update table statuses. A common database, represented by a cylinder, is accessed by all services, guaranteeing real -time data synchronisation. While the distinct division of duties reduces interdependencies and improves dependability and maintainability. This modular structure makes development and scaling easier.

This thorough design makes sure that the subsystem is secure, scalable and resilient enough to manage actual restaurant operations effectively.

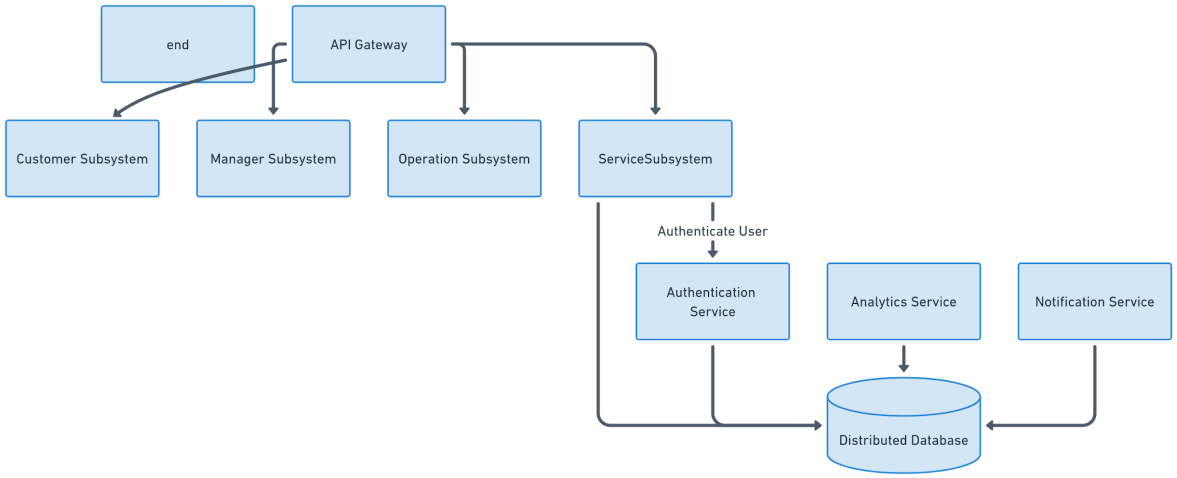
Overview

CloudTables-Customer, CloudTables-Service, CloudTables-Manager, and CloudTables-Operation are among the subsystems that are integrated into the CloudTables system. Every subsystem functions autonomously while communicating with one another via specified APIs and common services. The management of service staff operations, effective communication with managers and customers, and system-wide operations are the main objectives for the CloudTables-Service subsystem.

Key Interactions of CloudTables-Service Subsystem

1. With CloudTables-Customer Subsystem:
    - Receives the booking requests from customers and updates their statuses.
    - Sends the order and booking updates to the customers in real time.
2. With CloudTables-Manager Subsystem:
    - Shares booking and order data for the managerial reporting.
    - Gets menu updates so that ordering features are in sync.
3. With CloudTables-Operation Subsystem:
    - Gives information for advertising campaigns or system-wide events.
4. With Shared Services:
    - Authentication Service: Safeguards transactions and verifies employee credentials.
    - Analytics Service: Transmits information to get understanding about restaurant operations.
    - Notification Service: Triggers alerts for booking and orders.

Diagram: Whole System Architecture (Service Subsystem Focus)



Advantages of the Whole System Architecture

1. Integration: Having a well-defined APIs guarantees a smooth subsystem communication.
2. Modularity: Dependencies are reduced because each subsystem functions independently.
3. Scalability: Different needs are supported by independent subsystem scaling.
4. Security: Shared services such as Authentication ensure secure access and operations
5. Flexibility: Easy to add new subsystems or services as the system evolves.

Conclusion:

In order to facilitate effective service staff operations while interfacing with other subsystems and shared services, the CloudTables-Service subsystem easily integrates into the entire system. Scalibily, modularity, and real time operations are prioritised in the architectural design to guarantee that the subsystem and the system as a whole remain resilient and flexible enough to meet changing needs.

Task 5: Software Detailed Detailed Design

Structural Model

Managing reservations for tables is made easier with the booking service. Its internal structure is made up of several classes that cooperate to manage client notifications, booking generation, cancellations, and table availability checks. The characteristics, functions, and connections between these classes are shown in the class diagram.

Classes in the Booking Service

1. BookingManager(Main Class):
    - Responsibilities:
        - Acts as the central point for booking-related operations.
        - Interfaces with the other classes(e.g., Table, Customer) and external services such as NotificationService.
    - Attributes:
        - bookingID: int: Unique identifier for a booking.
        - customerID: int: ID of the customer making the booking.
        - tableID: int: ID of the table being reserved.
        - timeSlot: String: Time slot for the booking.
        - Status: String: Current status of the booking such as ``confirmed" or "cancelled".
    - Methods:
        - createBooking(customerID, tableID, timeSlot): boolean: Creates a new booking.
        - cancelBooking(bookingID): boolean: Cancels a booking which already exists.
        - updateBooking(bookingID, newTimeSlot): boolean: Updates the time slot for a booking
        - getBookingDetails(bookingID): Booking: Brings back the details for a booking.

2. Table(Support Class):
    - Responsibilities:
        - Represents the table being booked and manages the status.
    - Attributes:
        - tableID: int: Unique identifier for the table.
        - capacity: int: Total number of seats which are available at the table .
        - status: String: Current status of the table, for example, “available” or “booked”.
    - Methods:
        - isAvailable(timeSlot): boolean: Checks for given time slots for the table.
        - updateStatus(newStatus): void: Updates the tables status.
3. Customer(Support Class):
    - Responsibilities:
        - Stores the customers information for booking purposes.
    - Attributes:
        - customerID: int: Unique identifier for the customer.
        - name: String: Name of the customer.
        - contactInfo: String: Contact information for the customer.
    - Methods:
        - getContactInfo(): String: Retrieves the customers contact information.
4. NotificationService(External Dependency):
    - Responsibilities:
        - Sends any booking related notifications to the customers.
    - Methods:
        - sendNotification(message): void: Sends a notification with the custom message.

Diagram:


FeedbackManager
-String CurrentcustomerID
-List feedbackList.
+retrieveFeedback()
+submitFeedback()
+analyseFeedback()
+storeFeedback()
uses
associates
manages
uses
FeedbackDatabase
-String databaseConnection
+saveEntry()
+deleteEntry()
+fetchEntries()
Customer
-String customerID
-String email
-String phoneNumber
-String name
+updateProfile()
+submitFeedback()
FeedbackEntry
-int feedbackID
-String comments
-String customerID
-int rating
-Date timestamp
+editFeedback()
+displayFeedback()
AnalysisTool
-List Trends
-String report
+analyseFeedback()
+sendReport()
+generateReport()
ProductFeedback
-int porductRating
-String productName
+calculateProductScore
ServiceFeedback
-int serviceRating
+calculateServiceScore()

2. Behavioral Model

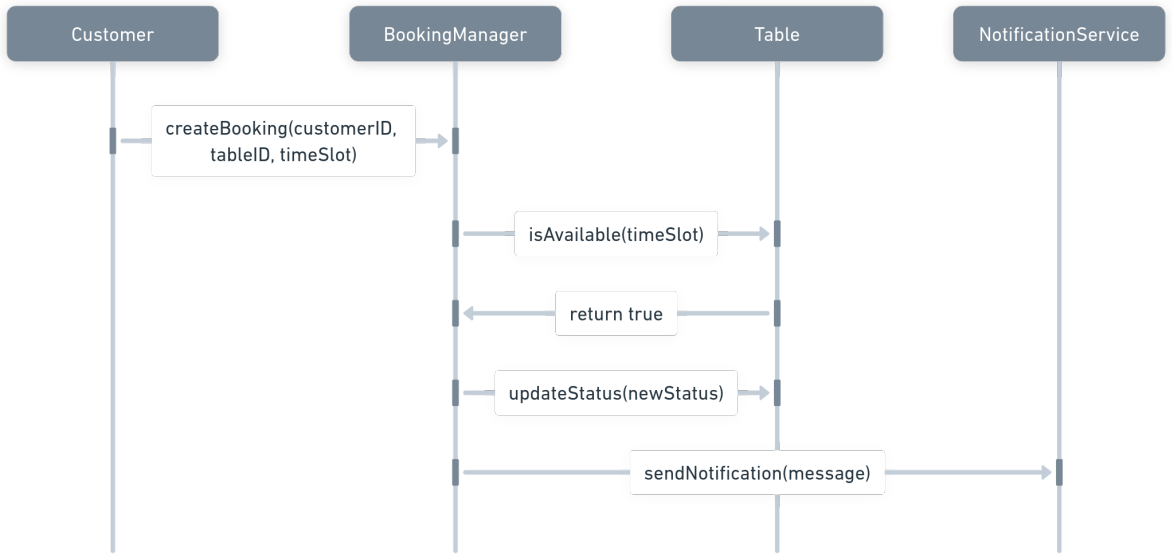
Overview of the Booking Service Behaviour

The way items interact during a normal booking operation is shown in the sequence diagram. It offers a detailed overview of how the system manages the booking making process, from receiving the requests to notifying the user.

Steps in the Booking Sequence

1. Customer Requests Booking:
    - By providing the BookingManager with their information, the table ID, and the preferred time slot, the customer starts the booking process.
2. Verify Table Availability:
    - The Booking Manager uses the isAvailable function of the Table class to determine whether the table is available or not.
3. Create Booking:
    - When a table becomes available, the BookingManager makes a reservation and changes the tables' state to "booked".
4. Update Table Status:
    - The updateStatus function is used by the BookingManager to update the Table status.
5. Send Notification:
    - To notify the customer that the booking has been successful, the BookingManager calls the NotificationService's sendNotification function.

Sequence Diagram:



Advantages:

1. Clarity:
    - The Booking Service's core components and their interconnections are clearly defined by the class diagram
2. Modularity:
    - The system is simpler to comprehend and manage because of the designs division of duties among classes.
3. Dynamic Behaviour:
    - By capturing the interactions in real time, the sequence diagram makes sure engineers can comprehend how the things work.
4. Scalability:
    - The service is made to manage situations with high demand by breaking down essential features into separate parts.

Conclusion

The Booking Services structural and behavioural models offer a thorough comprehension of its layout and operation. The Sequence Diagram shows the dynamic handling of a booking operation by the service, while the Class Diagram outlines the essential elements and their connections. Together, these models guarantee that the CloudTables-Service subsystem is implemented in a reliable, scalable, and maintained manner.

Reflection on Project Management and Collaboration

Project Management

In order to guarantee consistency with the overall system architecture, managing the CloudTables-Service subsystem required planning, designing, and collaborating with the team members. Among my responsibilities were:

- Task Prioritization: Dividing up the requirements for the subsystems into double tasks, including creating microservices (like the Booking, Order, Payment, and State Tracker Services), defining APIs, and incorporating shared services.
- Time management: By planning my work into distinct milestones and completing each component on schedule, I was able to meet all the deadlines.

Collaboration with the Team

Because the system architecture called for the integration of all the subsystems, teamwork was essential for this project. Our final architecture's scope was impacted because we were unable to build the CloudTables-Operation subsystem because our group only had three members. Despite this, we successfully assigned responsibilities to the remaining three subsystems and made sure that all of the features were covered.

Key Learnings and Improvements

This project taught me alot about team dynamics and project management.

1. Strengths:
    - My organizational skills and attention to detail ensured that the service subsystem met the project's requirements.
2. Areas for improvement:
    - Even while I remained focused on my own tasks, I realised that in order to see possible problems earlier, I needed to devote more time to going over other people's work.

Conclusion

Overall, my experience managing and designing the subsystem was both challenging and rewarding. I contributed significantly to the project's success by ensuring the subsystems functioned efficiently and integrated seamlessly with the rest of the system.

Links:

Google Doc repository:

https://docs.google.com/document/d/1H5DtZVsAx7gwFdsem1sPCpGxzrGeOtS5Wv8xLs_v2MQ/edit?usp=sharing

GitHub repository:

https://github.com/mujtaba5555/5047-group-meetings.git

References:

1) What is OAuth 2.0?
   https://auth0.com/intro-to-iam/what-is-oauth-2

2) What is TLS (Transport Layer Security)?
   https://www.cloudflare.com/en-gb/learning/ssl/transport-layer-security-tls/

3) Advanced Encryption Standard (AES)

   https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard#:~:text=The%20Advanced%20Encryption%20Standard%20(AES)%20is%20a%20symmetric%20block%20cipher,world%20to%20encrypt%20sensitive%20data.

4) General Data Protection Regulation (GDPR): Meaning and Rules

   https://www.investopedia.com/terms/g/general-data-protection-regulation-gdpr.asp#:~:text=The%20General%20Data%20Protection%20Regulation%20(GDPR)%20is%20a%20legal%20framework,the%20European%20Union%20(EU).