OXFORD BROOKES UNIVERSITY

Assessment cover

Module No:	COMP5047	Module title:	Applied Software Engineering

Assessment title:	Resit Coursework - Software Engineering of a Modern Computer Application

Due date and time:	9:00am, 14th April, 2025

Estimated total time to be spent on assignment:	84 hours per student

LEARNING OUTCOMES

On successful completion of this assignment, students will be able to achieve the module's following learning outcomes (LOs):
1. Demonstrate an understanding of the role of requirements analysis and specification in software engineering and to be able to use this knowledge to create use case models and functional models of computer applications.
2. Demonstrate an understanding of the relationship between requirements and design and to be able to apply the knowledge to create structural and behavioural models of computer applications.
3. Critically evaluate and utilise design paradigms of object-oriented analysis and design, component-based design, and service-oriented design.
4. Use software modelling language such as UML and modelling tools in the context of model-driven software engineering.
5. Work in a group to apply the knowledge and skills developed in this module

Engineering Council AHEP4 LOs assessed	
C3	Select and apply appropriate computational and analytical techniques to model complex problems, recognising the limitations of the techniques employed
C5	Design solutions for complex problems that meet a combination of societal, user, business and customer needs as appropriate. This will involve consideration of applicable health & safety, diversity, inclusion, cultural, societal, environmental and commercial matters, codes of practice and industry standards
C6	Apply an integrated or systems approach to the solution of complex problems
C14	Discuss the role of quality management systems and continuous improvement in the context of complex problems
C16	Function effectively as an individual, and as a member or leader of a team

Student Name: **Student Id:** **Subsystem:**

Statement of Compliance ***(please tick to sign)***

☐ I declare that the work submitted is my own and that the work I submit is fully in accordance with the University regulations regarding assessments (*www.brookes.ac.uk/uniregulations/current*)

RUBRIC OR EQUIVALENT:

Marking grid and marking form are available on Moodle website of the module.

FORMATIVE FEEDBACK OPPORTUNITIES

(a) Discuss your work with your practical class tutor during practical classes;
(b) Discuss your work with lecturer and/or practical class tutor in drop-in hours.

SUMMATIVE FEEDBACK DELIVERABLES

Deliverable content and standard description and criteria
Please see attached file of *COMP6030 Coursework Marking and Feedback* for feedbacks on your coursework, which include:
(a) Breakdown of marks on each assessment criterion
(b) Comments on each aspect of the assessment against assessment criteria
(c) Annotations on your submitted work

Task 2: Software Quality Requirements for CloudTables-Customer

2.1 Security and Privacy

The CloudTables-Customer application handles sensitive personal and transactional information, including user identification details, booking records, and payment interactions. To ensure confidentiality, integrity, and compliance with data protection regulations, the system must meet the following security and privacy requirements:

- Authenticated restaurant customers shall have the ability to create, read, update, and delete (CRUD) their own data, including their profiles, table bookings, food pre-orders, reviews, and payment history. No other user type (e.g., restaurant staff, managers, or other customers) shall have access to customer data through the CloudTables-Customer interface. Role-based access control (RBAC) shall be used to enforce these permissions across all system operations.
- All communication between the mobile application and the backend system must be secured using HTTPS with TLS 1.3 or higher. Any unencrypted transmission shall be considered a failure.
- All sensitive customer data stored in the backend database, such as booking history and contact details, must be encrypted using AES-256 encryption. Plaintext storage must be avoided.
- Sessions must expire after 15 minutes of inactivity. Any access beyond this period must prompt re-authentication to continue the session.
- The application must allow users to delete their accounts and export their stored personal data in a human-readable format, thereby complying with data privacy regulations such as GDPR.

2.2 Performance

Performance is critical for ensuring a smooth and responsive user experience when searching for restaurants, booking tables, and viewing menus. The system must deliver low latency and efficient processing under normal and peak loads. The following requirements define the expected performance standards:

- The system shall return restaurant search results within 1 second for at least 95% of user requests under normal usage.
- Booking a table, including form submission and confirmation, shall complete within 2 seconds for at least 95% of transactions.

- The mobile app shall launch and load its home interface within 3 seconds on devices using a 3G or faster network.
- Table availability updates must reflect on the customer-facing application within 2 seconds of any changes made at the restaurant.
- Menu content, including images and descriptions, must load within 1.5 seconds after a customer accesses the relevant section.

2.3 Reliability

Reliability refers to the system's ability to operate consistently and recover gracefully from faults. For customers relying on the app for bookings and pre-orders, uninterrupted access and consistent data handling are essential. The following requirements establish the system's expected reliability:

- The system shall maintain an uptime of 99.9% over any calendar month. This allows for a maximum of 43 minutes of downtime per month.
- The application shall recover from server or client-side crashes within 5 minutes, preserving session data where applicable.
- In the event of backend failure, customer booking and order data must be recoverable from automated backups performed every 24 hours.
- If an operation fails (e.g., booking, pre-order), the app must notify the user with a clear error message and present an option to retry.
- All bookings and pre-orders shall be implemented using atomic transactions to ensure no partial state occurs in case of failure.

2.4 Scalability

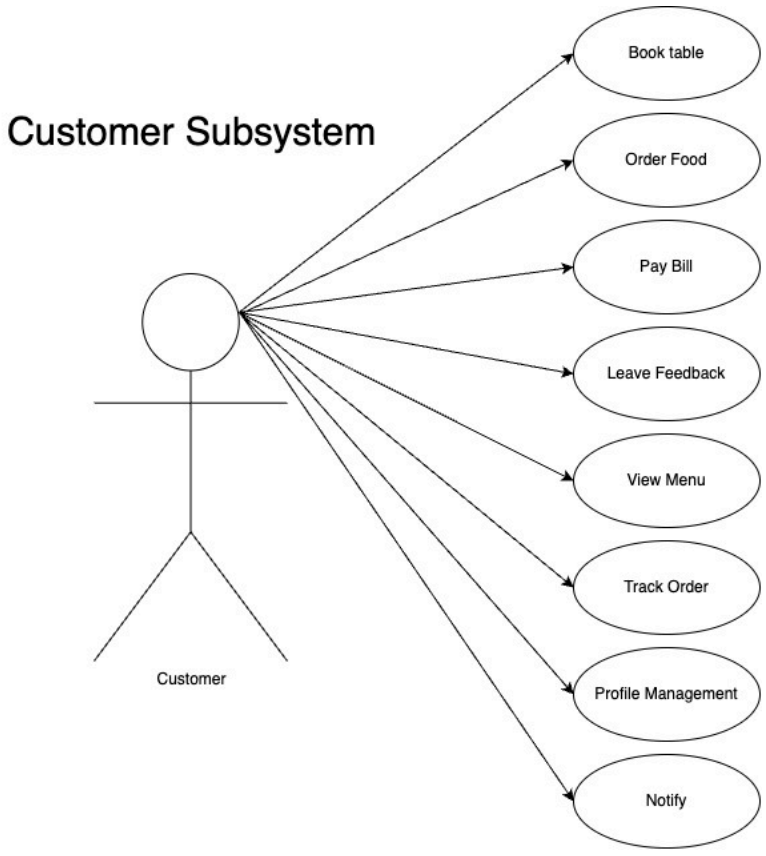
To accommodate growth in users and participating restaurants, the CloudTables-Customer system must scale horizontally and maintain performance as demand increases. Scalability ensures that the app functions well under load and can evolve with business needs. The following requirements define scalability expectations:

- The system must support at least 5,000 concurrent users without degrading average response times beyond 3 seconds.
- Load balancing mechanisms must distribute traffic across multiple servers using algorithms such as round-robin or least-connections.
- The backend services must support horizontal scaling through containerisation technologies (e.g., Docker, Kubernetes) to manage growing demand.

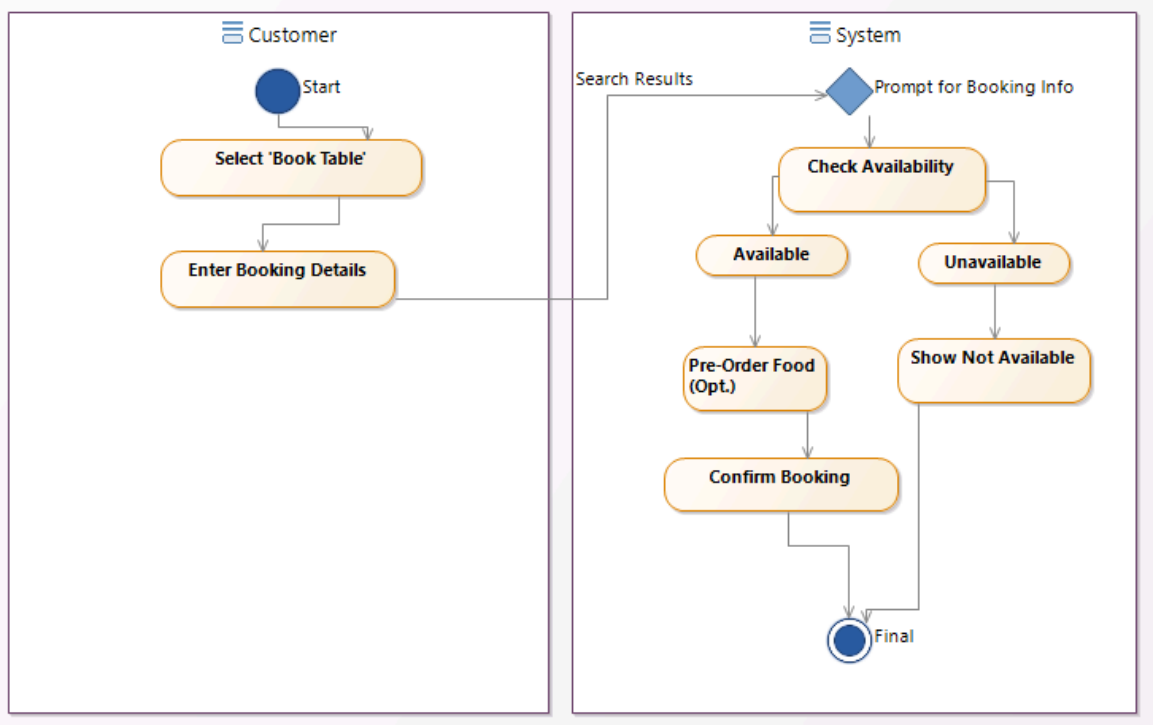
- - The system database must support partitioning and replication to maintain performance as the number of daily active users exceeds 10,000.
- - The mobile app must maintain responsive UI and functionality across devices with screen sizes ranging from 5 to 13 inches.

Task 3: Software Modelling and Specification

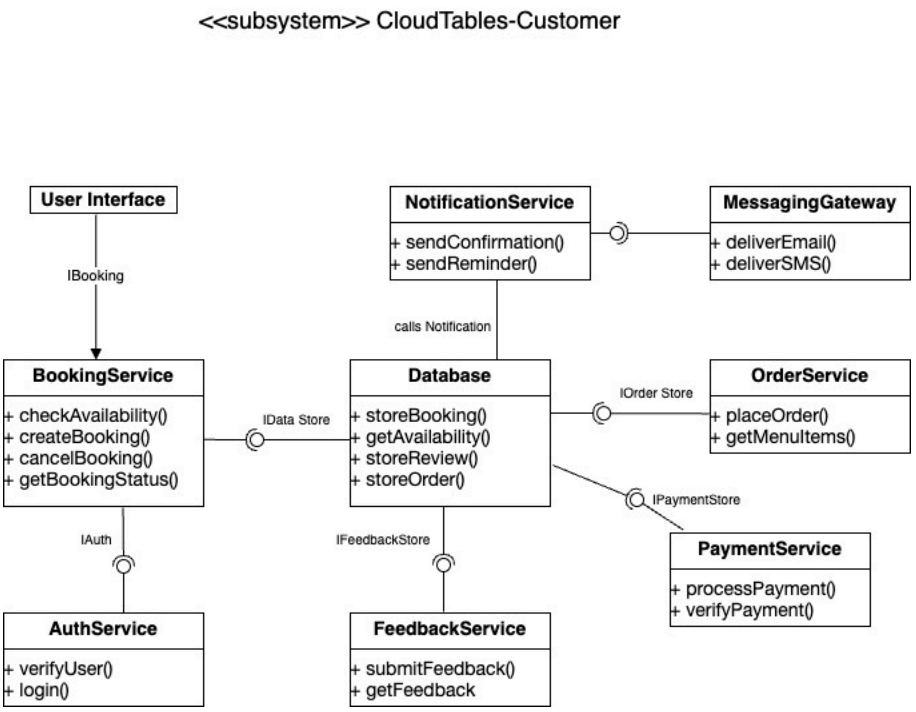
3.1 –



3.2 –


Customer
Start
Select 'Book Table'
Enter Booking Details
System
Search Results
Prompt for Booking Info
Check Availability
Available
Unavailable
Pre-Order Food (Opt.)
Show Not Available
Confirm Booking
Final

Task 4: Software Architectural Design



Subsystem Overview:

The CloudTables-Customer subsystem is a critical part of the broader CloudTables platform, designed to manage customer-facing operations such as table bookings, order placement, payments, feedback collection, and notifications. The subsystem follows a microservices architectural style, where independent services communicate via well-defined interfaces, promoting modularity, scalability, and resilience.

BookingService:

Responsibilities: Manages the full lifecycle of customer bookings, including checking availability, creating new bookings, updating or cancelling existing ones, and retrieving booking status.
Methods:
- checkAvailability()
- createBooking()
- cancelBooking()
- getBookingStatus()

Interfaces:
- Provided: IBooking (consumed by the User Interface)
- Required: IDataStore (Database), IAuth (AuthService)

Dependencies:
- Requires user authentication via AuthService.
- Stores and retrieves booking data through the Database.

OrderService:

Responsibilities: Handles the creation and management of customer food orders, including menu retrieval and order placement.
Methods:
- placeOrder()
- getMenuItems()

Interfaces:
- Required: IOrderStore (provided by Database)

PaymentService:

Responsibilities: Processes and verifies customer payments securely.
Methods:
- processPayment()
- verifyPayment()

Interfaces:
- Required: IPaymentStore (provided by Database)

FeedBackService:

Responsibilities: Allows customers to submit and view feedback related to their experience.
Methods:
- submitFeedback()
- getFeedback()

Interfaces:
- Required: IFeedbackStore (provided by Database)

NotificationService:

Responsibilities: Sends confirmations and reminders to customers based on key events (e.g., booking created, order placed).
Methods:
- sendConfirmation()
- sendReminder()

Interfaces:
- Required: IMessaging (provided by MessagingGateway)

MessagingGateway:

Responsibilities: Abstracts external messaging APIs for email and SMS delivery.
Methods:
- deliverEmail()
- deliverSMS()

Database:

Responsibilities: Centralized storage of all customer-related data including bookings, feedback, orders, and availability.
Methods:
- storeBooking()
- getAvailability()
- storeReview()
- storeOrder()

Interfaces:
- Provided: IDataStore, IOrderStore, IPaymentStore, IFeedbackStore

AuthService:

Responsibilities: Manages authentication of users through login and verification processes.
Methods:
- verifyUser()
- login()

Interfaces:
- Provided: IAuth

User Interface:

Responsibilities: Acts as the entry point for customers via web or mobile apps.
Interfaces:
- Consumes: IBooking (from BookingService)

Communication Between Components:

Services interact via explicitly defined interfaces rather than direct method calls. The system is loosely coupled using REST APIs. BookingService, OrderService, and FeedbackService interact with the centralized Database. NotificationService delegates email/SMS delivery to the external-facing MessagingGateway.

Security Measures:

Authentication:
- All user requests are authenticated using JWT tokens through AuthService.

Encryption:
- TLS is enforced for API communications.
- Sensitive data (e.g., payment info) is stored encrypted (AES-256).

Access Control:
- Role-based access rules govern which services or users can access specific endpoints or methods.

Scalability and Fault Tolerance:

Scalability:
- Stateless microservices (e.g., NotificationService) can be replicated horizontally.
- Services are deployed in containers and orchestrated via Kubernetes (or similar), enabling elastic scaling.

Fault Tolerance:
- Circuit breaker patterns and retry mechanisms are used in external-facing services (e.g., MessagingGateway).
- BookingService and PaymentService are designed to handle partial failures gracefully.

Data Flow Summary:

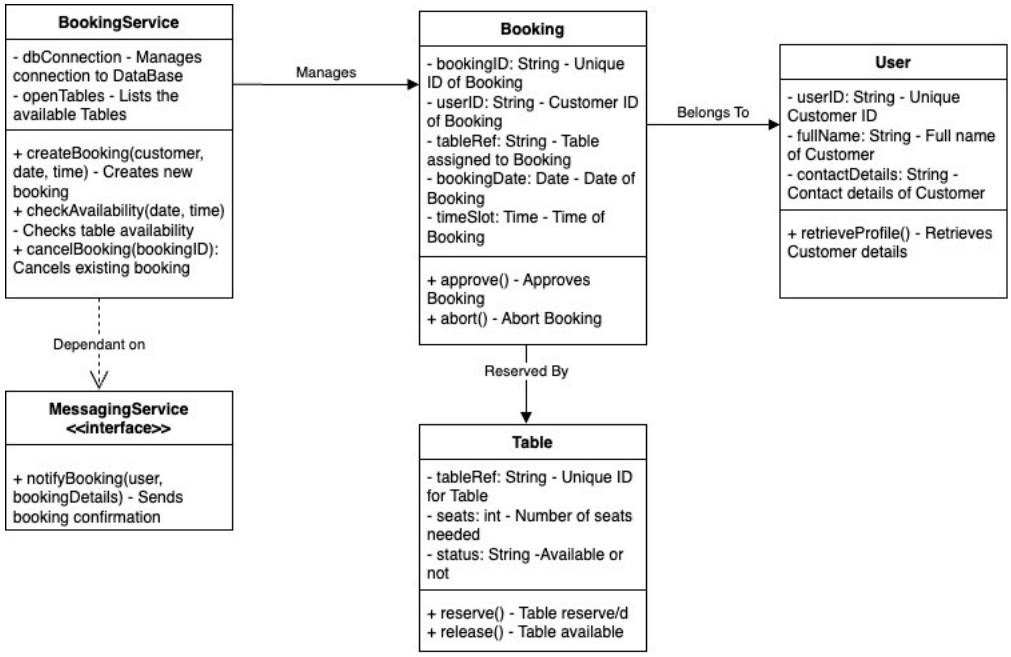
Input:
- User initiates a request via the interface (e.g., booking or order).

Processing:
- BookingService, OrderService, etc., validate and process the request, storing necessary information in the database.

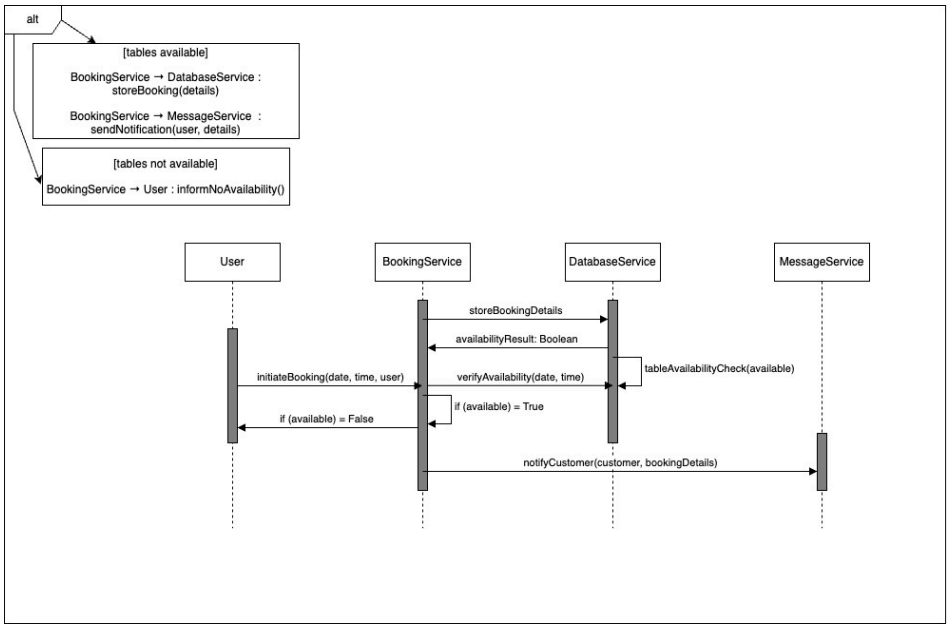
Output:
- NotificationService sends confirmation via email or SMS through MessagingGateway.

Task 5: Software Detailed Design

5.1 –



5.2 –



Main Scenario

The User initiates a booking by providing the date, time, and their user information.

The BookingService receives the request and verifies table availability by querying the DatabaseService.

If tables are available:

- The BookingService stores the booking details in the DatabaseService.

- The BookingService then sends a confirmation message to the user via the MessageService.

The booking process is completed successfully.

Alternate Path

If no tables are available – The BookingService notifies the User that the booking could not be completed.

Actors and Objects

User, BookingService, DatabaseService, MessageService.

Key Messages

- initiateBooking(date, time, user)

- verifyAvailability(date, time)

- availabilityResult: Boolean

- storeBookingDetails(details)

- sendMessage(customer, bookingDetails)

- informNoAvailability()