UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

Informatics Institute of Technology

Software Engineering Principles and Practice

Report on the design of a Hotel booking system

5SENG007C

Coursework 2

Module : 5SENG007C SOFTWARE ENGINEERING PRINCIPLES

Date of submission : 07 /01 / 2025

UOW ID: w2052726

Student ID : 20220857

Student Name : B . Oshadha Kumarinda Fernando

Module Leader: Cassim Farook

Table of Contents

Table of Figures

1. Reflection on software engineering methodology and lifecycle.

The Agile methodology was chosen for this project because it provides the flexibility and adaptability needed to handle changing requirements. Unlike traditional methods, Agile thrives on continuous user feedback, making it ideal for projects where meeting user expectations is a top priority. The process is broken down into manageable cycles called sprints, where each phase planning, designing, development, testing, and review is revisited iteratively. Compared to other methodologies like Waterfall, Scrum, or Spiral, Agile offers a more dynamic and responsive approach, which suits modern software development.

1.1 Reflection on the Agile Methodology

Agile stands out as a powerful method for managing software projects, especially those that require constant updates and revisions. Its focus on iterative improvement ensures that the final product doesn't just meet requirements but aligns closely with user needs. I found that Agile's collaborative nature fosters strong communication between developers, stakeholders, and users, which makes the process feel more inclusive and efficient [1].

One of the biggest advantages of Agile is its flexibility. Traditional methods like Waterfall often stick rigidly to an initial plan, which can cause delays when things need to change. Agile, on the other hand, welcomes changes even late in the development process. This adaptability is a real lifesaver in fast-paced environments where new ideas or challenges frequently arise. By delivering small, functional increments during short sprints, stakeholders can see tangible progress and provide immediate feedback. This minimizes wasted effort and keeps the project moving in the right direction [1].

Agile, in my opinion, effectively balances creativity with structure. Agile keeps things realistic and user-focused, whereas approaches like Scrum or Spiral may concentrate more on in-depth planning or analysis. It's not flawless the continuous iteration might occasionally feel too much to handle but it's a process that truly works when the objective is to produce something

significant and user-focused. Agile also reduces risks by breaking the project up into smaller, easier-to-manage components [1].

By segmenting the project into smaller, more manageable parts, agile also lowers risks. Instead than delaying testing till the end, as is the case with Waterfall, testing is incorporated into every sprint, enabling teams to find and fix problems early. This method of continuous testing guarantees those issues are fixed promptly, resulting in a better product. Agile also places a strong emphasis on teamwork and candid communication, creating an atmosphere that stimulates innovation and problem-solving [1].

1.2 Reflection on the Agile Lifecycle

Sprints, lasting two to four weeks, form the heart of Agile. Each sprint includes planning, design, development, testing, and review, ensuring steady progress and flexibility to adapt to changes. The planning phase focuses on setting goals and breaking them into manageable tasks, keeping the team aligned. Unlike rigid methods like Waterfall, Agile's incremental planning allows teams to adjust priorities based on feedback, making it ideal for dynamic projects.

During **the design and development phases** functional components are delivered with an emphasis on rapid feature implementation. During each sprint, Agile promotes the development of "minimum viable products" (MVPs) that may be tested and refined in later cycles. This is in contrast to approaches like Waterfall, which requires that design be finished before any development can start, or Spiral, where thorough risk analysis may cause implementation to be delayed [2].

The testing phase is a crucial part of the Agile lifecycle because it's integrated into every sprint. By verifying each increment thoroughly before moving forward, it helps prevent serious issues or faults from piling up over time. This proactive approach to testing ensures that problems are caught and resolved early, saving both time and effort. In contrast, traditional methods like Waterfall leave testing until the end of the process, often leading to costly rework when issues are discovered late. Agile's continuous testing approach is not just more efficient—it's a smarter way to maintain quality throughout the development cycle [2].

The **review and feedback phase** concludes each sprint. During this phase, stakeholders evaluate the progress and provide feedback. This ongoing feedback loop ensures that the project remains aligned with user needs and expectations. Unlike Scrum, which requires highly structured review processes, Agile keeps the feedback mechanism flexible and adaptable, making it easier to implement changes quickly [2].

1.3 Reflection on Other Software Engineering Methodologies

Waterfall methodology, which follows a rigid, linear process, Agile allows for iterative development and continuous improvement. Waterfall requires completing one phase entirely before moving to the next, making it challenging to adapt to changes once development has begun. In contrast, Agile enables flexibility by allowing adjustments to features or design based on feedback during the development process, which ensures that the product evolves effectively [3] [4].

The **Scrum methodology**, while a subset of Agile, introduces a higher level of structure, including defined roles such as Scrum Master and Product Owner. While Scrum provides many of the iterative benefits of Agile, its strict adherence to roles and daily meetings can make it less practical for smaller teams or simpler projects. Agile, by comparison, maintains the flexibility and iterative process without the need for such formality, making it easier to adapt to the unique needs of a project [3].

The **Spiral methodology** combines iterative development with a strong focus on risk analysis and management. While this is beneficial for large-scale or high-risk projects, it introduces significant complexity and time requirements. For smaller projects or those with manageable risks, Spiral's emphasis on risk evaluation in every phase can slow down progress unnecessarily. Agile, on the other hand, balances iterative development and flexibility without the additional burden of constant risk assessments [3] [4].

2. System architecture

2.1 Architecture diagram

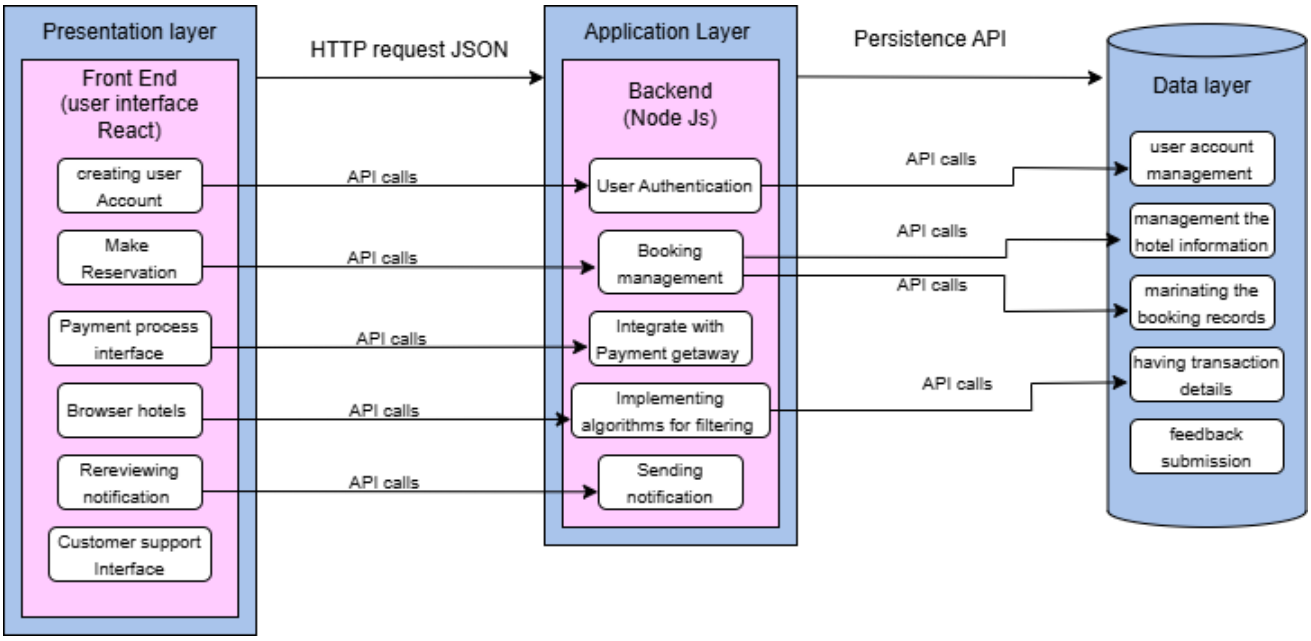

Figure 1 : Architectural Diagram

2.1.1 Explanation of the Diagram

Frontend (User Interface - React)

- This layer represents the part of the application where users interact with the system.
- It includes the following key interfaces and features:
    - **Creating User Account**: Allows users to register and log in.
    - **Make Reservation**: Enables users to search for hotels, select a room, and confirm the booking.
    - **Payment Process Interface**: Facilitates the payment process after room selection.
    - **Browser Hotels**: Provides an interface to search and browse available hotels with filtering options.

    - **Reviewing Notifications**: Displays notifications related to booking confirmations and cancellations.
- **Communication with Backend**: All interactions are made via **HTTP requests** using **RESTful APIs**, and data is exchanged in **JSON format**.

Backend (Node.js)

- This layer handles all business logic and manages data processing. It is composed of several core modules:
    1. **User Authentication**: Ensures secure login and registration for users.
    2. **Booking Management**: Manages the entire reservation process, including room availability checks and booking confirmations.
    3. **Integrate with Payment Gateway**: Facilitates secure payment transactions with external payment providers.
    4. **Implementing Algorithms for Filtering**: Applies filtering criteria (price range, amenities, location) when users search for hotels.
    5. **Sending Notification**: Manages notification services, sending real-time updates about bookings and cancellations.
- **Communication with Database**: Backend interacts with the database via **API calls** to retrieve and update data.

Data Layer (Database)

- This layer stores and manages all data related to the hotel booking system:
    - **User Account Management**: Stores user information such as account details and login credentials.
    - **Hotel Information Management**: Maintains data about hotels, including room types, prices, and amenities.
    - **Booking Records**: Keeps track of all reservations, including past and active bookings.
    - **Transaction Details**: Logs payment transactions for tracking and auditing.
    - **Feedback Submission**: Stores user reviews and feedback.

2.2 Completeness of the Architecture

The architecture comprehensively addresses the functional and non-functional requirements of the hotel booking system:

1. **User Roles and Interaction**:
    - Customers can:
        - Browse hotels and check room availability.
        - Make, modify, or cancel reservations.
        - Provide feedback and view notifications.
    - Hotel Managers can:
        - Manage hotel data, including room availability and pricing.
        - Monitor reservations and respond to customer feedback.
    - Administrators can:
        - Manage user accounts and oversee the system's performance.
2. **Data Flow**:
    - Data flows from the **frontend** to the **backend**, where business logic is applied.
    - Backend retrieves and updates information from the **database** and sends the processed data back to the frontend.
    - Notifications are sent asynchronously, ensuring users stay updated without interrupting ongoing interactions.
3. **Security Measures**:
    - Sensitive data, including user credentials and payment information, is encrypted both during transmission (**HTTPS**) and at rest.
    - Secure login mechanisms, such as **multi-factor authentication (MFA)** and **password hashing**, are implemented to ensure only authorized users can access their data.
4. **Integration with External Services**:
    - The backend integrates with external payment gateways for secure transactions.
    - Notifications (e.g., email or SMS) are sent via external messaging services, ensuring timely updates for users.

2.3 Justification of the Architecture

Tiers

The architecture follows a **three-tier model**:

1. **Presentation Layer**: The frontend, built using **React**, handles user interaction.
2. **Application Layer**: The backend, developed in **Node.js**, handles business logic and processes user requests.
3. **Data Layer**: The database layer stores and manages persistent data.

This separation of concerns improves the **scalability**, **maintainability**, and **flexibility** of the system.

Platform

- A **web-based platform** is chosen to maximize accessibility across different devices, including desktops, tablets, and smartphones.
- Using **React** ensures a responsive and user-friendly interface, enhancing the user experience.

Components

1. **Frontend Components**:
    - Includes features like **search filters**, **booking forms**, and a **dashboard** for user management.
2. **Backend Components**:
    - Comprises modules for **authentication**, **booking management**, **payment handling**, and **notification services**.
3. **Database Components**:
    - Consists of tables for storing:
        - User accounts
        - Hotel details
        - Booking records

- Payment transactions
- Feedback submissions

Connectors

1. **Frontend-Backend Connector**:
    - Uses **RESTful APIs** for communication.
    - Data is exchanged in **JSON format**.
2. **Backend-Database Connector**:
    - The backend interacts with the database using a persistence API (e.g., **Sequelize ORM** for SQL databases or **Mongoose** for MongoDB).
3. **Backend-External Service Connectors**:
    - Secure API calls are used for:
        - Payment gateway integration.
        - Sending notifications via external services (e.g., email or SMS providers).

Communication Type

1. **Synchronous Communication**:
    - Used for real-time interactions, such as booking confirmations and user authentication.
2. **Asynchronous Communication**:
    - Employed for tasks like sending notifications to ensure they don't block user interactions.

3. Testing of your architectural design.

3.1 Demonstration of Functional Requirements

1. **Real-time Room Availability**

   The **backend (Node.js)** communicates with the **database** through API calls to retrieve real-time room availability data. The frontend (React) requests this data via RESTful APIs and displays updated information to the user.

2. **Secure Payment Processing**

   The **backend** integrates with an external **payment gateway** through secure API calls. After the user selects a room and confirms the reservation, the backend transmits payment details to the payment gateway using encrypted communication to ensure secure handling of sensitive data.

3. **User Account Management**

   The **user authentication module** in the backend handles user registration, login, and profile management. The frontend provides an intuitive interface for users to manage their profiles, while the backend interacts with the **database** to create, update, or retrieve user information.

4. **Reservation Management**

   Admins and hotel managers can use the **dashboard interface** to update room availability, adjust pricing, and monitor reservations. The dashboard communicates with backend APIs, which handle updates to the **database** regarding room availability, pricing, and reservation status.

5. **Notification Management**

   For booking confirmations and cancellations, the backend uses asynchronous communication to send real-time notifications via an external notification service (e.g., email or SMS). This ensures timely updates to users without affecting the performance of other operations.

6. **Customer Support Integration**

   A **customer support interface** on the frontend allows users to access live chat or a chatbot for assistance. The backend communicates with the support service to handle user inquiries and provide immediate responses.

3.2 Demonstration of Non-Functional Requirements

1. **Scalability**

   The system follows a **three-tier architecture** with a clear separation of concerns between the **presentation layer (frontend)**, **application layer (backend)**, and **data layer (database)**. This design ensures that each layer can be scaled independently to handle increased loads without performance degradation. For example, the backend can be horizontally scaled using a load balancer, while the database can be sharded or replicated as needed.

2. **Performance**

   Backend performance is optimized by implementing caching mechanisms (e.g., in-memory caching for frequently accessed data) and using asynchronous communication for non-critical tasks such as sending notifications. This reduces latency and improves response times for users.

3. **Security**
   - User inputs are validated on both the **frontend** and **backend** to prevent malicious data from entering the system.
   - Sensitive data, including payment information, is encrypted before being transmitted to the external payment gateway.
   - All communication between the **frontend**, **backend**, and external services uses secure **HTTPS** protocols to protect data in transit.

4. **Usability**

   The frontend is developed using **React**, following a responsive design approach to ensure a consistent user experience across devices (desktops, tablets, and smartphones). This enhances accessibility and usability for all users, providing a seamless interface for tasks such as browsing hotels, making reservations, and submitting feedback.

4. Class diagram and justification.

4.1 Class Diagram

Figure 2 : Class Diagram

4.2 Justification for the Inclusion of Each Class, Its Attributes, and Methods

1. Customer Class

- **Attributes**
  - customerID: Uniquely identifies each customer and tracks their interactions with the system.
  - name, email, phone: Essential contact details for account management and personalized communication.
  - paymentDetails: Stores customer's payment information for completing transactions.
- **Methods**
  - register(): Enables a customer to create an account in the system.
  - searchForHotelRooms(): Allows customers to explore available rooms in various hotels.
  - selectRooms(), cancelBooking(): Manage the reservation process, including selecting and canceling bookings.
  - provideFeedback(): Allows the customer to share their experience after their stay.
  - enterPaymentMethods(): Used to add or update payment details before making a reservation.

2. Booking Class

- **Attributes**
  - bookingID: A unique identifier for each booking made by customers.
  - checkInDate, checkOutDate: Critical for scheduling the duration of a customer's stay.
  - totalPrice: Represents the total cost of the booking.
  - status: Tracks the current state of the booking (e.g., confirmed, pending, or canceled).

- **Methods**
    - confirmBooking(), modifyBooking(), cancelBooking(): Manage the booking lifecycle by confirming, modifying, or canceling reservations.
    - notifyRoomReservation(): Sends a notification to the customer when a booking is confirmed.

3. Room Class

- **Attributes**
    - roomID: Uniquely identifies each room in the system.
    - roomType: Specifies the type of room (e.g., single, double, suite).
    - price: The cost per night for staying in the room.
    - availability: A boolean attribute indicating whether a room is available for booking.
- **Methods**
    - checkAvailability(): Checks if a specific room is available for a given date range.
    - updateAvailability(): Updates the room's status based on booking activity.
    - showAvailability(): Displays available rooms to the customer.
    - reserveRoom(): Reserves a room when a customer confirms a booking.

4. Hotel Class

- **Attributes**
    - hotelID: Uniquely identifies each hotel.
    - name, location: Essential for identifying and locating hotels.
    - rate: Reflects the hotel's overall rating, based on customer feedback.
    - rooms: A list of rooms associated with the hotel.
- **Methods**
    - managePromotions(): Allows hotel managers to create and manage special offers.
    - updateDetails(): Enables hotel managers to update hotel information, such as name, location, or available amenities.

5. Payment Class

- **Attributes**
  - paymentID: Uniquely identifies each payment transaction.
  - amount: Represents the amount paid by the customer.
  - paymentDate: The date when the payment was made.
  - paymentMethod: Specifies the method of payment (e.g., credit card, PayPal).
- **Methods**
  - processPayment(), validatePayment(): Ensure that payments are processed securely and validated correctly.
  - makePayment(): Initiates the payment process.
  - savePayment(): Stores payment details in the database.
  - sendConfirmationMessage(): Sends a message to the customer confirming the success or failure of the payment.

6. Feedback Class

- **Attributes**
  - feedbackID: Uniquely identifies each piece of feedback.
  - rating: Captures a numerical rating provided by the customer.
  - comments: Contains detailed feedback from the customer.
  - dateSubmitted: Tracks when the feedback was provided.
- **Methods**
  - submitFeedback(): Allows customers to submit their feedback after a stay.
  - viewFeedback(): Displays collected feedback for analysis and reporting.

7. Admin Class

- **Attributes**

- adminID: Uniquely identifies each admin.
- username, password: Used for secure authentication of administrators.

- **Methods**
  - manageUsers(): Provides functionality for admins to manage customer accounts.
  - manageBookings(): Allows admins to view, update, or delete bookings.
  - generateReports(): Enables admins to generate reports for system analysis and decision-making.

8. ChatSupport Class

- **Attributes**
  - supportID: Uniquely identifies each chat support session.
  - type: Specifies the type of support interaction (e.g., live chat or chatbot).
  - query: Contains the customer's query details.
- **Methods**
  - initiateChat(): Starts a new chat session with a support agent or chatbot.
  - respondToQuery(): Allows the support system to respond to customer queries in real time.

9. Notification Class

- **Attributes**
  - notificationID: Uniquely identifies each notification.
  - message: Contains the content of the notification.
  - timestamp: Records when the notification was sent.
- **Methods**
  - sendNotification(): Sends a notification to the user regarding booking status or payment confirmation.
  - logNotification(): Logs notification details for future reference or auditing.

4.3 Correct UML Notation

1. **Class Representation**

   Each class is represented with three sections:

   - **Class name** at the top.
   - **Attributes** in the middle, prefixed with visibility modifiers (+ for public, - for private).
   - **Methods** at the bottom, also prefixed with visibility modifiers.

2. **Relationships and Multiplicities**

   - Relationships between classes (associations) are clearly depicted using lines.
   - Multiplicities are correctly labeled at both ends of the relationships (e.g., 1..*, 0..1), providing insight into the cardinality of the relationships.

3. **Dashed Arrows for Dependencies**

   Dashed arrows represent dependencies between classes where appropriate (e.g., when one class uses another class's methods).

5. Testing the class design.

5.1 Correctness and completeness of the sequence diagram.

5.1.1 Sequence diagram

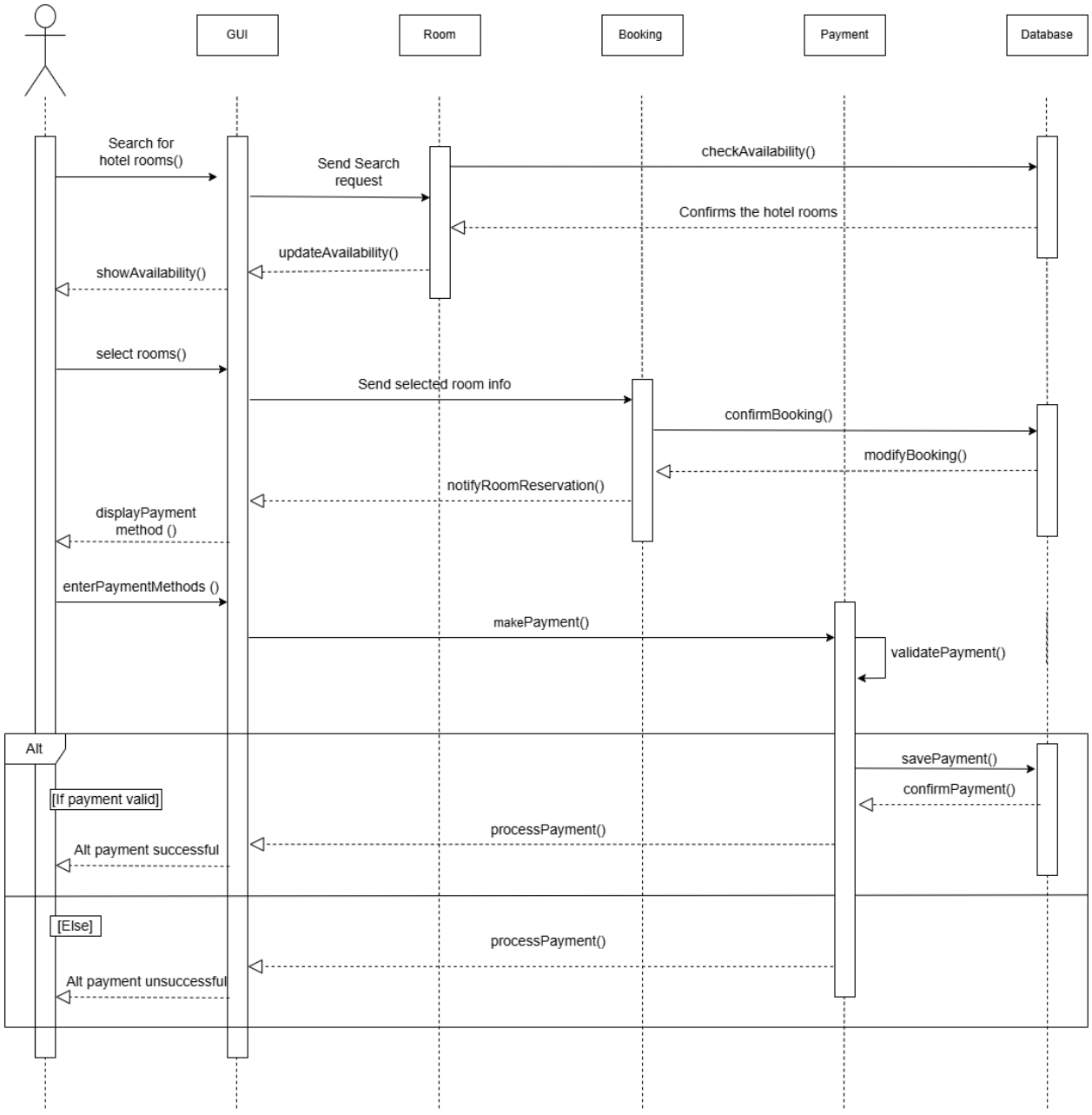

Figure 3 : Sequence Diagram

5. 2 Correctness and Completeness of the Sequence Diagram

1. Accurate Representation of Interaction

The sequence diagram accurately depicts interactions between the following components:

- **GUI (Graphical User Interface)**
- **Room**
- **Booking**
- **Payment**
- **Database**

The sequence of method calls and the flow of information among these components are correctly represented.

2. Ensures the Logical Flow

- **Searching for hotel rooms:**
  The process begins when the customer initiates a search using the searchHotelRooms() method.
  The GUI sends a search request to the **Room** component, which invokes checkAvailability() to verify room availability by interacting with the **Database**.
  Upon confirmation from the database, room availability is updated via updateAvailability() and displayed to the user through showAvailability().
- **Selecting and booking rooms:**
  Once the customer selects the desired room using selectRooms(), the GUI sends the selected room details to the **Booking** component via confirmBooking().
  The **Booking** component interacts with the **Database** by invoking modifyBooking() to create a booking record.
- **Payment process:**
  After booking, the system prompts the customer to enter payment details using displayPaymentMethod() and enterPaymentMethods().
  The **Payment** component receives the payment information and processes it by invoking

makePayment(), followed by validatePayment() which ensures that the payment details are valid.

If validation is successful, the system records the payment in the **Database** using savePayment() and confirms the payment using confirmPayment().

- **Final confirmation:**

  If the payment is successful, the **Booking** component notifies the customer via notifyRoomReservation().

  In case the payment is unsuccessful, the system repeats the processPayment() step or displays a failure message to the user.

3. Inclusion of All Relevant Classes

All key components involved in the hotel booking process are represented in the diagram, ensuring a complete and clear understanding of the interactions.

4. Database Interaction

The sequence diagram highlights interactions with the **Database** at critical steps:

- Checking room availability
- Modifying booking details
- Validating payment
- Saving payment records

This ensures that the data flow between the front-end, back-end, and database is properly addressed.

5.3 Required Classes and Their Support for the Functionality

1. **Customer Class**
    - **Attributes:** customerID, name, email, phone, paymentDetails
    - **Methods:**
        - searchHotelRooms(): Initiates a search for available rooms.
        - selectRooms(): Selects rooms for booking.
2. **Room Class**
    - **Attributes:** roomID, roomType, price, availability
    - **Methods:**
        - checkAvailability(): Verifies room availability.
        - updateAvailability(): Updates the room status once booked.
        - showAvailability(): Displays available rooms to the user.
3. **Booking Class**
    - **Attributes:** bookingID, checkInDate, checkOutDate, totalPrice, status
    - **Methods:**
        - confirmBooking(): Confirms the selected room for booking.
        - notifyRoomReservation(): Sends a notification to the customer for payment.
4. **Payment Class**
    - **Attributes:** paymentID, amount, paymentDate, paymentMethod
    - **Methods:**
        - makePayment(): Initiates the payment process.
        - validatePayment(): Validates the provided payment details.
        - processPayment(): Processes the payment and interacts with the database.
5. **Database**
    - Handles data storage and retrieval operations for room availability, booking confirmation, and payment validation.

6. Consideration of modern tools.

6.1 Development and testing tools.

Frameworks/Libraries/Plugins:

- **Node.js** (for backend development): Node.js is a fast, event-driven JavaScript runtime environment that allows you to handle asynchronous requests efficiently. It's ideal for building scalable APIs and handling user registration, hotel search, and reservations in real-time.
- **Express.js** (for API creation): Express is a lightweight and flexible Node.js framework that simplifies the development of robust APIs. It will be used to handle HTTP requests, manage routes for booking and reservations, and ensure smooth data flow between the frontend and backend.
- **React** (for frontend development): React's component-based architecture makes it easy to build interactive UIs. It is highly suitable for the hotel booking app as it offers a fast, dynamic user experience with smooth state management.
- **Redux** (for state management in React): Redux is useful for managing the global state of your application. It will help in maintaining data consistency, especially when managing user details, reservation data, and hotel listings across different components.
- **Material-UI** or **React Bootstrap** (for UI styling): Both libraries provide pre-styled components and themes for building a sleek, responsive UI. Material-UI is highly customizable and fits well with React applications, while React Bootstrap integrates seamlessly with the bootstrap framework.
- **Axios** (for HTTP requests): Axios is a promise-based HTTP client for the browser and Node.js. It will be used to send API requests to your Node.js backend, for actions like retrieving hotel data, making bookings, and processing payments.

2. **Version Control Systems (VCS):**
    - **GitHub**: Essential for version control, GitHub helps manage your code, track changes, and collaborate with your team. It also integrates with deployment pipelines for easy deployment to cloud platforms.

3. **Dependency Management Tools:**
    - **npm** (for Node.js dependencies): npm is the default package manager for Node.js and will help you manage libraries and dependencies such as Express, React, and Axios.
    - **yarn** (alternative to npm): Yarn is another dependency management tool that can help with faster installs and lock file management, ensuring that your dependencies are consistent across environments.

4. **API Testing Tools:**
    - **Postman**: A versatile tool to test your REST APIs. With Postman, you can make requests to your Node.js API endpoints and verify the responses. It also allows you to automate API testing for different scenarios such as booking creation, cancellations, and payment processing.

5. **Testing Frameworks:**
    - **Jest** (for unit and integration testing): Jest is a JavaScript testing framework that works well with both Node.js and React. You can use it to write unit tests for your backend logic (like reservation handling) and frontend components (like booking forms or hotel listings).
    - **Supertest** (for API testing in Node.js): Used alongside Jest, Supertest helps you write tests for your Express API to ensure that the endpoints work correctly, such as making reservations, modifying bookings, etc.

- **React Testing Library** (for frontend testing): This library helps you test React components by simulating user interactions and ensuring your UI components render correctly.

Implementation Tools

1. **Operating Systems:**
   - **Linux/Ubuntu**: Ideal for hosting the backend services and deploying the full application on a cloud server. It's widely used in production environments due to its stability and security.
   - **Windows/macOS**: These operating systems can be used for development environments, depending on your personal preference.
2. **Virtual Machines:**
   - **Docker**: Docker is an essential tool for containerizing your app. You can containerize both the backend (Node.js/Express) and frontend (React) parts of your app, ensuring consistency across development, staging, and production environments.
3. **Servers:**
   - **NGINX**: NGINX is a lightweight, high-performance server that you can use as a reverse proxy to forward requests to your Node.js app. It also handles load balancing and can serve static files for React.
4. **Cloud-Based Tools/Platforms:**
   - **Heroku**: A cloud platform that makes it easy to deploy Node.js apps. It integrates well with GitHub for continuous deployment and is suitable for smaller-scale applications.
   - **AWS or Google Cloud**: These platforms offer scalable solutions for hosting your app. AWS has services like EC2 for running servers and S3 for static file hosting, while Google Cloud offers similar options.
   - **Firebase**: Firebase can be used for authentication (e.g., with Google or Facebook login)
   - and real-time data management if you prefer using a backend-as-a-service approach.

5. **APIs:**
    - **Stripe/PayPal APIs** (for payment processing): Stripe and PayPal offer secure APIs for handling payments. These APIs will integrate into your backend for booking payment processing, allowing users to securely complete transactions.
    - **Google Places API**: Useful for retrieving detailed information about hotels, including their location, reviews, amenities, and photos. It can be integrated with your backend to provide users with up-to-date hotel data.

7. Considerations of Software Engineering Principles in the Design

Separation of Concerns

Separation of concerns is a key software engineering principle aimed at dividing a system into distinct sections, each responsible for a specific functionality. In the hotel booking system, this principle is implemented using a layered architecture:

- **Presentation Layer**: This layer handles user interaction and displays information to the customer through the **GUI** component.
- **Business Logic Layer**: The **Room**, **Booking**, and **Payment** components manage core business processes such as checking availability, confirming bookings, and processing payments.
- **Data Layer**: The **Database** component stores and retrieves information related to room availability, booking details, and payment records.

This separation allows developers to modify or improve one layer (e.g., updating the GUI) without affecting the business logic or database structure. As a result, the system becomes more maintainable and scalable.

Modularity

The design of the hotel booking system follows a modular approach by dividing the system into independent components or classes:

- **Customer**, **Room**, **Booking**, **Payment**, and **Notification** are separate modules, each encapsulating its data and behavior.
- For example, the **Room** class manages availability and room details, while the **Payment** class handles the payment process independently.
- This modular structure promotes code reuse and simplifies debugging by allowing developers to work on individual modules without needing to understand the entire system.
- Testing is also more efficient, as each module can be tested separately before integrating it with other components.

Abstraction

Abstraction simplifies complex operations by modeling essential entities and hiding unnecessary details.

- The system uses classes to represent real-world entities like customers, rooms, bookings, and payments.
- For instance, the **Booking** class provides high-level methods such as confirmBooking() and notifyRoomReservation(), abstracting the internal complexities of managing reservations.
- This abstraction ensures that users and developers interact with simplified, high-level interfaces, improving usability and maintainability.
- By focusing on the essential properties and behaviors of entities, the design reduces complexity and enhances clarity.

Anticipation of Change

The system is designed with flexibility to accommodate future changes in requirements or technology.

- For example, if new payment methods need to be integrated, the **Payment** component can be extended without impacting the rest of the system due to its modular structure.
- An Agile development approach has been adopted, enabling iterative development cycles. During each iteration, customer feedback is gathered and incorporated, ensuring that the system remains aligned with evolving user needs.
- Design patterns and flexible architecture further allow the system to adapt to changes with minimal disruption.

Incremental Development

Incremental development is employed by delivering the system in small, functional increments rather than attempting to deliver the entire system at once.

- Each iteration focuses on a specific set of features, such as room search, booking, or payment processing.
- For example, in the initial sprint, the team may focus on implementing room search and availability checking, while later sprints address booking confirmation and payment processing.
- This approach reduces risk by allowing continuous testing, validation, and feedback at each stage. Stakeholders can observe tangible progress frequently and provide input, ensuring that the final product meets their expectations.

References

[1] D. Young, "Software Development Methodologies," 2013.

[2] K. E. Risener, "A study of Software Development Methodologies," 04 2022. [Online]. Available: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://scholarworks.uark.edu/cgi/viewcontent.cgi?article=1105&context=csceuht. [Accessed 04 01 2025].

[3] V. Niitin, "Itransistion," 16 04 2024. [Online]. Available: https://www.itransition.com/software-development/methodologies. [Accessed 05 01 2025].

[4] M. I. D. Rathnayake, 2020. [Online]. Available: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://ijariie.com/AdminUploadPdf/A_Review_of_Software_Development_Methodologies_in_Software_Engineering_ijariie12553.pdf?srsltid=AfmBOoqxnkGzHgtmRxKGMXvWLFcXqEcK--QoNB5q9AjMdbc6CjahS8EK. [Accessed 05 01 2025].