**Overall Architecture:**

1. **Presentation Layer:**
   * This layer is responsible for handling user interaction and presenting information to the user.
   * Components: User interface (UI), presentation logic.
   * Interactions: Receives user input, displays reservation options, collects user preferences, and presents booking confirmation.
2. **Application Layer:**
   * This layer contains the business logic of the reservation system.
   * Components: Reservation management, availability checking, pricing calculation, payment processing.
   * Interactions: Receives requests from the presentation layer, interacts with the data layer, and orchestrates the overall booking process.
3. **Domain Layer:**
   * This layer encapsulates the domain objects and business rules of the reservation system.
   * Components: Entities (e.g., Room, Reservation), services (e.g., ReservationService, PaymentService).
   * Interactions: Implements business rules, performs validation, and enforces constraints on data manipulation.
4. **Data Access Layer:**
   * This layer handles data storage and retrieval.
   * Components: Data access objects (DAOs), database management.
   * Interactions: Communicates with the database to perform CRUD (Create, Read, Update, Delete) operations on reservation-related data.

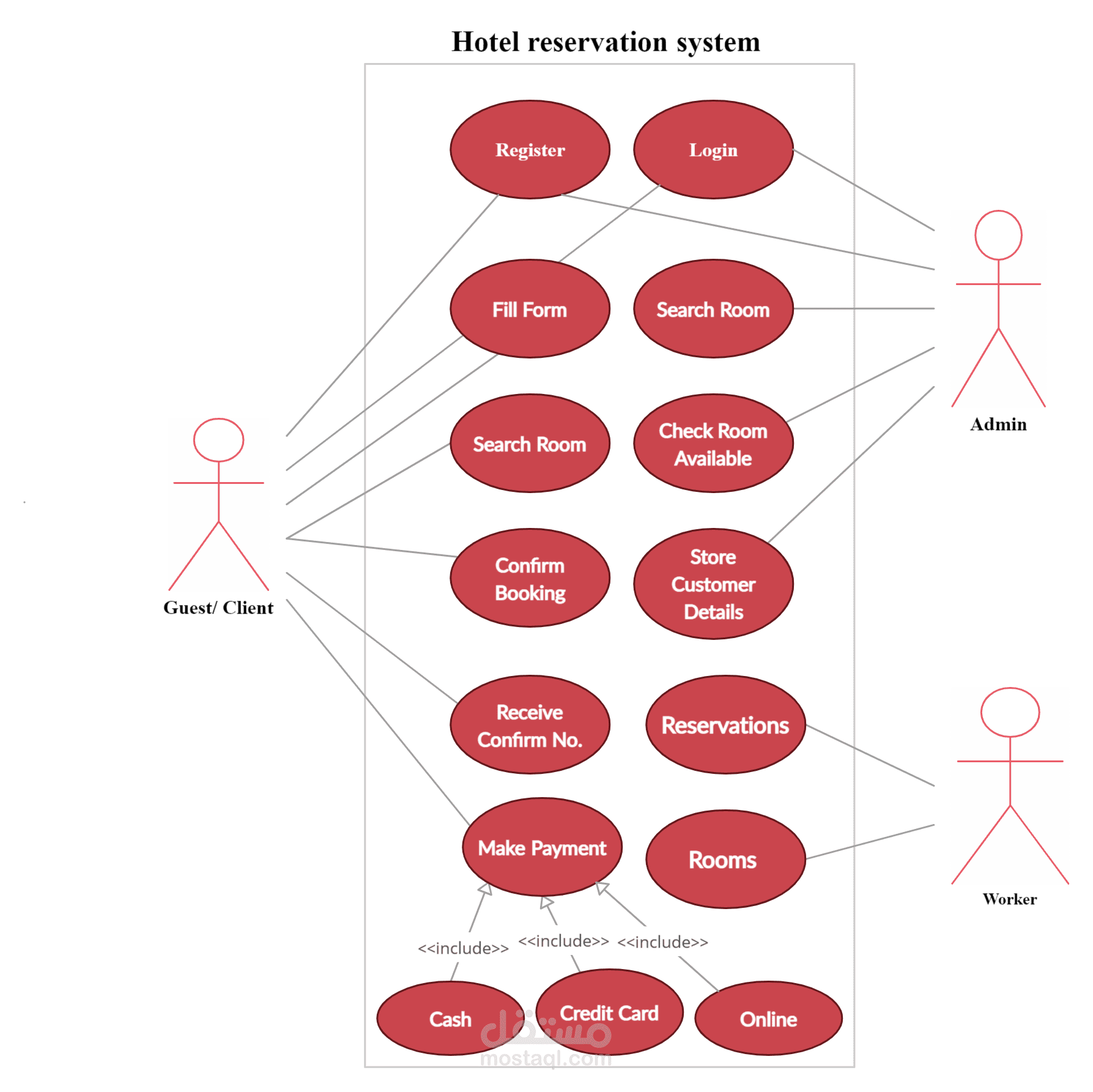
**Diagrams:**

**1. Component Diagram:**

**2. Sequence Diagram (Reservation Process):**

**3. Class Diagram (Simplified):**

These diagrams provide a visual representation of the software architecture, illustrating the components, modules, and interactions within the reservation system. Each layer has well-defined responsibilities, promoting maintainability, scalability, and modifiability of the software solution



1. **Programming Languages:**
   * **Java or C#:** These languages are commonly used for building enterprise-level applications due to their strong support for object-oriented programming, robustness, and scalability.
2. **Backend Framework:**
   * **Spring Boot (Java) or ASP.NET Core (C#):** These frameworks provide a comprehensive set of tools for building web applications, including RESTful APIs, dependency injection, and support for microservices architecture.
3. **Frontend Framework:**
   * **React.js or Angular:** These frontend frameworks are popular for building interactive user interfaces. They offer component-based architecture, state management, and efficient rendering, enhancing the user experience.
4. **Database:**
   * **MySQL, PostgreSQL, or MongoDB:** These databases are commonly used for storing reservation data. MySQL and PostgreSQL are relational databases suitable for structured data, while MongoDB is a NoSQL database suitable for handling unstructured or semi-structured data.
5. **ORM (Object-Relational Mapping):**
   * **Hibernate (Java) or Entity Framework (C#):** ORMs simplify database interactions by abstracting away the details of SQL queries. They map database tables to domain objects, making it easier to work with relational data in object-oriented programming.
6. **Web Server:**
   * **Apache Tomcat (Java) or IIS (Internet Information Services) (C#):** These web servers are used to deploy and host the reservation system, providing HTTP request handling and serving static content.
7. **API Documentation:**
   * **Swagger/OpenAPI:** Swagger/OpenAPI is used to document RESTful APIs. It provides a standardized way to describe API endpoints, request/response payloads, and authentication mechanisms, making it easier for developers to understand and use the API.
8. **Version Control:**
   * **Git:** Git is a distributed version control system used for tracking changes to source code. It facilitates collaboration among team members, versioning of codebase, and rollback to previous states if necessary.
9. **Build Automation:**
   * **Maven (Java) or MSBuild (C#):** These build automation tools are used to manage project dependencies, compile source code, run tests, and package the application into deployable artifacts.
10. **Containerization:**
    * **Docker:** Docker is used for containerizing the application, allowing it to run consistently across different environments (development, testing, production) and simplifying deployment.
11. **Continuous Integration/Continuous Deployment (CI/CD):**
    * **Jenkins, GitLab CI/CD, or Azure DevOps:** These CI/CD tools automate the process of building, testing, and deploying the application. They enable frequent integration of code changes, early detection of issues, and rapid delivery of updates.

**Hardware Requirements:**

* A computer with sufficient processing power and memory to run the development tools and database server.
* Internet connectivity for downloading dependencies and accessing resources.

**Software Requirements:**

1. **Operating System:**
   * Windows, macOS, or Linux (Ubuntu, CentOS, etc.)
2. **Development Tools:**
   * Java Development Kit (JDK) or .NET SDK depending on the chosen programming language.
   * Integrated Development Environment (IDE) such as IntelliJ IDEA or Eclipse for Java, or Visual Studio for C#.
   * Node.js and npm (Node Package Manager) for frontend development if using React.js or Angular.
   * Docker for containerization (optional but recommended).
3. **Database:**
   * MySQL, PostgreSQL, or MongoDB (Choose based on project requirements).
4. **Version Control:**
   * Git (optional but recommended).

**Installation and Setup:**

**1. Development Environment Setup:**

* Install JDK or .NET SDK based on the chosen programming language.
* Install your preferred IDE and configure it for development.
* Install Node.js and npm if frontend development is required.

**2. Database Setup:**

* Install and configure the preferred database server (MySQL, PostgreSQL, or MongoDB).
* Create a new database schema for the hotel reservation system.
* Configure database connection properties in the application configuration files.

**3. Backend Setup:**

* Clone or download the backend source code repository.
* Install project dependencies using Maven (Java) or NuGet (C#).
* Configure application properties such as database connection details, server port, etc.
* Run database migrations to create tables and seed initial data if necessary.
* Build and run the backend application locally using the IDE or command-line tools.

**4. Frontend Setup (if applicable):**

* Clone or download the frontend source code repository.
* Install project dependencies using npm.
* Configure API endpoint URLs in the frontend environment files.
* Build and run the frontend application locally using npm scripts.

**5. Docker Setup (optional but recommended):**

* Install Docker on your system.
* Dockerize the backend and database components of the application.
* Use Docker Compose to manage the containerized services and dependencies.

**Testing:**

* Perform integration tests to ensure that the backend and frontend components interact correctly.
* Test the application functionality thoroughly to identify and fix any bugs or issues.