**Integration**

**1. Overview of the Integration Process**

The integration of frontend, backend, and database involves three primary layers:

* **Frontend (Client-side):** Handles user interaction and sends requests to the backend.
* **Backend (Server-side):** Processes requests, interacts with the database, and returns responses to the frontend.
* **Database (Persistent Storage):** Stores user data, room details, booking information, and other application-specific data.

The integration ensures that when a user performs an action on the frontend (e.g., booking a room), the request is processed by the backend, which then interacts with the database to store or retrieve the necessary data.

**2. Frontend-Backend Integration**

**2.1 Communication via HTTP/HTTPS:**

* **RESTful APIs:** The backend should expose RESTful APIs that the frontend can interact with using HTTP/HTTPS requests. Common operations include:
  + **GET:** Retrieve data (e.g., get available rooms).
  + **POST:** Submit data (e.g., create a new booking).
  + **PUT/PATCH:** Update data (e.g., edit room details).
  + **DELETE:** Remove data (e.g., delete a booking).
* **AJAX/Fetch API:** On the frontend, you can use AJAX (Asynchronous JavaScript and XML) or the modern Fetch API to make asynchronous requests to the backend APIs. This allows the application to send data to and receive data from the server without needing to reload the page.

**2.2 Example Workflow:**

1. **User Action:** A user (Manager) clicks the "Book Room" button on the frontend.
2. **Frontend Request:** The frontend captures the room ID and user details, then sends a POST request to the backend API endpoint /api/bookRoom.
3. **Backend Processing:** The backend API receives the request, verifies the user’s role and available credits, and then interacts with the database to create the booking.
4. **Response to Frontend:** The backend sends a response (e.g., success or failure message) back to the frontend.
5. **Frontend Update:** The frontend processes the response and updates the user interface accordingly (e.g., showing a confirmation message or error alert).

**3. Backend-Database Integration**

**3.1 Database Connection Setup:**

* **JDBC/ORM Framework:** In a Java-based backend, you typically use JDBC (Java Database Connectivity) or an ORM (Object-Relational Mapping) framework like Hibernate to connect to the database.
* **Configuration:** Database connection details (URL, username, password) are configured in the backend application, usually in a configuration file (e.g., application.properties for Spring Boot).

**3.2 CRUD Operations:**

* **Create (Insert):** Inserting new records (e.g., creating a new booking).
* **Read (Select):** Retrieving data (e.g., fetching room availability).
* **Update:** Modifying existing records (e.g., updating room details).
* **Delete:** Removing records (e.g., canceling a booking).

**3.3 Example Workflow:**

1. **API Request:** The backend receives a request to book a room.
2. **Database Query:** The backend queries the database to check room availability and user credits.
3. **Data Update:** If the room is available and the user has sufficient credits, the backend inserts a new booking record into the database.
4. **Return Response:** The backend then returns the result of the operation (e.g., booking success or failure) to the frontend.

**4. Full Integration Workflow**

**4.1 User Registration Example:**

1. **User Action (Frontend):** A new user fills out the registration form and submits it.
2. **API Call (Frontend to Backend):** The frontend sends a POST request to /api/registerUser with the user details.
3. **Backend Processing:**
   * The backend validates the input data.
   * The backend checks if the username already exists in the database.
   * If valid, the backend saves the new user details into the database.
4. **Database Interaction:**
   * An INSERT SQL command is executed to store the user details in the Users table.
5. **Response Handling (Backend to Frontend):**
   * The backend returns a success message if registration is successful, or an error message if there was an issue (e.g., username already taken).
6. **Frontend Update:**
   * The frontend displays a confirmation message to the user or asks them to try again with a different username.

**4.2 Booking Room Example:**

1. **Search Rooms (Frontend):** The Manager searches for rooms using specific criteria.
2. **API Call (Frontend to Backend):** The frontend sends a GET request to /api/searchRooms with the search parameters.
3. **Backend Query:**
   * The backend queries the database for rooms that match the criteria (e.g., capacity, available dates).
4. **Database Interaction:**
   * The database returns a list of rooms that match the search criteria.
5. **Response Handling (Backend to Frontend):**
   * The backend returns the list of available rooms to the frontend.
6. **Booking Process:**
   * The Manager selects a room and initiates the booking process.
   * The frontend sends a POST request to /api/bookRoom with the room ID and Manager details.
   * The backend checks availability and user credits, then updates the database to create the booking.
   * The backend returns a booking confirmation to the frontend.
   * The frontend updates the UI to reflect the booking confirmation.

**5. Security Considerations**

**5.1 Authentication and Authorization:**

* **JWT (JSON Web Tokens):** Use JWT for secure authentication between frontend and backend. Upon successful login, the backend issues a JWT, which the frontend includes in the headers of subsequent requests.
* **Role-Based Access Control (RBAC):** Ensure that different API endpoints are protected based on user roles (Admin, Manager, Member).

**5.2 Secure Data Transmission:**

* **HTTPS:** Always use HTTPS for secure communication between frontend and backend, preventing data interception.
* **Input Validation:** Validate all inputs on both frontend and backend to prevent injection attacks.

**6. Deployment and Environment Setup**

**6.1 Development Environment:**

* **Local Servers:** Use local servers like Apache Tomcat for Java (backend) and local databases like MySQL or PostgreSQL during development.
* **Mock APIs:** Use mock APIs to simulate backend responses during frontend development before full integration.

**6.2 Production Environment:**

* **Cloud Services:** Deploy the backend and database on cloud platforms (e.g., AWS, Azure). Use managed database services like Amazon RDS for scalability.
* **CI/CD Pipeline:** Set up a CI/CD pipeline for automated testing, building, and deployment of the application.

**6.3 Environment Variables:**

* **Configuration:** Use environment variables to manage sensitive information (e.g., database credentials, API keys) across different environments (development, staging, production).