Gym Management System

1. Project Overview

The Gym Management System is a **microservices-based application** designed to manage all gym operations efficiently.

Key Features

- User authentication and authorization with JWT
- Trainer and member management
- Class bookings and schedules
- Inventory management for gym equipment
- Attendance tracking for members and trainers

Microservices

Service	Description
API Gateway	Entry point for all requests, routes them to appropriate microservices.
Auth Service	Handles user registration, login, JWT generation, and validation.
Member Service	Manages members, memberships, and plans.
Attendance Service	Tracks member attendance (check-in/check-out).
Trainer Service	Manages trainers and their assignments to members.
Class & Booking Service	Handles gym classes, sessions, and bookings.
Inventory Service	Manages gym assets, maintenance logs, and inventory.

---

2. Architecture

- **API Gateway** routes requests to the microservices.
- Each service follows **layered architecture**:
  - **Controller** – REST API endpoints
  - **Service / ServiceImpl** – Business logic
  - **Repository** – Database access
  - **Entity** – Database mapping
  - **DTO** – Request/response objects
  - **Exception Handling** – Global exceptions using `@ControllerAdvice`
  - **Security** – JWT authentication for protected endpoints
- **Inter-service communication**: Services use **WebClient** .
- **"Microservices trust each other".**

---

3. Prerequisites

- **Java 17**
- **Eclipse IDE**
- **Maven**
- **MySQL**
- **Postman / Browser** for testing APIs

4. Database Setup

1. Create databases for each microservice:

```
CREATE DATABASE gym_auth;
CREATE DATABASE gym_member_db;
CREATE DATABASE gym_attendance_db;
CREATE DATABASE gym_trainer_db;
CREATE DATABASE gym_class_db;
CREATE DATABASE gym_inventory_db;
```

2. Set environment variables:

```
export DB_HOST=127.0.0.1
export DB_USERNAME=your_db_user
export DB_PASSWORD='your_db_password'
```

3. Update `application.properties` in each service:

```
spring.datasource.url=jdbc:mysql://${DB_HOST}:3306/auth_service_db
spring.datasource.username=${DB_USERNAME}
spring.datasource.password=${DB_PASSWORD}
spring.jpa.hibernate.ddl-auto=update
```

---

5. Running the Microservices Locally (Eclipse / Java 17)

Step-by-Step

1. Open the project in **Eclipse**.
2. Set **Java version to 17** (Project → Properties → Java Compiler → 17).
3. **Run API Gateway** first.
4. **Run Auth Service** next.
5. Run remaining services in the recommended order:
   - Member Service
   - Attendance Service
   - Trainer Service
   - Class & Booking Service
   - Inventory Service

⚠ **Important:** API Gateway must be running first to route requests correctly.

---

6. API Testing

- **Swagger UI** for each service:

```
http://localhost:{port}/swagger-ui.html
```

Default Ports

Service	Port
API Gateway	No
Auth Service	No
Member Service	8082
Attendance Service	8083
Trainer Service	8084
Class & Booking Service	8085
Inventory Service	8086

Authentication

1. Register via `/api/public/register`
2. Login via `/api/public/login` → get JWT
3. Use JWT in `Authorization: Bearer <token>` for all protected APIs

---

7. Step-by-Step Microservice Setup (Reference from PDF)

Auth Service

- **Setup**: Create project, configure DB, add dependencies (Web, Security, JWT, JPA, Validation)
- **Database**: Tables: users, roles, permissions, user_roles, role_permissions
- **Service**: Methods: register(), login(), validateToken(), getUserById()
- **Security**: JwtProvider, AuthFilter, SecurityConfig
- **Endpoints**:
  - POST `/auth/register`
  - POST `/auth/login`
  - GET `/auth/validate-token`
  - GET `/auth/user/{id}`

Member Service

- **Depends on**: Auth Service (JWT validation)
- **Database**: members, memberships, plans
- **Service Methods**: createMember(), updateMember(), getMember(), getAllMembers()

- **Endpoints**:
  - POST `/members`
  - GET `/members/{id}`
  - PUT `/members/{id}`
  - GET `/members`

Attendance Service

- **Depends on**: Member Service
- **Database**: attendance_logs (id, memberId, checkInTime, checkOutTime)
- **Service Methods**: checkIn(memberId), checkOut(memberId), getAttendanceByMember()
- **Endpoints**:
  - POST `/attendance/check-in`
  - POST `/attendance/check-out`
  - GET `/attendance/member/{memberId}`

Trainer Service

- **Depends on**: Member Service
- **Database**: trainers, staff, trainer_assignments
- **Service Methods**: createTrainer(), assignTrainerToMember(), getTrainerByMember()
- **Endpoints**:
  - POST `/trainers`
  - GET `/trainers`
  - POST `/trainers/assign`
  - GET `/trainers/member/{memberId}`

Class & Booking Service

- **Depends on**: Member + Trainer Service
- **Database**: gym_classes, class_sessions, bookings
- **Service Methods**: createClass(), createSession(), bookSession(), getBookingsByMember()
- **Endpoints**:
  - POST `/classes`
  - POST `/classes/{sessionId}/book`
  - GET `/classes/member/{memberId}`

Inventory Service

- **Least dependency**
- **Database**: assets, maintenance_logs
- **Service Methods**: createAsset(), updateStatus(), logMaintenance(), getMaintenanceHistory()
- **Endpoints**:
  - POST `/assets`
  - GET `/assets`
  - PUT `/assets/{id}/status`

    - GET `/assets/maintenance`

---

8. Sample API Endpoints

Service	Endpoint	Method	Description	Auth Required
Auth	`/auth/register`	POST	Register user	No
Auth	`/auth/login`	POST	Login, get JWT	No
Member	`/members`	POST	Create member	Yes
Member	`/members/{id}`	GET	Get member by ID	Yes
Attendance	`/attendance/check-in`	POST	Member check-in	Yes
Attendance	`/attendance/check-out`	POST	Member check-out	Yes
Trainer	`/trainers`	GET	Get all trainers	Yes
Trainer	`/trainers/assign`	POST	Assign trainer to member	Yes
Class Booking	`/classes`	POST	Create a class	Yes
Class Booking	`/classes/{sessionId}/book`	POST	Book class session	Yes
Inventory	`/assets`	POST	Add new asset	Yes
Inventory	`/assets/{id}/status`	PUT	Update asset status	Yes

Expand with Swagger for full endpoint documentation.

---

9. Integration Testing

1. Start all services (API Gateway first)
2. Register a user → login → get JWT
3. Create a member
4. Assign trainer to member
5. Book class for member
6. Check-in / check-out attendance
7. Create / update gym asset

If all flows work → ✅ System complete

---

10. Troubleshooting

- **Database connection issues**: Check DB_HOST, DB_USERNAME, DB_PASSWORD
- **Port conflicts**: Ensure each service uses its designated port
- **JWT issues**: Ensure Auth Service and API Gateway are running first
- **Inter-service errors**: Ensure dependent services (Member/Trainer) are running before calling endpoints

FRONTEND

1. **Install Node.js**

```
node -v
npm -v
```

2. **Create Vite React App**

```
npm create vite@latest my-react-app
```

(Choose React + JS/TS)

3. **Go to Project Folder**

```
cd my-react-app
```

4. **Install Dependencies**

```
npm install
```

5. **Run Development Server**

```
npm run dev
```

(Open http://localhost:3000)

6. **Build for Production**

```
npm run build
```

7. **Preview Production Build**

```
npm run preview
```