Perfect. Now we'll turn your clean architecture into an **execution roadmap**.

We'll go service by service and define:

- ✅ Step-by-step development order
- 🧠 What to implement first
- 🔗 When to connect to other services
- 🧪 What to test
- 🚀 When it's "production-ready"

We'll build in the **correct dependency order**.

🔵0️⃣ Build Order (Important)

Because services depend on Auth + Member, the correct build sequence is:

1. **Auth Service**
2. **Member Service**
3. **Attendance Service**
4. **Trainer Service**
5. **Class & Booking Service**
6. **Inventory Service**

1️⃣ AUTH SERVICE — Step-by-Step

This must be fully working before other services.

STEP 1 — Project Setup

- Create project
- Configure database
- Add dependencies:
    - Web
    - Security
    - JWT
    - JPA
    - Validation

STEP 2 — Database Design

Create tables:

- users
- roles
- permissions
- user_roles (many-to-many)
- role_permissions

STEP 3 — Model Layer

Create:

- `User`
- `Role`

- `Permission`

Add relationships:

- User ↔ Roles
- Role ↔ Permissions

STEP 4 — Repository Layer

- `UserRepository`
- `RoleRepository`
- `PermissionRepository`

STEP 5 — Service Layer

Implement:

`AuthService`

Methods:

- register()
- login()
- validateToken()
- getUserById()

Logic:

- Password encryption
- Role assignment
- JWT generation

STEP 6 — Security Layer

Create:

- `JwtProvider`
- `AuthFilter`
- `SecurityConfig`

Flow:

1. Extract token
2. Validate token
3. Load user details
4. Set authentication context

STEP 7 — Controller Layer

Create `AuthController`

Endpoints:

- POST /auth/register
- POST /auth/login
- GET /auth/validate-token
- GET /auth/user/{id}

STEP 8 — Testing

Test:

- Register
- Login → get JWT
- Pass JWT to protected endpoint

If JWT works → ✅ Auth Service complete

2️⃣ MEMBER SERVICE — Step-by-Step

Depends on Auth Service (JWT validation)

STEP 1 — Project Setup

Add:

- Web
- JPA
- Validation
- Feign or RestTemplate for AuthClient

STEP 2 — Database Design

Tables:

- members
- memberships
- plans

Relationships:

- Member → Membership
- Membership → Plan

STEP 3 — Model Layer

Create:

- Member

- Membership
- Plan

STEP 4 — Repository Layer

- MemberRepository
- MembershipRepository
- PlanRepository

STEP 5 — AuthClient

Create HTTP client:

Call:
GET /auth/validate-token

Before executing any endpoint.

STEP 6 — Service Layer

Create `MemberService`

Methods:

- createMember()
- updateMember()
- getMember()
- getAllMembers()

Add validation:

- Only ADMIN or STAFF can create/update

STEP 7 — Controller

Create `MemberController`

Endpoints:

- POST /members
- GET /members/{id}
- PUT /members/{id}
- GET /members

STEP 8 — Testing

Test:

- JWT required
- Member creation
- Membership assignment

If working → ✅ Member Service complete

3️⃣ ATTENDANCE SERVICE — Step-by-Step

Depends on Member Service

STEP 1 — Setup

Add:

- Web
- JPA
- HTTP client (MemberClient)

STEP 2 — Database

Table:

- attendance_logs

Fields:

- id
- memberId
- checkInTime
- checkOutTime

STEP 3 — Model

Create:

- AttendanceLog

STEP 4 — Repository

- AttendanceRepository

STEP 5 — MemberClient

Call:
GET /members/{id}

Before check-in.

STEP 6 — Service Layer

Create:

Methods:

- checkIn(memberId)
- checkOut(memberId)

- getAttendanceByMember()

Logic:

- Prevent double check-in
- Check membership active (optional)

STEP 7 — Controller

Endpoints:

- POST /attendance/check-in
- POST /attendance/check-out
- GET /attendance/member/{memberId}

STEP 8 — Testing

Test:

- Check-in
- Check-out
- Invalid member
- Already checked-in case

If working → ✅ Attendance Service complete

4️⃣ TRAINER SERVICE — Step-by-Step

Depends on Member Service

STEP 1 — Setup

Add:

- Web
- JPA
- MemberClient

STEP 2 — Database

Tables:

- trainers
- staff
- trainer_assignments

STEP 3 — Models

Create:

- Trainer
- Staff
- TrainerAssignment

STEP 4 — Repository

- TrainerRepository
- StaffRepository
- AssignmentRepository

STEP 5 — Service Layer

Methods:

- createTrainer()

- assignTrainerToMember()
- getTrainerByMember()

Validate:

- Member exists (MemberClient)

STEP 6 — Controller

Endpoints:

- POST /trainers
- GET /trainers
- POST /trainers/assign
- GET /trainers/member/{memberId}

STEP 7 — Testing

Test:

- Assign trainer
- Duplicate assignment
- Invalid member

If working → ✅ Trainer Service complete

5️⃣ CLASS & BOOKING SERVICE — Step-by-Step

Depends on Member + Trainer

STEP 1 — Setup

Add:

- Web
- JPA
- MemberClient
- TrainerClient

STEP 2 — Database

Tables:

- gym_classes
- class_sessions
- bookings

STEP 3 — Models

Create:

- GymClass
- ClassSession
- Booking

STEP 4 — Repositories

- ClassRepository
- SessionRepository
- BookingRepository

STEP 5 — Service Layer

Methods:

- createClass()
- createSession()
- bookSession()
- getBookingsByMember()

Logic:

- Verify member
- Verify trainer
- Check capacity
- Prevent double booking

STEP 6 — Controller

Endpoints:

- POST /classes
- POST /classes/{sessionId}/book
- GET /classes/member/{memberId}

STEP 7 — Testing

Test:

- Capacity limit
- Double booking
- Invalid trainer

If working → ✅ Class Service complete

6️⃣ INVENTORY SERVICE — Step-by-Step

Least dependency

STEP 1 — Setup

Add:

- Web
- JPA
- AuthClient

STEP 2 — Database

Tables:

- assets
- maintenance_logs

STEP 3 — Models

Create:

- Asset
- MaintenanceLog

STEP 4 — Repository

- AssetRepository
- MaintenanceRepository

STEP 5 — Service Layer

Methods:

- createAsset()
- updateStatus()
- logMaintenance()
- getMaintenanceHistory()

Restrict:

- ADMIN / STAFF only

STEP 6 — Controller

Endpoints:

- POST /assets
- GET /assets
- PUT /assets/{id}/status
- GET /assets/maintenance

STEP 7 — Testing

Test:

- Status change
- Maintenance logging
- Unauthorized access

If working → ✅ Inventory Service complete

🔥 **Final Step — Integration Testing**

After all services:

1. Start all services

2. Register user
3. Login → get JWT
4. Create member
5. Assign trainer
6. Book class
7. Check-in attendance
8. Create asset

If all flows work → 🎉 System complete

🧠 **Final Advice (Professional Level)**

After completion, optionally add:

- API Gateway
- Centralized logging
- Docker
- Service discovery
- Config server
- Circuit breaker
- OpenAPI documentation