Tài liệu này thiết kế **MuTraPro** đúng theo đề: **Server: Spring Boot**, **CSDL: SQL Server**, **Client: ReactJS**. Bạn sẽ chạy được hệ thống microservices: `discovery-server (Eureka)`, `api-gateway (Spring Cloud Gateway + JWT)`, `auth-service`, `customer-service` (orders/requests), `transcription-service`, `arrangement-service`, `recording-service`, `scheduling-service`, `notification-service`, `payment-service (mock)`, `file-service` (MinIO/S3), cùng **React web**.

---

0) Mục tiêu & phạm vi

- Chạy được **end-to-end**: Khách hàng đăng ký/đăng nhập → tạo **Yêu cầu dịch vụ** (upload audio/video) → Điều phối viên phân công → Chuyên gia ký âm/ phối khí xử lý → (tuỳ chọn) đặt lịch phòng thu & thu âm → Khách hàng duyệt & thanh toán (mock) → nhận sản phẩm cuối.
- Phù hợp **3.2** (UML + tài liệu SRS/Thiết kế/Testing/Installation) – checklist ở cuối.

---

1) Kiến trúc tổng thể

```
mutrapro/
  docker-compose.yml
  .env
  discovery-server/            # Eureka
  api-gateway/                 # Spring Cloud Gateway + JWT filter
  auth-service/                # Users + Roles + JWT
  customer-service/            # Orders/Requests + trạng thái
  transcription-service/       # Nhiệm vụ ký âm (deliverable PDF/MusicXML)
  arrangement-service/         # Nhiệm vụ phối khí (mixdown/stems)
  recording-service/           # Tác vụ thu âm nghệ sĩ (upload takes)
  scheduling-service/          # Lịch phòng thu (rooms/equipments/bookings)
  notification-service/        # Email/WebSocket thông báo
  payment-service/             # Thanh toán (mock/sandbox)
  file-service/                # Presigned URL upload/download tới MinIO
  web-frontend/                # React (Vite) UI đa vai trò
```

Nguyên tắc - Mỗi service là **Spring Boot app** độc lập, **mỗi DB riêng** trong SQL Server. - **React** chỉ gọi **API Gateway** (`http://localhost:8080/api/v1/**`). - **JWT** xác thực ở gateway → chèn header `X-User` / `X-Role` cho downstream. - **Optional**: RabbitMQ để phát sự kiện (order.created, task.updated) phục vụ notification/ghi log.

---

2) Chuẩn bị môi trường

- JDK 21, Maven 3.9+, Node.js 20 LTS + npm, Docker Desktop.
- Postman/Insomnia.
- IDE: IntelliJ hoặc VS Code.

Kiểm tra:

```
java -version
mvn -v
node -v
docker -v
```

---

3) Hạ tầng Docker: SQL Server + MinIO (+ RabbitMQ tuỳ chọn)

`docker-compose.yml` **(root)**

```
version: '3.9'
services:
  sqlserver:
    image: mcr.microsoft.com/mssql/server:2022-latest
    environment:
      - ACCEPT_EULA=Y
      - SA_PASSWORD=${SA_PASSWORD:-YourStrong!Passw0rd}
      - MSSQL_PID=Developer
    ports: ["1433:1433"]
    healthcheck:
      test: ["CMD-SHELL", "/opt/mssql-tools18/bin/sqlcmd -S localhost -U sa -
P $${SA_PASSWORD} -C -Q 'SELECT 1' || exit 1"]
      interval: 10s
      timeout: 5s
      retries: 10
  minio:
    image: minio/minio:latest
    environment:
      MINIO_ROOT_USER: ${MINIO_ROOT_USER:-minio}
      MINIO_ROOT_PASSWORD: ${MINIO_ROOT_PASSWORD:-minio123}
    command: server /data --console-address ":9001"
    ports: ["9000:9000", "9001:9001"]
  rabbitmq:
    image: rabbitmq:3-management
    ports: ["5672:5672", "15672:15672"]
```

Chạy:

```
docker compose up -d
# tạo DB cho từng service
DBS="auth_db customer_db transcription_db arrangement_db recording_db
scheduling_db notification_db payment_db file_db"
for d in $DBS; do docker exec -it $(docker ps -qf name=sqlserver) /opt/mssql-
```

```
tools18/bin/sqlcmd -S localhost -U sa -P YourStrong!Passw0rd -C -Q "IF
DB_ID('$d') IS NULL CREATE DATABASE $d;"; done
```

MinIO: mở `http://localhost:9001` đăng nhập `minio/minio123` → tạo buckets: `mutrapro-uploads`, `mutrapro-deliverables`.

---

4) Khởi tạo các Spring Boot service

Tạo dự án từ Spring Initializr: - **Dependencies chung** (tuỳ service): `Spring Web`, `Spring Data JPA`, `Validation`, `Spring Security` (auth/gateway), `Spring Cloud Netflix Eureka Client/Server`, `Spring Cloud Gateway` (gateway), `SQL Server Driver`, `Lombok`, `Actuator`. - **RabbitMQ** (nếu dùng): `spring-boot-starter-amqp`.

BOM Spring Cloud (pom.xml)

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>2024.0.0</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

---

5) Eureka – discovery-server

Main

```
@EnableEurekaServer
@SpringBootApplication
public class DiscoveryServerApplication { public static void main(String[]
args){ SpringApplication.run(DiscoveryServerApplication.class,args);} }
```

`application.yml`

```
server: { port: 8761 }
spring: { application: { name: discovery-server }, main: { web-application-
type: reactive } }
eureka:
  client:
```

```
    register-with-eureka: false
    fetch-registry: false
```

---

6) API Gateway (Spring Cloud Gateway + JWT)

`application.yml`

```
server: { port: 8080 }
spring:
  application: { name: api-gateway }
  cloud:
    gateway:
      default-filters: [ RemoveRequestHeader=Cookie ]
      routes:
        - id: auth-service
          uri: lb://auth-service
          predicates: [ Path=/api/v1/auth/** ]
        - id: customer-service
          uri: lb://customer-service
          predicates: [ Path=/api/v1/orders/** ]
          filters: [ JwtAuth ]
        - id: transcription-service
          uri: lb://transcription-service
          predicates: [ Path=/api/v1/transcription/** ]
          filters: [ JwtAuth ]
        - id: arrangement-service
          uri: lb://arrangement-service
          predicates: [ Path=/api/v1/arrangement/** ]
          filters: [ JwtAuth ]
        - id: recording-service
          uri: lb://recording-service
          predicates: [ Path=/api/v1/recordings/** ]
          filters: [ JwtAuth ]
        - id: scheduling-service
          uri: lb://scheduling-service
          predicates: [ Path=/api/v1/scheduling/** ]
          filters: [ JwtAuth ]
        - id: payment-service
          uri: lb://payment-service
          predicates: [ Path=/api/v1/payments/** ]
          filters: [ JwtAuth ]
        - id: file-service
          uri: lb://file-service
          predicates: [ Path=/api/v1/files/** ]
          filters: [ JwtAuth ]

jwt:
```

```
  secret: ChangeMeToLongRandomSecretKey_!@#123
  issuer: mutrapro
  expiration: 3600

eureka.client.service-url.defaultZone: http://localhost:8761/eureka
```

Filter JWT (JwtAuthGatewayFilterFactory.java) – xác thực token, đính `X-User`, `X-Role`:

```
@Component
public class JwtAuthGatewayFilterFactory extends
AbstractGatewayFilterFactory<JwtAuthGatewayFilterFactory.Config> {
  private final JwtService jwt; public
JwtAuthGatewayFilterFactory(JwtService jwt){ super(Config.class); this.jwt =
jwt; }
  public static class Config {}
  @Override public GatewayFilter apply(Config config) { return (exchange,
chain) -> {
    String path = exchange.getRequest().getPath().toString();
    if (path.startsWith("/api/v1/auth/")) return chain.filter(exchange);
    String auth =
exchange.getRequest().getHeaders().getFirst(HttpHeaders.AUTHORIZATION);
    if (auth==null || !auth.startsWith("Bearer ")) {
exchange.getResponse().setStatusCode(HttpStatus.UNAUTHORIZED); return
exchange.getResponse().setComplete(); }
    String token = auth.substring(7);
    if (!jwt.validate(token)) {
exchange.getResponse().setStatusCode(HttpStatus.UNAUTHORIZED); return
exchange.getResponse().setComplete(); }
    String user = jwt.getUsername(token); String role = jwt.getRole(token);
    return chain.filter(exchange.mutate().request(r-> r.headers(h->{
h.add("X-User", user); h.add("X-Role", role);})).build());
  }; }
}
```

`JwtService` dùng `jjwt` (HS256) như hướng dẫn trước.

---

7) Auth-service (Users/Roles/JWT)

Entity

```
@Entity @Table(name="users")
public class AppUser { @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
Long id; @Column(unique=true) String email; String password; String role;
Instant createdAt; }
```

Repo + Service + Controller: đăng ký, đăng nhập, trả `{ token }` với claims `{sub, email, role}`.
`application.yml` (kết nối `auth_db`).

8) File-service (MinIO/S3 presigned URL)

- Endpoint: `POST /api/v1/files/presign` với body `{bucket, key, contentType, method}` trả presigned URL.
- Lưu metadata file (owner, orderId, type: SOURCE|SCORE|MIXDOWN|FINAL...).
  Kết nối MinIO: dùng SDK S3 (AWS Java v2) với `endpointOverride`, `credentials` từ `.env`.

9) Customer-service (Orders/Requests)

Bảng `orders`

```
- id (PK), customerEmail, type: TRANSCRIBE|ARRANGE|RECORD|FULL
- sourceKey (S3 key), scoreKey (optional)
- status: NEW|ASSIGNED|TRANSCRIBED|ARRANGED|RECORD_SCHEDULED|RECORDED|READY|
DELIVERED|CANCELLED
- price, currency, paymentStatus: UNPAID|PAID
- notes, deadline, createdAt, updatedAt
```

API - `POST /api/v1/orders` (CUSTOMER): tạo order + đính `sourceKey`.
- `GET /api/v1/orders?mine=true` (CUSTOMER) hoặc filter theo trạng thái (COORDINATOR).
- `PATCH /api/v1/orders/{id}/status` (COORDINATOR).
- (Optional) phát sự kiện RabbitMQ `order.created`.

10) Transcription-service

Bảng: `transcription_tasks(id, orderId, transcriberEmail, status, deliverableKey, comments)`. **API** - `POST /api/v1/transcription/tasks/assign` (COORDINATOR) → gán người ký âm.
- `GET /api/v1/transcription/tasks?assignee=me` (TRANSCRIBER).
- `POST /api/v1/transcription/tasks/{id}/submit` → cập nhật `deliverableKey` (PDF/MusicXML trên MinIO) & trạng thái `TRANSCRIBED`.

11) Arrangement-service

Bảng: `arrangement_tasks(id, orderId, arrangerEmail, status, stemsKey, scoreKey, mixdownKey)`. **API**: gán/submit như trên → cập nhật order sang `ARRANGED`.

12) Scheduling-service & Recording-service

Scheduling-service - Bảng: `rooms(id,name,location)`, `equipments`, `bookings(id,roomId,start,end,engineerEmail,orderId,status)`. - API: `POST /api/v1/scheduling/bookings`, `GET /...` lọc theo phòng/ngày.
Recording-service - Bảng: `recording_tasks(id, orderId, artistEmail, bookingId, status, takeKeys[])`.
- API: upload takes (.wav) qua `file-service`, chọn best take → `RECORDED`.

---

13) Notification-service

- Nhận sự kiện (AMQP) hoặc subscribe DB changes → gửi **Email** (JavaMailSender) và **WebSocket (STOMP)** cho web.
- Bảng `notifications(userEmail, message, read, createdAt)`.

---

14) Payment-service (mock)

- `POST /api/v1/payments/checkout` nhận `orderId, amount` → trả URL sandbox (giả).
- `GET /api/v1/payments/callback?orderId=...&status=success` → cập nhật `UNPAID→PAID`.

---

15) Cấu hình kết nối SQL Server (mẫu cho mọi service)

```
spring:
  datasource:
    url: jdbc:sqlserver://localhost:
1433;databaseName=customer_db;encrypt=true;trustServerCertificate=true
    username: sa
    password: ${SA_PASSWORD:YourStrong!Passw0rd}
  jpa:
    hibernate:
      ddl-auto: update   # DEV; PROD dùng Flyway
    show-sql: true
    properties:
      hibernate.dialect: org.hibernate.dialect.SQLServerDialect

eureka.client.service-url.defaultZone: http://localhost:8761/eureka
```

---

16) Chạy local theo thứ tự

1) `docker compose up -d` (SQL Server, MinIO, RabbitMQ).
2) `discovery-server` (8761).
3) `auth-service` (8081), `file-service` (8084), `customer-service` (8082), `transcription-service` (8085), `arrangement-service` (8086), `scheduling-service` (8087), `recording-service` (8088), `notification-service` (8089), `payment-service` (8090).
4) `api-gateway` (8080).

Test nhanh 1. `POST /api/v1/auth/register` → `POST /api/v1/auth/login` lấy `{token}`.
2. `POST /api/v1/files/presign` → upload audio đến MinIO.
3. `POST /api/v1/orders` (Authorization: Bearer <token>) tạo order.
4. (COORDINATOR) gán task ký âm; (TRANSCRIBER) submit PDF/MusicXML.
5. (ARRANGER) submit mixdown; (STUDIO_ADMIN) tạo booking; (ARTIST) upload takes.
6. (CUSTOMER) duyệt & thanh toán mock → tải sản phẩm cuối.

---

17) React Web (Vite + TS)

```
cd web-frontend
npm create vite@latest . -- --template react-ts
npm i axios react-router-dom zustand
```

`src/api.ts`

```
import axios from 'axios';
const api = axios.create({ baseURL: 'http://localhost:8080/api/v1' });
api.interceptors.request.use(cfg => { const t =
localStorage.getItem('token'); if (t) cfg.headers.Authorization = `Bearer $
{t}`; return cfg;});
export default api;
```

Màn hình gợi ý - **Customer**: Tạo yêu cầu (upload → presign), Theo dõi đơn, Duyệt & Thanh toán, Tải kết quả.
- **Coordinator**: Hộp yêu cầu mới, Phân công, Theo dõi tiến độ.
- **Transcriber/Arranger/Artist**: Nhiệm vụ của tôi, Gửi sản phẩm.
- **Studio Admin**: Lịch phòng thu (calendar view).
- **Notifications**: Realtime với WebSocket.

---

18) Bảo mật & phân quyền

- JWT (HS256), claim `sub,email,role`.
- Gateway kiểm, downstream tin `X-User` + `X-Role`.
- **Method security** (@PreAuthorize("hasRole('COORDINATOR')")) ở service.

- Audit log: lưu thao tác (ai, khi nào, đối tượng gì).

---

19) Hiệu năng & độ tin cậy

- Upload qua **presigned URL** (không qua service) → giảm tải.
- **Async**: sự kiện RabbitMQ cho notification/long task.
- **Resilience**: retry, circuit breaker (Resilience4j) cho call giữa services nếu cần.
- **Observability**: Actuator + Prometheus/Grafana (tuỳ chọn).

---

20) Migrations & dữ liệu

- DEV có thể dùng `ddl-auto: update`.
- PROD dùng **Flyway**: tạo `V1__init.sql` cho mỗi service.
- Seed data (vai trò, phòng thu mẫu) qua `data.sql` hoặc runner.

---

21) Testing

- **Unit**: service/mapper.
- **Integration**: controller + JPA (Testcontainers cho SQL Server, hoặc profile H2 cho nhanh).
- **Contract**: Gateway ↔ services (OpenAPI).
- **E2E**: kịch bản từ tạo order đến giao hàng/ thanh toán.

---

22) Tài liệu đáp ứng mục 3.2

- **User Requirements**: mô tả vai trò & user stories (Customer/Coordinator/Transcriber/Arranger/Artist/StudioAdmin).
- **SRS**: use case, luồng nghiệp vụ, đặc tả API (OpenAPI), NFR (usability, reliability, performance, scalability).
- **Architecture Design**: C4 (Context/Container/Component), sequence diagrams cho các luồng chính.
- **Detailed Design**: ERD (mỗi service), bảng trạng thái đơn hàng, hợp đồng sự kiện (nếu dùng RabbitMQ).
- **System Implementation**: hướng dẫn build/run, cấu trúc mã.
- **Testing Documentation**: test plan, test cases, kết quả.
- **Installation Guide**: Docker Compose (dev), ghi chú deploy prod.
- **Source & Artifacts**: mã nguồn + gói jar/docker images + `.env.example`.

---

23) Docker hóa mỗi service (mẫu Dockerfile)

```
FROM maven:3.9.8-eclipse-temurin-21 AS build
WORKDIR /app
COPY pom.xml .
RUN mvn -q -DskipTests dependency:go-offline
COPY src ./src
RUN mvn -q -DskipTests package

FROM eclipse-temurin:21-jre
WORKDIR /app
COPY --from=build /app/target/*.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java","-jar","/app/app.jar"]
```

Cập nhật `docker-compose.yml` thêm các service (ports map cho gateway và discovery; các service nội bộ không cần public port).

---

24) Checklist chạy end-to-end

1. `docker compose up -d` hạ tầng → tạo DB & buckets.
2. Chạy `discovery-server`.
3. Chạy **auth**, **file**, **customer**, **transcription**, **arrangement**, **scheduling**, **recording**, **notification**, **payment**.
4. Chạy **api-gateway**.
5. Đăng ký/đăng nhập → presign upload → tạo order.
6. Phân công & submit deliverables theo vai trò.
7. Thanh toán mock → giao sản phẩm (link tải MinIO).
8. Kiểm tra notifications, logs.

---

25) Hướng mở nâng cao

- Keycloak thay auth-service tự viết; 2FA.
- Workflow/BPMN (Camunda/Flowable) cho quy trình phức tạp.
- Chuyển storage sang S3 thật (AWS) + CDN.
- K8s + Ingress + autoscaling.
- Tích hợp AI hỗ trợ tách stems, nhận diện tempo/key (worker riêng).

> Cứ làm lần lượt từ mục **3 → 6 → 7…**, bạn sẽ có prototype hoàn chỉnh đúng công nghệ yêu cầu: **Spring Boot + SQL Server + React**. Khi cần, mình có thể đóng gói **starter repo** với skeleton tất cả service + gateway + docker-compose để bạn clone về chạy ngay.