SP: Render WalkThrough

☕ 1. Spring Boot Project Setup (Maven)

✅ Option A: Use Spring Initializr

Choose:

- **Project**: Maven
- **Language**: Java
- **Spring Boot Version**: Latest stable
- **Dependencies**:
  - Spring Web
  - Spring Boot DevTools
  - Spring Data JPA
  - PostgreSQL Driver
  - Spring Security (optional if adding auth handling)
  - Lombok (optional for reducing boilerplate)
  - Apache Commons FileUpload (manual later)
  - Quartz (for reminders, optional)

Click **Generate**, unzip the project, and open it in IntelliJ / VSCode.

✅ Directory Structure (Important Bits)

```
src/
├── main/
│   ├── java/
│   │   └── com/example/studypilot/
│   │       ├── StudyPilotApplication.java
│   │       ├── controller/
```

```
│   │       ├── service/
│   │       ├── model/
│   │       └── repository/
│   └── resources/
│       ├── application.properties
```

🛠️ 2. Configure `application.properties`

```
# Server
server.port=8080

# PostgreSQL (Supabase or external DB for now)
spring.datasource.url=jdbc:postgresql://localhost:5432/studypilot
spring.datasource.username=YOUR_DB_USER
spring.datasource.password=YOUR_DB_PASSWORD
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

# CORS (allowing frontend to access)
spring.web.cors.allowed-origins=http://localhost:3000
spring.web.cors.allowed-methods=GET,POST,PUT,DELETE
```

🧪 3. Test Locally

1. Create a simple REST controller:

```
@RestController
@RequestMapping("/api/test")
public class TestController {

    @GetMapping
    public String ping() {
        return "Hello from StudyPilot!";
```

```
    }
}
```

1. Run the app:

```
./mvnw spring-boot:run
```

1. Go to `http://localhost:8080/api/test`

   You should see: `Hello from StudyPilot!`

🚀 4. Deploy to Render

🔧 Step-by-Step Guide:

📝 A. Create `render.yaml` (optional, for Infrastructure as Code)

```
services:
  - type: web
    name: studypilot-backend
    env: java
    buildCommand: "./mvnw clean install"
    startCommand: "java -jar target/*.jar"
    plan: free
    envVars:
      - key: SPRING_DATASOURCE_URL
        value: YOUR_REMOTE_DB_URL
      - key: SPRING_DATASOURCE_USERNAME
        value: YOUR_DB_USER
      - key: SPRING_DATASOURCE_PASSWORD
        value: YOUR_DB_PASSWORD
```

🌀 B. Set up on Render:

1. Go to https://render.com

2. Click **New Web Service**
3. Connect your GitHub repo
4. Choose:
    - **Environment**: Java
    - **Build Command**: `./mvnw clean install`
    - **Start Command**: `java -jar target/*.jar`
    - **Instance Type**: Free or Starter
5. Add **Environment Variables** in Render UI:
    - `SPRING_DATASOURCE_URL`
    - `SPRING_DATASOURCE_USERNAME`
    - `SPRING_DATASOURCE_PASSWORD`

> 💡 You can also point this to a Supabase PostgreSQL DB if using Supabase only for storage and auth.

🔄 5. Enable CORS for Frontend Access (Axios from Next.js)

Add this to your Spring config:

```
@Configuration
public class WebConfig implements WebMvcConfigurer {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
                .allowedOrigins("https://your-frontend-url.com", "http://localhost:30
00")
                .allowedMethods("GET", "POST", "PUT", "DELETE");
    }
}
```

📦 6. Connect From Next.js

Install Axios:

```
npm install axios
```

Sample call in your React component:

```
const res = await axios.get('https://your-render-url.onrender.com/api/test');
console.log(res.data); // Should log "Hello from StudyPilot!"
```

🧹 Optional: Add `.renderignore` File

Exclude build files or local configs you don't want deployed.

```
.idea/
target/
*.log
.env
```

✅ You're Done!

Once deployed:

- Your Spring Boot backend is live
- It's communicating with your frontend via Axios
- You can now begin building out endpoints like `/api/syllabus/upload`, `/api/study-plan/generate`, etc.

Would you like me to:

- Scaffold a boilerplate Spring Boot repo for you?
- Show how to parse PDFs and return topic lists?
- Help you configure Google Calendar API for reminder sync?

Let me know how deep you wanna go next!