**Fitness Website Project Overview**

**🎯 Project Title: FitLife Backend – Spring Boot REST API**

**📦 Modules / Features**

| **Module** | **Features** |
| --- | --- |
| **User Module** | Register, Login, View Profile |
| **Workout Module** | CRUD workouts, associate with user |
| **Diet Module** | Add/view meal plans, calorie tracking |
| **Trainer Module** | Assign trainer to users, view clients |
| **Report Module** | Show BMI, weekly progress, etc. |

**🧱 Technologies to be Used**

| **Area** | **Stack** |
| --- | --- |
| Language | Java 17 (can backport to Java 8) |
| Core Concepts | OOPs, Collections, Multithreading, Exception Handling |
| Framework | Spring Boot |
| REST APIs | @RestController, @RequestMapping, DTOs |
| DI & Bean Lifecycle | @Autowired, @Component, @PostConstruct, @PreDestroy |
| Database | MySQL/PostgreSQL (Normalized schema + indexing + joins) |
| Build Tool | Maven |
| Version Control | Git |
| API Testing | Postman |
| Unit Testing | JUnit 5 + Mockito |
| Design Thinking | Layered Architecture (Controller → Service → Repo) |
| Optional Enhancements | Swagger UI, Docker (optional), Authentication with JWT |

**🧠 How Core Concepts Will Be Applied**

| **Concept** | **Application** |
| --- | --- |
| **OOPs** | Entities like User, Workout, Trainer with inheritance/interfaces |
| **Collections** | List of workouts per user, Map of trainer-client, etc. |
| **Multithreading** | Simulate background task (e.g., email/sms sending) |
| **Exception Handling** | Global exception handler (@ControllerAdvice) for 404/500 errors |
| **SQL Joins** | Join User + Trainer tables to fetch trainer-client mappings |
| **Indexing** | Index on user email, workout name |
| **Normalization** | Tables split across users, workouts, trainers, logs |

**📁 Project Structure (Maven)**

pgsql

CopyEdit

com.fitlife

│

├── controller

│ └── UserController.java

│

├── service

│ └── UserService.java

│

├── repository

│ └── UserRepository.java

│

├── model

│ └── User.java

│

├── dto

│ └── UserDTO.java

│

├── config

│ └── AppConfig.java

│

├── exception

│ └── GlobalExceptionHandler.java

│

├── test

│ └── UserServiceTest.java

│

└── Application.java

**✅ Next Step:**

I will now create:

1. A **starter project template** (pom.xml, base packages)
2. Sample entities and controllers (User/Workout)
3. A few examples for collections, threading, exceptions, and JUnit

Would you like me to:

* Share this as downloadable ZIP, GitHub link?
* Or build it together step-by-step here with explanations?

Let me know your preferred mode and I’ll start immediately! 💻✅

**You said:**

guide me with step by step , i will implement. i will provide github link where you can review

**ChatGPT said:**

Perfect! That’s the best way to learn — by building it yourself with expert guidance. 💪  
We’ll follow a structured, **phase-wise implementation plan** where each step builds on the previous one.

**✅ Phase 1: Project Setup (Spring Boot + Maven)**

**🎯 Goal:**

* Create a Spring Boot Maven project
* Initialize Git & connect to GitHub
* Setup MySQL/PostgreSQL
* Test a basic "Hello World" REST API

**✅ Step-by-Step Instructions**

**🧱 Step 1: Create Spring Boot Maven Project**

You can use:

* [**Spring Initializr**](https://start.spring.io/)
* Or IntelliJ / Eclipse Spring Starter Wizard

**Choose:**

* Project: Maven
* Language: Java
* Spring Boot: 3.1.x or 2.7.x
* Group: com.fitlife
* Artifact: fitlife-backend
* Dependencies:
  + Spring Web
  + Spring Data JPA
  + MySQL Driver or PostgreSQL Driver
  + Spring Boot DevTools
  + Lombok (optional but helpful)

🎯 Result: You get a generated project zip. Extract it into your workspace and open it in your IDE.

**📄 Step 2: Set up pom.xml**

Make sure your pom.xml includes:

xml

CopyEdit

<dependencies>

<!-- Spring Boot Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Boot JPA -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- MySQL Driver -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

<!-- JUnit 5 -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

**⚙️ Step 3: application.properties setup**

properties

CopyEdit

# Database config

spring.datasource.url=jdbc:mysql://localhost:3306/fitlife

spring.datasource.username=root

spring.datasource.password=your\_password

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

# Server config

server.port=8080

Replace your\_password with your actual MySQL root password.

✅ Make sure you create a DB called fitlife in MySQL:

sql

CopyEdit

CREATE DATABASE fitlife;

**🔧 Step 4: Create a Sample Controller**

java

CopyEdit

@RestController

@RequestMapping("/api/test")

public class TestController {

@GetMapping("/hello")

public String hello() {

return "Hello, FitLife is up!";

}

}

**🧪 Step 5: Run Your App**

Run FitlifeBackendApplication.java →  
Open browser/Postman: http://localhost:8080/api/test/hello  
✅ You should see: Hello, FitLife is up!

**🗃️ Step 6: Initialize Git**

bash

CopyEdit

git init

git add .

git commit -m "Initial FitLife Spring Boot project"

Then create a GitHub repo and connect it:

bash

CopyEdit

git remote add origin https://github.com/your-username/fitlife-backend.git

git branch -M main

git push -u origin main

**✅ Now Your Turn:**

Please complete these steps and push your initial working project to GitHub.  
Then reply with your **GitHub link** — I’ll review and give feedback or we’ll move to:

**Phase 2: Core Java Concepts – OOPs, Collections, Exceptions, Multithreading via Workout Module**

Ready? Go for it — and paste your GitHub link here once done. 🔗💪

Top of Form

Bottom of Form