Spring Boot Layered Project Structure

Traditional 3-tier architecture with Controller, Service, and Repository layers. Clear separation of concerns for team development.

#spring-boot #java #rest #api #layered #jpa

PNG PDF Copy Prompt

Project Directory

```
myproject/
  src/
    main/
      java/
        com/example/demo/  Base package
          DemoApplication.java  @SpringBootAppl…
          config/  @Configuration …
            SecurityConfig.java
            WebConfig.java
          controller/  @RestController…
            UserController.java
            ProductController.java
          service/  @Service classes
            UserService.java
            ProductService.java
          repository/  Spring Data JPA
            UserRepository.java
            ProductRepository.java
          model/  @Entity classes
            User.java
            Product.java
          dto/  Request/Respons…
            UserDto.java
            CreateUserRequest.java
          exception/  Custom exceptio…
            ResourceNotFoundException.java
            GlobalExceptionHandler.java  @ControllerAdvi…
      resources/
        application.yml  Main config
        application-dev.yml  Dev profile
        application-prod.yml  Prod profile
    test/
      java/
        com/example/demo/
          DemoApplicationTests.java
          controller/
            UserControllerTest.java  @WebMvcTest
          service/
            UserServiceTest.java  Unit tests
          repository/
            UserRepositoryTest.java  @DataJpaTest
  pom.xml
  .gitignore
  README.md
```

Why This Structure?

The classic 3-tier architecture that most Spring developers know. Controllers handle HTTP, services contain business logic, repositories manage data access. Each layer only talks to the one below it, making code testable and maintainable.

Key Directories

controller/ - REST endpoints, request validation, response mapping

service/ - Business logic, transaction boundaries, orchestration

repository/ - Spring Data JPA interfaces for database operations

model/ - JPA entities mapped to database tables

dto/ - Data Transfer Objects for API contracts

exception/ - Custom exceptions and global error handling

Getting Started

1. Configure `application.yml` with database settings
2. `./mvnw spring-boot:run`
3. Visit http://localhost:8080/api/users
4. Use `-Dspring.profiles.active=dev` for dev profile

Layer Dependencies

Controller - Never calls Repository directly

Service - Only layer with business logic

Repository - No business logic, only data access

Model - JPA entities, no behavior

DTO - Never expose entities in API responses

When To Use This

- CRUD-heavy applications
- Team projects with clear responsibilities
- Projects with moderate complexity
- When you need testable, maintainable code
- Standard enterprise Java projects

Trade-offs

Anemic domain - Entities are data bags—logic lives in services

Rigid layers - Can lead to pass-through methods with no logic

Package by layer - Related features spread across packages