Spring Boot Hexagonal Architecture Structure

Ports and Adapters architecture with domain at the center. Infrastructure concerns are isolated from business logic.

#spring-boot #java #hexagonal #ddd #clean-architecture #ports-adapters

PNG PDF Copy Prompt

Project Directory

```
myproject/
  src/
    main/
      java/
        com/example/demo/
          DemoApplication.java
          domain/  Pure business l…
            model/  Entities, Value…
              User.java  Rich domain mod…
              UserId.java  Value Object
              Email.java  Value Object
            port/  Interfaces
              in/  Driving ports
                CreateUserUseCase.java
                GetUserUseCase.java
              out/  Driven ports
                UserRepository.java  Interface only
                NotificationSender.java
            service/  Use case implem…
              UserService.java  Implements ports
            exception/
              UserNotFoundException.java
          adapter/  Infrastructure
            in/  Driving adapters
              web/  REST controllers
                UserController.java
                dto/
                  UserResponse.java
                mapper/
                  UserWebMapper.java
            out/  Driven adapters
              persistence/  Database
                UserPersistenceAdapter.java  Implements port
                UserJpaRepository.java  Spring Data
                UserEntity.java  JPA entity
                UserPersistenceMapper.java
              notification/
                EmailNotificationAdapter.java
          config/  Spring configur…
            BeanConfiguration.java  Wire domain ser…
      resources/
        application.yml
    test/
      java/
        com/example/demo/
          domain/  Pure unit tests
            UserServiceTest.java
          adapter/  Integration tes…
            in/
              UserControllerTest.java
            out/
              UserPersistenceAdapterTest.java
  pom.xml
  .gitignore
  README.md
```

Why This Structure?

The domain is the center of your application, free from framework dependencies. Ports define what the domain needs (interfaces), adapters implement those ports with real infrastructure. Your business logic is testable without Spring, databases, or HTTP.

Key Directories

`domain/` - Pure Java, no Spring annotations, no framework imports

`domain/port/in/` - Use cases—what the application can do

`domain/port/out/` - What the domain needs from outside (repositories, APIs)

`adapter/in/` - How the outside world talks to us (REST, CLI, events)

`adapter/out/` - How we talk to external systems (DB, email, APIs)

Getting Started

1. Start with domain model and use cases
2. Define ports as interfaces in domain
3. Implement adapters in infrastructure layer
4. `./mvnw spring-boot:run`

The Dependency Rule

Domain has zero dependencies on adapters or Spring. Dependencies point inward: Adapters → Domain ← Ports. The domain defines interfaces (ports), adapters implement them. This is enforced by package structure, not magic.

When To Use This

- Complex domain logic worth protecting
- Long-lived projects that will evolve
- Multiple entry points (REST, CLI, events)
- Need to swap infrastructure easily
- Teams practicing Domain-Driven Design

Trade-offs

`More indirection` - Mapping between domain and adapter models

`Learning curve` - Team needs to understand ports/adapters concept

`Overkill for CRUD` - Simple apps don't benefit from this complexity

Testing Strategy

`Domain tests` - Pure unit tests, no Spring context, fast

`Adapter tests` - Integration tests with @WebMvcTest, @DataJpaTest

`E2E tests` - @SpringBootTest with full context