MASTERING SPRING BOOT

A COMPLETE GUIDE
FROM FUNDAMENTALS TO PRODUCTION MICROSERVICES

- BUILD MODERN APPLICATIONS
- SECURE BY DESIGN
- TEST WITH CONFIDENCE
- DEPLOY TO PRODUCTION
- OPERATE AND OBSERVE

SPRING BOOT 4.1 | JAVA 17+ | CURRENT SPRING ECOSYSTEM

EDWARE WALLACE

Mastering Spring Boot

A Complete Guide from Fundamentals to Production Microservices

Steve Publications

This book is available at https://leanpub.com/masteringspringboot

This version was published on 2026-08-13

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

A Complete Guide from Fundamentals to Production Microservices

This book takes you from the basics of Java and Spring through to building, securing, testing, deploying, and operating production-grade Spring Boot applications. Whether you are new to Java or an experienced developer adopting modern Spring Boot practices, this guide provides the knowledge, patterns, and real-world examples you need to build reliable, scalable systems using Spring Boot 4.1, Java 17+, and the current Spring ecosystem.

Introduction

Every technology decision in software development starts with a question: what problem am I trying to solve? For building enterprise-grade backend systems in Java, that answer has increasingly been Spring Boot. But Spring Boot does not exist in isolation. It sits on top of decades of Java evolution, web standards, database theory, and distributed systems patterns. Understanding those foundations is what separates developers who can follow tutorials from engineers who can design robust systems.

This book is structured to take you through that complete journey. We begin with the Java language features that Spring Boot relies on most heavily. Then we move through build tools, web fundamentals, and database concepts before diving into Spring Framework itself and finally Spring Boot in depth. From there we cover REST API development, data access, security, testing, messaging, performance optimization, microservices architecture, containerization, CI/CD pipelines, observability, and advanced topics including reactive programming and native compilation.

The book targets Spring Boot 4.1 and Java 17+. If you are coming from older versions, note the major shifts: Spring Boot 3.x introduced Jakarta EE namespace changes (jakarta.* instead of javax.*), mandatory Java 17 baseline, and significantly updated security defaults. Spring Boot 4.x continues this modernization with refined auto-configuration, improved observability integration, and better container support. The code examples throughout reflect current best practices for these versions.

Throughout the book you will find complete, realistic code examples rather than isolated snippets. Each major topic includes explanations of what the feature does, why it exists, how to implement it, common mistakes to avoid, testing strategies, security implications, and production considerations. The goal is not just to show you how to make something work but to help you understand when and why to use each approach.

You can read this book sequentially from beginning to end if you are starting fresh. Experienced developers can jump directly to chapters relevant to their needs, though some later chapters assume familiarity with earlier material on core Spring concepts.

Let us begin with the Java foundation.

Chapter 1: The Java Foundation for Spring Boot

Spring Boot applications are written in Java. While the framework handles much of the infrastructure complexity, your day-to-day work involves writing Java classes that define domain models, services, controllers, repositories, and configurations. A solid understanding of modern Java is essential not just for using Spring Boot effectively but for writing code that is maintainable, performant, and idiomatic.

This chapter covers the Java language features most relevant to Spring Boot development: object-oriented fundamentals, interfaces and polymorphism, collections and generics, exception handling, annotations, lambdas and streams, and recent Java features like records and sealed classes. If you already know these topics well, you can skim or skip this chapter and return to it as a reference.

Why Java and Why Spring Boot Today

Java remains one of the most widely used languages for backend development, particularly in enterprise environments. Several factors explain its continued dominance: strong typing that catches errors at compile time, mature tooling and IDE support, excellent concurrency primitives, backward compatibility across versions, and an enormous ecosystem of libraries and frameworks. The Java Virtual Machine itself provides platform independence, garbage collection, JIT compilation for performance optimization, and robust memory management.

Spring Boot has become the de facto standard framework for building Java backend applications. It simplifies Spring Framework configuration through sensible defaults and auto-configuration while remaining fully compatible with the underlying Spring ecosystem. Key advantages include rapid project setup via Spring Initializr, embedded web servers that eliminate deployment complexity, production-ready monitoring through Actuator, seamless integration

with databases via Spring Data, comprehensive security support, and native image compilation support for cloud-native deployments.

As of mid-2026, the current stable release is Spring Boot 4.1.0, which requires Java 17 as a minimum and supports up to Java 26. This version works with Spring Framework 7.0.8+, uses Jakarta EE namespaces throughout, and provides refined support for containerization, observability, and native compilation via GraalVM.

Object-Oriented Programming Essentials

Spring Boot applications are built on object-oriented principles. Classes define the structure and behavior of your application components: domain entities that represent business concepts, service classes that implement business logic, controller classes that handle HTTP requests, and configuration classes that wire everything together. Understanding how to design clean, cohesive classes is fundamental.

A class defines a blueprint with fields (state) and methods (behavior). Fields store data, while methods operate on that data. Access modifiers control visibility: private fields encapsulate internal state, protected members are accessible within packages and subclasses, package-private members are visible only within the same package, and public members are accessible from anywhere. Spring Boot heavily relies on this encapsulation pattern, particularly through dependency injection where objects receive their dependencies through constructors or setters rather than creating them directly.

Consider a typical domain entity in a Spring Boot application:

```
public class Product {
    private Long id;
    private String name;
    private BigDecimal price;
    private boolean active;

    public Product(String name, BigDecimal price) {
        this.name = name;
        this.price = price;
        this.active = true;
    }

    public Long getId() { return id; }
```

```
    public String getName() { return name; }
    public BigDecimal getPrice() { return price; }
    public boolean isActive() { return active; }

    public void deactivate() {
        this.active = false;
    }
}
```

This class encapsulates product data with controlled access through getters and a behavioral method (deactivate). Spring Boot frameworks like JPA will interact with such classes through reflection, reading fields and calling methods to map between Java objects and database rows. The constructor pattern shown here is common in modern Spring development: a parameterized constructor for creating new instances and generated or explicit getters for reading values.

Inheritance allows one class to extend another, inheriting its fields and methods while adding specialized behavior. Spring Framework itself uses inheritance extensively: controllers extend base classes, repositories extend interfaces that inherit from core repository types, and configuration classes build on framework abstractions. However, in application code, composition is often preferred over deep inheritance hierarchies for maintainability.

Interfaces, Abstract Classes, and Polymorphism

Interfaces define contracts without implementation. A class implements an interface by providing concrete implementations for all declared methods. Spring Boot relies heavily on interfaces because they enable loose coupling: code depends on abstractions rather than concrete classes, making it easier to swap implementations, write tests with mocks, and organize complex systems.

Consider how Spring Data JPA defines repositories as interfaces:

```
public interface UserRepository extends JpaRepository<User, Long> {
    List<User> findByEmail(String email);
    boolean existsByEmail(String email);
}
```

You declare what operations you need without writing any implementation code. Spring Boot creates the implementation at runtime through proxies. This

pattern appears throughout the framework: controllers implement request handling contracts, services define business logic interfaces, and configuration classes work with bean definitions rather than concrete instances.

Abstract classes provide partial implementation along with abstract methods that subclasses must implement. Use an abstract class when you have common behavior shared across related types but need flexibility for specific implementations. For example, a base service class might provide common transaction handling or logging while leaving business-specific operations to subclasses:

```
public abstract class BaseService<T, ID> {
    protected abstract JpaRepository<T, ID> getRepository();

    public T findById(ID id) {
        return getRepository().findById(id)
            .orElseThrow(() -> new EntityNotFoundException(id));
    }

    public List<T> findAll() {
        return getRepository().findAll();
    }
}
```

Polymorphism allows code to work with different types through a common interface or base class. When Spring Boot's dependency injection resolves a bean, it often injects an implementation that satisfies an interface type. The consuming code works with the interface without knowing which specific implementation is provided. This flexibility is crucial for testing: you can inject mock implementations in tests while using real implementations in production, all without changing the consumer code.

Collections Framework and Generics

Java's Collections Framework provides data structures essential for any application: lists for ordered sequences, sets for unique elements, maps for key-value associations, queues for processing order, and specialized variants for concurrent access. Spring Boot applications use these extensively to manage collections of entities, configuration values, request parameters, and cached data.

Lists maintain insertion order and allow duplicates. ArrayList is the most common implementation, optimized for fast random access. LinkedList provides faster insertions and deletions at arbitrary positions but slower access by index. In Spring Boot, you will frequently work with lists when returning collections of entities from repositories or processing arrays of request parameters:

```
// Repository returns a list of products
List<Product> products = productRepository.findAll();

// Filtering in service layer
List<Product> activeProducts = products.stream()
    .filter(Product::isActive)
    .toList();
```

Sets guarantee uniqueness. HashSet provides fast lookup based on hash codes, while TreeSet maintains elements in sorted order. You might use sets for tracking unique user roles, validating that no duplicate entries exist, or implementing allowlists and blocklists:

```
Set<String> allowedOrigins = Set.of("https://app.example.com",
↪  "https://admin.example.com");
boolean isAllowed = allowedOrigins.contains(origin);
```

Maps associate keys with values. HashMap is the standard implementation for fast key-based lookups. TreeMap keeps keys sorted. LinkedHashMap preserves insertion order. Spring Boot uses maps internally for configuration property binding, request attribute storage, and caching:

```
Map<String, Object> config = Map.of(
    "cache.ttl", 3600L,
    "max.retries", 3,
    "feature.new-checkout", true
);
```

Generics provide type safety for collections and other parameterized types. They allow you to specify the type of elements a collection holds, catching type errors at compile time rather than runtime. Spring Boot frameworks extensively use generics: repositories are parameterized by entity type and ID type (JpaRepository<User, Long>), responses wrap typed payloads, and configuration properties bind to strongly-typed classes.

```java
// Generic repository interface
public interface CrudRepository<T, ID> extends Repository<T, ID> {
    <S extends T> S save(S entity);
    Optional<T> findById(ID id);
    Iterable<T> findAll();
}
```

When working with generics in Spring Boot, pay attention to type erasure: generic type information is available at compile time but not at runtime. This matters when using reflection or creating instances generically. Framework code handles this through Class objects passed as parameters or TypeToken patterns.

Exception Handling Best Practices

Robust exception handling is critical for production applications. Java provides two categories of exceptions: checked exceptions that must be declared and handled (like SQLException), and unchecked exceptions (RuntimeException subclasses) that represent programming errors or unexpected conditions. Spring Boot follows the convention of using unchecked exceptions for most application-level error conditions, making code cleaner while still allowing centralized error handling.

Define domain-specific exception classes for your business rules. This enables precise error handling and clear communication of what went wrong:

```java
public class ResourceNotFoundException extends RuntimeException {
    public ResourceNotFoundException(String resource, Long id) {
        super(String.format("%s not found with id %d", resource, id));
    }
}

public class DuplicateResourceException extends RuntimeException {
    public DuplicateResourceException(String resource, String identifier) {
        super(String.format("Duplicate %s with identifier: %s", resource,
        ↪  identifier));
    }
}
```

In Spring Boot web applications, you handle exceptions globally using @ControllerAdvice and @ExceptionHandler. This keeps error handling logic centralized rather than scattered across controllers:

```java
@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<ErrorResponse>
    ↪ handleNotFound(ResourceNotFoundException ex) {
        ErrorResponse error = new ErrorResponse(
            HttpStatus.NOT_FOUND.value(),
            ex.getMessage(),
            LocalDateTime.now()
        );
        return new ResponseEntity<>(error, HttpStatus.NOT_FOUND);
    }

    @ExceptionHandler(DuplicateResourceException.class)
    public ResponseEntity<ErrorResponse>
    ↪ handleDuplicate(DuplicateResourceException ex) {
        ErrorResponse error = new ErrorResponse(
            HttpStatus.CONFLICT.value(),
            ex.getMessage(),
            LocalDateTime.now()
        );
        return new ResponseEntity<>(error, HttpStatus.CONFLICT);
    }
}
```

When working with external resources like databases or APIs, wrap checked exceptions in unchecked ones to avoid cluttering method signatures. Spring's DataAccessException hierarchy already does this for database operations, converting SQLException and other checked exceptions into RuntimeException subclasses that you can handle uniformly.

Never catch exceptions without purpose. Catch only what you can meaningfully recover from, log meaningful context about failures, and let unexpected errors propagate so they are visible rather than silently swallowed. In Spring Boot, the default error handler will return a 500 response for unhandled exceptions while logging the stack trace, which is appropriate behavior for most unexpected failures during development.

Annotations: The Language of Spring

Annotations provide metadata about code elements without changing their logic. They are declarations that frameworks like Spring read to configure behavior automatically. Spring Boot's entire architecture relies on annotations:

@SpringBootApplication marks your main class, @RestController identifies web controllers, @Service and @Repository mark service layers and data access components, @Autowired enables dependency injection, @ConfigurationProperties binds configuration values, and dozens more control specific behaviors.

Understanding how annotations work helps you use Spring Boot effectively and debug issues when they arise. Annotations have targets (what they can annotate: classes, methods, fields, parameters) and retention policies (when they are available: source time only, class file level, or runtime via reflection). Spring annotations typically use RUNTIME retention so the framework can inspect them at startup.

Spring Boot uses meta-annotations extensively: one annotation that is itself annotated with other annotations. @SpringBootApplication is a prime example – it combines @Configuration (marks a class as a configuration source), @EnableAutoConfiguration (activates auto-configuration), and @ComponentScan (enables component scanning). This pattern keeps commonly used combinations concise while maintaining flexibility for individual pieces when needed.

You can create custom annotations for your own purposes. For example, you might define a @CacheableForUser annotation that combines caching behavior with user-specific key generation:

```
@Target({ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
@Cacheable(cacheNames = "user-data")
public @interface CacheableForUser {
    String keyGenerator() default "#userId";
}
```

Spring Boot reads annotations during application startup through component scanning and configuration processing. When a class is annotated with @Component or any of its specializations (@Service, @Repository, @Controller), Spring registers it as a bean in the application context. Annotations on methods and fields control dependency injection, transaction boundaries, caching behavior, security constraints, and many other aspects of runtime behavior.

Lambdas, Streams, and Modern Java

Java 8 introduced lambda expressions and the Stream API, fundamentally changing how developers write functional-style code in Java. Spring Boot applications make extensive use of these features for concise, expressive data processing, event handling, and configuration.

Lambda expressions provide a compact way to represent anonymous functions. Where you previously needed verbose anonymous inner classes, lambdas express the same logic concisely:

```
// Before (anonymous class)
Comparator<Product> byPrice = new Comparator<Product>() {
    @Override
    public int compare(Product p1, Product p2) {
        return p1.getPrice().compareTo(p2.getPrice());
    }
};

// After (lambda)
Comparator<Product> byPrice = (p1, p2) ->
↪  p1.getPrice().compareTo(p2.getPrice());

// Even shorter with method reference
Comparator<Product> byPrice = Comparator.comparing(Product::getPrice);
```

Spring Framework uses functional interfaces extensively. Event listeners can be lambda expressions, stream operations use lambdas for filtering and mapping, configuration builders accept lambda-based customization, and reactive streams rely entirely on functional composition.

The Stream API enables declarative data processing pipelines. Instead of writing explicit loops, you chain operations that describe what transformation to perform:

```
List<Order> expensiveActiveOrders = orderRepository.findAll().stream()
    .filter(Order::isActive)
    .filter(o -> o.getTotal().compareTo(new BigDecimal("1000")) > 0)
    .sorted(Comparator.comparing(Order::getTotal).reversed())
    .limit(10)
    .toList();
```

This code reads naturally: filter active orders, keep those over 1000, sort by total descending, take the top 10. Streams support both intermediate operations (filter, map, sorted, distinct) that return new streams and terminal operations (toList, collect, forEach, reduce) that produce results.

In Spring Boot, streams are particularly useful in service layers for transforming entity collections to DTOs, filtering data based on complex criteria, aggregating statistics, and preparing paginated responses:

```
public List<ProductDto> getActiveProducts() {
    return productRepository.findAll().stream()
        .filter(Product::isActive)
        .map(product -> new ProductDto(
            product.getId(),
            product.getName(),
            product.getPrice()
        ))
        .toList();
}
```

Optional provides a clean way to represent values that may be absent without using null. Spring Data repositories return Optional for findById operations, and using Optional throughout your codebase reduces NullPointerExceptions:

```
public ProductDto getProductOrThrow(Long id) {
    return productRepository.findById(id)
        .map(ProductDto::new)
        .orElseThrow(() -> new ResourceNotFoundException("Product", id));
}
```

Records, Sealed Classes, and Recent Java Features

Java 17 introduced several features that Spring Boot applications leverage for cleaner, more maintainable code. These features reduce boilerplate while improving type safety and expressiveness.

Records provide concise immutable data carriers. Where you previously wrote full classes with fields, a constructor, getters, equals, hashCode, and toString, a record declares all of this in one line:

```
public record ProductDto(Long id, String name, BigDecimal price) {}
```

This generates an immutable class with final fields, a canonical constructor, accessor methods (id(), name(), price()), and correct implementations of equals, hashCode, and toString. Records are ideal for DTOs (Data Transfer Objects), value objects, configuration data, and API request/response bodies in Spring Boot applications:

```
public record CreateUserRequest(String email, String name, String password) {}
public record UserResponse(Long id, String email, String name, LocalDateTime
↪  createdAt) {}
```

Spring Boot integrates seamlessly with records. They work as controller request bodies via JSON deserialization, as method return types for automatic serialization, and as configuration property targets with @ConfigurationProperties. The framework treats them like any other class for dependency injection and bean management.

Sealed classes restrict which other classes may extend them, making your type hierarchies explicit and easier to reason about:

```
public sealed interface PaymentMethod permits CreditCardPayment, PayPalPayment,
↪  BankTransferPayment {}

public final class CreditCardPayment implements PaymentMethod { /* ... */ }
public final class PayPalPayment implements PaymentMethod { /* ... */ }
public final class BankTransferPayment implements PaymentMethod { /* ... */ }
```

Pattern matching for instanceof simplifies type checking and casting:

```java
// Before
if (obj instanceof String) {
    String s = (String) obj;
    // use s
}

// After (Java 16+)
if (obj instanceof String s) {
    // use s directly
}
```

Switch expressions with pattern matching (Java 21+) further simplify type-based logic:

```java
return switch (paymentMethod) {
    case CreditCardPayment ccp -> processCreditCard(ccp);
    case PayPalPayment pp -> processPayPal(pp);
    case BankTransferPayment btp -> processBankTransfer(btp);
};
```

Text blocks (Java 15+) make multi-line strings readable, useful for SQL queries, JSON templates, and HTML content:

```java
String sql = """
    SELECT p.id, p.name, p.price
    FROM products p
    WHERE p.active = true
    ORDER BY p.name
    """;
```

These modern Java features combine to produce Spring Boot applications that are more concise, safer, and easier to maintain than those written with older Java versions. Using them is not just about following trends – it is about writing code that communicates intent clearly and reduces the surface area for bugs.

Chapter 2: Build Tools, Dependencies, and Project Setup

Before writing Spring Boot code, you need a build system to compile your application, manage its dependencies, run tests, and package the final artifact. Java has two dominant build tools: Maven and Gradle. Both integrate deeply with Spring Boot through official plugins and are fully supported by IDEs and CI/CD pipelines. This chapter explains how each works, when to choose one over the other, and how to configure them for professional Spring Boot development.

Maven vs Gradle: Choosing Your Build Tool

Maven has been the standard Java build tool for decades. It uses XML-based configuration files (pom.xml) and follows a convention-over-configuration philosophy. Project structure, build lifecycle phases, and plugin behavior are largely predetermined, which means new developers can understand any Maven project quickly by knowing the conventions. Maven's central repository is mature and reliable, its dependency resolution is deterministic, and its ecosystem of plugins is extensive.

Gradle emerged as an alternative that addresses Maven's verbosity and inflexibility. It uses a Groovy or Kotlin DSL for build scripts (build.gradle or build.gradle.kts) that are more concise and programmable than XML. Gradle supports incremental builds through fine-grained change detection, making large projects significantly faster to rebuild. Its dependency management is powerful and flexible, and it handles multi-project builds elegantly.

For Spring Boot development, both tools work equally well. The official Spring Boot documentation provides examples for both, Spring Initializr generates projects with either build system, and all Spring Boot plugins support both. Your choice should consider:

- Team familiarity: if your team knows Maven well, use Maven

- Project complexity: Gradle shines in large multi-module projects where its programmability matters
- Build speed requirements: Gradle's incremental builds can be significantly faster for large codebases
- Ecosystem constraints: some organizations standardize on one tool

This book provides examples for both tools. If you prefer one over the other, focus on those examples and refer to the alternative only when needed. The underlying Spring Boot concepts are identical regardless of build tool.

Maven Fundamentals and POM Configuration

Maven organizes projects around a Project Object Model defined in pom.xml. This file declares project metadata, dependencies, plugins, and build configuration. Every Maven project has coordinates: groupId, artifactId, and version, which uniquely identify it in repositories.

A minimal Spring Boot Maven project looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>4.1.0</version>
        <relativePath/>
    </parent>

    <groupId>com.example</groupId>
    <artifactId>my-application</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>my-application</name>
    <description>My Spring Boot Application</description>

    <properties>
        <java.version>17</java.version>
    </properties>

    <dependencies>
```

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <scope>runtime</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

Key elements explained:

- The parent POM (spring-boot-starter-parent) provides sensible defaults for compiler settings, dependency versions, plugin configurations, and resource filtering. It manages transitive dependency versions so you rarely need to specify versions explicitly in your own dependencies.
- Properties define configurable values used throughout the POM. Setting java.version to 17 tells Maven to compile with Java 17 language features.
- Dependencies list libraries your project needs. Notice that most do not have version elements – the parent POM manages those versions through its dependencyManagement section, ensuring compatible versions across all Spring Boot dependencies.

- The spring-boot-maven-plugin packages your application as an executable JAR (also called an uber jar or fat jar) containing your code, all dependencies, and an embedded web server. This is what makes Spring Boot applications runnable with a simple java -jar command.

Maven's build lifecycle consists of phases: validate, compile, test, package, verify, install, deploy. Running mvn clean package compiles the code, runs tests, and creates the executable JAR in the target directory. The clean phase removes previous build artifacts to ensure a fresh build.

For multi-module projects, Maven uses a parent POM that aggregates child modules. Each module has its own pom.xml but inherits from the parent, sharing dependency management and plugin configurations:

```
<!-- Parent POM -->
<packaging>pom</packaging>
<modules>
    <module>api-gateway</module>
    <module>user-service</module>
    <module>order-service</module>
    <module>shared-models</module>
</modules>
```

Gradle Fundamentals and Build Scripts

Gradle uses build scripts written in Groovy (build.gradle) or Kotlin DSL (build.gradle.kts). The Kotlin DSL provides type safety, better IDE support, and compile-time checking of your build configuration. For new Spring Boot projects, the Kotlin DSL is recommended.

Here is the equivalent Spring Boot project using Gradle with Kotlin DSL:

```
plugins {
    java
    id("org.springframework.boot") version "4.1.0"
    id("io.spring.dependency-management") version "1.1.7"
}

group = "com.example"
version = "0.0.1-SNAPSHOT"

java {
    toolchain {
        languageVersion = JavaLanguageVersion.of(17)
    }
}

repositories {
    mavenCentral()
}

dependencies {
    implementation("org.springframework.boot:spring-boot-starter-web")
    implementation("org.springframework.boot:spring-boot-starter-data-jpa")
    runtimeOnly("org.postgresql:postgresql")
    testImplementation("org.springframework.boot:spring-boot-starter-test")
    testRuntimeOnly("org.junit.platform:junit-platform-launcher")
}

tasks.withType<Test> {
    useJUnitPlatform()
}
```

Key elements explained:

- The plugins block declares the Spring Boot plugin and the dependency management plugin. These replace Maven's parent POM functionality.
- The java block with toolchain configuration specifies Java 17, using Gradle's toolchain feature that can automatically download the required JDK if not present locally.
- Repositories define where Gradle finds dependencies. mavenCentral() is the standard central repository.
- Dependencies use a concise notation: configuration("group:artifact"). The configurations (implementation, runtimeOnly, testImplementation) control when and how dependencies are used – implementation dependencies are available at compile time and runtime, runtimeOnly dependencies are only needed at runtime, and testImplementation dependencies are for tests only.

Gradle tasks replace Maven's lifecycle phases. Common tasks include:

- build: compiles code, runs tests, and creates the package
- bootJar: creates the executable Spring Boot JAR
- bootRun: runs the application directly from source without packaging
- clean: removes build artifacts
- test: runs all tests

For multi-module projects, Gradle uses a settings.gradle.kts file to define included projects:

```
rootProject.name = "my-platform"
include("api-gateway")
include("user-service")
include("order-service")
include("shared-models")
```

Each module has its own build.gradle.kts, and shared configuration can be applied through convention plugins or a buildSrc directory.

Gradle's incremental build system tracks file changes and only rebuilds what has changed, which can dramatically reduce build times for large projects. Its dependency caching also speeds up subsequent builds by reusing downloaded artifacts.

Dependency Management Strategies

Spring Boot simplifies dependency management through its starters and version alignment. A starter is a curated set of dependencies that you include with a single declaration to get everything needed for a specific use case: web development, database access, security, testing, messaging, and more.

Common Spring Boot starters include:

- spring-boot-starter-web: Spring MVC, embedded Tomcat, JSON serialization
- spring-boot-starter-webmvc: explicit naming in newer versions for Spring MVC web support
- spring-boot-starter-data-jpa: JPA support with Hibernate

- spring-boot-starter-security: Spring Security authentication and authorization
- spring-boot-starter-test: JUnit, Mockito, Hamcrest, and Spring Test utilities
- spring-boot-starter-validation: Bean Validation (Jakarta Validation)
- spring-boot-starter-cache: Spring Cache abstraction
- spring-boot-starter-mail: JavaMail and Spring Email support
- spring-boot-starter-actuator: production monitoring endpoints
- spring-boot-starter-oauth2-resource-server: OAuth2/JWT resource server support
- spring-boot-starter-kafka: Apache Kafka integration
- spring-boot-starter-amqp: RabbitMQ integration

Third-party starters exist for many integrations, but be cautious: only use well-maintained starters from trusted sources. The spring-boot prefix is reserved for official Spring Boot starters.

Spring Boot's dependency management ensures that all managed dependencies use compatible versions. When you add spring-boot-starter-web and spring-boot-starter-data-jpa, the Tomcat version, Hibernate version, Jackson version, and all transitive dependencies are aligned to work together. This eliminates one of Java development's most frustrating problems: version conflicts between libraries.

When you need a specific library not managed by Spring Boot, declare it with an explicit version. If you need to override a managed version (rarely necessary), use dependencyManagement in Maven or the versions plugin in Gradle:

```
<!-- Maven: override a managed version -->
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.18.0</version>
        </dependency>
    </dependencies>
</dependencyManagement>
```

```
// Gradle: override a managed version
dependencies {
    constraints {
        implementation("com.fasterxml.jackson.core:jackson-databind:2.18.0")
    }
}
```

Dependency scopes control when dependencies are available:

- compile/implementation: needed at compile time and runtime (default)
- provided/runtimeOnly: only needed at runtime (e.g., database drivers)
- test: only for test compilation and execution
- annotationProcessor: for annotation processing (e.g., Lombok, MapStruct)

Understanding scopes helps keep your application's classpath lean and avoids packaging unnecessary libraries into your final artifact.

Managing Versions and Dependency Conflicts

Despite Spring Boot's careful version management, conflicts can still occur when different dependencies require incompatible versions of the same library. Maven and Gradle provide tools to diagnose and resolve these issues.

In Maven, run mvn dependency:tree to see the full dependency graph:

```
[INFO] com.example:my-application:jar:0.0.1-SNAPSHOT
[INFO] +- org.springframework.boot:spring-boot-starter-web:jar:4.1.0:compile
[INFO] |  +- org.springframework.boot:spring-boot-starter:jar:4.1.0:compile
[INFO] |  \-
↪  org.springframework.boot:spring-boot-starter-tomcat:jar:4.1.0:compile
[INFO] \- org.postgresql:postgresql:jar:42.7.3:runtime
```

Look for duplicate libraries with different versions. Maven's default conflict resolution uses the "nearest definition" rule: the version closest to your project in the dependency tree wins. You can override this explicitly in dependencyManagement.

In Gradle, run ./gradlew dependencies to see the dependency graph:

```
+--- org.springframework.boot:spring-boot-starter-web -> 4.1.0
|    +--- org.springframework.boot:spring-boot-starter -> 4.1.0
|    \- org.springframework.boot:spring-boot-starter-tomcat -> 4.1.0
\- org.postgresql:postgresql -> 42.7.3
```

Gradle uses the "first declaration" rule by default. You can use dependency constraints or force specific versions when needed.

Common conflict scenarios and solutions:

- Different Spring versions: ensure all Spring dependencies come from Spring Boot's managed set, not declared independently
- Jackson version conflicts: usually resolved by using Spring Boot's managed version; only override if you need a specific bug fix
- Logging framework conflicts: Spring Boot standardizes on Logback with SLF4J; exclude conflicting logging implementations from third-party libraries

When excluding transitive dependencies that cause conflicts:

```xml
<!-- Maven -->
<dependency>
    <groupId>com.some.library</groupId>
    <artifactId>problematic-lib</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

```
// Gradle
implementation("com.some.library:problematic-lib") {
    exclude(group = "org.slf4j", module = "slf4j-log4j12")
}
```

Project Structure Conventions

A consistent project structure makes Spring Boot applications easier to navigate, maintain, and extend. Spring Boot follows standard Java conventions with some framework-specific patterns.

Standard directory layout:

```
my-application/
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com/example/myapp/
│   │   │       ├── MyApplication.java          # Main application class
│   │   │       ├── config/                     # Configuration classes
│   │   │       ├── controller/                 # Web controllers
│   │   │       ├── service/                    # Business logic services
│   │   │       ├── repository/                 # Data access repositories
│   │   │       ├── model/                      # Domain entities and DTOs
│   │   │       │   ├── entity/                 # JPA entities
│   │   │       │   └── dto/                    # Data transfer objects
│   │   │       ├── security/                   # Security configuration
│   │   │       └── exception/                  # Exception handling
│   │   └── resources/
│   │       ├── application.properties          # Application configuration
│   │       ├── application-dev.properties      # Development profile config
│   │       ├── application-prod.properties     # Production profile config
│   │       └── db/migration/                   # Database migration scripts
│   └── test/
│       ├── java/
│       │   └── com/example/myapp/              # Test classes mirror main
↪  structure
│       └── resources/
│           └── application-test.properties     # Test configuration
├── pom.xml                                     # Maven configuration
├── build.gradle.kts                            # Gradle configuration
↪  (alternative)
├── Dockerfile                                  # Container definition
└── README.md                                   # Project documentation
```

Package organization follows these principles:

- Place the main application class at the root of your package hierarchy so component scanning finds all sub-packages by default
- Group classes by layer (controller, service, repository) rather than by feature for small to medium applications

- For larger applications, consider organizing by domain or feature module instead of strict layers
- Keep configuration classes in a dedicated config package
- Separate entities (database-mapped objects) from DTOs (API request/response objects)

The main application class serves as the entry point and typically looks like this:

```
package com.example.myapp;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

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

The @SpringBootApplication annotation enables auto-configuration and component scanning for the package where this class resides and all its sub-packages. This is why placing it at the root of your package hierarchy matters: it ensures Spring discovers all your components without additional configuration.

Chapter 3: Web Fundamentals for Spring Boot Developers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

HTTP Protocol Essentials

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

REST Architecture and API Design Principles

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

JSON and Data Serialization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

SQL Fundamentals and Relational Databases

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Database Clients and Connection Basics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

API Clients and Testing Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Chapter 4: Spring Framework Core Concepts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Inversion of Control and Dependency Injection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

The Spring Application Context

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Beans: Definition, Scopes, and Lifecycle

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Component Scanning and Stereotype Annotations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Configuration with Java Config and XML

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Profiles and Conditional Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Externalized Configuration and Properties

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Chapter 5: Spring Boot Fundamentals

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

What Spring Boot Solves and How It Works

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Creating Projects with Spring Initializr

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Auto-Configuration Mechanics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Starters: Curated Dependency Sets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Embedded Servlet Containers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Configuration Properties and Binding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Running, Packaging, and Distributing Applications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Chapter 6: Building REST APIs with Spring MVC

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Spring MVC Architecture Overview

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Controllers and Request Mapping

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Path Variables, Query Parameters, and Request Bodies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Response Types and Content Negotiation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Input Validation with Bean Validation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Global Exception Handling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Filters, Interceptors, and Handler Mappings

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

File Uploads and Downloads

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Chapter 7: Designing Professional REST APIs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

RESTful API Design Principles in Practice

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Resource Modeling and Naming Conventions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Pagination, Sorting, and Filtering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

API Versioning Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

OpenAPI Documentation with SpringDoc

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

HATEOAS for Hypermedia APIs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

CORS Configuration and Cross-Origin Security

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Chapter 8: Data Access with Spring Data JPA

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

JPA and Hibernate Fundamentals

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Entity Mapping and Relationships

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Spring Data Repositories and Derived Queries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Custom Queries with JPQL and Native SQL

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Specifications and Dynamic Queries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Transaction Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Database Migrations with Flyway and Liquibase

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Connection Pooling and Performance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Chapter 9: Advanced Data Access Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Optimistic and Pessimistic Locking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Lazy Loading, Eager Loading, and N+1 Problems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Second-Level Cache and Query Cache

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Batch Operations and Bulk Updates

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

When to Consider NoSQL Options

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Database Indexing Strategies for JPA

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Chapter 10: Security with Spring Security

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Spring Security Architecture Overview

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Authentication and Authorization Fundamentals

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Password Encoding and User Details Service

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Session Management and CSRF Protection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Method-Level Security

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

OAuth 2.0 and OpenID Connect

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

JWT-Based API Authentication

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Resource Servers and Token Validation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Security in Production: Best Practices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Chapter 11: Application Architecture and Design Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Layered Architecture in Spring Boot

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Domain-Driven Design Concepts for Spring

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

The Repository Pattern Deep Dive

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

DTOs, Mappers, and Separation of Concerns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

SOLID Principles in Practice

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Modular Monolith Organization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Hexagonal and Clean Architecture Approaches

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Chapter 12: Testing Spring Boot Applications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Testing Philosophy and Pyramid

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Unit Testing with JUnit and Mockito

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Integration Testing with Spring Boot Test

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Web Layer Testing with MockMvc

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Database Testing with Testcontainers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Security Testing Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

API Contract Testing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Test Configuration, Profiles, and Fixtures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Chapter 13: Async Processing, Messaging, and Events

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Scheduled Tasks and Async Methods

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Spring Application Events

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Message Brokers: Kafka vs RabbitMQ

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Producer and Consumer Patterns with Kafka

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Event-Driven Architecture with Spring

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Retries, Dead-Letter Queues, and Idempotency

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Transaction Boundaries in Distributed Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Chapter 14: Caching, Performance, and Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Spring Cache Abstraction

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Redis Integration for Distributed Caching

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Cache Invalidation Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

JVM Tuning and Memory Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Connection Pooling Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Application Profiling Techniques

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Concurrency and Thread Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Chapter 15: Observability, Logging, and Monitoring

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Structured Logging Best Practices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Logback Configuration and Levels

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Spring Boot Actuator Endpoints

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Metrics with Micrometer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Health Checks and Liveness Probes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Distributed Tracing Concepts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Correlation IDs and Request Tracking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Alerting and Production Diagnostics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Chapter 16: Integration Patterns and External Services

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

REST Clients with WebClient and RestTemplate

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Resilience Patterns: Retries, Timeouts, Circuit Breakers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Rate Limiting and Backpressure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Spring Batch for Batch Processing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

File Storage Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Email Integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Webhooks and Callback Handling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Chapter 17: Microservices with Spring Boot

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Monolith vs Microservices Decision Framework

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Service Decomposition Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Inter-Service Communication Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

API Gateways and Routing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Service Discovery and Registry

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Centralized Configuration Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Distributed Tracing Across Services

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Security in Microservice Architectures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Chapter 18: Containerization, Orchestration, and Cloud Deployment

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Docker Fundamentals for Java Applications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Writing Efficient Dockerfiles for Spring Boot

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Multi-Stage Builds and Layer Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Kubernetes Deployment Basics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Health Probes and Liveness Checks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Configuration and Secrets in Kubernetes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Scaling Strategies and Rolling Updates

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Cloud Deployment Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Chapter 19: CI/CD and Release Engineering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Git Workflow Strategies for Teams

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Build Pipeline Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Automated Testing in Pipelines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Artifact Management and Versioning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Environment Promotion Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Blue-Green and Canary Deployments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Rollback Strategies and Safety Nets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Infrastructure as Code Considerations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Chapter 20: Advanced Spring Boot Development

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Custom Auto-Configuration Deep Dive

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Building Your Own Starters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Advanced Bean Scopes and Lifecycle Hooks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Aspect-Oriented Programming with Spring AOP

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Custom Annotations and Meta-Annotations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

GraalVM Native Compilation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Reactive Programming with WebFlux

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

When to Choose Reactive Over MVC

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Chapter 21: Configuration Management Mastery

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Properties Files and YAML Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

@ConfigurationProperties and Binding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Profiles and Environment-Specific Config

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Environment Variables and Command-Line Overrides

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Configuration Precedence Rules

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Externalized Configuration Sources

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Secrets Management Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Configuration Validation at Startup

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Chapter 22: Troubleshooting and Production Operations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Startup Failures and Dependency Conflicts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Bean Creation Errors and Circular Dependencies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Database Connection and Transaction Problems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Security Configuration Issues

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

REST API Debugging Techniques

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Performance Bottleneck Diagnosis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Memory Leaks and GC Tuning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Concurrency Issues

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Deployment Failures and Container Problems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Production Incident Response Procedures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/masteringspringboot.