When starting development using Spring Boot, it's essential to have a solid understanding of several basic concepts:

**1. Spring Framework:**

Understanding the core concepts of the Spring Framework is crucial. These include:

* Dependency Injection (DI) and Inversion of Control (IoC): Spring's DI container manages the dependencies of application objects.
* Beans: Java objects managed by the Spring container.
* AOP (Aspect-Oriented Programming): Spring supports AOP, which allows you to modularize cross-cutting concerns in your application.

**2. Spring Boot Basics:**

* **Starter Dependencies**: Spring Boot starters provide pre-configured dependencies for commonly used libraries and frameworks.
* **Auto-configuration**: Spring Boot automatically configures your application based on the dependencies and properties you have included.
* **Embedded Servers**: Spring Boot includes support for embedded servers like Tomcat, Jetty, and Undertow.

**3. Spring MVC:**

Spring MVC is a widely-used framework for building web applications. Key concepts include:

* **Controllers**: Components that handle incoming HTTP requests.
* **Views**: Templates or views used to generate the response.
* **Model**: Data that is passed between the controller and the view.

**4. Data Access:**

Spring Boot supports various data access technologies, including:

* **JPA (Java Persistence API)**: Object-relational mapping framework for managing relational data in Java applications.
* **Spring Data JPA**: Simplifies data access with JPA repositories.
* **Hibernate, JDBC**: Other options for interacting with databases.

**5. Security:**

Understanding how to secure your Spring Boot application is important. Key concepts include:

* **Authentication**: Verifying the identity of users.
* **Authorization**: Controlling access to resources based on user roles and permissions.
* **Spring Security**: Powerful and customizable security framework provided by Spring.

**6. Testing:**

Spring Boot provides support for testing your applications. Important testing concepts include:

* **Unit Testing**: Testing individual components or units of code.
* **Integration Testing**: Testing interactions between components.
* **Spring Test**: Provides annotations and utilities for testing Spring applications.

**7. Configuration:**

Spring Boot allows you to externalize configuration using application.properties or application.yml files. Understanding how to configure your application using these files is important.

**8. Actuator:**

Spring Boot Actuator provides endpoints for monitoring and managing your application. This includes health checks, metrics, environment details, etc.

**9. RESTful Web Services:**

Spring Boot simplifies the development of RESTful APIs. Understanding how to create REST controllers, handle requests, and produce responses is essential.

**10. Dependency Management:**

Understanding how to manage dependencies using Maven or Gradle is important for building and managing Spring Boot projects.

Examples

**1. Spring Framework:**

// Bean class

public class MyService {

public void sayHello() {

System.out.println("Hello from MyService!");

}

}

// Configuration class

@Configuration

public class AppConfig {

@Bean

public MyService myService() {

return new MyService();

}

}

// Main class

public class MainApp {

public static void main(String[] args) {

ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);

MyService service = context.getBean(MyService.class);

service.sayHello();

}

} **2. Spring Boot Basics:**

@SpringBootApplication

public class MyApplication {

public static void main(String[] args) {

SpringApplication.run(MyApplication.class, args);

}

} **3. Spring MVC:**

@RestController

public class HelloController {

@GetMapping("/hello")

public String hello() {

return "Hello, World!";

}

} **4. Data Access (JPA):**

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String username;

private String email;

// Getters and setters

}

public interface UserRepository extends JpaRepository<User, Long> {

}

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

public List<User> getAllUsers() {

return userRepository.findAll();

}

} **5. Security:**

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.antMatchers("/public/\*\*").permitAll()

.anyRequest().authenticated()

.and()

.formLogin();

}

} **6. Testing:**

@RunWith(SpringRunner.class)

@SpringBootTest

public class MyServiceTest {

@Autowired

private MyService myService;

@Test

public void testSayHello() {

String message = myService.sayHello();

assertEquals("Hello from MyService!", message);

}

}

**7. Configuration:**

application.properties

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

spring.datasource.username=root

spring.datasource.password=secret

**8. Actuator:**

<dependency>

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

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

</dependency>

**9. RESTful Web Services:**

javaCopy code

@RestController public class BookController { @GetMapping("/books") public List<Book> getAllBooks() { // Code to fetch all books from database } @PostMapping("/books") public ResponseEntity<Book> createBook(@RequestBody Book book) { // Code to create a new book } }

**10. Dependency Management:**

<dependency>

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

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

</dependency>

These examples cover basic concepts of Spring Boot. You can build upon them as you delve deeper into Spring Boot development.