SPRING BOOT: A Saga do Multiverso Java

MARCELO WILLIAM

Guia Prático de Spring Boot

Introdução

Spring Boot é um framework baseado no Spring que facilita a criação de aplicativos stand-alone e produção prontos. Ele elimina a necessidade de configurações extensivas, oferecendo uma configuração padrão, permitindo que você comece a desenvolver rapidamente.

01

CONFIGURAÇÃO AUTOMÁTICA

Configuração Automática

Uma das principais vantagens do Spring Boot é a configuração automática. O Spring Boot tenta configurar automaticamente seus componentes com base nas dependências presentes no classpath. Isso elimina a necessidade de escrever configurações tediosas e permite que você se concentre no desenvolvimento de sua aplicação.

Exemplo: Conexão com Banco de Dados

application.yml — YAML

```yaml
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/mydb
    username: user
    password: password
    driver-class-name: com.mysql.cj.jdbc.Driver
```

UserRepository.java — Java

```java
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
}
```

02

DEPENDÊNCIA E GERENCIAMENTO DE PACOTES

Dependência e Gerenciamento de Pacotes com Spring Boot Starter

Os starters do Spring Boot são um conjunto de dependências que você pode adicionar ao seu projeto para incluir um grupo específico de bibliotecas e funcionalidades. Eles são projetados para facilitar a configuração inicial de um projeto.

Exemplo: Spring Boot Starter Web
Para criar uma aplicação web, você pode adicionar a seguinte dependência ao seu pom.xml:

pom.xml XML

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

03

DESENVOLVIMENTO DE APIs RESTful

Desenvolvimento de APIs RESTful

Spring Boot facilita a criação de APIs RESTful, permitindo criar endpoints de forma simples e rápida. A anotação @RestController é usada para marcar uma classe como um controlador que manipula solicitações HTTP e retorna dados JSON.

Exemplo: Endpoint de Usuário
Aqui está um exemplo de como criar um controlador simples para gerenciar usuários:

UserController.java — Java

```java
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/users")
public class UserController {

    private final UserRepository userRepository;

    public UserController(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @GetMapping("/{id}")
    public User getUser(@PathVariable Long id) {
        return userRepository.findById(id).orElseThrow(() -> new UserNotFoundException(id));
    }

    @PostMapping
    public User createUser(@RequestBody User user) {
        return userRepository.save(user);
    }
}
```

04

SPRING DATA JPA

Spring Data JPA

Spring Data JPA simplifica a implementação de repositórios baseados em JPA, reduzindo o boilerplate e proporcionando um modelo de acesso a dados mais simples. Com ele, você pode facilmente interagir com o banco de dados sem precisar escrever consultas SQL.

Exemplo: Entidade e Repositório
Primeiro, defina a entidade que será persistida no banco de dados:

User.java — Java

```java
import javax.persistence.*;

@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;

    // getters and setters
}
```

Spring Data JPA

Em seguida, crie o repositório correspondente:

UserRepository.java — Java

```java
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {
}
```

05

SEGURANÇA COM SPRING SECURITY

Segurança com Spring Security

Spring Security fornece autenticação e controle de acesso robustos. Ele pode ser configurado rapidamente com Spring Boot, permitindo proteger suas APIs e controlar quem pode acessar o que.

Exemplo: Configuração Básica de Segurança
A configuração abaixo mostra como proteger todos os endpoints da aplicação, exceto o endpoint /public, que será acessível sem autenticação:

SecurityConfig.java — Java

```java
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
            .antMatchers("/public").permitAll()
            .anyRequest().authenticated()
            .and()
            .formLogin();
    }
}
```

AGRADECIMENTOS

OBRIGADO POR LER ATÉ AQUI !

Esse ebook foi gerado por IA, e diagramado por um humano.

Esse conteúdo foi gerado com fins de construção, não foi realizado uma validação cuidadosa humana no conteúdo e pode conter erros gerados por uma IA.