

Tutorial: Criando um Aplicativo CRUD com Spring Boot e MySQL

Introdução

Neste tutorial, você aprenderá como criar um aplicativo CRUD (Create, Read, Update, Delete) básico usando Spring Boot e MySQL. Um aplicativo CRUD é essencial para muitos sistemas, pois permite a manipulação de dados de uma maneira intuitiva e eficiente.

Ferramentas Utilizadas

Spring Boot: Um framework Java que facilita a criação de aplicativos Java de maneira rápida e simples. Junto de suas dependências.

H2 Database: Um banco de dados em memória que é amplamente utilizado para desenvolvimento e teste de aplicativos Java. Ele fornece uma solução leve e rápida para armazenamento temporário de dados durante o desenvolvimento.

Maven: Uma ferramenta de automação de compilação e gerenciamento de dependências para projetos Java.

IDE de sua escolha (Eclipse, IntelliJ IDEA, Visual Studio Code etc.): Para desenvolvimento de código. A escolhida neste tutorial foi o Visual Studio Code, por preferência mesmo, mesmo nele sendo necessário a adição de algumas extensões. (Recomendo um pack do próprio spring boot que já configura automático)

POSTMAN: Para testar os endpoints da aplicação.

Passo a Passo Configurações.

1. Configuração do Ambiente

Certifique-se de ter o JDK (Java Development Kit) instalado em sua máquina. Instale uma IDE de sua preferência, se ainda não tiver uma.

2. Criação do Projeto Spring Boot

Abra um navegador da web e vá para o site oficial do Spring Initializr.

No Spring Initializr, você pode configurar seu projeto Spring Boot selecionando as dependências e configurações desejadas. Certifique-se de incluir as dependências Spring Data JPA.

Após configurar seu projeto, clique no botão "Generate" para baixar um arquivo ZIP contendo o esqueleto do seu projeto Spring Boot.]

Extraia o arquivo ZIP em um diretório de sua escolha e abra o projeto em sua IDE.

Segue print do site Spring Initializr, se caso não adicionar dependências neste momento pode adicionar no decorrer do projeto, podendo remover e adicionar como desejar.

The screenshot shows the Spring Initializr web interface. On the left, under 'Project', 'Gradle - Maven' is selected. Under 'Language', 'Java' is selected. Under 'Spring Boot', '3.2.2' is selected. The 'Project Metadata' section contains the following fields: Group (com.example), Artifact (demo), Name (demo), Description (Demo project for Spring Boot), Package name (com.example.demo), Packaging (Jar), and Java version (17). On the right, the 'Dependencies' section is empty with the text 'No dependency selected'. At the bottom, there are three buttons: 'GENERATE CTRL + G', 'EXPLORE CTRL + SPACE', and 'SHARE...'.

Lembrando que a criação do projeto também pode ser feita nas próprias IDE 's, tendo as devidas extensões.

3. Configuração do Banco de Dados

Configure as propriedades do banco de dados no arquivo `application.properties` para se conectar ao seu banco h2.

Certifique-se de ter criado o banco de dados que será utilizado pelo aplicativo.

The screenshot shows the `application.properties` file in an IDE. The file content is as follows:

```
demo > src > main > resources > application.properties
1 spring.datasource.url=jdbc:h2:mem:crudb
2 spring.datasource.driverClassName=org.h2.Driver
3 spring.datasource.username=user
4 spring.datasource.password=password
5 spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
6
7 spring.jpa.hibernate.ddl-auto=update
8 spring.h2.console.enabled=true
9 spring.h2.console.path=/h2-console
10
```

4. Configuração do pom

Como antes comentado sobre as dependências irei deixar as mesmas usadas no projeto. Se caso estiver usando o Visual Studio Code, conforme voce vai acrescentando elas a IDE ja vai construindo o projeto conforme alterado. Voce pode conferir elas no arquivo `pom.xml` do repositório.

5. Criação da Entidade

Crie uma classe Java para representar a entidade que será manipulada pelo aplicativo.

Anote a classe com `@Entity` e adicione outras anotações JPA conforme necessário para mapeamento de banco de dados.

No caso estamos fazendo um CRUD de usuário onde ele possui somente id e nome. E já com esta classe mapeada já é criado também a tabela no nosso banco de memória. Segue print de parte da classe. podendo ver ela completa no repositório:

```
demo > src > main > java > borth > crud > demo > entity > J Usuario.java > ...
1  package borth.crud.demo.entity;
2
3
4  import jakarta.persistence.*;
5
6
7  @Entity
8  @Table(name = "usuario")
9  public class Usuario {
10     @Id
11     @GeneratedValue(strategy = GenerationType.IDENTITY)
12     @Column(name = "id")
13     private Integer id;
14
15     @Column(name = "nome", length = 150)
16     private String nome;
17
```

6. Criação do Repositório

Crie uma interface que estenda JpaRepository ou CrudRepository para a entidade criada anteriormente.

Esta interface fornece métodos para realizar operações CRUD no banco de dados.

Print de exemplo da classe mencionada:

```
1  package borth.crud.demo.repository;
2
3  import borth.crud.demo.entity.Usuario;
4  import org.springframework.data.jpa.repository.JpaRepository;
5  import org.springframework.stereotype.Repository;
6
7  → UsuarioController | ← 3 beans
8  @Repository
9  public interface UsuariosRepository extends JpaRepository<Usuario, Integer> {
10
11 }
```

7. Criação dos Controladores

Crie controladores para lidar com as requisições HTTP relacionadas às operações CRUD. Anote os métodos nos controladores com @PostMapping, @GetMapping, @PutMapping, @DeleteMapping conforme apropriado.

Aqui acrescentamos um pouco de lógica para salvar, ler, alterar e deletar do nosso banco. Podendo ver ela completa no repositório.

```
import org.springframework.data.repository.UserRepository;

// UsuariosRepository
@RestController
@RequestMapping("/usuario") // Define o prefixo para todas as rotas definidas neste controller
public class UsuarioController {

    // UsuariosRepository
    @Autowired
    private UserRepository usuariosRepository;

    http://127.0.0.1:8080/usuario/save (Count=1 Total=0,16s Max=0,00s)
    @PostMapping("/save") // Criar um novo usuário
    @ResponseStatus(HttpStatus.CREATED)
    public Usuario save(@RequestBody Usuario usuario) {
        return usuariosRepository.save(usuario);
    }

    http://127.0.0.1:8080/usuario/{id}
    @GetMapping("/{id}") // Buscar por id
    public Usuario getUsuarioById(@PathVariable Integer id) {
        return usuariosRepository
            .findById(id)
            .orElseThrow(() -> new RuntimeException(HttpStatus.NOT_FOUND,
                "Usuario não encontrado"));
    }
}
```

8. Testando o Aplicativo

Inicie o aplicativo e teste as operações CRUD usando a ferramenta Postman.

Fazendo as requisições no post da seguinte forma:

<http://localhost:8080/> seguido do endpoint

POST	/usuario/save: Cria um novo usuário.
GET	/usuario/{id}: Busca um usuário pelo ID.
PUT	/usuario/update/{id}: Atualiza um usuário existente pelo ID.
DELETE	/usuario/{id}: Deleta um usuário pelo ID.
GET	/usuario/all: Retorna todos os usuários cadastrados.

Conclusão

Neste tutorial, você aprendeu como criar um aplicativo CRUD básico usando Spring Boot. Você agora tem uma base sólida para expandir e personalizar o aplicativo de acordo com suas necessidades específicas.