API RESTful em Java 17 com Spring Boot e STS utilizandoH h2

Este artigo apresenta um guia técnico detalhado para criar uma API RESTful usando **Java 17**, **Spring Boot 3.0**, e o **Spring Tool Suite (STS)**, com suporte para banco de dados em memória H2.

Essa API permitirá gerenciar informações de clientes por meio de operações CRUD (Create, Read, Update, Delete).

Passo 1: Configurando o Projeto no Spring Tool Suite (STS)

1. **Abrir o STS** e selecionar **File > New > Spring Starter Project**.
2. Nomeie o projeto e selecione as dependências essenciais:
   - **Spring Web** (para construir a API REST).
   - **Spring Data JPA** (para interagir com o banco de dados).
   - **H2 Database** (para usar um banco de dados em memória).
3. **Configurar pom.xml**: Confirme se as dependências estão corretas e que a versão do Java é 17:

   xml

```
<properties>
   <java.version>17</java.version>
</properties>
<dependencies>
   <!-- Dependência para API REST -->
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
   </dependency>
   <!-- Dependência para JPA -->
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
   </dependency>
   <!-- Dependência para banco de dados H2 -->
   <dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <scope>runtime</scope>
   </dependency>
</dependencies>
```

Passo 2: Configuração do Banco de Dados no application.properties

Configure as propriedades da aplicação para usar o H2 como banco de dados em memória e habilite o console H2 para visualizar os dados:

properties

```
server.port=8080
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:testdb
```

Passo 3: Criação da Classe Modelo Cliente

A classe Cliente representa o modelo de dados, com mapeamento JPA e validação de campos.

java

```
package com.project.jpa.JavaJPA.model;

import javax.persistence.*;
import javax.validation.constraints.NotNull;

@Entity
public class Cliente {

    // ID único para cada cliente
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String nome;

    @NotNull // Campo obrigatório
    private String email;

    // Getters e Setters para manipulação dos atributos
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getNome() { return nome; }
    public void setNome(String nome) { this.nome = nome; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
}
```

Passo 4: Criação do Repositório Clientes

A interface Clientes extende JpaRepository, facilitando as operações CRUD sem a necessidade de implementar métodos básicos.

Java

```
package com.project.jpa.JavaJPA.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import com.project.jpa.JavaJPA.model.Cliente;

// Repositório para gerenciar as operações CRUD da entidade Cliente
public interface Clientes extends JpaRepository<Cliente, Long> { }
```

Passo 5: Implementação do Controlador ClientesController

O controlador ClientesController define os métodos CRUD e os endpoints para manipulação dos clientes.

java

```
package com.project.jpa.JavaJPA.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import javax.validation.Valid;
import com.project.jpa.JavaJPA.model.Cliente;
import com.project.jpa.JavaJPA.repository.Clientes;
import java.util.List;

@RestController
@RequestMapping("api/v1/clientes")
public class ClientesController {

    @Autowired
    private Clientes clientes;

    // Listar todos os clientes (GET)
    @GetMapping
    public List<Cliente> listar() {
        return clientes.findAll();
    }

    // Adicionar novo cliente (POST)
    @PostMapping
    public Cliente adicionar(@Valid @RequestBody Cliente cliente) {
        return clientes.save(cliente);
    }

    // Buscar cliente por ID (GET)
    @GetMapping("/{id}")
    public ResponseEntity<Cliente> buscar(@PathVariable Long id) {
```

```
        return clientes.findById(id)
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.notFound().build());
    }

    // Atualizar cliente por ID (PUT)
    @PutMapping("/{id}")
    public ResponseEntity<Cliente> atualizar(@PathVariable Long id, @Valid @RequestBody
Cliente cliente) {
        return clientes.findById(id).map(clienteExistente -> {
            cliente.setId(clienteExistente.getId());
            clientes.save(cliente);
            return ResponseEntity.ok(cliente);
        }).orElse(ResponseEntity.notFound().build());
    }

    // Deletar cliente por ID (DELETE)
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deletar(@PathVariable Long id) {
        return clientes.findById(id).map(cliente -> {
            clientes.delete(cliente);
            return ResponseEntity.noContent().build();
        }).orElse(ResponseEntity.notFound().build());
    }
}
```

Exemplos de JSON para Requisições e Respostas

Listar Clientes (GET /api/v1/clientes)

Request:
Sem corpo.

Response (200 OK):

json

```
[
    {
        "id": 1,
        "nome": "João Silva",
        "email": "joao.silva@email.com"
    },
    {
        "id": 2,
        "nome": "Maria Oliveira",
        "email": "maria.oliveira@email.com"
    }
]
```

Adicionar Cliente (POST /api/v1/clientes)

Request:

json
```
{
  "nome": "Ana Costa",
  "email": "ana.costa@email.com"
}
```

Response (201 Created):

json
```
{
  "id": 3,
  "nome": "Ana Costa",
  "email": "ana.costa@email.com"
}
```

Buscar Cliente por ID (GET /api/v1/clientes/{id})

Request:
Sem corpo.

Response (200 OK):

json
```
{
  "id": 1,
  "nome": "João Silva",
  "email": "joao.silva@email.com"
}
```

Atualizar Cliente (PUT /api/v1/clientes/{id})

Request:

json
```
{
  "nome": "João Santos",
  "email": "joao.santos@email.com"
}
```

Response (200 OK):

json
```
{
  "id": 1,
  "nome": "João Santos",
  "email": "joao.santos@email.com"
}
```

Deletar Cliente (DELETE /api/v1/clientes/{id})

Request:
Sem corpo.

Response (204 No Content):
Sem corpo.

Com este guia passo a passo, você construiu uma API RESTful básica em Java 17 com Spring Boot, usando o Spring Tool Suite e o banco de dados H2.