

Web Services - REST e CRUD

Sistemas Distribuídos e Mobile

Prof. Me. Gustavo Torres Custódio

gustavo.custodio@anhembi.br

Conteúdo

CRUD com REST

Netbeans



CRUD com REST

Web Services ~ REST e CRUD

CRUD com REST

- Criamos um exemplo de API básica na aula anterior.
 - Agora precisamos adicionar **persistência** em nossa API.
 - Armazenar elementos em banco de dados e acessá-los.

CRUD com REST

- CRUD é uma sigla que designa quatro operações em bancos de dados relacionais:
 - Criação (Create);
 - Leitura (Read);
 - Atualização (Update);
 - Remoção (Delete).

CRUD com REST

- Vamos fazer com que nossa API realize as operações CRUD em um banco de dados.



Web Services ~ REST e CRUD

Netbeans

API REST com CRUD

- Vamos criar uma API REST que realiza operações de CRUD.
- Vamos cadastrar e consultar os nomes de usuários.

API REST com CRUD

- A API terá 3 funcionalidades (*endpoints*):
 - Cadastrar um usuário.
 - Listar todos os usuários.
 - Listar um usuário específico.

API REST com CRUD

- Utilizaremos o Postman para requisições do tipo POST.
 - <https://www.postman.com/>




API REST com CRUD

- Entre no *Spring Initializr* para criar um projeto (verifique a versão correta do Java).
- Adicione as dependências:
 - *Spring Web*.
 - *DevTools*.

API REST com CRUD

- *Spring Data JPA - Java Persistence API*
 - responsável por armazenamento em bancos de dados.
- *H2 Database*
 - H2 é um banco de dados relacional escrito em Java, fácil de integrar com projetos em Java (executado na memória).

API REST com CRUD



Project
☒ Maven Project
☐ Gradle Project

Language
☒ Java ☐ Kotlin
☐ Groovy

Spring Boot
☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (M5) ☐ 2.7.5 (SNAPSHOT)
☐ 2.7.4 ☐ 2.6.13 (SNAPSHOT) ☒ 2.6.12

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Dependencies ADD DEPENDENCIES... CTRL + B

Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Boot DevTools DEVELOPER TOOLS

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Spring Data JPA SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.



H2 Database SQL

Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

GENERATE CTRL + ⌘

EXPLORE CTRL + SPACE

SHARE...



API REST com CRUD

- Crie uma classe chamada Usuario dentro de um arquivo Usuario.java.

```
@Entity
@Table(name = "usuario")
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column
    private String nome;
```

API REST com CRUD

- A anotação `Entity` serve para relacionar o modelo com o banco de dados
- A anotação `Table` define que os dados serão armazenados na tabela “user”.
 - Neste caso, uma tabela que possui um **campo id** e um **campo nome**.
- `GeneratedValue` estabelece que cada elemento da tabela possui um **id único** e esse id é gerado automaticamente.

API REST com CRUD

- Vamos gerar também os **getters** e **setters** para os atributos da tabela.

```
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;  
}
```


API REST com CRUD

- Para realizar operações CRUD em uma entidade Usuario, é necessário criar um **repositório** para Usuario.
 - Crie um arquivo chamado UsuarioRepository.java:

```
public interface UsuarioRepository extends CrudRepository<Usuario, Long>{  
}
```

API REST com CRUD

- Por último, precisamos criar o **controlador**, onde cada um dos **endpoints** serão criados.
- Crie um arquivo chamado `RestcrudController.java`.

API REST com CRUD

```
@RestController
public class RestcrudController {
    @Autowired
    private UsuarioRepository repositórioUsuario;

    @GetMapping("/usuarios")
    public Iterable<Usuario> listarTodosUsuarios() {
        // Implementar
        return null;
    }

    @GetMapping("/usuario/{id}")
    public ResponseEntity<Usuario> buscarUsuarioPorId(@PathVariable long id) {
        // Implementar
        return null;
    }

    @PostMapping("/novo_usuario")
    public Usuario salvarUsuario(@Validated @RequestBody Usuario usuario) {
        // Implementar
        return null;
    }
}
```

API REST com CRUD

- Neste caso, temos três *endpoints*:
 - Listar todos os usuários (/usuarios).
 - Listar somente um usuário (/usuario/{id}).
 - Adicionar um usuário (/novousuario).
- GetMapping define *endpoints* que aceitam requisições do tipo GET.
- PostMapping define *endpoints* que aceitam requisições do tipo POST.

API REST com CRUD

- Primeiro vamos adicionar um novo usuário.

```
@PostMapping("/novousuario")
public Usuario salvarUsuario(@Validated @RequestBody Usuario usuario) {
    return repositorioUsuario.save(usuario);
}
```

API REST com CRUD

- Vamos utilizar o Postman para adicionar um usuário em nosso banco de dados.
- Enviamos uma requisição em formato JSON do **tipo POST**.
- Quando o usuário é adicionado, a API retorna o id e o nome do usuário adicionado em formato JSON.

API REST com CRUD

The screenshot displays a REST client interface with the following components:

- URL Bar:** Shows the endpoint `localhost:8080/novousuario`. Buttons for `Save` and `Send` are present.
- Method:** A dropdown menu is set to `POST`.
- Body Tab:** The `Body` tab is selected, showing a JSON payload:

```
1 {  
2   "nome": "Gustavo"  
3 }
```
- Format Selection:** Radio buttons for `none`, `form-data`, `x-www-form-urlencoded`, `raw`, `binary`, and `GraphQL` are available. The `JSON` format is selected.
- Response Section:** Below the request body, the `Body` tab is selected, showing the response:

```
1 {  
2   "id": 2,  
3   "nome": "Gustavo"  
4 }
```
- Status Bar:** Indicates `Status: 200 OK`, `Time: 30 ms`, and `Size: 189 B`. A `Save Response` button is also visible.

API REST com CRUD

- Agora listamos todos os usuários no banco de dados.

```
@GetMapping("/usuarios")  
public Iterable<Usuario> listarTodosUsuarios() {  
    return repositorioUsuario.findAll();  
}
```

- O método `findAll` é um método do `CrudRepository` que retorna todos os elementos em uma tabela.

API REST com CRUD

- Todos os elementos da lista são mostrados acessando a URL:
 - localhost:8080/usuarios

```
[  
  {"id":1, "nome":"Gustavo"},  
  {"id":2, "nome":"João"},  
  {"id":3, "nome":"Silva"}  
]
```

API REST com CRUD

- Por último, implementamos a última rota que busca um usuário pelo id.

```
@GetMapping("/usuario/{id}")
public ResponseEntity<Usuario> buscarUsuarioPorId(@PathVariable long id) {
    Optional<Usuario> usuario = repositorioUsuario.findById(id);

    if (usuario.isPresent()) {
        return ResponseEntity.ok().body(usuario.get());
    } else {
        return ResponseEntity.notFound().build();
    }
}
```

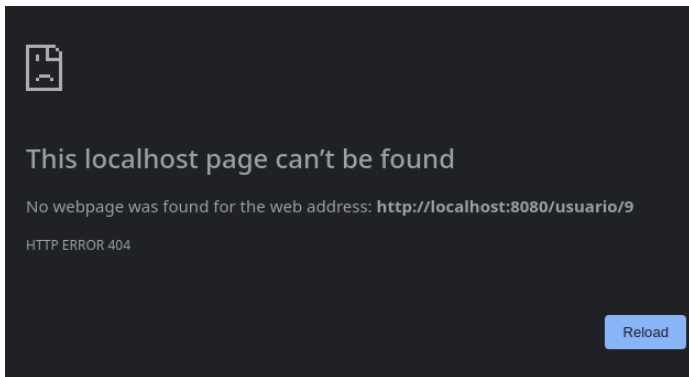
API REST com CRUD

- Caso o usuário seja encontrado, o *endpoint* devolve as informações do usuário em formato JSON.

```
{"id":2,"nome":"João"}
```

API REST com CRUD

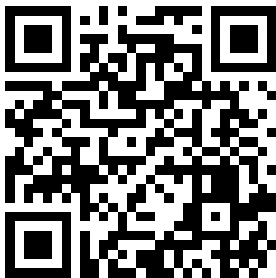
- Caso o usuário não seja encontrado, o *endpoint* devolve um erro HTTP 404 (página não encontrada).



Referências

- <https://www.infoq.com/minibooks/emag-03-2010-rest>
- <http://blog.obeautifulcode.com/API/Learn-REST-In-18-Slides/>
- <http://courses.ischool.berkeley.edu/i290-rmm/s12/slides/Lecture3%20REST.pdf>
- <https://hevodata.com/learn/spring-boot-rest-api/>

Conteúdo



<https://gustavotcustodio.github.io/sdmobile.html>

Obrigado

gustavo.custodio@anhembi.br