

Curso de Jogos Digitais

Disciplina de Tecnologias Web

Aula 16

Integração do Back-end com Front-end



Professor: André Flores dos Santos



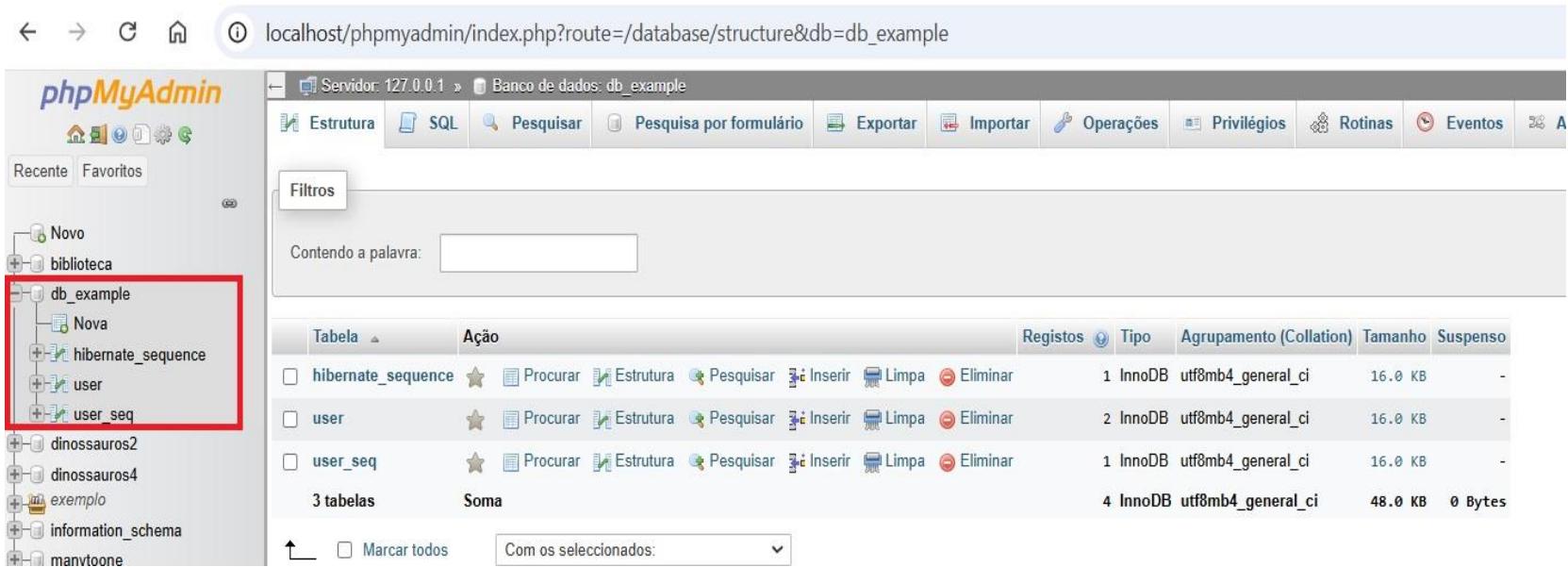
Para fazermos a integração Front-end e Back-end devemos ter alguns itens prontos:

- 1) O banco de dados estar criado e online, no nosso caso MariaDb.
 - User: root
 - Pass:
 - Porta: 3306
- 2) O back-end implementado e rodando na porta 8080, no nosso caso vamos implementar 3 endPoints:
 - <http://localhost:8080/hello> para mostrar a msg hello world
 - <http://localhost:8080/demo/all> para buscar os usuários
 - <http://localhost:8080/hello/add> para adicionar os usuários
- 3) Fron-end implementado com HTML/css e Javascript com alguma tecnologia para usar o protocolo HTTP, ou algum tipo de Framework que também utilize de preferência protocolo HTTP.

Arquivos do projeto: https://github.com/andreflores2009/TecnologiasWeb_SpringBoot.git

No banco de dados devemos criar uma base de dados.

Nome ‘db_example’ ou o que desejar, porém em outros lugares deve lembrar este nome para configurar, um deles é dentro do projeto no Spring boot arquivo ‘application.Properties’.



The screenshot shows the phpMyAdmin interface for the database 'db_example'. A red box highlights the database structure under the 'db_example' schema, which includes tables 'Nova', 'hibernate_sequence', 'user', and 'user_seq'. The main table list shows three tables: 'hibernate_sequence', 'user', and 'user_seq', all of which are InnoDB type with utf8mb4_general_ci collation. The 'hibernate_sequence' table has 1 record, 'user' has 2 records, and 'user_seq' has 1 record, totaling 4 records across 3 tables.

| Tabela | Ação | Registros | Tipo | Agrupamento (Collation) | Tamanho | Suspenso |
|--------------------|---|-----------|--------|-------------------------|---------|----------|
| hibernate_sequence | Procurar Estrutura Pesquisar Inserir Limpa Eliminar | 1 | InnoDB | utf8mb4_general_ci | 16.0 KB | - |
| user | Procurar Estrutura Pesquisar Inserir Limpa Eliminar | 2 | InnoDB | utf8mb4_general_ci | 16.0 KB | - |
| user_seq | Procurar Estrutura Pesquisar Inserir Limpa Eliminar | 1 | InnoDB | utf8mb4_general_ci | 16.0 KB | - |

Configurar o 'Application Properties' dentro do Projeto do SpringBoot

Caminho src/main/resources/application.properties.xml

Colocamos as configurações e credenciais para conectar no banco de dados MariaDb

User: root

Port:3306

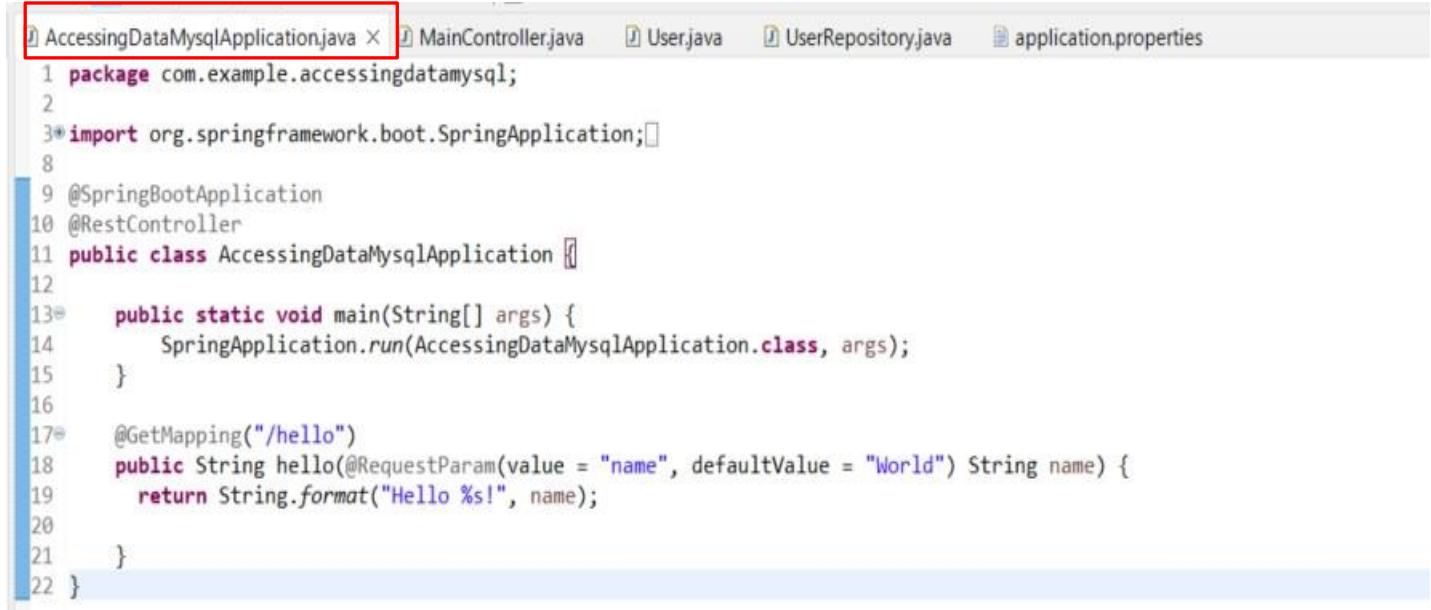
Pass:

The screenshot shows the Eclipse IDE interface with the following details:

- Package Explorer:** Shows the project structure with a red box highlighting the `application.properties` file under the `src/main/resources` directory.
- Editor View:** Displays the contents of the `application.properties` file. The configuration for connecting to MariaDB is highlighted with a red box:

```
1 spring.jpa.hibernate.ddl-auto=update
2 spring.datasource.url=jdbc:mysql://localhost:3306/db_example
3 spring.datasource.username=root
4 spring.datasource.password=
5 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
6 spring.jpa.show-sql: true
7 spring.jpa.hibernate.ddl-auto=update
```

Vamos relembrar os códigos do nosso projeto



The screenshot shows a Java code editor with the following file structure:

- AccessingDataMySQLApplication.java (highlighted with a red border)
- MainController.java
- User.java
- UserRepository.java
- application.properties

The content of AccessingDataMySQLApplication.java is:

```
1 package com.example.accessingdatamysql;
2
3 import org.springframework.boot.SpringApplication;
4
5 @SpringBootApplication
6 @RestController
7 public class AccessingDataMySQLApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(AccessingDataMySQLApplication.class, args);
11     }
12
13     @GetMapping("/hello")
14     public String hello(@RequestParam(value = "name", defaultValue = "World") String name) {
15         return String.format("Hello %s!", name);
16     }
17 }
```

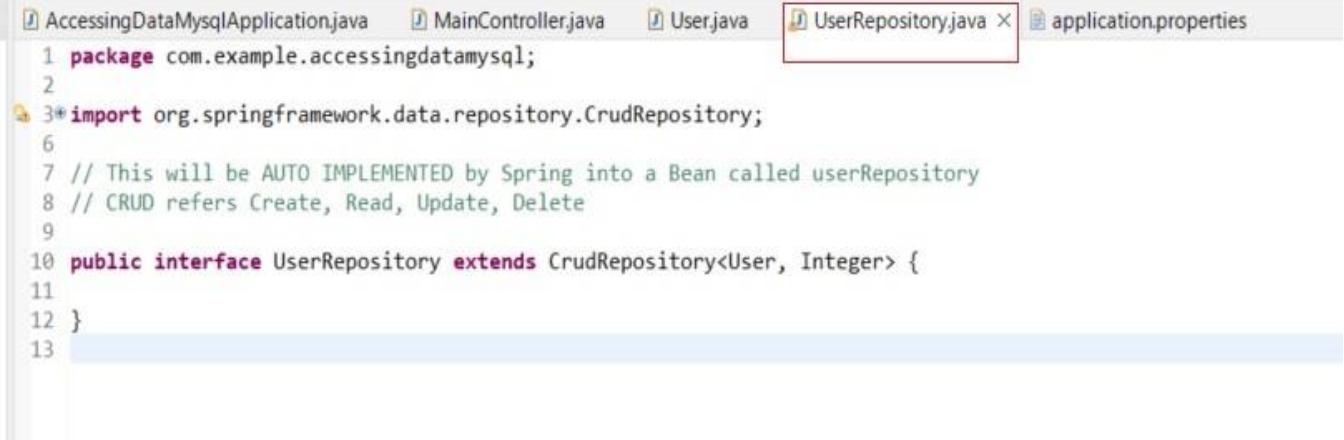
Vamos relembrar os códigos do nosso projeto

```
1 AccessingDataMySQLApplication.java  2 MainController.java X  3 User.java  4 UserRepository.java  5 application.properties
 1 package com.example.accessingdatamysql;
 2
 3 import org.springframework.beans.factory.annotation.Autowired;□
 4
 5 //para acessar via http com os arquivos Front-end fora da pasta do projeto
 6 @CrossOrigin(origins = "*") // Permite requisições de qualquer origem
 7
 8
 9 @Controller // This means that this class is a Controller
10 @RequestMapping(path="/demo") // This means URL's start with /demo (after Application path)
11 public class MainController {
12     @Autowired // This means to get the bean called userRepository
13         // Which is auto-generated by Spring, we will use it to handle the data
14     private UserRepository userRepository;
15
16     @PostMapping(path="/add") // Map ONLY POST Requests
17     public @ResponseBody String addNewUser (@RequestParam String name
18             , @RequestParam String email) {
19         // @ResponseBody means the returned String is the response, not a view name
20         // @RequestParam means it is a parameter from the GET or POST request
21
22         User n = new User();
23         n.setName(name);
24         n.setEmail(email);
25         userRepository.save(n);
26         return "Saved";
27     }
28
29     @GetMapping(path="/all")
30     public @ResponseBody Iterable<User> getAllUsers() {
31         // This returns a JSON or XML with the users
32         return userRepository.findAll();
33     }
34 }
```

Vamos relembrar os códigos do nosso projeto

```
1 AccessingDataMySQLApplication.java  2 MainController.java  3 User.java X  4 UserRepository.java  5 application.properties
 1 package com.example.accessingdatamysql;
 2
 3 import jakarta.persistence.Entity;
 4
 5 @Entity // This tells Hibernate to make a table out of this class
 6 public class User {
 7
 8     @Id
 9     @GeneratedValue(strategy=GenerationType.IDENTITY) // Usa auto-incremento para primary key no banco de dados
10     private Integer id;
11
12     private String name;
13
14     private String email;
15
16     public Integer getId() {
17         return id;
18     }
19
20     public void setId(Integer id) {
21         this.id = id;
22     }
23
24     public String getName() {
25         return name;
26     }
27
28     public void setName(String name) {
29         this.name = name;
30     }
31
32     public String getEmail() {
33         return email;
34     }
35
36     public void setEmail(String email) {
37         this.email = email;
38     }
39
40 }
41
42 <
```

Vamos relembrar os códigos do nosso projeto

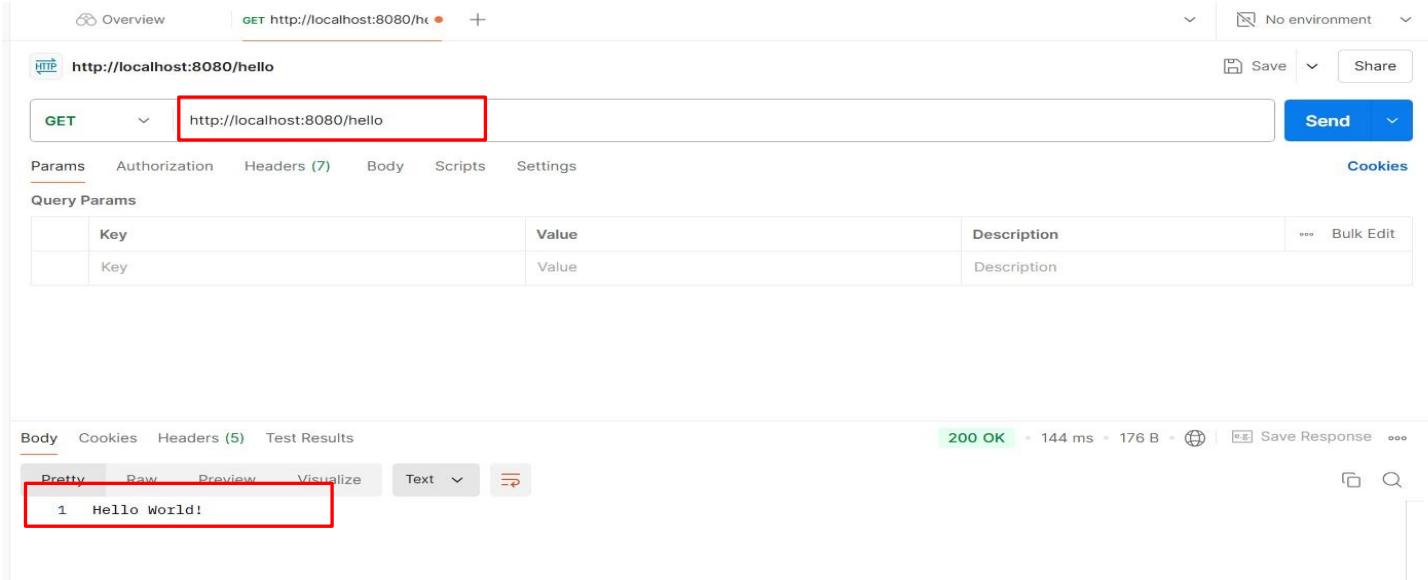


```
AccessingDataMySQLApplication.java  MainController.java  User.java  UserRepository.java X  application.properties
1 package com.example.accessingdatamysql;
2
3 import org.springframework.data.repository.CrudRepository;
4
5 // This will be AUTO IMPLEMENTED by Spring into a Bean called userRepository
6 // CRUD refers Create, Read, Update, Delete
7
8 public interface UserRepository extends CrudRepository<User, Integer> {
9
10 }
```

Feito isso vamos testar através de requisições HTTP, via postaman ou de forma manual no navegador.

Lembrar de rodar o projeto antes e verificar se não tem erros e deixar o banco de dados ‘on’!

- Vamos testar com uma requisição ‘hello world’: <http://localhost:8080/hello>



The screenshot shows the Postman application interface. At the top, it displays the URL `GET http://localhost:8080/hello`. Below the URL input field, there is a red box highlighting the URL itself. To the right of the URL input is a blue "Send" button. Underneath the URL input, there are tabs for "Params", "Authorization", "Headers (7)", "Body", "Scripts", and "Settings". The "Headers (7)" tab is currently selected. Below these tabs is a section titled "Query Params" with a table:

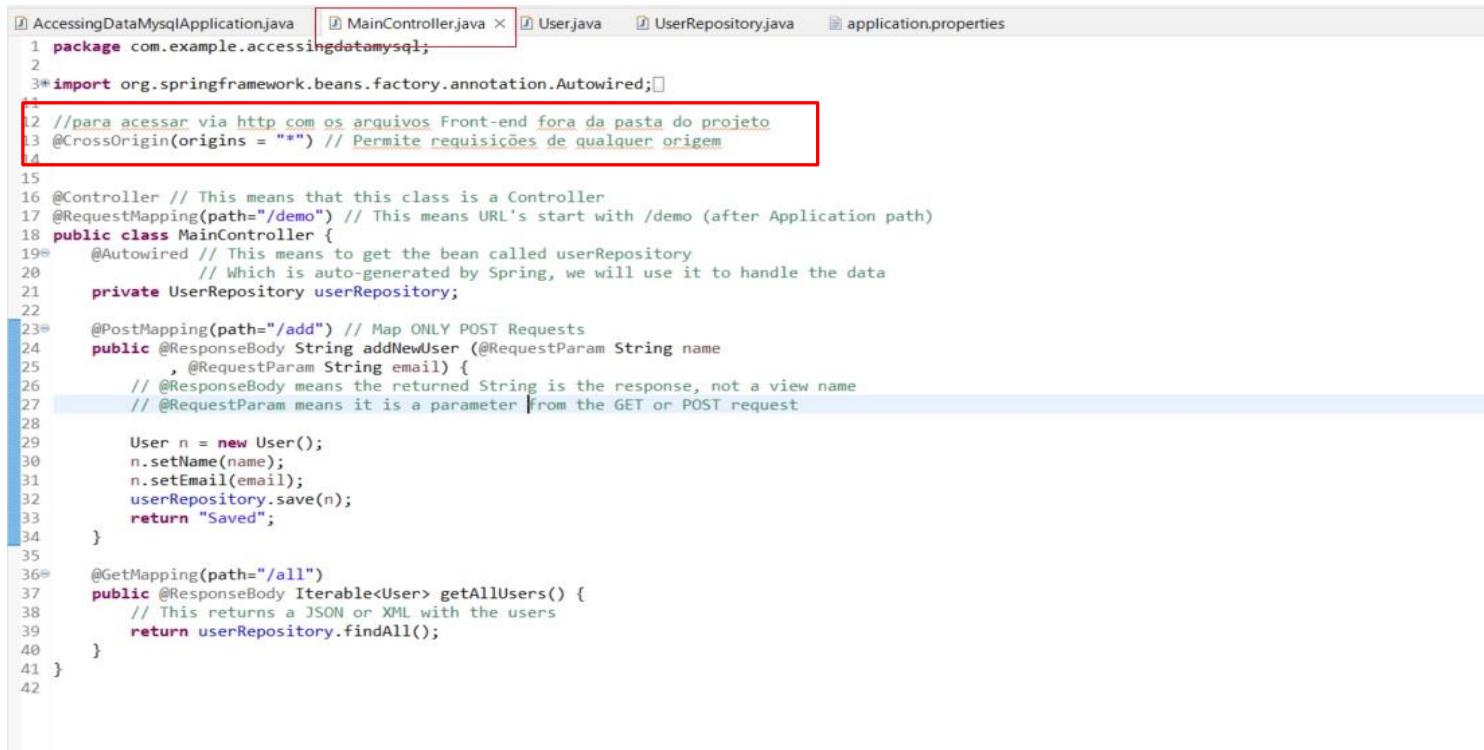
| Key | Value | Description | ... | Bulk Edit |
|-----|-------|-------------|-----|-----------|
| Key | Value | Description | ... | |

At the bottom of the interface, there is a "Body" tab which is currently active, indicated by a blue underline. The body content area shows the response from the server:

```
1 Hello World!
```

Agora vamos fazer a programação HTML+css+Javascript para enviar e receber dados nos end points que temos programados.

OBS: No Spring Boot se a pasta com os arquivos do Front-end estiver fora da pasta do projeto devemos configurar o ‘@CrossOrigin(origins = “*”) dentro do ‘MainController’.



```
AccessingDataMySQLApplication.java MainController.java ✘ User.java UserRepository.java application.properties

1 package com.example.accessingdatamysql;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 //para acessar via http com os arquivos Front-end fora da pasta do projeto
6 @CrossOrigin(origins = "*") // Permite requisições de qualquer origem
7
8
9
10 @Controller // This means that this class is a Controller
11 @RequestMapping(path="/demo") // This means URL's start with /demo (after Application path)
12 public class MainController {
13     @Autowired // This means to get the bean called userRepository
14         // Which is auto-generated by Spring, we will use it to handle the data
15     private UserRepository userRepository;
16
17     @PostMapping(path="/add") // Map ONLY POST Requests
18     public @ResponseBody String addNewUser (@RequestParam String name
19             , @RequestParam String email) {
20         // @ResponseBody means the returned String is the response, not a view name
21         // @RequestParam means it is a parameter from the GET or POST request
22
23         User n = new User();
24         n.setName(name);
25         n.setEmail(email);
26         userRepository.save(n);
27         return "Saved";
28     }
29
30
31     @GetMapping(path="/all")
32     public @ResponseBody Iterable<User> getAllUsers() {
33         // This returns a JSON or XML with the users
34         return userRepository.findAll();
35     }
36
37 }
```

Agora vamos ver nosso código HTML

```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <meta charset="UTF-8">
    <title>Gerenciamento de Usuários</title>
    <!-- Link para o arquivo CSS externo -->
    <link rel="stylesheet" href="styles.css">
  </head>
  <body>
    <!-- Formulário para adicionar um novo usuário -->
    <h1>Adicionar Novo Usuário</h1>
    <form id="addUserForm">
      <label for="name">Nome:</label>
      <input type="text" id="name" name="name" required>
      <br>
      <label for="email">E-mail:</label>
      <input type="email" id="email" name="email" required>
      <br>
      <button type="submit">Adicionar Usuário</button>
    </form>

    <!-- Botão para obter todos os usuários -->
    <h1>Lista de Usuários</h1>
    <button id="getAllUsersBtn">Obter Todos os Usuários</button>
    <!-- Tabela para exibir os usuários -->
    <table id="usersTable">
      <thead>
        <tr>
          <th>ID</th>
          <th>Nome</th>
          <th>E-mail</th>
        </tr>
      </thead>
      <tbody>
        <!-- Linhas dos usuários serão inseridas aqui via JavaScript -->
      </tbody>
    </table>

    <!-- Script JavaScript externo -->
    <script src="script.js"></script>
  </body>
</html>
```

Agora vamos ver nosso código CSS

```
1  /* Estilos básicos para o corpo da página */
2  body {
3      font-family: Arial, sans-serif;
4      margin: 20px;
5  }
6
7  /* Estilo para os títulos */
8  h1 {
9      color: #333;
10 }
11
12 /* Estilo para o formulário */
13 form {
14     margin-bottom: 20px;
15 }
16
17 /* Estilo para os labels e inputs do formulário */
18 label {
19     display: inline-block;
20     width: 50px;
21 }
22
23 input {
24     margin-bottom: 10px;
25 }
26
27 /* Estilo para a tabela de usuários */
28 table {
29     width: 100%;
30     border-collapse: collapse;
31 }
32
33 /* Estilo para o cabeçalho da tabela */
34 th, td {
35     border: 1px solid #ccc;
36     padding: 8px;
37     text-align: left;
38 }
39
40 /* Estilo para linhas alternadas da tabela */
41 tbody tr:nth-child(even) {
42     background-color: #f9f9f9;
43 }
```

Agora vamos ver nosso código JavaScript

```
// URL base do backend
const backendUrl = 'http://localhost:8080';

// Seleciona o formulário de adicionar usuário
const addUserForm = document.getElementById('addUserForm');
// Seleciona o botão de obter todos os usuários
const getAllUsersBtn = document.getElementById('getAllUsersBtn');
// Seleciona o corpo da tabela onde os usuários serão inseridos
const usersTableBody = document.querySelector('#usersTable tbody');

// Adiciona um listener para o evento de envio do formulário
addUserForm.addEventListener('submit', function(event) {
    event.preventDefault(); // Evita o envio padrão do formulário

    // Obtém os valores dos campos do formulário
    const name = document.getElementById('name').value;
    const email = document.getElementById('email').value;

    // Envia uma requisição POST para adicionar um novo usuário
    fetch(`${backendUrl}/demo/add`, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/x-www-form-urlencoded',
        },
        body: `name=${encodeURIComponent(name)}&email=${encodeURIComponent(email)}`,
    })
    .then(response => response.text())
    .then(data => {
        alert(data); // Exibe uma mensagem de sucesso
        addUserForm.reset(); // Limpa o formulário
    })
    .catch(error => {
        console.error('Erro:', error);
    });
});

// Adiciona um listener para o botão de obter todos os usuários
getAllUsersBtn.addEventListener('click', function() {
    // Envia uma requisição GET para obter todos os usuários
    fetch(`${backendUrl}/demo/all`)
    .then(response => response.json())
    .then(users => {
        // Limpa o conteúdo atual da tabela
        usersTableBody.innerHTML = '';

        // Itera sobre cada usuário e cria uma nova linha na tabela
        users.forEach(user => {
            const row = document.createElement('tr');

            // Cria as células para ID, nome e e-mail
            const idCell = document.createElement('td');
            idCell.textContent = user.id;

            const nameCell = document.createElement('td');
            nameCell.textContent = user.name;

            const emailCell = document.createElement('td');
            emailCell.textContent = user.email;

            // Adiciona as células à linha
            row.appendChild(idCell);
            row.appendChild(nameCell);
            row.appendChild(emailCell);

            // Adiciona a linha ao corpo da tabela
            usersTableBody.appendChild(row);
        });
    })
    .catch(error => {
        console.error('Erro:', error);
    });
});
});
```

Pronto já podemos abrir o ‘index.html’ e testar uma consulta ou adicionar um usuário para verificar se funciona. OBS: Lembrar de deixar a aplicação rodando, o banco de dados ‘on’. Se não houver usuários cadastrados no banco de dados, cadastrar.

Adicionar Novo Usuário

Nome:

E-mail:

Lista de Usuários

| ID | Nome | E-mail |
|----|-------|-------------------------|
| 1 | Andre | andre.santos@ufn.edu.br |
| 2 | Ana | ana@ufn.edu.br |

Existe a opção de colocar os arquivos ‘Front-end’ dentro do Projeto Spring Boot ao invés de deixar eles numa pasta em separado

Se a pasta ‘static’ não existir, crie ela e carregue os arquivos do Front-end para dentro.

Passos para Colocar os Arquivos Dentro do Projeto:

1. Coloque os arquivos na pasta correta:

- `index.html`, `styles.css` e `script.js` devem ser colocados na pasta `src/main/resources/static` do seu projeto Spring Boot.



Existe a opção de colocar os arquivos ‘Front-end’ dentro do Projeto Spring Boot

Atualize as end-points dentro do arquivo ‘script.js’, pois agora vamos rodar dentro do projeto.
E comente ou retire o backendUrl.

2. Atualize o código JavaScript (script.js):

- Como a página será servida pela mesma origem que o backend, você pode usar URLs relativas nas chamadas `fetch`.
- Remova ou comente a linha que define a variável `backendUrl`.

javascript

Copiar código

```
// const backendUrl = 'http://localhost:8080'; // Remova esta Linha
```

- Atualize as chamadas `fetch` para usar URLs relativas:

javascript

Copiar código

```
// Envia uma requisição POST para adicionar um novo usuário
fetch('/demo/add', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/x-www-form-urlencoded',
    },
    body: `name=${encodeURIComponent(name)}&email=${encodeURIComponent(email)}`,
})
```

E

javascript

Copiar código

```
// Envia uma requisição GET para obter todos os usuários
fetch('/demo/all')
```

Agora vamos testar

Agora como os arquivos Front-end estão dentro do projeto podemos acessar da seguinte forma:

<http://localhost:8080/index.html> no navegador.

Lembre-se de reiniciar a aplicação e o banco de dados para não ter problemas antes de testar.

Resultado

Conseguimos rodar o Front-end com os arquivos dentro do projeto.



Adicionar Novo Usuário

Nome:

E-mail:

[Adicionar Usuário](#)

Lista de Usuários

[Obter Todos os Usuários](#)

| ID | Nome | E-mail |
|----|-------|-------------------------|
| 1 | Andre | andre.santos@ufn.edu.br |
| 2 | Ana | ana@ufn.edu.br |

Resultado

Agora é com você construa a sua ferramenta do Trabalho Final!!!!

Vamos construir o método delete e atualizar dentro desse projeto?

Quais são os passos?

Vamos começar a adaptar o back-end e depois o front-end

Modificar o Controlador MainController.java

Vamos criar o método atualizar e deletar.

Adicionar no Controlador ‘MainController.java’

Método atualizar:

```
// Endpoint para atualizar um usuário existente
@PutMapping(path = "/update/{id}") // Mapeia uma requisição PUT para /demo/update/{id}
public @ResponseBody String updateUser(@PathVariable Integer id, @RequestParam String name, @RequestParam
String email) {
    // Busca o usuário pelo ID; se não encontrar, retorna uma mensagem de erro
    User user = userRepository.findById(id).orElse(null); //pode retornar 'null'
    if (user == null) {
        return "Usuario não encontrado!!";
    }

    // Atualiza o nome e o email do usuário existente
    user.setName(name);
    user.setEmail(email);

    userRepository.save(user); // Salva as alterações no banco de dados

    // Retorna uma mensagem de sucesso
    return "Usuário Atualizado com sucesso!";
}
```

Adicionar no Controlador ‘MainController.java’

Método atualizar:

Explicação das Novas Funcionalidades:

Atualização de Usuário (updateUser):

Rota: PUT /demo/update/{id}

Parâmetros:

@PathVariable Integer id: O ID do usuário a ser atualizado.

@RequestParam String name e @RequestParam String email: Novos valores para name e email.

Funcionamento:

O método tenta buscar um usuário com o ID especificado.

Se o usuário existir, ele atualiza o nome e o email com os novos valores e salva as alterações no banco de dados.

Se o usuário não for encontrado, retorna a mensagem "User not found".

Adicionar no Controlador ‘MainController.java’

Método deletar:

```
// Endpoint para deletar um usuário existente
@DeleteMapping(path = "/delete/{id}") // Mapeia uma requisição DELETE para
/demo/delete/{id}
public @ResponseBody String deleteUser(@PathVariable Integer id) {
// Verifica se o usuário existe pelo ID; se não existir, retorna uma mensagem de erro
if (!userRepository.existsById(id)) {
return "Usuario não encontrado";
}

// Deleta o usuário pelo ID
userRepository.deleteById(id);

// Retorna uma mensagem de sucesso
return "Deletado com sucesso!!!";
}
```

Adicionar no Controlador ‘[MainController.java](#)’

Método deletar:

Exclusão de Usuário (deleteUser):

Rota: DELETE /demo/delete/{id}

Parâmetro:

@PathVariable Integer id: O ID do usuário a ser deletado.

Funcionamento:

- O método verifica se o usuário com o ID especificado existe no banco de dados.
- Se existir, o método deleta o usuário.
- Se não existir, retorna a mensagem "User not found".

Vamos testar pelo Postman um update e um delete para ver se funciona

Testar o Endpoint de Atualização (PUT)

Método: **PUT**

URL: <http://localhost:8080/demo/update/{id}>

Substitua {id} pelo ID do usuário que deseja atualizar (por exemplo,
<http://localhost:8080/demo/update/1>).

insira os parâmetros:

name: o novo nome do usuário (por exemplo, "Carlos")

email: o novo email do usuário (por exemplo, "carlos@example.com")

Descrição: Esse endpoint atualiza o name e email do usuário com o ID especificado. O backend retornará "User not found" se o usuário não existir.

Vamos testar pelo Postman um update

Resultado para id=1

The screenshot shows the Postman interface with a successful API call. The URL is `PUT localhost:8080/demo/update/1?name=Paula&email=paula@ufn.edu.br`. The request body contains the following parameters:

| Key | Value | Description |
|---|------------------|-------------|
| <input checked="" type="checkbox"/> name | Paula | |
| <input checked="" type="checkbox"/> email | paula@ufn.edu.br | |
| <input type="checkbox"/> id | 1 | |
| Key | Value | Description |

The response status is 200 OK, with a response time of 214 ms and a response size of 285 B. The response body is: "1 Usuário Atualizado com sucesso!" (User updated successfully!).

Vamos testar pelo Postman um update e um delete para ver se funciona

Testar o Endpoint de Deleção (DELETE)

Método: **DELETE**

URL: <http://localhost:8080/demo/delete/{id}>

Substitua {id} pelo ID do usuário que deseja deletar

<http://localhost:8080/demo/delete/1>

Descrição: Esse endpoint deleta o usuário com o ID especificado. Retorna "User not found" se o usuário não existir.

Vamos testar pelo Postman um Delete

Resultado para id=7 (verificar se o id existe no banco)

The screenshot shows the Postman interface with the following details:

- Method:** DELETE
- URL:** localhost:8080/demo/delete/7
- Query Params:** id = 1
- Response Body:** 1 Deletado com sucesso!!!

Pronto nosso back-end está funcionando!! Vamos atualizar o nosso front-end

Resultado

Adicionar/Atualizar Usuário

Nome:

E-mail:

Lista de Usuários

| ID | Nome | E-mail | Ações | |
|----|----------|--------------------|---------------------------------------|--|
| 1 | Paula | paula@ufn.edu.br | <input type="button" value="Editar"/> | <input type="button" value="Excluir"/> |
| 2 | Ana | ana@ufn.edu.br | <input type="button" value="Editar"/> | <input type="button" value="Excluir"/> |
| 3 | Carolina | carol@ufn.edu.br | <input type="button" value="Editar"/> | <input type="button" value="Excluir"/> |
| 5 | Gabriel | gabriel@ufn.edu.br | <input type="button" value="Editar"/> | <input type="button" value="Excluir"/> |
| 8 | Maria | maria@ufn.edu.br | <input type="button" value="Editar"/> | <input type="button" value="Excluir"/> |

Tarefa: Tente executar todos os passos vistos na aula de hoje e deixar o projeto funcionando.

Na próxima aula teremos o trabalho final.

Referências

- COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. Distributed Systems: Concepts and Design. 4. ed. Addison-Wesley Publishers, 2002.
- TANENBAUM, Andrew S. Distributed Operating Systems. Prentice-Hall, 1995.
- TANENBAUM, Andrew S. Sistemas Distribuídos: Princípios e Paradigmas. 2. ed. Prentice-Hall, 2007.
- DANTAS, Mario. Computação Distribuída de Alto Desempenho- Redes, Grids e Clusters Computacionais. 2. ed. Editora Axcel Books. 2005.
- JALOTE, Pankaj. Fault tolerance in distributed systems. New Jersey: Prentice-Hall, 1998. 432 p.
- NUTT, Gary J. Operating Systems: a modern perspective. Reading: Addison-Wesley, 1997.
- SILBERSCHATZ, Abraham; GALVIN, Peter Baer. Operating system concepts. 5. ed. Reading: Addison - Wesley, 1998.
- TOSCANI, Simão S.; OLIVEIRA, Rômulo S. de; CARISSIMI, Alexandre da S. Sistemas operacionais e programação concorrentes. Porto Alegre: Sagra Luzzatto, 2003.
- Adaptação de aulas Professor Guilherme Kurtz, 2023.
- <https://br.freepik.com/fotos-vetores-gratis/sistemas-de-informacao>
- <https://www.postman.com/>

Thank you for your attention!!



Email: andre.flores@ufn.edu.br