Curso de Jogos Digitais

Disciplina de Tecnologias Web

Aula 16

Integração do Back-end com Front-end

Professor: André Flores dos Santos

Santa Maria – RS 2025

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

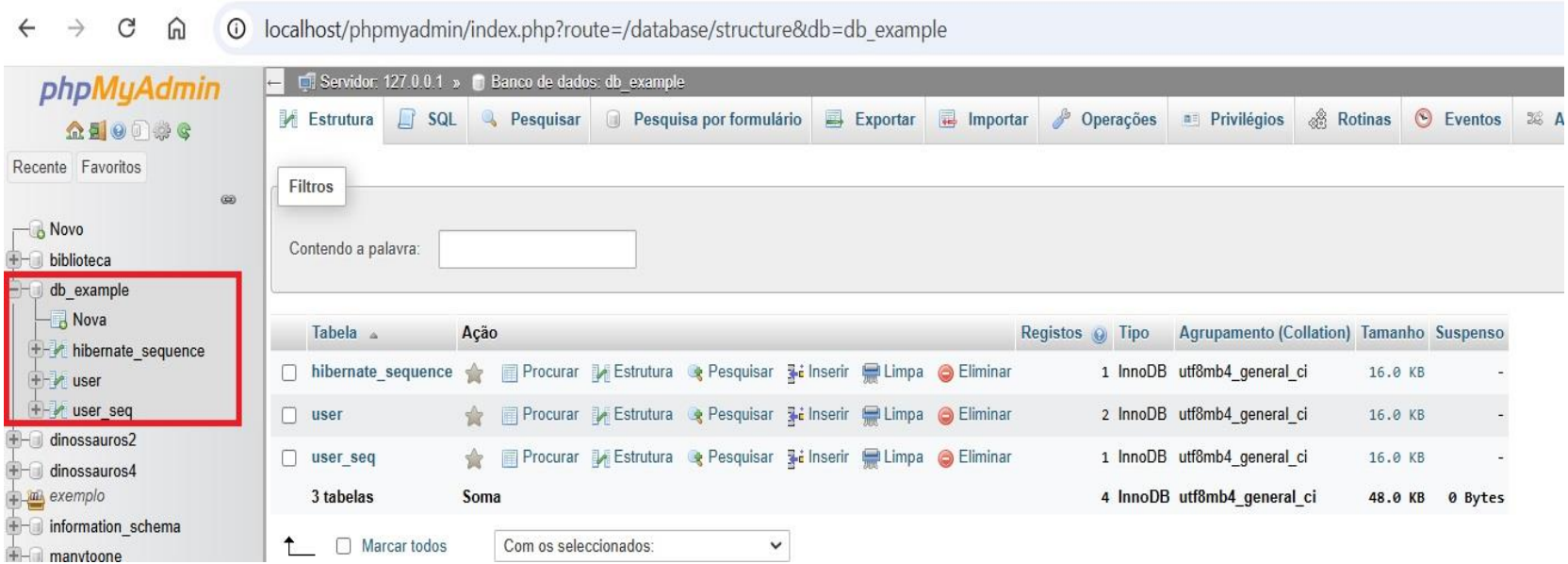
2) O back-end implementado e rodando na porta 8080, no nosso caso vamos implementar 3 endPoints:
   - http://localhost:8080/hello para mostrar a msg hello world
   - http://localhost:8080/demo/all para buscar os usuários
   - http://localhost:8080/hello/add para adicionar os usuários

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.

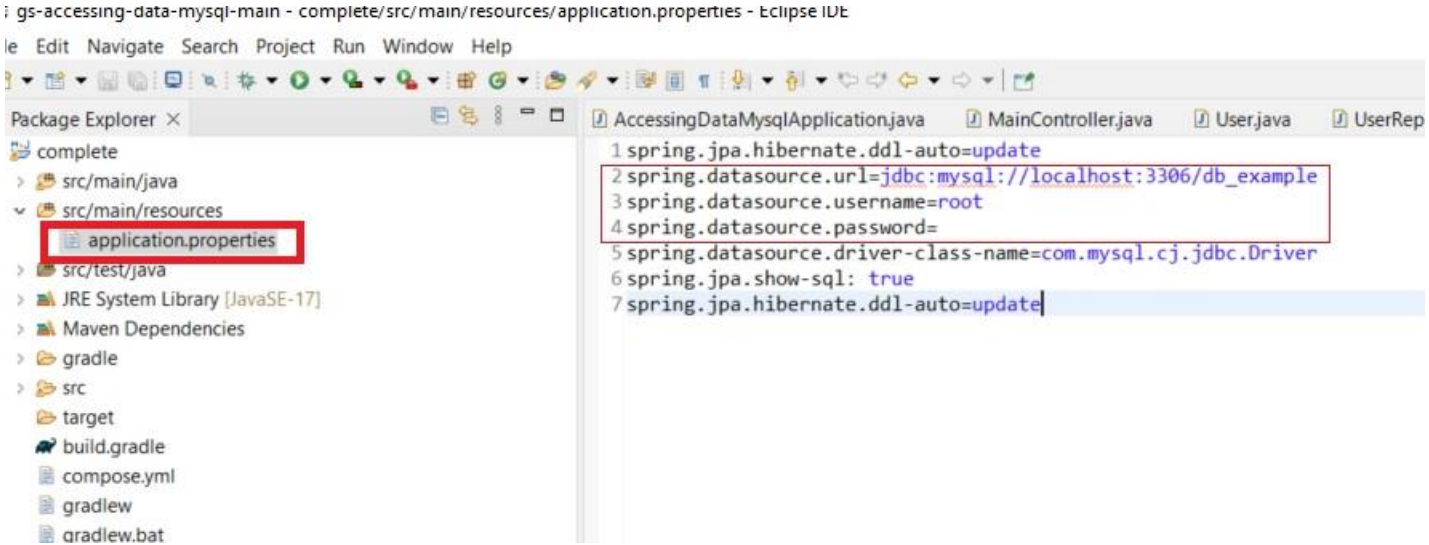
Nome ‘db_example’ ou o que desejar, porém em outros lugares deve lembrar este nome para configurar, um deles é dentro do projeto no Spring boot arquivo ‘application.Properties’.



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:



Vamos relembrar os códigos do nosso projeto

AccessingDataMysqlApplication.java | MainController.java | User.java | UserRepository.java | application.properties

```java
package com.example.accessingdatamysql;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
@RestController
public class AccessingDataMysqlApplication {

    public static void main(String[] args) {
        SpringApplication.run(AccessingDataMysqlApplication.class, args);
    }

    @GetMapping("/hello")
    public String hello(@RequestParam(value = "name", defaultValue = "World") String name) {
      return String.format("Hello %s!", name);

    }
}
```

Vamos relembrar os códigos do nosso projeto

AccessingDataMysqlApplication.java | MainController.java | User.java | UserRepository.java | application.properties

```java
package com.example.accessingdatamysql;

import org.springframework.beans.factory.annotation.Autowired;

//para acessar via http com os arquivos Front-end fora da pasta do projeto
@CrossOrigin(origins = "*") // Permite requisições de qualquer origem


@Controller // This means that this class is a Controller
@RequestMapping(path="/demo") // This means URL's start with /demo (after Application path)
public class MainController {
    @Autowired // This means to get the bean called userRepository
               // Which is auto-generated by Spring, we will use it to handle the data
    private UserRepository userRepository;

    @PostMapping(path="/add") // Map ONLY POST Requests
    public @ResponseBody String addNewUser (@RequestParam String name
            , @RequestParam String email) {
        // @ResponseBody means the returned String is the response, not a view name
        // @RequestParam means it is a parameter from the GET or POST request

        User n = new User();
        n.setName(name);
        n.setEmail(email);
        userRepository.save(n);
        return "Saved";
    }

    @GetMapping(path="/all")
    public @ResponseBody Iterable<User> getAllUsers() {
        // This returns a JSON or XML with the users
        return userRepository.findAll();
    }
}
```

Vamos relembrar os códigos do nosso projeto

AccessingDataMysqlApplication.java | MainController.java | User.java | UserRepository.java | application.properties

```java
package com.example.accessingdatamysql;

import jakarta.persistence.Entity;

@Entity // This tells Hibernate to make a table out of this class
public class User {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY) // Usa auto-incremento para primary key no banco de dados
    private Integer id;

    private String name;

    private String email;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }

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

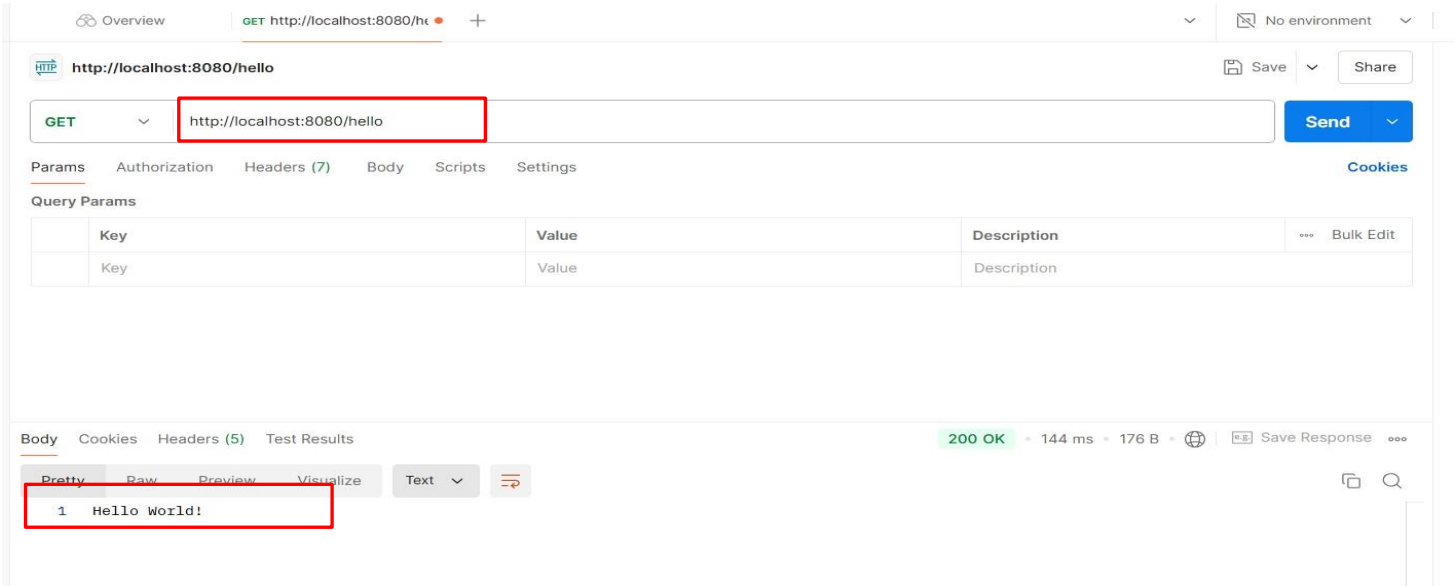
Vamos relembrar os códigos do nosso projeto

AccessingDataMysqlApplication.java | MainController.java | User.java | UserRepository.java | application.properties

```java
package com.example.accessingdatamysql;

import org.springframework.data.repository.CrudRepository;

// This will be AUTO IMPLEMENTED by Spring into a Bean called userRepository
// CRUD refers Create, Read, Update, Delete

public interface UserRepository extends CrudRepository<User, Integer> {

}
```

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



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

```java
package com.example.accessingdatamysql;

import org.springframework.beans.factory.annotation.Autowired;

//para acessar via http com os arquivos Front-end fora da pasta do projeto
@CrossOrigin(origins = "*") // Permite requisições de qualquer origem


@Controller // This means that this class is a Controller
@RequestMapping(path="/demo") // This means URL's start with /demo (after Application path)
public class MainController {
    @Autowired // This means to get the bean called userRepository
               // Which is auto-generated by Spring, we will use it to handle the data
    private UserRepository userRepository;

    @PostMapping(path="/add") // Map ONLY POST Requests
    public @ResponseBody String addNewUser (@RequestParam String name
            , @RequestParam String email) {
        // @ResponseBody means the returned String is the response, not a view name
        // @RequestParam means it is a parameter from the GET or POST request

        User n = new User();
        n.setName(name);
        n.setEmail(email);
        userRepository.save(n);
        return "Saved";
    }

    @GetMapping(path="/all")
    public @ResponseBody Iterable<User> getAllUsers() {
        // This returns a JSON or XML with the users
        return userRepository.findAll();
    }
}
```

Agora vamos ver nosso código HTML

```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

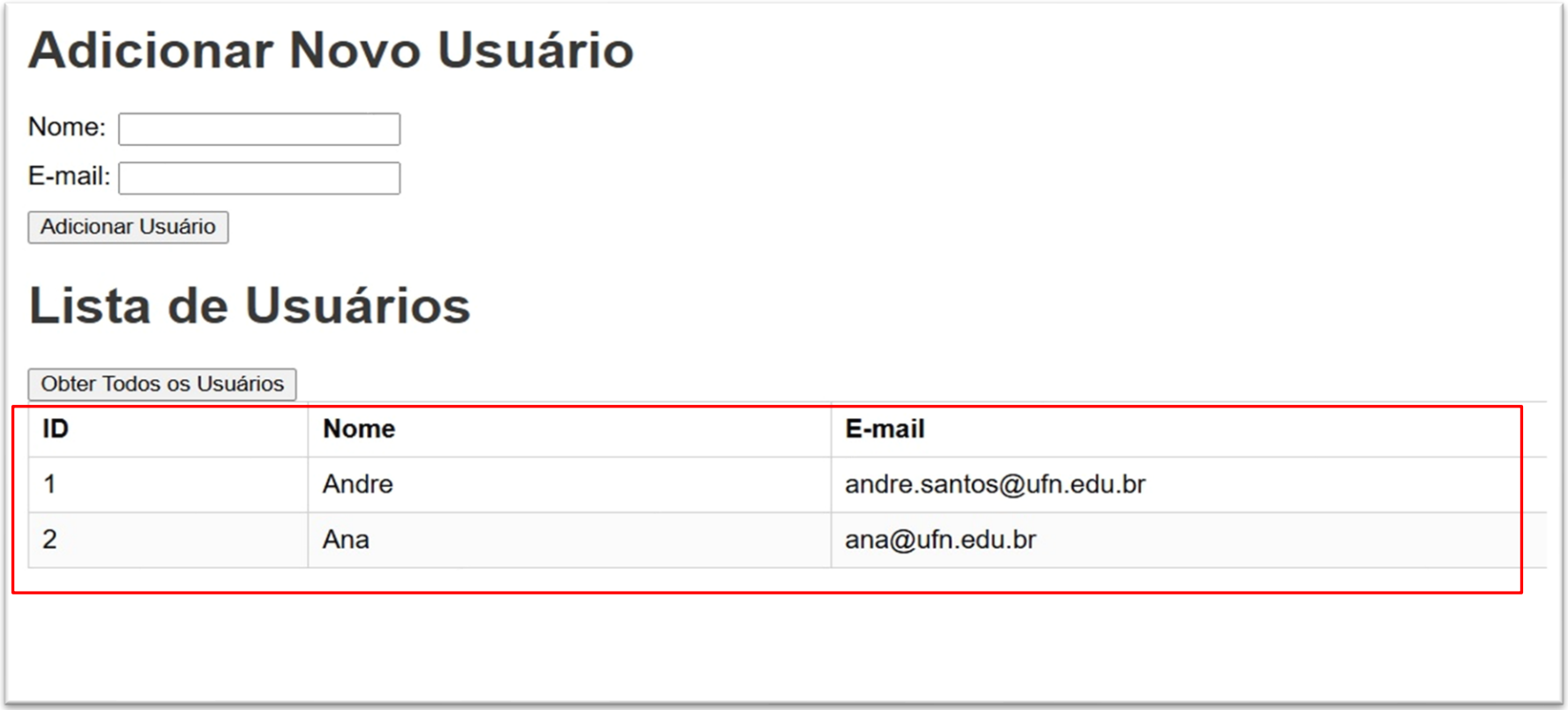
```css
/* Estilos básicos para o corpo da página */
body {
    font-family: Arial, sans-serif;
    margin: 20px;
}

/* Estilo para os títulos */
h1 {
    color: #333;
}

/* Estilo para o formulário */
form {
    margin-bottom: 20px;
}

/* Estilo para os labels e inputs do formulário */
label {
    display: inline-block;
    width: 50px;
}

input {
    margin-bottom: 10px;
}

/* Estilo para a tabela de usuários */
table {
    width: 100%;
    border-collapse: collapse;
}

/* Estilo para o cabeçalho da tabela */
th, td {
    border: 1px solid #ccc;
    padding: 8px;
    text-align: left;
}

/* Estilo para linhas alternadas da tabela */
tbody tr:nth-child(even) {
    background-color: #f9f9f9;
```

Agora vamos ver nosso código JavaScript

```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);
    });
});
```

```javascript
// 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.



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.

scss — Copiar código

```
seu-projeto/
├── src/
│   ├── main/
│   │   ├── java/ (código Java)
│   │   └── resources/
│   │       ├── application.properties
│   │       └── static/
│   │           ├── index.html
│   │           ├── styles.css
│   │           └── script.js
```

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

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

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

Agora vamos testar

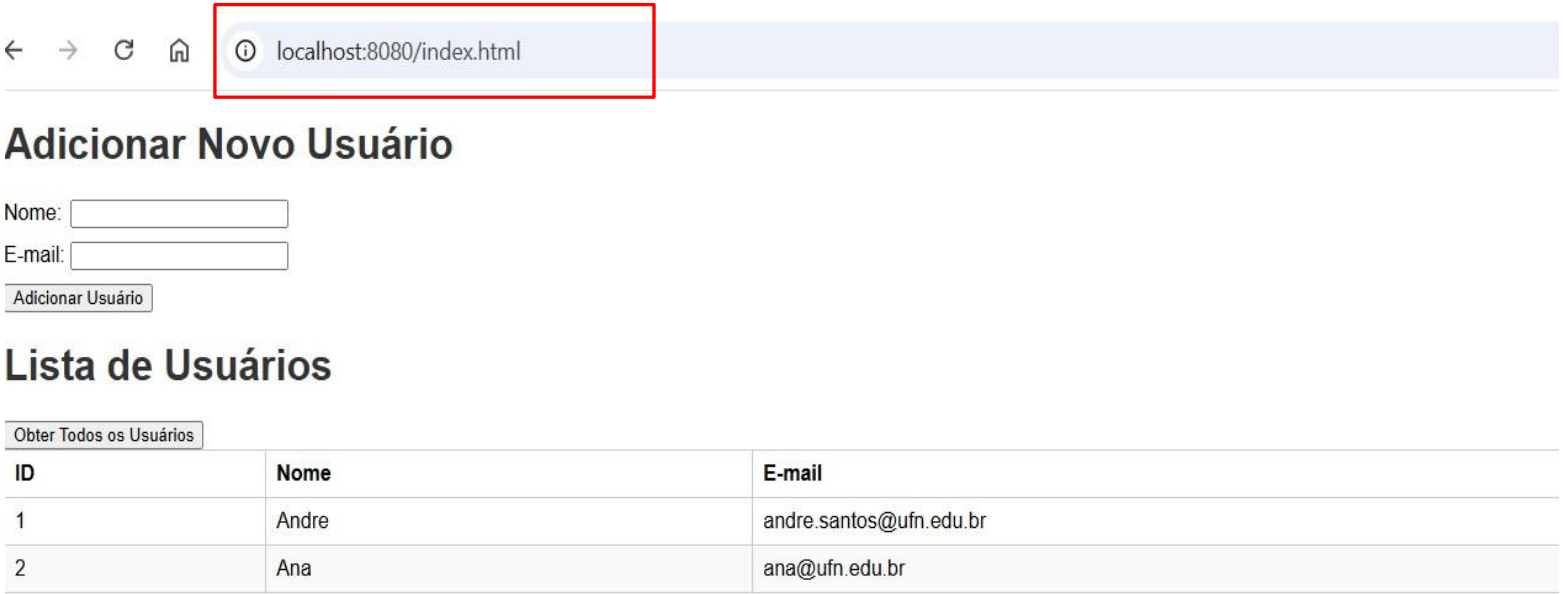
Agora como os arquivos Front-end estão dentro do projeto podemos acessar da seguinte forma:

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

Lembre-se de reinicar 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.



Resultado

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

Desafio

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)

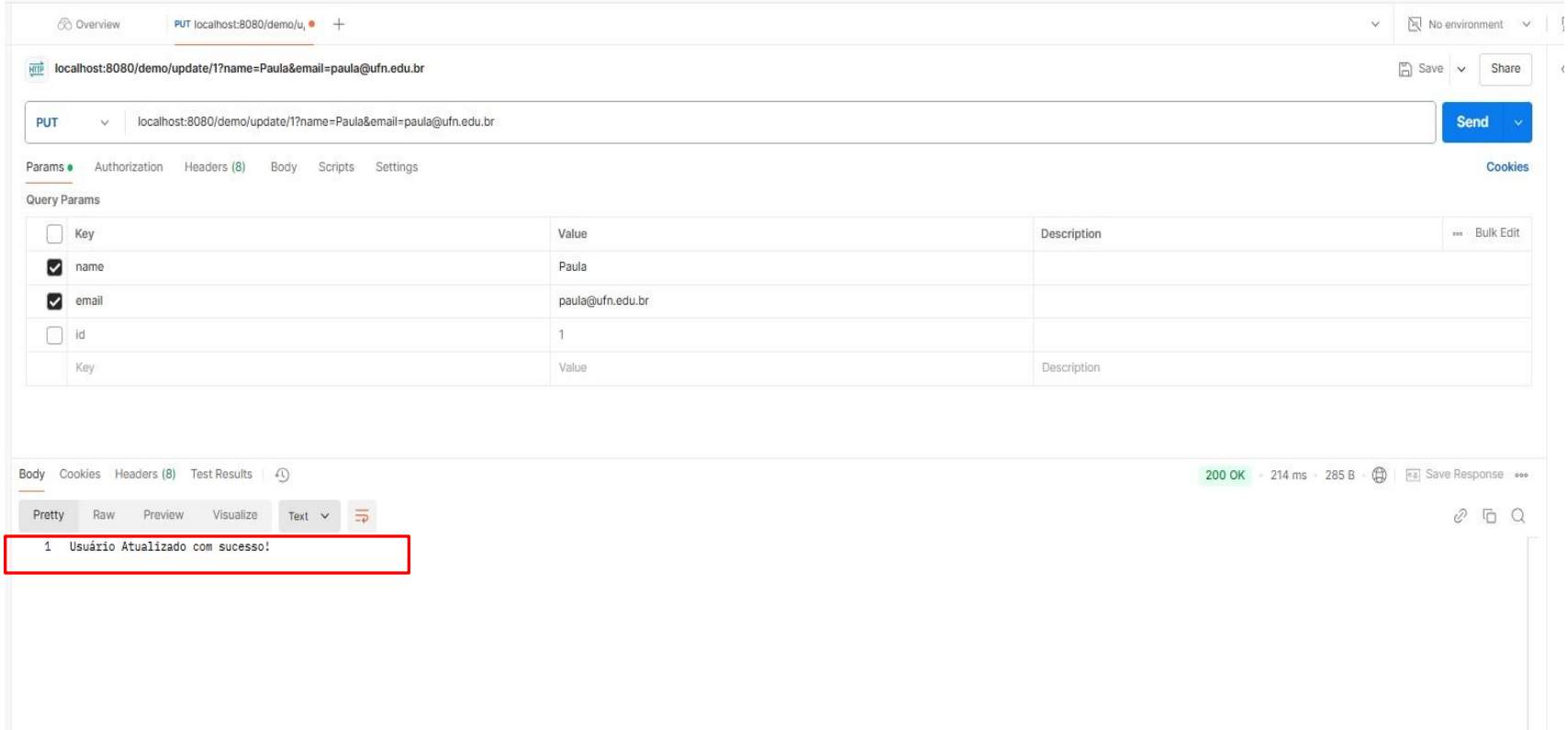
Método: PUT
URL: http://localhost:8080/demo/update/{id}

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



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

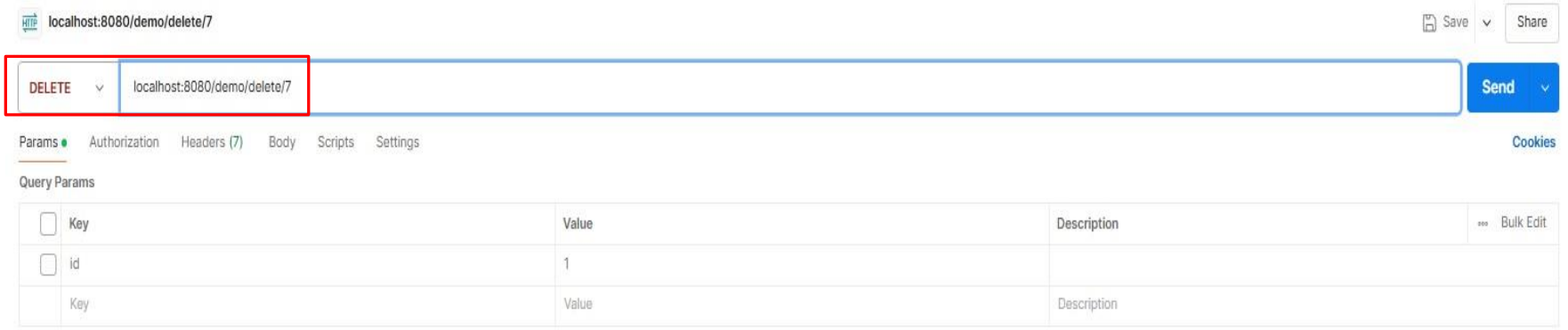
Testar o Endpoint de Deleção (DELETE)
Método: DELETE

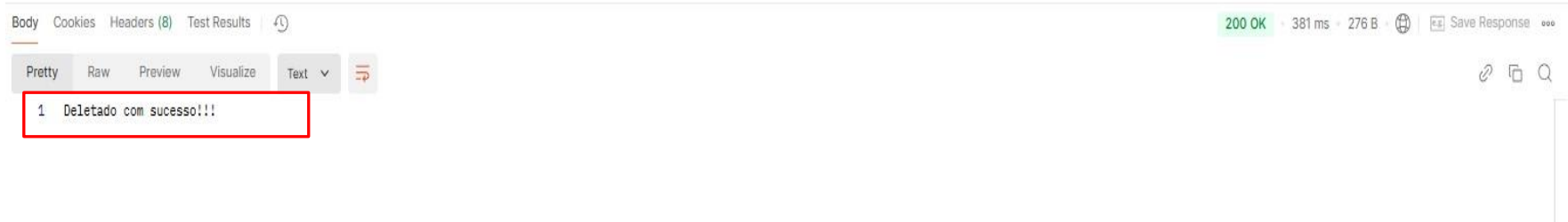
URL: http://localhost:8080/demo/delete/{id}

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)





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

Resultado

Adicionar/Atualizar Usuário

Nome:
E-mail:
Salvar Usuário

Lista de Usuários

Obter Todos os Usuários

ID	Nome	E-mail	Ações	
1	Paula	paula@ufn.edu.br	Editar	Excluir
2	Ana	ana@ufn.edu.br	Editar	Excluir
3	Carolina	carol@ufn.edu.br	Editar	Excluir
5	Gabriel	gabriel@ufn.edu.br	Editar	Excluir
8	Maria	maria@ufn.edu.br	Editar	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**