Integração do Front-End em React com Back-End

Primeiro Workshop da Computação

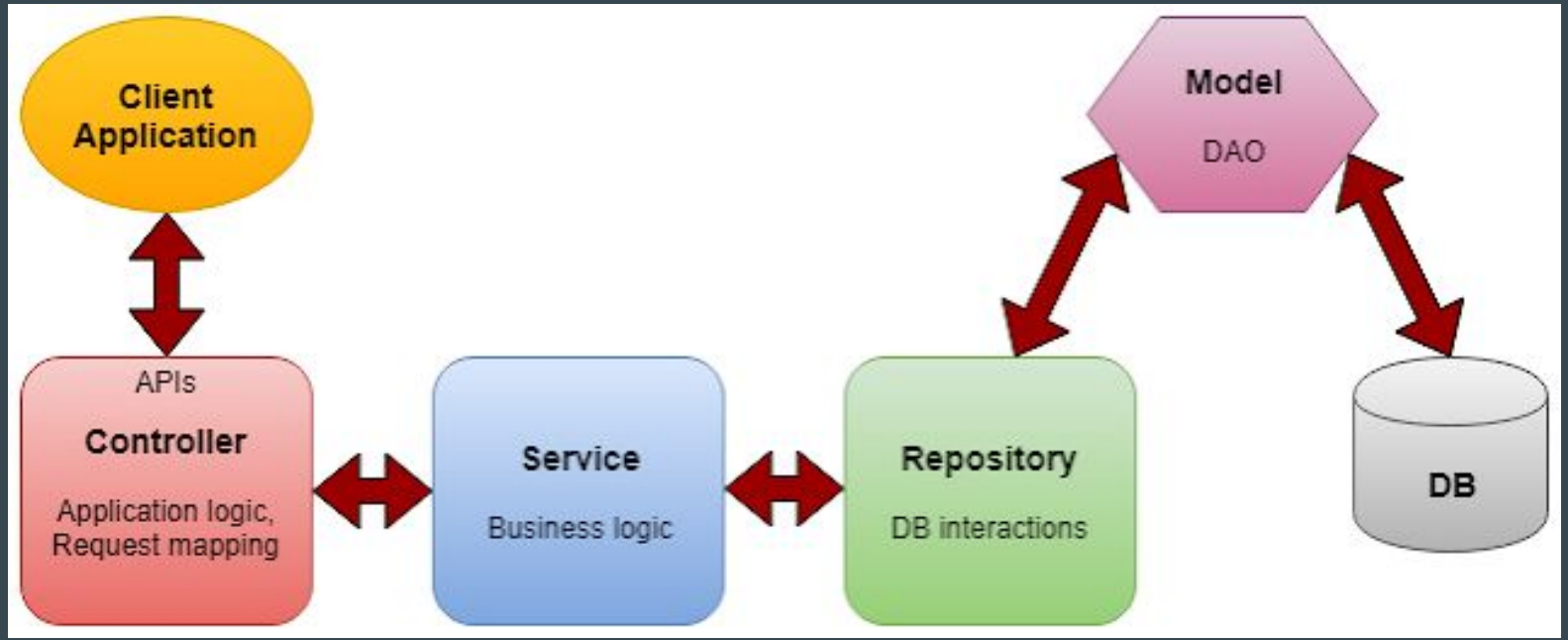
Vinícius Eduardo de Souza Honório
Luiz Felipe dos Santos Nogueira

Conteúdo do Primeiro Dia

- Introdução à Arquitetura do Spring Boot 3
- Visão Geral do Projeto Back-End
- Apresentação da Biblioteca ReactJS
- Visão Geral da Estrutura de Pastas e do Funcionamento do React

Introdução ao Projeto Base Back-End

Arquitetura do Spring Boot



Schema da Entidade Produtos

```
CREATE TABLE products (
                        id INT NOT NULL AUTO_INCREMENT,
                        name varchar(100) NOT NULL,
                        description varchar(255) NULL,
                        price float NOT NULL,
                        creation DATE DEFAULT NULL,
                        PRIMARY KEY (id)
);
```

DTO da Entidade Produto

```java
@AllArgsConstructor @NoArgsConstructor
@Getter @Setter
public class ProductsDTO {
    private Long id;
    @NotNull(message = "products.error.nameIsNull") @NotBlank(message = "products.error.nameIsNull")
    @Size(min = 3, max = 100, message = "products.error.nameSize")
    private String name;
    @Size(max = 255, message = "products.error.description")
    private String description;
    @Min(value = 0, message = "products.error.price")
    private Float price;
    @JsonProperty(access = JsonProperty.Access.READ_ONLY)
    private LocalDate creation;

    4 usages  lipenog
    public ProductsDTO(Products products){
        this.id = products.getId();
        this.name = products.getName();
        this.description = products.getDescription();
        this.price = products.getPrice();
        this.creation = products.getCreation();
    }
}
```

Products Controller

```java
@CrossOrigin
@RestController
public class ProductsController {
    8 usages
    private final ProductsService productsService;

    lipenog
    @Autowired
    public ProductsController(ProductsService productsService) { this.productsService = productsService; }

    lipenog *
    @PostMapping("/products")
    public ResponseEntity<ProductsDTO> productsPost(@RequestBody ProductsDTO productsDTO) throws InvalidDtoException {
        List<String> violations = verifyDTO(productsDTO);
        if(!violations.isEmpty()){
            throw new InvalidDtoException(violations);
        }
        productsDTO.setId(null);
        Products products = productsService.saveProduct(productsDTO);
        ProductsDTO response = new ProductsDTO(products);
        return new ResponseEntity<>(response, HttpStatus.CREATED);
    }
```

Products Service

```java
@Service
public class ProductsService {
    5 usages
    private final ProductsRepository productsRepository;

    lipenog
    @Autowired
    public ProductsService(ProductsRepository productsRepository) { this.productsRepository = productsRepository; }

    2 usages  lipenog
    public Products saveProduct(ProductsDTO productsDTO){
        Products productsEntity = new Products(productsDTO);
        return productsRepository.save(productsEntity);
    }
```

Products Repository

```java
@Repository
public interface ProductsRepository extends JpaRepository<Products, Long> {
}
```

Products Entity

```java
@Entity
@NoArgsConstructor @AllArgsConstructor
@Getter @Setter
public class Products {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String description;
    private Float price;
    private LocalDate creation;
    1 usage  lipenog
    public Products(ProductsDTO productsDTO){
        this.id = productsDTO.getId();
        this.name = productsDTO.getName();
        this.description = productsDTO.getDescription();
        this.price = productsDTO.getPrice();
    }

    lipenog
    @PrePersist @PreUpdate
    private void setCreationDate() { this.creation = LocalDate.now(ZoneId.of( zoneId: "UTC")); }

}
```

Introdução ao Projeto Base Front-End

Apresentação da Biblioteca ReactJS

Conceitos Básicos

ReactJS é uma biblioteca desenvolvida pelo Facebook, especialmente utilizada para construção de aplicações de página única (SPA)

NodeJS é um ambiente de execução JavaScript fora do navegador e de alta performance. Geralmente é utilizado para construção de aplicações Back-End

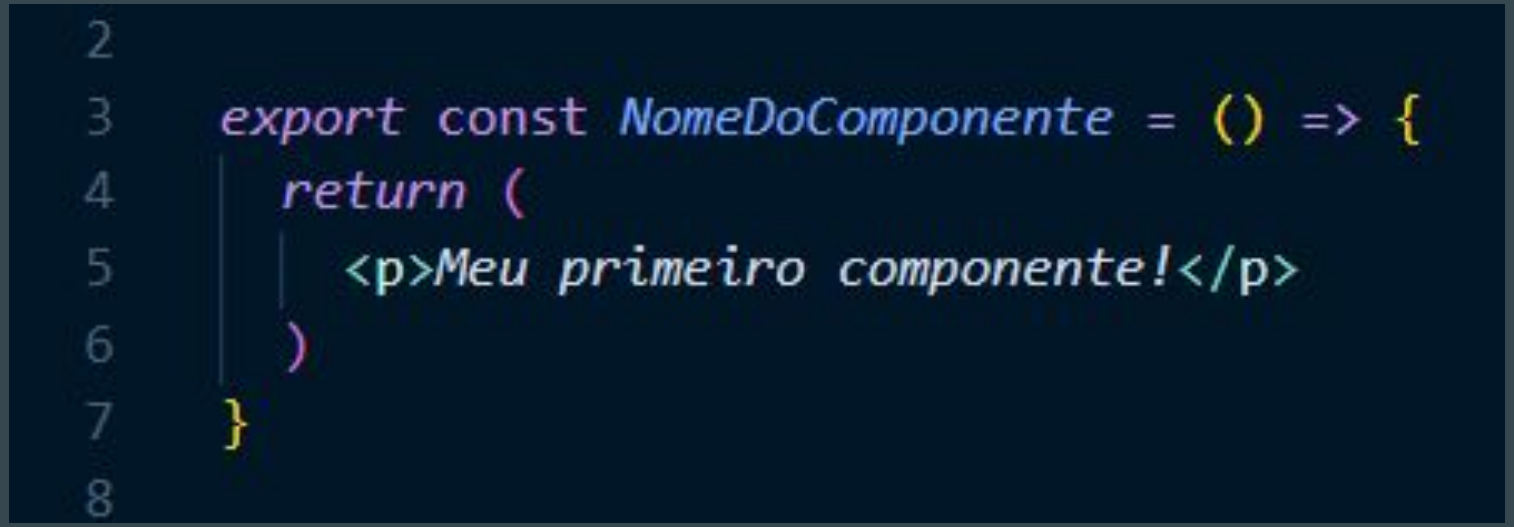
NPM é o gerenciador de pacotes do Node, utilizado para organizar as dependências do projeto e facilitar seu compartilhamento entre ambientes

Conceitos Básicos

A construção de uma aplicação React se baseia na utilização de componentes para gerar a interface final

Componentes são blocos de código HTML definidos em uma sintaxe especial do React (JSX), envolvidos por uma única tag pai

Componentes (exemplo)



Componentes (exemplo)

```
3   // Isso não funciona!
4
5   export const NomeDoComponente = () => {
6     return (
7       <p>Parede</p>
8       <p>Gaveta</p>
9     )
10  }
11
12  // Isso funciona!
13  export const NomeDoCorretoComponente = () => {
14    return (
15      <div>
16        <p>Parede</p>
17        <p>Gaveta</p>
18      </div>
19    )
20  }
21
```

Componentes

```
2
3   // Isso também funciona!
4   export const NomeDoCorretoComponente = () => {
5     return (
6       <>
7         <p>Parede</p>
8         <p>Gaveta</p>
9       </>
10    )
11  }
12
```

Caso necessário exportar um componente com várias tags irmãs sem pai, podemos utilizar a **tag vazia do React (fragment)** , para evitar problemas de sintaxe do código

Primeira Atividade Prática

Primeiros Passos

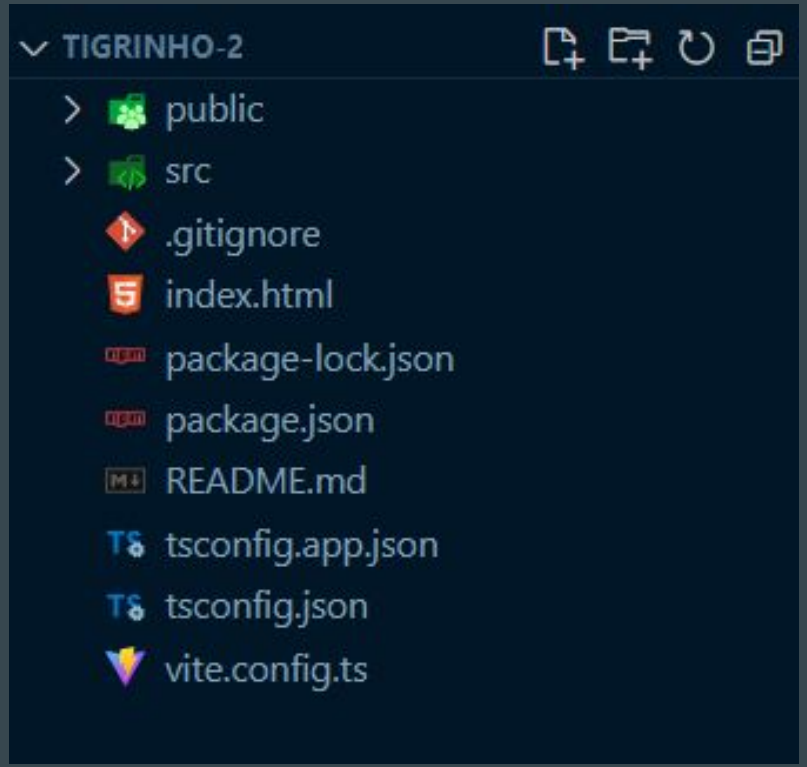
- Faça o download do diretório base do projeto no endereço: https://github.com/megaVE/workshop-front-end

- Extraia o conteúdo do arquivo compactado e abra a pasta **“starter-version”** no seu Visual Studio Code

- Abra o terminal do VS Code na mesma pasta e digite o comando **npm install** , para instalar as dependências do projeto

Visão Geral da Estrutura de Pastas e seu Funcionamento

Principais Arquivos e Estruturas do React



Principais Arquivos e Estruturas do React

index.html - Arquivo HTML base onde a aplicação será renderizada

/public - Diretório destinado a arquivos estáticos da aplicação (ex: favicon, fontes,...)

/src - Diretório base onde ficarão os arquivos do projeto

/src/assets - Diretório destinado a arquivos de mídia (ex: imagens, animações) do projeto

Principais Arquivos e Estruturas do React

.gitignore - Permite definir quais arquivos não devem ser rastreados pelo Git

package.json e **package-lock.json -** Listam as dependências e bibliotecas do projeto

/node_modules - Armazena as dependências e bibliotecas do projeto

tsconfig.json e **tsconfig.node.json -** Arquivos de configuração do TypeScript (quando utilizado)

AVISO: Não se esqueça NUNCA de remover o rastreio da pasta “node_modules” com o **.gitignore**

main.tsx

```tsx
import { createRoot } from 'react-dom/client'
import { App } from './app.tsx'
import './index.css'

createRoot(document.getElementById('root')!).render(<App />)

```

Este script faz com que a aplicação React seja renderizada dentro da tag

** **

do HTML base, visto anteriormente

Introdução aos Conceitos Básicos do React

Propriedades de Componentes

Props

Um recurso amplamente utilizado para criar componentes customizáveis e reutilizáveis são as propriedades de componente **(props)**

Estas propriedades são **declaradas nos parâmetros do Componente** , assim como os parâmetros de uma função

Estas propriedades são atribuídas como **atributos do Componente** , assim como os atributos de um elemento HTML

Props (exemplo)

```
3   // Definição na Declaração
4   const TelaDeUsuario = ({ name }: { name: string }) => {
5     return <p>Olá, {name}, seja bem-vindo(a) ao salão psiquiátrico!</p>;
6   };
7
8   // Atribução na Chamada
9   <TelaDeUsuario name={"Hello Kitty Girl"} />;
10
```

Props

Todos os valores de props (variáveis, números, funções,...) devem ser passados dentro de chaves, como visto anteriormente, obedecendo a sintaxe:

propriedade={valor}

Exceção: as chaves podem ser omitidas quando se estiver utilizando uma string com aspas simples ou duplas

Props

A tipagem das propriedades do componente deve ser feita com sintaxe de objeto para facilitar a utilização e evitar erros de escrita de código

Esta tipagem pode ser feito por meio da tipagem tradicional do TypeScript ou por meio de interfaces

Props (exemplo)

```tsx
// Tipagem com tipo
type TypeComponentProps = {
  name: string;
  age: number;
};

const TypeComponent = ({ name, age }: TypeComponentProps) => {
  return (
    <div>
      <p>{name}</p>
      <p>{age}</p>
    </div>
  );
};

```

Props (exemplo)

```tsx
// Tipagem com interface
interface InterfaceComponentProps {
  name: string;
  age: number;
}

const InterfaceComponent = ({ name, age }: InterfaceComponentProps) => {
  return (
    <div>
      <p>{name}</p>
      <p>{age}</p>
    </div>
  );
};

```

Props (exemplo)

```
3   // Definição na Declaração
4   const TelaDeUsuario = ({ name }: { name: string }) => {
5     return <p>Olá, {name}, seja bem-vindo(a) ao salão psiquiátrico!</p>;
6   };
7
8   // Atribução na Chamada
9   <TelaDeUsuario name={"Hello Kitty Girl"} />;
10
```

Prop Children

Um recurso muito útil para criar componentes containers e organizar o código de forma mais visível é o de ***children props***

Com este recurso, é possível colocar **componentes dentro de componentes** , sem limite de aninhamento

Este tipo de propriedade, definido no TypeScript como **ReactNode** , recebe o nome de "**children** ", por convenção

Prop Children (exemplo)

```tsx

import type { ReactNode } from "react";

const Componente = ({ children }: { children: ReactNode }) => {
  return (
    <div>
      <section>{children}</section>
    </div>
  );
};

const OutroComponente = () => {
  return <p>Tem gosto de giz de cera</p>;
};

```

Prop Children (exemplo)

```
4   // Sintaxe Padrão
5
6   (
7     <Componente
8       children={<OutroComponente />}
9     />
10  )
11
12  // Sintaxe "children"
13
14  (
15    <Componente>
16      <OutroComponente />
17    </Componente>
18  )
```

Conteúdo do Segundo Dia

- Depuração da Aplicação Back-End com IntelliJ
- Visão Geral do Funcionamento do Postman e das Requisições HTTP
- Introdução aos Conceitos Básicos do React

Visão Geral do Funcionamento da API Back-End com Postman

Gerenciamento de Estados

State

Alterações de variáveis da forma tradicional **não** possui impacto imediato sobre a interface HTML do projeto

Se precisamos definir uma informação dinâmica na interface, precisamos criar um **estado (state)** através do **hook** “useState”

State (exemplo)

```
2
3   // Isso não funciona!
4   let value = 0
5
6   export const Component = () => {
7     return (
8       <div>
9         <button onClick={() => { value += 1 }}>+</button>
10        <p>{value}</p>
11      </div>
12    )
13  }
```

State (exemplo)

```
3
4    import { useState } from "react"
5
6    const [value, setValue] = useState<string>("")
7
```

State (exemplo)

```tsx

import { useState } from "react"

const [value, setValue] = useState<number>(0)

export const Component = () => {
  return (
    <div>
      <button onClick={() => { setValue(value + 1) }}>+</button>
      <p>{value}</p>
    </div>
  )
}

```

Monitoramento de Estados

Monitoramento de State

Algumas vezes na aplicação, pode ser necessário executar uma função quando um estado (variável) é alterado ou assume determinado valor

Para este fim utilizamos o **hook** “useEffect”, que permite definir quais estados serão monitorados e executa uma função definida toda vez que algum deles é alterado

Monitoramento de State

```tsx

import { useState, useEffect } from "react"

const [value, setValue] = useState<number>(0)

useEffect(() => {
  console.log(`O valor do contador foi alterado para ${value}.`)
}, [value])

export const Component = () => {
  return (
    <div>
      <button onClick={() => { setValue(value + 1) }}>+</button>
      <p>{value}</p>
    </div>
  )
}

```

Operadores Ternários

Operadores Ternários (AND)

```
3   export const Component = ({ condition }: { condition: boolean }) => {
4     return (
5       <div>
6         {condition && <p>A condição é verdadeira!</p>}
7       </div>
8     )
9   }
10
```

Operadores Ternários (IF ELSE)

```tsx
export const Component = ({ condition }: { condition: boolean }) => {
  return (
    <div>
      {condition ? (
        <p>A condição é verdadeira!</p>
      ) : (
        <p>A condição não é verdadeira...</p>
      )}
    </div>
  );
};

```

Aplicando CSS em ReactJS

Utilização de CSS

Existem 3 principais formas de aplicar CSS em uma aplicação React:

1. Importando um arquivo .css de estilos em uma aplicação, semelhante ao padrão do HTML “vanilla”

2. Aplicando CSS inline com a sintaxe adaptada do JSX

3. Utilizando o recurso de módulo CSS do React, que permite a importação seleta de classes para cada componente

Utilização de CSS (Importação Padrão)

```
2
3     import "./styles.css"
4     // É só isso mesmo...
5
```

Utilização de CSS (Inline no JSX)

```
4   export const Component = () => {
5     return (<div>
6       <h1 styles={{ color: "red", backgroundColor: "black"}}>UNIbet - Faça seu jogo aqui!</h1>
7
8       <p style={{ fontWeight: "bold"}}>Quanto tempo vai atrasar o ônibus das 8h45?</p>
9     </div>)
10  }
11
```

As propriedades CSS são **CamelCase** na sintaxe JSX

Utilização de CSS (Module CSS)

```
2
3   import styles from "styles.module.css"
4
5   export const Component = () => {
6     return (
7       <div>
8         <h1 className={styles.title}>UNIbet - Faça seu jogo aqui!</h1>
9         <p className={styles.secondTitle}>Quanto tempo vai atrasar o ônibus das 8h45?</p>
10      </div>
11    )
12  }
13
```

Listas em React

Listas em React

Para exibir um array em um projeto React utilizamos do método **map** do JavaScript, que retorna outro array com os itens do array original manipulados por uma função

Listas em React

Para exibir um array em um projeto React utilizamos do método **map** do JavaScript, que retorna outro array com os itens do array original manipulados por uma função

```
3   const randomList = [1, 2, 3]
4
5   const Lista = () => {
6     return (
7       <ul>
8         {randomList.map((item) => {
9           return <li>{item}</li>
10        })}
11      </ul>
12    )
13  }
14
```

```
15  const MesmaLista = () => {
16    return (
17      <ul>
18        <li>1</li>
19        <li>2</li>
20        <li>3</li>
21      </ul>
22    )
23  }
```

Listas em React

LEMBRETE: quando utilizando o map, devemos sempre adicionar o atributo “key” na tag pai retornada, o qual contém um valor único para cada valor do array

```
const randomListUpgrade = [
  {id: "faoudosadaiksdosa", value: 1},
  {id: "çclasçdlasçdlasld", value: 2},
  {id: "asdoasoasoasxzcas" , value: 3}]

const Lista = () => {
  return (
    <ul>
      {randomListUpgrade.map((item) => {
        return (
          <li key={item.id}>
            {item.value}
          </li>
        )
      })}
    </ul>
  )
}
```

Conteúdo do Terceiro Dia

- Utilizando o Front-End para Requisições no Back-End
- Atividade Prática com Ngrok
- Conclusões e Considerações Finais

Requisições HTTP e Integração com Back-End

Requisições de API no Front-End

Uma forma de comunicar uma aplicação Front-End com uma aplicação Back-End é por meio de **chamadas e respostas HTTP**

Estas chamadas contém informações (rota, método, conteúdo do corpo) que devem expressar o que o **usuário (cliente)** deseja acessar no **provedor (servidor)**

Requisições de API no Front-End (exemplo)

```
3   const response = await fetch(
4     "http://localhost:3000/users",
5     {
6       method: "POST",
7       headers: { "Content-Type": "application/json" },
8       body: JSON.stringify({
9         name: "Macaco Engenheiro",
10        age: 23,
11        isEmployed: true
12      }),
13    }
14  );
15  const dataObject = await response.json();
16
17  return dataObject
```