**Be The Hero – Semana OmniStack 11**

- Fazendo uso do npm nesse projeto.

- Iniciando o projeto dentro da pasta backend. npm init -y

- Realizo a instalação do express: npm install express

- Crio a index.js na pasta backend.

- npx executa um pacote externo sem instalar na máquina: npx create-react-app frontend na pasta raiz.

**Tipos de parâmetros**

**Query params**: 90% das vezes usado nos metódos GET e ficam visíveis na URL é o que aparece após o “?”

**Route params**: parâmetros usados para identificar recursos das rotas.

EX: app/users

**Request body:** Utilizado para criar ou alterar recursos.

Comando de start: **npm start**

**Configurando o BD**

Banco de dados utilizado: **SQLite**

Utilização do query builder KNEX.js

Instalar o knex e sqlite: <http://knexjs.org/>

Para iniciar o arquivo do knex: **Npx knex init**

Possui migrations que é o controle de versões das minhas tabelas do banco de dados.

**Modelagem do BD**

ONG

caso

**Funcionalidades**

- Login de ONG

- Listagem de casos

- Listagem especifica de um caso

- Cadastramento de ONG

- Cadastramento de caso

- Atualização de caso

- Atualização de dados de ONG

- Deletar caso

- Entrar em contato ONG

**Criando migration e a estrutura das tabelas**

- Cria uma pasta chamada migration em database

- Criando a migration da ong: **npx knex migrate:make create\_nomeTabela**

- Criando a estrutura da tabela em: 20200410180220\_create\_ong

- Consolidando a criação da tabela: **npx knex migrate:latest**

**OBS:** O latest não agiu como o esperado e só criou o bd de ‘caso’, pois ele foi o último. Melhor utilizar o up que sobe individualmente!

**Testando a inserção de dados**

- Para a ID uso o crypto que já vem no node.

**Estabelecer a conexão com o banco de dados**

- Crio o arquivo connection.js na pasta database

-

**Testando a inserção de dados para os Casos**

**-** No caso do id eu tenho que pegar a id da ong que vai realizar a inserção e farei isso por meio do *request.headers*. Aí eu possuo alguns dados de autenticação, dados do idioma e outras informações.

**-** No body eu só mando os dados que são específicos da tabela caso.

**-** A autenticação eu pego no header “Authorization”

**Deletar um caso**

- Para deletar um caso preciso passar o id como parâmetro na rota.

- Para coletar um parâmetro da rota uso o *request.params*

- Faço uma verificação se o caso a ser deletado, realmente pertence a essa ONG.

**OBS: Um controle tem que ser a central das principais ações, a partir do momento que tenho por exemplo, dois selects é melhor eu jogar um desses para um novo controller.**

**Login e logout**

- Controller e route: session

- Logout é feita no front-end

**Paginação para os incidentes**

**Contagem total de Casos**

- Armazeno numa variável, mas pego esse valor no X-Total-Count que vem no header da resposta.

**Faço o JOIN dos dados do caso com as informações das ONGs**

**Instalação do CORS**

- npm install cors

**Interface WEB**

PRINCIPAIS CONCEITOS DO REACT

Componente (C), Estado (E) e Propriedade (P)

**C:**

Função que retorna um conteúdo. A function App() é um C.

UM COMPONENTE POR ARQUIVO.

Para utilizar vários componentes um abaixo do outro, como vários Headers eu não preciso usar <div> eu utilizo fragments que são simplesmente tags sem nomenclatura, ou seja, apenas abrem e fecham.

**P:**

**Conceito:** São os atributos utilizados nos C`s.

Dentro das functions posso passar propriedades (props). Que são parâmetros. Para mostrar que quero inserir um conteúdo JS, eu utilizo {}. Exemplo: {props.title}.

**E:**

**Conceito:** Informação que é manipulada por um COMPONENTE. Lembrar sempre da IMUTABILIDADE

React não monitora mudanças de valores de variáveis.

useState. Importo isto junto com o React:

import React, { useState } from 'react';

**Usando o useState:**

function App() {

const [counter, setCounter] = useState(0);

// Estado inicial e 0 e o retorno e um vetor

function incrementCounter(){

counter++;

}

return (

<h2>Conta</h2>

<button onClick={incrementCounter}>Incrementar</button>

);

}

REACT trabalha com IMUTABILIDADE, ou seja, ele nunca altera um dado ele cria um novo de acordo com um valor anterior.

function App() {

const [counter, setCounter] = useState(0);

// Estado inicial e 0 e o retorno e um vetor

function incrementCounter(){

setCounter(counter + 1);

}

return (

<>

<h2>Conta</h2>

<button onClick={incrementCounter}>Incrementar</button>

</>

);

}

**Página de login**

•

**Trabalhar com rotas na aplicação**

npm install react-router-dom

• **Browser Router:**

• **Switch:**

• **Route:**

**Ligar o backend com o frontend**

• Dar o start no backend

• Instalar o axios no frontend: npm install axios

• Crio a função que irá cuidar da inserção do registro

• Todos os inputs ficarão dentro de um estado (useState)

• Os inputs terão um value e um onChange que fara com que uma arrow function seja chamada e pegue o value do input

**-- e.target.value –**

**Cuidado com os nomes das Keys no JSON**

• Depois de cadastrar retornar o usuário para a index

**Login**

**•** Após passar o id eu preciso fazer com que essas infos da sessão fiquem armazenadas (ID e nome).

**•**