**Conhecendo a OmniStack**

**Criando a base da aplicação**

Rota é o caminho completo. Ex: http://localhost:3333/users

O “/users” é o recurso que se quer acessar

Métodos HTTP:

GET: Busca uma informação do Back-end

POST: Criar uma informação no back-end

PUT: Alterar uma informação no back-end

DELETE: Deletar uma informação no back-end

O navegador toda vez que requisita uma URL acessa, acessa determinada página usa apenas o método GET. Então para testar todos os métodos, durante a etapa de desenvolvimento usaremos o programa Insomnia.

No Insomnia, para testarmos criamos uma nova request e no campo post colocamos a URL para acessar o resurso. Ex: http://localhost:3333/users

Tipos de parâmetros:

Querry: parâmetros nomeados enviados na rota após o “?” (Filtros, páginação)

Ex:<http://localhost:3333/users?name=Victor> (No Insomnia)

Nesse caso queremos de retorno apenas os ussuários com nome “Victor”. É possível também usar “&” um operador que serve como “e”.

Ex: Ex:<http://localhost:3333/users?name=Victor$idade=21>

No código, o parâmetro “request” da função guarda todos os dados que vem através da nossa requisição

Rout: parâmetros utilizados para identificar recursos.

Ex:[http://localhost:3333/](http://localhost:3333/users?name=Victor)1 (No Insomnia)

Usuário que possui o id 1

Request Body: corpo da requisição, utilizado para criar ou alterar recursos

Nodemon:

Instalando o nodemon: npm install nodemon -D

A opção “-D” torna o nodemon uma opção apenas de desenvolvimento, não para o usuário final.

Tal pacote faz a reinicialização do servidor após alterações feitas no código automaticamente. Não sendo necessário parar com o Ctrl+C e inicializar de novo com o node index.js

Para executar o nodemon criamos um script personalizado no package.json:

“start”: “nodemon index.js”

E no terminal executamos o script: npm start

Tipos de bancos de dados:

Se encontra hoje dois tipos de banco de dados:

SQL: MySQL, SQLite, PostgreeSQL, Oracle, Microsoft SQL Server (relacionais)

NoSQL: MongoDB, CouchDB (não relacionais)

Será utilizado o SQLite, que não precisará instalação adicional, é armazenado como um arquivo com a extensão .sqlite

Configuração do Banco de Dados:

Driver direto do banco: SELECT \* FROM users (linguagem SQL)

Querry Builder: table(‘users’).select(‘\*’).where() (linguagem JS)

A ferramenta de querry builder será o knex.

Para instalar:

npm install knex

E o driver referente ao banco, no caso será o:

npm install sqlite3

Na página do knex é tudo explicado.

Após feito as instalações, inicializa-se:

npx knex init

Para um código mais organizado criamos uma pasta na raiz chamada “src” que colocaremos todos os arquivos criados por nós.

Criaremos também um arquivo específico para as rotas e importaremos ela para o arquivo index e também a pasta “database” do banco de dados.

PS: É necessário fazer todas as alterações de caminhos nos arquivos.

Pensando nas entidades e funcionalidades:

Entidades: (precisarão ser armazenados no banco de dados)

* Ong
* Caso (incident)

Fincionalidades:

* Login da Ong
* Logout da Ong
* Cadastro da Ong
* Cadastrar novos casos
* Deletar casos
* Listar casos específicos de uma Ong
* Listar todos os casos
* Entrar em contato com a Ong

Para criar as tabelas do Banco de Dados será utilizado um recurso do knex, as “migrations”

Para isso cria-se uma nova pasta dentro de “database” com o nome “migrations” e feito alterações no knexfile.js

Para a criação do arquivo:

npx knex migrate:make create\_ongs

Consertar o aviso colocando no arquivo knexfile – useNullAsDefault: true,

Olhar no site do knex o createtable

Criou-se controllers para cada método para desacoplar o código

Adicionando módulo CORS;

Módulo de segurança que controlará quem irá acessar a aplicação

Para instalar:

npm install cors

**Construindo a interface Web**

Primeiro foi feito uma “limpeza” na pasta do frontend, excluindo alguns arquivos que já foram criados e não serão necessários

Conceitos do React:

Um componente no React é uma função que retorna HTML, todos os componentes começam com letra maiúscula.

JSX seria o JavaScript integrado com XML(sintaxe do HTML).

Caso algum trecho de código se repita várias vezes, como é o caso de um Header numa página, é interessante criar um arquivo separado e depois integrá-lo ao código

Para que a página renderize uma mudança há o conceito de estado. Em sua mudança, a página carrega uma nova função. Porém por causa de performance, há o conceito de imutabilidade, não se pode alterar o estado de uma forma direta, é necessário sobrepor o valor da variável que quisermos.

A função useState() retorna um array, sendo a primeira posição o valor da virável e a segunda uma função responsável por alterar o valor da mesma. Toda vez que precisarmos alterar algo faremos isso através do conceito de estado e não de uma simples variável

Construindo rotas:

Instalar: npm install react-router-dom

Criar o arquivo de rotas “routes.js” na src

Conectando frontend e backend

Para isso será instalado um client http, responsável por por fazer as chamadas ao backend. Será utilizado o axios

npm install axios

Após cria-se em src uma pasta services e dentro, api.js

Cada váriavel digitada pelo usuário será armazenada em um estado

**Desenvolvendo o app mobile**

Primeiro deve-se instalar o expo:

npm install g expo-cli

**Funcionalidades Avançadas:**

Para validação usou-se o celebrate

npm install celebrate

Para testes usou-se o jest

npm install jest