**Aplicação Oministack**

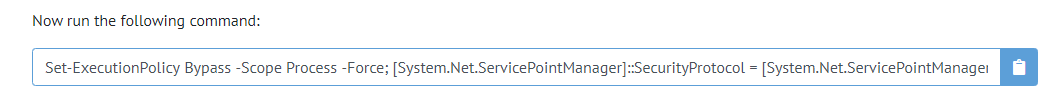
Iniciando os trabalhos

ATENÇÃO: crie uma pasta para criação dos APP, use pastas sem caracteres especiais ou espaços... o caminho todo deve ser assim.

**Instalações de ferramentas...**

**NODE** –para construção de aplicações JAVA em backend

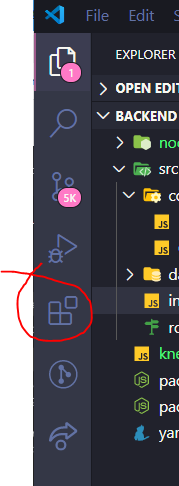
* Motor de reconhecimento do JAVA no navegador (end), interpretando JAVA por fora do browser.
* <https://nodejs.org/en/> - utilizar versão LTS (estável)
  + Esta instalação é fixa e mais complexa para atualizações
  + Devemos instalar o “gerenciador de pacotes” (others downloads) ([Installing Node.js via package manager](https://nodejs.org/en/download/package-manager/))
  + Instalar (modos de visualização e controles)
    - Alternatives
    - Using [**Chocolatey**](https://chocolatey.org/)(<https://chocolatey.org/install>):
    - cinst nodejs
    - # or for full install with npm
    - cinst nodejs.install
    - Using **[Scoop](https://scoop.sh/)**:
    - scoop install nodejs
  + Abre o powershell do Windows em modo admin.
  + Digite <Get-ExecutionPolicy> se o retorno = restrito <Set-ExecutionPolicy AllSigned> se retorno <> próximo passo.
  + Execute o comando para instalação do site...



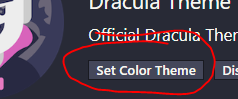
* + Após instalação digite <choco> para conferir.
  + Depois digite (choco install nodejs-lts ou sinsc nodejs-lts)
  + Feche o terminal (powershell) e abra novamente digite <node –v> para verificar a versão
    - Instalar o gerenciador de pacotes do NODE
      * npm -v (sdk conjunto de aplicações para uso da aplicação)
* **Instalação do Visual Studio Code (VSC)** (<https://code.visualstudio.com/>)

Abrir o VSC

Configurando temas (procure vídeo para isto se precisar):



* + Em extencions (para facilitar uso)
    - Procure drácula <install> depois configure em (drácula)



* + - Procure Material IconTheme (habilita os ícones dos arquivos)
* No powershell, dentro da pasta do projeto (**cd** <pasta> vai para a pasta), (**cd..** vai para a pasta anterior), (**md** <pasta> cria pasta), (**ls** lista arquivos e pastas)
  + Faça uma pasta (backend) para instalação dos pacotes do APP
  + Digite npm init –y (aceita perguntas de instalação)
    - Cria arquivo package.json



* No Visual Studio Code
  + Abrir a pasta do projeto para instalar framework (micro framework, somente ferramentas básicas) para interpretar informações do browser e outros (comunicação back e frontend)
  + Usar o terminal do VSC acessando <ctrl + “>
    - Digitar npm install express

Instala dentro do package.json

Cria uma pasta (armazena código fonte de dependências) e um arquivo (configura e chama as dependências (informações de cache)

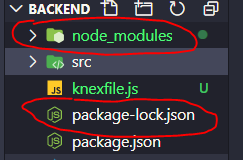


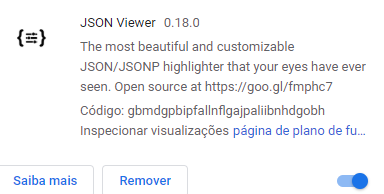
Figura 1 - instalando o express

* + Criar um arquivo index.js (arquivo principal da aplicação)



Figura 2 - arquivo principal da aplicação

* No Chrome instalar a extensão JSON Viewer



Utilize a opção drácula

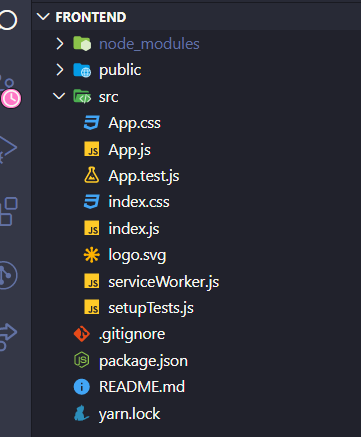
* **Instalação do react**

É usado para acelerar a aplicação e a comunicação no browser de forma a gerir a aplicação no computador do usuário com SPA (single page aplication)

* + No powershell ou no NODE digitar (na página principal do projeto)

npx creat react app frontend

Para verificar a instalação basta entrar na pasta criada com o VSC



Para executar o react, dentro da pasta execute pelo VSC

npm start

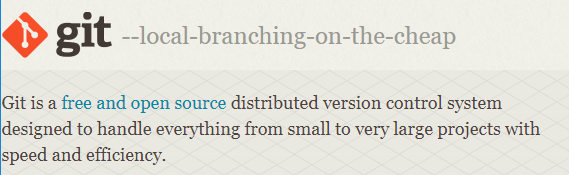
Uso do EXPO para facilitar o react e react native (instalar Expo)

+++++++++++++++++++++++++++++++++++++++++++

GitHub 🡪 página de acompanhamento e armazenamento de códigos de programação

<https://github.com/>

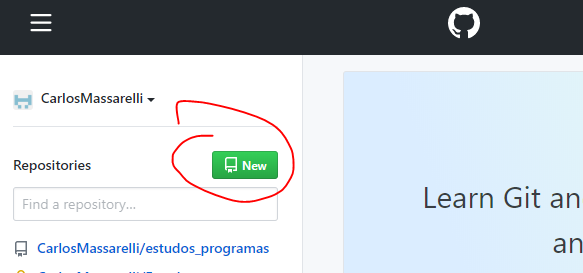
Para guardar seu código no GIT-HUB, instale o git-scm (<https://git-scm.com/>) na parte de dowloads, basta fazer a instalação automática.



Para conferir a instalação, no powershell digite o comando: git –version

A resposta será a versão do git = git version 2.26.2.windows.1

Log no git-hub e crie o repositório para os programas no ícone abaixo:



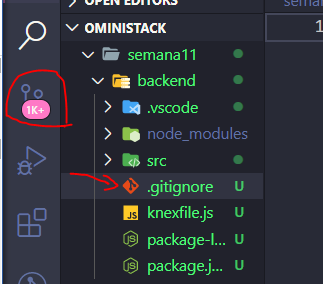
Deixe o repositório público para as empresas verem...

Após a instalação crie o arquivo .gitignore (atenção... esta pasta será criada somente após a instalação do VSC, e será criada na pastas BACKEND (na pasta SRC criada mais a frente) e FRONTEND

++++++++++++++++++++++++++++++++++++++++++++

**Abrir** projeto em construção

* Pode acessar diretamente da pasta, pelo arquivo (abrindo o arquivo, mas não abre o diretório no VSC.
* Pelo terminal, dentro da pasta do projeto, digite <code .> este comando abre o VSC e abre o arquivo index.js na pasta do projeto (muito melhor)
  + Caso não funcione, deve instalar o “code” pelo VSC, pelo <command palette> em <view> (atalho ctrl+shift+P) e procure “install code” (se estiver instalado não aparecerá), após isto, clique no ícone lateral hachurado abaixo:



Esta pasta irá impedir que o github receba alguns arquivos, como o node.modules que é instalado quando da construção do projeto.

No powershell (modo administrador) digite:

git commit -m "first commit"

Se a resposta for:

\*\*\* Please tell me who you are.

Run

git config --global user.email "you@example.com"

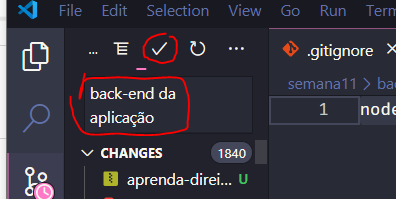
git config --global user.name "Your Name"

to set your account's default identity.

Omit --global to set the identity only in this repository.

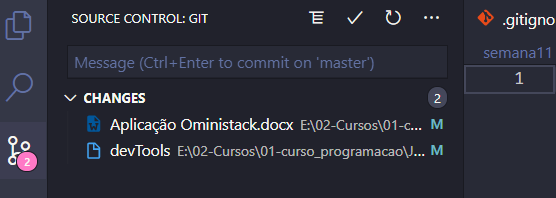
Deverá rodar os dois comandos em azul para identificar-se... alterando o e-mail e o nome.

Após isso vc cria o nome para o seu GIT... e clique no ícone de checagem:

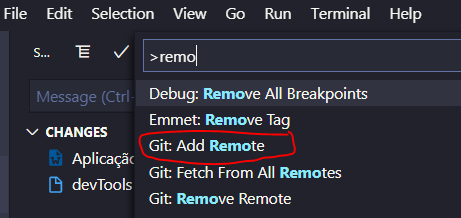


Após isto sua aplicação está comitada.

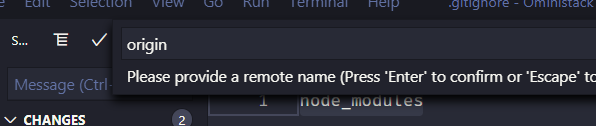
Se tudo der certo os arquivos serão retirados da tela, porque estarão “comitados”... como se fosse uma marcação na história do projeto.



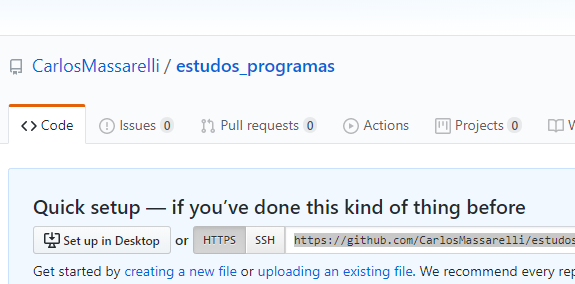
Depois, no VSC, com a tecla <shift+ctrl+P> busque por Git: Add Remote



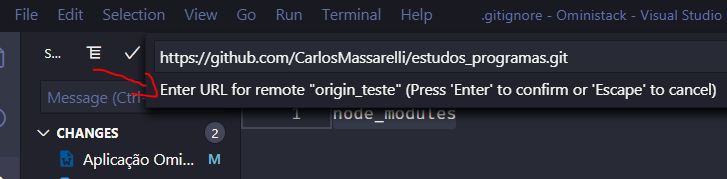
Dê um nome para o remote... em regra é “origin”



Na página do Git-Hub, entre no repositório criado e copie o link para arquivar os arquivos do APP



Insira após colocar o nome do remote no VSC...



Nas configurações (...) ao lado do nome do arquivo, procure a opção <Push>

+++++++++++++++++++++++++++++++++++++++++++++++++++

Para reinicilizar o browser automaticamente (auto execução do projeto)

* pacotes de instalação (na pasta do projeto)

npm install nodemon -D (somente para desenvolvimento)

no arquivo package.jason criar atalho

"scripts": {

"start": "nodemon src/index.js"

},

* + para chamar a aplicação depois de instalada no terminal do VSC

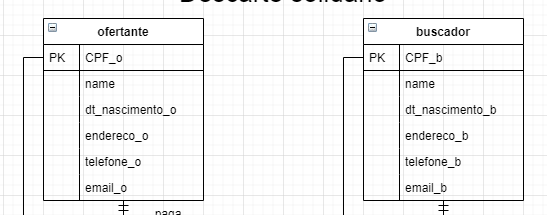
npm **start**

***Construindo a aplicação inicial***

CUIDADO 🡪 qualquer sílaba diferente em um comando, caminho ou arquivo ocasionará um erro.

Formas de construir 🡪 duas realidades:

* layout **pronto** (entregue pelo usuário)
  + pensa as “entidades” (quem usa o BD) e funcionalidades (como usa o BD) da aplicação com base no projeto entregue (visão do cliente para a realidade da aplicação)
    - cliente
      * usuário – quem manipula e usufrui do APP
      * contratante do serviço – quem dispõe o APP para o usuário
    - entidades
      * grupo de dados que serão armazenados para uso do APP
    - funcionalidades
      * o que cada entidade faz/guarda dentro da aplicação
* sem layout
  + Cria o layout e segue o passo acima
  + Constrói APP com base em funcionalidades e velocidades da aplicação (pensamento pragmático)
  + Constrói APP com base na necessidade do mercado e aplicabilidade comercial (pensamento funcionalista) 🡪 volta para o passo anterior e depois para o passo de layout pronto.



Ofertante, buscador, produto, produto\_necessidade (**entidades**)

Login/logout, cadastro (inclusão/exclusão), listar dados, etc (**funcionalidade**)

Estas informações são genéricas por conta da versatilidade da operação

Após todas as instalações teremos a pastas:

* BACKEND (dentro desta pasta teremos os trabalhos iniciais)
  + index.js (Figura 2 - arquivo principal da aplicação)

**criar** uma pasta para organizar os arquivos que criamos... facilita a organização

* **src** (pasta source (fonte) dentro BACKEND) – pela instalação dos pacotes ela terá um ícone próprio
  + Transfira os arquivos que criar para este diretório (ex. index.js)

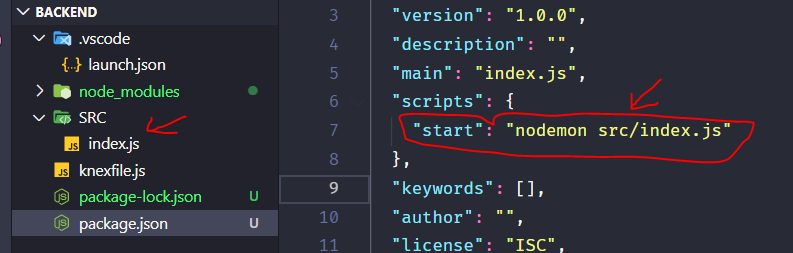


Figura 3 - organizando pastas no projeto

Não esquecer de alterar o endereço dentro do package.json para iniciar a aplicação com a alteração do diretório.

* + **Criar** um arquivo para rotas (routes.js dentro da pasta **src**) que armazenará TODAS as rotas (Será usado mais a frente)
* **controlers** (pasta para controle de rotas dentro da pasta **src**) – pela instalação dos pacotes ela terá um ícone próprio
  + **database** (pasta do banco de dados dentro da pasta **src**) – pela instalação dos pacotes ela terá um ícone próprio

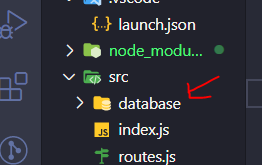


Figura 4 - organizando pastas no projeto

* FRONTEND

No arquivo index.js

* + Importando funcionalidade do express (instaladas anteriormente)

Cria variável ‘express’ que recebe funcionalidade ‘express’

const express = require(‘express’);

variável que armazena a aplicação

const app = express();

aplicação ‘ouve’ a porta (3333) pode ser qualquer porta, esta é a mais usada no browser (localhost:3333) o ideal é separar as aplicações por portas ex. node 3333, react 3000.

app.listem(3333);

Executando aplicação (no terminal do VSC acessando <ctrl + “>)

node index.js

Abrir o browser e chamar a porta da aplicação (localhost:3333)

cannot get = rota executando sem dados (outra resposta = não executando)

Para fechar a aplicação <ctrl + c>

* + Funcionamento da rota (deve cancelar o servido e reiniciar quando fizer)

App.get (‘/’ , (request, response ) -> {

return response.send (‘hello world’);

para formato json usar abaixo...

return.response.json ({

evento: ‘semana oministeck’,

aluno: ‘nome aluno’

});

});

ATENÇÃO... para chamar a rota no insomnia e no browser deve ser sem a barra final (<http://localhost:3333>) caso não tenha um recurso específico

Na construção do projeto, preciso acessar uma “rota” ex. localhost <http://localhost:3333/> deve informar no projeto o caminho ex.

app.get('/', (*request*, *response*) => {

  return *response*.json ({

    evento: 'Semana OmniStack 11.0',

    aluno: 'Carlos'

  });

});

Figura 5- código java

Métodos http:

GET = busca/lista informação do backend (único método requisitado pelo navegador)

* Instalar o INSOMNIA (<https://insomnia.rest/>) aplicativo que possibilita o uso e controle dos outros métodos da aplicação (simula os acessos dos usuários e respostas do browser.

POST = cria informação para frontend no backend

PUT = altera informação do backend

DELETE = deleta informação do backend

Parâmetros usados dentro da URL que encaminham ou recebem informações para tratamento:

* São recebidos/enviados pelos (request, response) no java (Figura 2- código java)
  + response.json ({conjunto de dados a ser enviado})

return **response**.json ({

evento: 'Semana OmniStack 11.0',

aluno: 'Carlos'

});

* + const params = **request**.query; (acessa todos os parâmetros)

console.log(params); (exibe no console a informação recebida)

**QUERY** Params = parâmetros nomeados e enviados na rota (enviado pelo insomnia e capturado pelo app) após a "?" usado para filtros e paginação (possível anexar parâmetros com o “&”)

Usado somente com GET

Acessado por **request.query**

+++++++++++++++++++++ NO JAVA

app.post('/users', (request, response) => {

const params = **request**.query; (acessa todos os parâmetros do query)

console.log(params); (exibe informação recebida no console)

return response.json ({

evento: 'Semana OmniStack 11.0',

aluno: 'Carlos'

});

});

app.listen(3333);

++++++++++++++++++++++++

**ROUT** params = utilizados para identificar recursos específicos (app.get('/cadastro/:id', (...)) (entidade ou tabela com dados específico)

Usado somente com GET

Acessado por **request.params**

+++++++++++++++++++++

app.get('/users/:id', (request, response) -> {

const **params** = request.params;

(recebe paramentos de todos os id)

(**id** para id específico))

console.log(params);

return response.jason({

evento: 'teste de retorno',

aluno: 'nome1'

});

});

+++++++++++++++++++++++++++

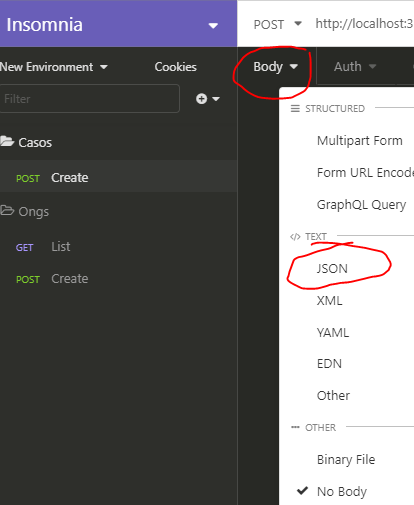
**REQUEST BODY** = corpo da requisição para alterar ou criar **recursos**

(app.post('/cadastro/:id', (...)) (entidade ou tabela com dados específico)

Usado somente com POST / PUT / DELETE

Acessado por **request.params**

Para enviar é preciso configurar o formato do corpo, escolhendo o método e o corpo (body)



**ATENÇÃO**: antes de enviar a requisição deve informar para o APP que as requisições serão JAVA, inserindo no início do programa a informação:

app.use(express.json()); (informa que a requisição será enviada em java)

+++++++++++++++++++++

app.post('/users’, (request, response) -> {

const **body** = request.**body**;

console.log(**body**);

return response.jason({

evento: 'teste de retorno',

aluno: 'nome1'

});

});

TRABALHANDO COM SQLite (faz um banco dentro da aplicação e facilita o uso, posteriormente pode-se usar o código sem alteração para o MySQL.

**Formas de acessar o SQL pelo APP**:

* baixar o driver 🡪 usar comandos convencionais no KDE do banco de dados (BD)

SELECT \* FROM users (nome do banco de dados) WHERE (dados da busca)

* query builder 🡪 (usa comandos java para BD sem usar a estrutura do BD, desta forma não é necessário alterar o código para acessar o banco

ex. table (‘users’).select(‘\*’).where()

* + biblioteca mais utilizada = KNEX.JS (<http://knexjs.org/>)

npm install knex (instala a biblioteca dentro da pasta BACKEND)

* + - instalar a biblioteca específica
      * **npm** install sqlite3
      * npm install mysql
  + CONEXÃO COM BD

Começando o uso, inicie o pacote instalado

**npx** knex init (cria um arquivo knexfile.js, tendo o caminhos para os ambientes do APP), dentro tem o seguinte informação:

* Development 🡪 ambiente de desenvolvimento
* Staging 🡪 simula ambiente de uso (para desenvolvimento)
* Production 🡪 ambiente de uso (acesso cliente)

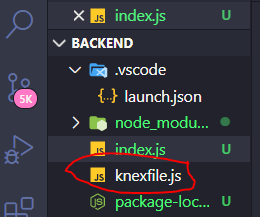


Figura 6 - cria ambiente do banco de dados

**Organizando aplicação**:

Transfira as rotas para o arquivo routes.js, criado anteriormente (Figura 3 - organizando pastas no projeto), com os comandos abaixo:

/\*importa a função express\*/

const express = require('express');

/\*desacopla rotas do express\*/

const routes = express.Router();

app.post('/users', (*request*, *response*) => {

  const body = *request*.body;

  console.log(body);

  return *response*.json ({

    evento: 'Semana OmniStack 11.0',

    aluno: 'Carlos'

  });

});

Retira do arquivo index.js a rota app... trocando app por routes (rota do aplicativo):

routes.post('/users', (*request*, *response*) => {

  const body = *request*.body;

Exporta a variável do arquivo routes.js para o index.js (para ele reconhecer as rotas) pelo comando:

*module*.*exports* = routes;

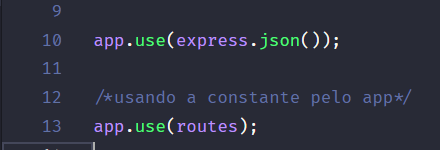
Observe que **routes** é a constante que recebe as aplicações, sendo transportada para fora.

Importando no index.js para ele saber onde será a o **arquivo** da rota.

/\*armazena a função express\*/

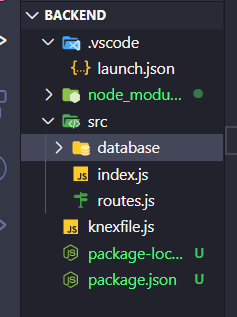
const routes = require('./routes');

Usando a constante com a informação da biblioteca, abaixo do uso de linguagem JAVA pelo aplicativo (as informações estão circulando em java)



É possível testar com o insomnia (usando o trabalho que já funcionava com os testes anteriores) (lembre de iniciar o nodemon (npm start)

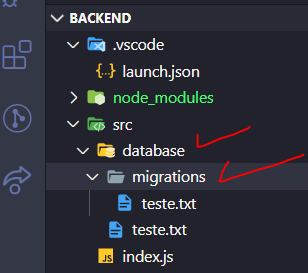
No arquivo knexfile.js (Figura 6 - cria ambiente do banco de dados), alterar o nome do arquivo do BD e local para a pasta “database” (Figura 4 - organizando pastas no projeto), assim a pasta BACKEND terá os arquivos abaixo:



Criar o BD (<http://knexjs.org/#Migrations>), temos um sistema de controle de alterações no BD (**cuidado**, permite acesso aos dados da aplicação)



Para uso desta funcionalidade, deve criar uma pasta (migrations) para os dados colhidos, dentro da pasta “database”.



Informar no knexfile.js (Figura 6 - cria ambiente do banco de dados) o caminho para migrations

  development: {

    client: 'sqlite3',

    connection: {

      filename: './src/database/db.sqlite'

    },

    migrations: {

      directory:'./src/database/migrations'

    },

    useNullAsDefault: true,

  },

**Organizando aplicação teste**:

Construindo aplicação de layout pronto

* **Entidades**
  + ONG
  + Casos (incidents)
* **Funcionalidades**
  + Login / logout ONG
  + Cadastro ONG
  + Cadastro / listar (geral / específico) / deletar (casos)
  + Entrar contato ONG (whatsapp / e-mail)

**Criando BD do APP teste:**

No terminal VSC digitar “npx knex migrate:make create\_ongs”, criando o arquivo de suporte do migrations.

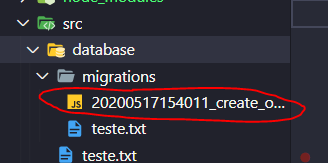
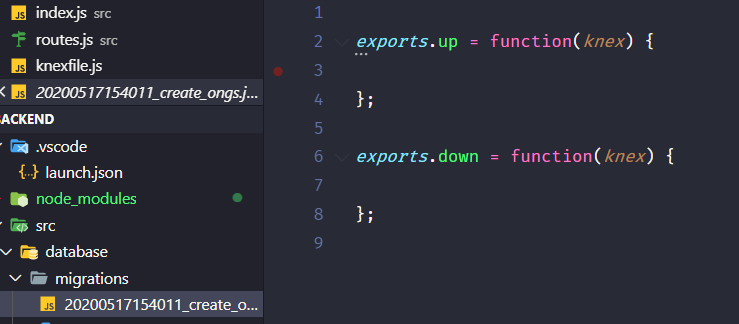


Figura 7 - criando arquivo migrate

Com os métodos de exportação pré-definidos abaixo:



Procurar no site knex as Schema Builder



Dentro do arquivo pré-definido, inserir a estrutura do BD para criação e configuração das tabelas:

*exports*.up = function(*knex*) {

  return *knex*.schema.createTable('ongs', function (*table*) {

*table*.string('id').primary();

*table*.string('name').notNullable();

*table*.string('email').notNullable();

*table*.string('whatsapp').notNullable();

*table*.string('city').notNullable();

*table*.string('uf').notNullable();

  })

};

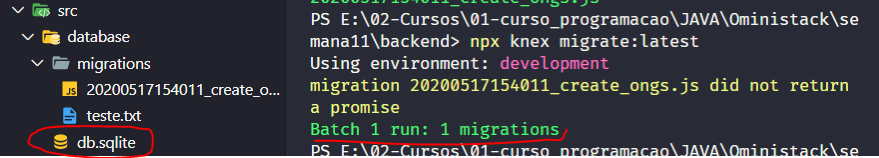
*exports*.down = function(*knex*) {

*knex*.schema.dropTable('ongs');

};

Após as alterações, chamar a função para a criação do BD

No terminal VSC digitar “npx knex migrate:latest”



Criando outro banco, no terminal VSC digitar “npx knex migrate:make create\_ongs”, criando o arquivo de suporte do migrations.(Figura 7 - criando arquivo migrate), seguir os passos até aqui...

Para o arquivo migrate deve ser usado os dados abaixo:

*exports*.up = function(*knex*) {

  return *knex*.schema.createTable('incidents', function (*table*) {

*table*.increments();

*table*.string('title').notNullable();

*table*.string('description').notNullable();

*table*.decimal('value').notNullable();

*table*.string('ong\_id').notNullable();

*table*.foreign('ong\_id').references('id').inTable('ongs');

  });

};

*exports*.down = function(*knex*) {

*knex*.schema.dropTable('incidents');

};

Figura 8 - criação do BD - incidents

ATENÇÃO: não há alteração de arquivos pq a tabela foi criada dentro do BD (estudo de banco de dados)

Caso queira desfazer o comando de criação que foi executado (npx knex – lista todas as funcionalidades do knex:

Usage: [options] [command]

Options:

migrate:rollback [options] Rollback the last batch of migrations performed.

migrate:down [<name>] Undo the last or the specified migration that was already run.

Para usar o comando (somente o migration anterior)

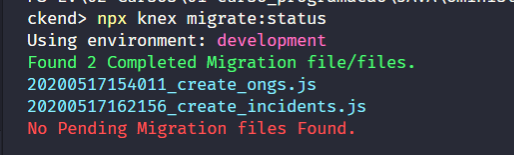
npx knex migrate:rollback (volta a ação imediatamente anterior)

Para voltar uma migração específica:

npx knex down: 20200517154011\_create\_ongs (nome da migration).

Para verificar migragions executadas:

Npx knex migrategra:status



Após criação do BD (construindo a aplicação efetiva)

routes.post('/ongs', (*request*, *response*) => {

  const data = *request*.body;

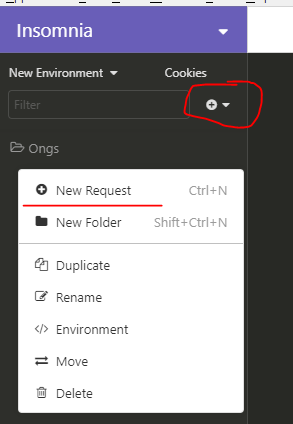
  console.log(data);

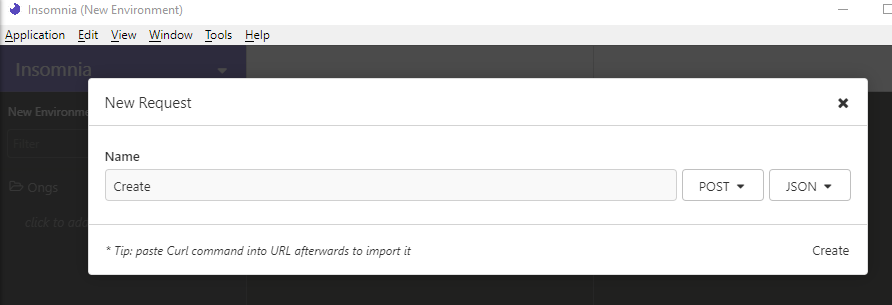
  return *response*.json ();

});

Figura 9 - alterações no arquivo routes.js

No insomnia criar uma pasta “Ongs” e uma requisição na pasta “Create” para o método POST em formato JSON (porque estamos enviando a informação FRONT 🡪 BACK)





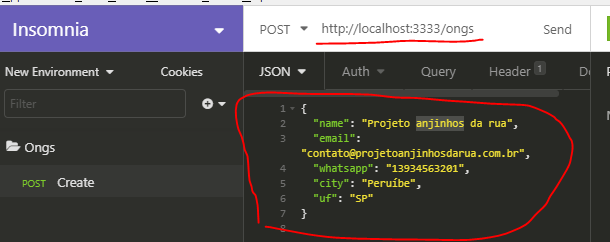


Figura 10 - dados do corpo (body)

{

"name": "Projeto anjinhos da rua",

"email": "contato@projetoanjinhosdarua.com.br",

"whatsapp": "13934563201",

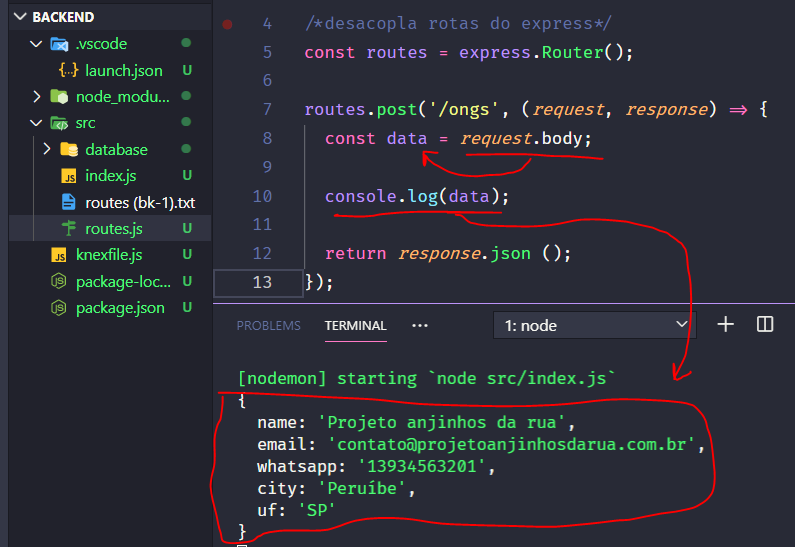
"city": "Peruíbe",

"uf": "SP"

}

Atenção: no insomnia é “ “ e não ‘ ‘ para strings, decimal não precisa de “ “.

A porta para encaminhamento serão os dados abaixo, dentro do recurso (Figura 8 - alterações no arquivo routes.js) que queremos acessar “ongs” (Figura 11 - alterações no arquivo routes.js), que vem no corpo (Figura 9 - dados do corpo (body)) <após clicar SEND> e é atribuído a constante “data” e listado no console (para conferência):



Neste caso todos os dados criam uma matriz dentro da constante “data”

Podemos criar uma variável para cada elemento, garantindo a informação enviada:

 const {name,email,whatsapp,city,uf} = *request*.body;

**Criação de id** pelo método crypto

/\*importa biblioteca para gerar string aleatória\*/

const crypto = require('crypto');

/\*desacopla rotas do express\*/

const routes = express.Router();

routes.post('/ongs', (*request*, *response*) => {

  const {name,email,whatsapp,city,uf} = *request*.body;

  /\*solicita um id (randomico) de 4 bites

    e transforma em string hexadecimal\*/

  const id = crypto.randomBytes(4).toString('HEX')

**Conexão com BD** (aplicação x cliente)

Na pasta “database” **criar** arquivo .js para conexão

Configurando e exportando informações:

/\*importa biblioteca knex\*/

const knex = require('knex');

/\*importa configuração knex\*/

const configuration = require('../../knexfile');

/\*importa dados para connections / migrations\*/

const connection = knex(configuration.development);

/\*exporta informações para routes\*/

*module*.*exports* = connection

A constante “connection” tem atribuição das informações da criação do banco de dados no arquivo referenciado (knexfile.js) que foi criado anteriormente (Figura 6 - cria ambiente do banco de dados)

Posteriormente, captura os dados no routes.js, da conexão do DB do arquivo connection (cuidado com as pastas):

/\*importa dados conexão BD do arquivo connection\*/

const conection = require('./database/connection');

Criando **rota de inclusão**

Desta forma o routes.js ficará como abaixo:

/\*importa a função express\*/

const express = require('express');

/\*importa biblioteca para gerar string aleatória\*/

const crypto = require('crypto');

/\*importa dados conexão BD do arquivo connection\*/

const connection = require('./database/connection');

/\*desacopla rotas do express\*/

const routes = express.Router();

routes.post('/ongs', async (*request*, *response*) => {

  const {name,email,whatsapp,city,uf} = *request*.body;

  /\*solicita um id (randomico) de 4 bites e transforma em string hexadecimal\*/

  const id = crypto.randomBytes(4).toString('HEX');

  /\*usa a conexão (connection) com o BD para inserir as informações entregues pelo usuário no corpo (body\*/

  await connection('ongs').insert({

    id,

    name,

    email,

    whatsapp,

    city,

    uf,

  })

  return *response*.json ({ id });

});

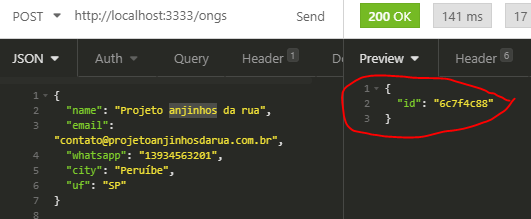
*module*.*exports* = routes;

Para testar a aplicação, basta enviar a requisição que foi criada no insomnia (Figura 9 - dados do corpo (body))

Se der erro, leia a informação no terminal, caso seja problema com o BD, crie novamente:

1. Exclua o BD
2. Rode migration rollback quantas vezes forem os BD’s
3. Rode a migration novamente (“npx knex migrate:latest”)

A resposta no insomnia será:



Criando rota de listagem, antes da rota de inclusão, como segue:

/\* rota de listagem \*/

routes.get('/ongs', async (*request*,*response*) => {

  const ongs = await connection('ongs').select('\*');

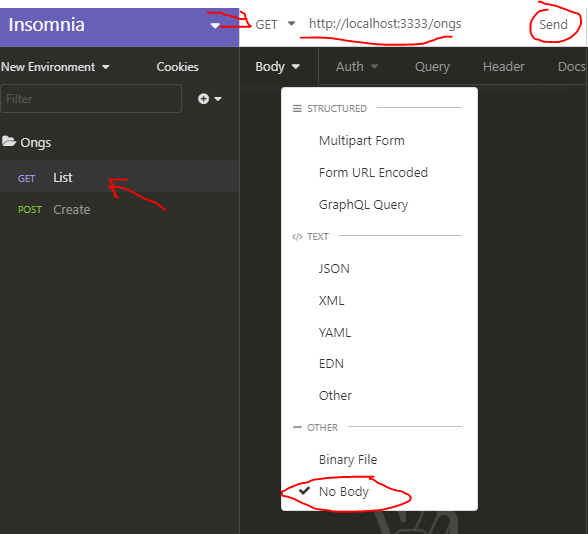
  return *response*.json(ongs);

});

/\* rota de inclusão \*/

No insomnia, criar a solicitação para o APP:

* Cria novo ambiente de solicitação (List) pelo método GET sem corpo (No Body



* A resposta será o número de cadastros prontos, como segue:

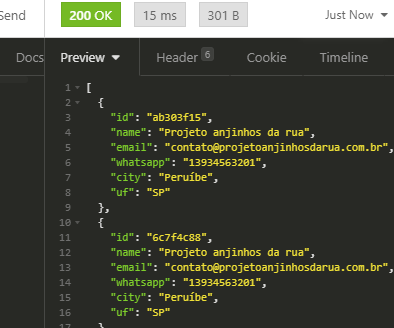


Figura 12 - resposta da requisição de criação

Criar **paginação** na resposta.

Dentro do método INDEX inserimos outras informações, como segue:

*module*.*exports* = {

  //método list

  async index(*request*,*response*) {

    // criando paginação

    const{ page = 1} = *request*.query;

    const [count] = await connection('incidents').count()

    const incidents = await connection('incidents')

      .limit(5)

      .offset((page - 1) \*5)

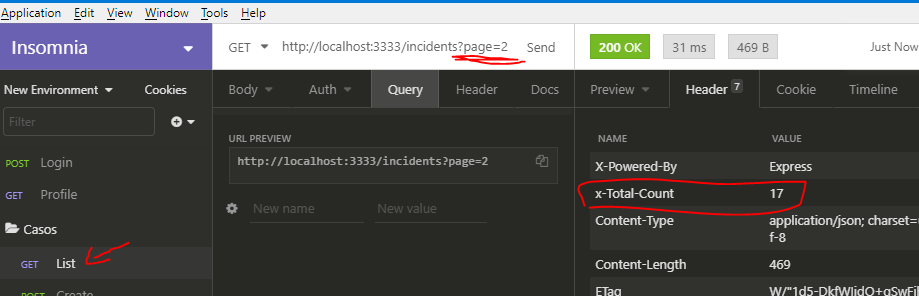
      .select('\*');

*response*.header('x-Total-Count', count['count(\*)'])

    return *response*.json(incidents);

  },

No insomnia, quando enviamos a listagem, ele limita a saída à página solicitada (acrescentamos a informação no endereço de rota (?page=2)), com a informação do total de registros no HEADER, como segue:



Para capturar as outras informações necessárias do BD, precisamos acessar a outra tabela (no caso a ONG) pois estamos recebendo a informação da tabela incidents.

Para isto utilizamos o JOIN (comando SQL) para juntar informações de outras tabelas e encaminhar para o FRONTEND, como segue:

    const incidents = await connection('incidents')

      .join('ongs', 'ongs.id', '=', 'incidents.ong\_id')

      .limit(5)

      .offset((page - 1) \*5)

      .select([

        'incidents.\*',

        'ongs.name',

        'ongs.email',

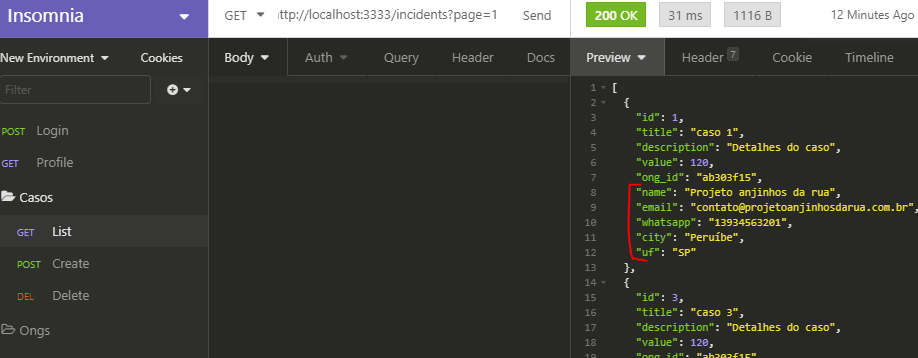
        'ongs.whatsapp',

        'ongs.city',

        'ongs.uf']);

*response*.header('x-Total-Count', count['count(\*)'])

Resposta encaminhada ao FRONTEND:



**Atualizando estrutura da aplicação**

Na pasta controlers (Figura 3 - organizando pastas no projeto)

Criar arquivo para cada controlador (OngControler.js)

Transferir as informações do routes.js para este arquivo, otimizando o código:

const crypto = require('crypto');

const connection = require('../database/connection');

*module*.*exports* = {

  async index(*request*,*response*) {

    const ongs = await connection('ongs').select('\*');

    return *response*.json(ongs);

  },

  async create(*request*, *response*) {

    const {name, email, whatsapp, city, uf} = *request*.body;

    /\*solicita um id (randomico) de 4 bites e transforma em string hexadecimal\*/

    const id = crypto.randomBytes(4).toString('HEX');

    /\*usa a conexão (connection) com o BD para

    inserir as informações entregues pelo

    usuário no corpo (body\*/

    await connection('ongs').insert({

      id,

      name,

      email,

      whatsapp,

      city,

      uf,

    })

    return *response*.json ({ id });

  }

};

O routes.js será modificado para:

/\*importa a função express\*/

const express = require('express');

/\*importa biblioteca para gerar string aleatória\*/

/\*enviada para controlers\*/

//const crypto = require('crypto');

const OngController = require('./controllers/OngController')

/\*importa dados conexão BD do arquivo connection\*/

//const connection = require('./database/connection');

// agora não é mais utilizada (esta no controller)

/\*desacopla rotas do express\*/

const routes = express.Router();

/\* rota de listagem (comentada para uso posterior\*/

routes.get('/ongs', OngController.index);

/\* rota de inclusão \*/

routes.post('/ongs', OngController.create);

/\*dados inseridos no controller.js \*/

*module*.*exports* = routes;

Para teste, basta criar novas ongs e lista-las pelo insomnia.

**Atualizando estrutura da aplicação e criando outros recursos (incidentes)**

Crie um arquivo na pasta (controllers) **IncidentController.js** para requisição (incidents = casos da ong)

A estrutura segue abaixo para o método CREATE no **IncidentController.js**:

const connection = require('../database/connection');

*module*.*exports* = {

  async create(*request*, *response*) {

    const { title, description, value } = *request*.body;

    //request.readers; (recebe os dados do cabeçalho da requisição)

    const ong\_id = *request*.headers.authorization;

    //cria const result = para pegar tudo

    const [id] = await connection('incidents').insert({

      title,

      description,

      value,

      ong\_id,

    });

    // const id = result[0]; (captura o primeiro elemento da matriz)

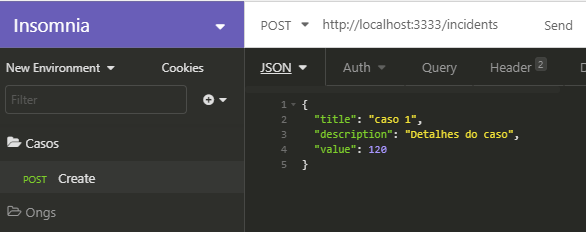
    // manda com {} para saber que é o id

    return *response*.json({ id })

  }

};

Construindo interação no insomnia:



Criar pasta <Casos> no método POST (envia a informação para criar/alterar) no endereço de requisição (<http://localhost:3333/incidents>), com as informações abaixo:

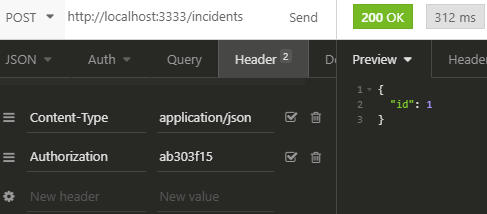
{

"title": "caso 1",

"description": "Detalhes do caso",

"value": 120

}



Na opção HEADER, incluir uma variável para receber o id da ong que está cadastrando o caso (authorization) (Figura 12 - resposta da requisição de criação)

No **routes.js** é necessário mostrar a rota para a requisição que foi criada:

/\*importa a função express\*/

const express = require('express');

const OngController = require('./controllers/OngController')

const IncidentController = require('./controllers/IncidentController')

/\*desacopla rotas do express\*/

const routes = express.Router();

/\* rota de listagem (comentada para uso posterior\*/

routes.get('/ongs', OngController.index);

/\* rota de inclusão \*/

routes.post('/ongs', OngController.create);

/\*dados inseridos no controller.js \*/

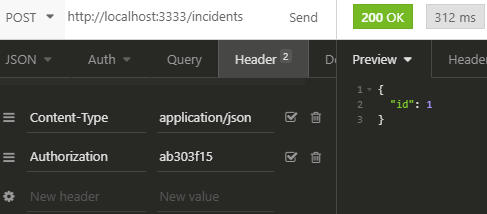
/\* rota de incidents \*/

routes.post('/incidents', IncidentController.create);

/\*dados inseridos no controller.js \*/

*module*.*exports* = routes;

Para testar, encaminhe a informação pelo SEND do insomnia



Em PREVIEW é exibido o id do caso criado (incremental), após o envio da requisição.

A estrutura segue abaixo para o método LIST:

Acrescente no **IncidentController.js**:

*module*.*exports* = {

  //método list

  async index(*request*,*response*) {

    const incidents = await connection('incidents').select('\*');

    return *response*.json(incidents);

  },

  //método create

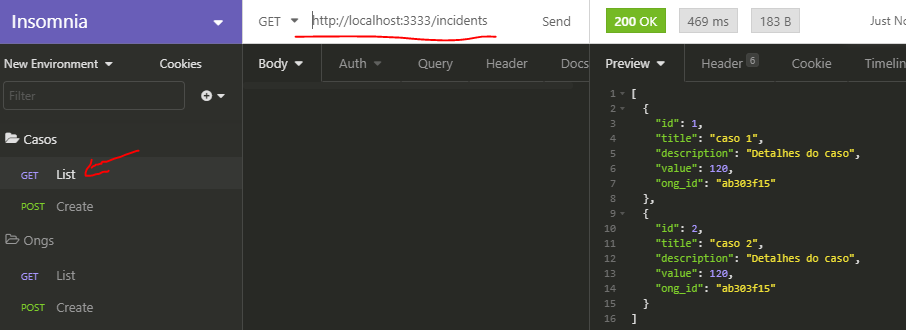
No **routes.js** é necessário mostrar a rota para a requisição que foi criada:

/\* rota de incidents \*/

routes.get('/incidents', IncidentController.index);

Para testar, inclua uma requisição no insomnia:

* Na pasta casos crie uma requisição LIST, com método GET, sem corpo (no body) para o endereço (<http://localhost:3333/incidents>), após enviar a resposta do APP será a lista dos casos cadastrados, como abaixo:



Criando o método DELETE (excluir casos)

A estrutura segue abaixo para o método DELETE

No routes.js, inserimos a rota para o método

/\* rota de incidents \*/

routes.get('/incidents', IncidentController.index);

routes.delete('/incidentes/:id', IncidentController.delete);

routes.post('/incidents', IncidentController.create);

/\*dados inseridos no controller.js \*/

Acrescente no **IncidentController.js após o método create**:

    });

    // const id = result[0]; (captura o primeiro elemento da matriz)

    // manda com {} para saber que é o id

    return *response*.json({ id })

  },

    //método delete

    async delete(*request*,*response*) {

      const {id} = *request*.params;

      const ong\_id = *request*.headers.authorization;

      const incidents = await connection('incidents')

        .where('id', id)

        .select('ong\_id')

        .first();

//retorna apenas um resultado

       if(incidents.ong\_id != ong\_id) {

        return *response*.status(401).json({

          error: 'Operação não permitida!'

        });

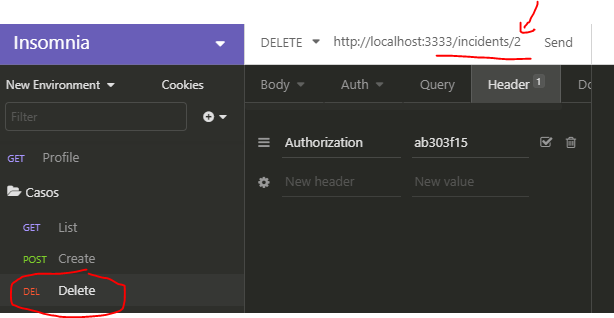
      }

      await connection('incidents').where('id', id).delete();

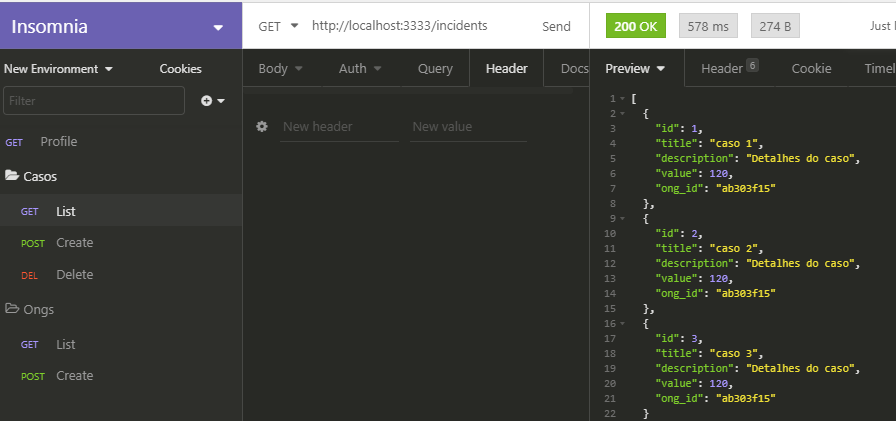
      return *response*.status(204).send();

    },

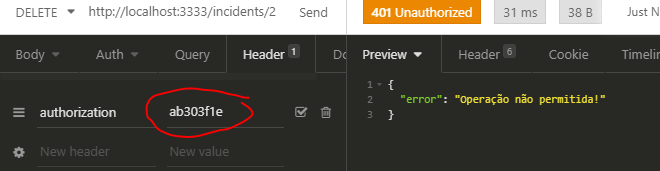
Para testar crie uma rota no insomnia



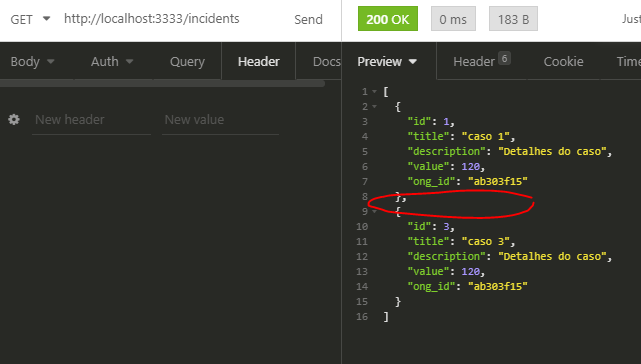
Até agora temos estes 3 casos que podem ser listados pelo método LIST, criado anteriormente:



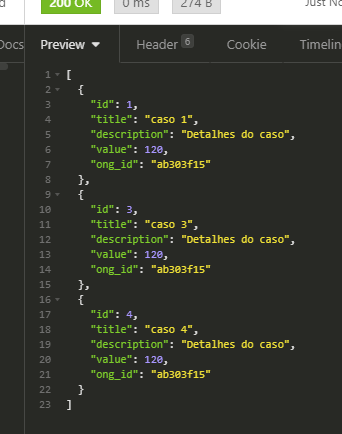
Vamos excluir o método 2 com um HEADER diferente enviará um erro



Se enviamos o HEADER correto, ele exclui o caso



O caso 2 foi excluído.



Os novos casos não ocupam o lugar vago... fundamentos SQL

Criando o método LIST específico para casos de todas as ONG.

Para a metodologia NVC não devemos ter mais que 5 métodos no controller, portanto vamos criar um novo controller.

Criamos o arquivo ProfileController.js

As informações serão semelhantes as anteriores, como segue:

const connection = require('../database/connection');

*module*.*exports* = {

  //método list específico

  async index(*request*,*response*) {

    const ong\_id = *request*.headers.authorization;

    const incidents = await connection ('incidents')

      .where('ong\_id', ong\_id)

      .select('\*');

    return *response*.json(incidents);

  },

};

No routes.js, inserimos a rota para o método e a rota de chamamento

/\*\*cria rota para ProfileController \*/

const ProfileController = require('./controllers/ProfileController')

/\*rota para listar casos da ong \*/

routes.get('/profile', ProfileController.index);

Criando o **método LOGIN** para o sistema.

Criamos uma rota para um novo arquivo (SessionController.js) que irá controlar as ONGs que estarão criando uma sessão para uso conforme abaixo:

const connection = require('../database/connection');

*module*.*exports* = {

  async create(*request*, *response*) {

    const { id } = *request*.body;

    const ong = await connection('ong')

      .where('id', id)

      .select('name')

      .first();

    if (!ong) {

      return *Response*.status(400).json({

        error: 'Não foi encontrada ONG para este ID'

      });

    }

    return *Response*.json(ong);

  }

}

No arquivo routes.js direcionamos a rota

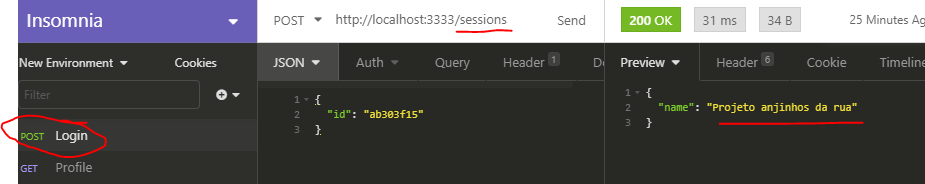
/\*\*cria rota para SessionController \*/

const SessionController = require('./controllers/SessionController')

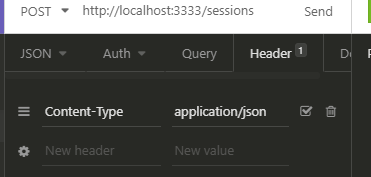
/\* cria sessão de LOGIN\*/

  routes.post('/sessions', SessionController.create);

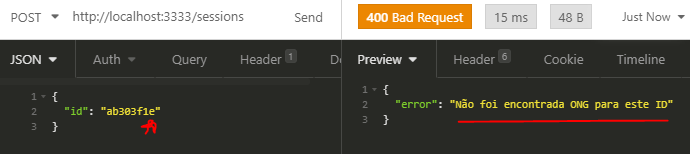
Para conferir, criamos um método para a informação no insomnia:



O conteúdo é json e deve ser especificado, como abaixo:



Se errar o ID ele retorna o erro



Inserindo **módulo de segurança**

No terminal, na pasta BACKEND, instalamos o pacote CORS, que irá limitar o acesso ao APP, com o comando abaixo:

npm install cors

No arquivo index.js importe o pacote e capture a informação para o APP, como segue:

/\*importa a função express\*/

const express = require('express')

const cors = require('cors');

/\*armazena a função express \*/

const routes = require('./routes');

/\*armazena a função express\*/

const app = express();

app.use(cors(/\*{

  origin: 'http://meuapp.com'

}\*/));