**# testejs**

JS

Instalar homebrew (win: chocolatey)

Instalar node lts

Instalar npm

Instalar nvm

Instalar VS Code

Tema Dracula

ícones Material Icon theme

Arquitetura da aplicacao

o que é back-end?

Regras de negócio

Conexao bd

envio de e-mail

comunicacao com webservices

autenticacao de usuario

criptografia e segurança

como criar uma API RESTFUL

Front-end

web

mobile

outros serviços

JSON - o que é?

Verificar versões

node -v

npm -v

nvm -v

Inicializar

npm init -y

Abrir pasta no VS code

package.json - armazena as infors das dependencias que forem instaladas

O que é rota

Instalar o Express - mini framework de node para desenvolvimento web - para lidar com rotas

npm install express

Criar pasta backend

Criar o arquivo index.js

const express = require('express');

const app = express();

app.listen(3333)

Rodar

node index.js

Abrir browser: localhost:3333

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

//return response.send("Hello World");

return response.json({

evento: 'Hello Worlddd!!',

nome: 'Fernando'

});

})

React

Abordagem tradicional - retorna HTML

Abordagem SPA - Single Page Application

O back-end só retorna JSON

Fica sempre na mesma página

sem responsabilidade na apresentação d página

executar npx - executa um pacote externa sem a necessidade de instalar

npx create-react-app frontend

npm start

localhost:3000

Abrir o App.js

Mudar o texto para Hello World

React Native

Não converte o código para o código nativo

Ele utiliza um motor de JSCore- ele é implementado dentro do app

Xamarin converte o código

Expo - framework para utilizar, conjunto de bibliotecas para utilizar camera, gps

feito para apps pequenos e menos escaláveis

==

Rotas

Métodos

\* GET: buscar/listar uma informação do back-end

\* POST: criar uma informação no back-end

\* PUT: alterar uma informação do back-end

\* DELETE: deletar uma informação do back-end

Instalar o Insomnia para poder interagir com o back-end no POST, PUT, DELETE

mudar o .get para .post

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

return response.json({

evento: 'Hello Worlddd!!',

nome: 'Fernando'

});

})

Tipos de parâmetros:

\* Query Params: parâmetros nomeados enviados na rota após ? (Filtros, paginação)

\* Route Params: parâmetros utilizados para identificar recursos

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

Query

http://localhost:3333/users?name=Fernando

http://localhost:3333/users?page=2&name=Fernando&idade=18

Route

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

return response.json({

evento: 'Hello Worlddd!!',

nome: 'Fernando'

});

})

http://localhost:3333/users/1

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

const params = request.query;

console.log(params);

return response.json({

evento: 'Hello Worlddd!!',

nome: 'Fernando'

});

})

http://localhost:3333/users?name=Fernando&idade=18

no console sai:

{ name: 'Fernando', idade: '18' }

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

const params = request.params;

console.log(params);

return response.json({

evento: 'Hello Worlddd!!',

nome: 'Fernando'

});

})

http://localhost:3333/users/456

no console sai { id: '456' }

Request Body

depois da inicialização

app.use(express.json());

e então

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

const body = request.body;

console.log(body);

return response.json({

evento: 'Hello Worlddd!!',

nome: 'Fernando'

});

})

no insomnia:

post http://localhost:3333/users

body json:

{

"usuario" :"Fernando",

"idade" : 18

}

Instalar o NODEMON

npm install nodemon -D

-D somente para quando estiver desenvolvendo, é uma dependencia de desenvolvimento

no package.json

"scripts": {

"start": "nodemon index.js"

},

depois dar um npm start

Bancos de dados

SQL : MYSQL,

NoSQL: MongoDB, CouchDB, etc

SQLITE

Instalar o Driver: pacote oficial para node

SELECT \* FROM users

Query Builder: utilizando JS

table('users').select('\*').where()

Instalar Knex - Query builder js

npm install knex

npm install sqlite3

npx knex init

criar pasta src e jogar o index.js

mudar o package.json - nodemon src/index.js

criar dentro do src o routes.js

const express = require('express');

const routes = express.Router();

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

const body = request.body;

console.log(body);

return response.json({

evento: 'Hello Worlddd!!',

nome: 'Fernando H'

});

})

module.exports = routes;

no index.js

const express = require('express');

const routes = require('./routes');

const app = express();

app.use(express.json());

app.use(routes)

e no knexfile.js

development: {

client: 'sqlite3',

connection: {

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

}

},

criar a pasta database dentro do src

Entidades

- ONG

- Caso (Incident)

Funcionalidades

- Login de ONG

- Logout

- Cadastro de ONG

- Cadastrar novos Casos

- Deletar casos

- Listar casos específicos de uma ong

- Listar todos os casos

- Entrar em contato com a ONG

Criar as tabelas no banco

Knex migrations - manter o histórico de alterações, deleções, etc

Migration CLI

dentro da database criar a pasta chamada migrations

no knexfile.js

module.exports = {

development: {

client: 'sqlite3',

connection: {

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

},

migrations: {

directory: './src/database/migrations'

}

},

npx knex migrate:make create\_ongs

vai dar um warning no sqlite:

sqlite does not support inserting default values. Set the `useNullAsDefault` flag to hide this warning. (see docs http://knexjs.org/#Builder-insert).

mudar o knexfile

useNullAsDefault: true,

na migration criada:

//criacao da tabela

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', 2).notNullable();

})

};

//se algo der errado

exports.down = function(knex) {

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

};

rodar com

npx knex migrate:latest

criar a migration create\_incidents (tabela incidents)

para voltar a migration:

npx knex migrate:rollback

para saber os status das migrations:

npx knex migrate:status

criar pasta ongs no insomnia

enviar um json

mudar o routes.js

const express = require('express');

const routes = express.Router();

//pacote que vem junto com node para criptografia

const crypto = require('crypto')

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

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

//cria um id que é uma string aleatória e converter para string hexadecimal

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

return response.json();

})

module.exports = routes;

dentro da pasta database, criar connection.js

const knex = require('knex');

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

const connection = knex(configuration.development);

module.exports = connection

mudar a routes

const express = require('express');

const routes = express.Router();

//pacote que vem junto com node para criptografia

const crypto = require('crypto')

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

//a função anônima deve ser assíncrona, pois\*

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

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

//cria um id que é uma string aleatória e converter para string hexadecimal

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

//\* ela deve aguardar a inserção no bd para retornar o response

//o node vai esperar o finalizar para continuar

await connection('ongs').insert({

id,

name,

email,

whatsapp,

city,

uf,

})

return response.json({ id });

})

module.exports = routes;

testar no insomnia e deve retornar um json com id

criar outra rota no routes.js

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

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

return response.json(ongs);

});

testar - deve retornar uma lista com todas as ongs

Tela de computador com fundo preto

Descrição gerada automaticamente

Criar pasta controllers dentro de src e OngController.js

Tela de computador com fundo preto

Descrição gerada automaticamente

Passar as funções para o OngController.js

routes.js

const express = require('express');

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

const routes = express.Router();

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

//a função anônima deve ser assíncrona, pois\*

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

*module*.*exports* = routes;

OngController.js

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

//pacote que vem junto com node para criptografia

const crypto = require('crypto')

*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;

//cria um id que é uma string aleatória e converter para string hexadecimal

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

//\* ela deve aguardar a inserção no bd para retornar o response

//o node vai esperar o finalizar para continuar

await connection('ongs').insert({

id,

name,

email,

whatsapp,

city,

uf,

});

return response.json({ id });

}

}

Criar o IncidentController dentro da pasta controllers

Tela de celular com aplicativo aberto

Descrição gerada automaticamente

Para criar, o json deve ser

{

"title": "Caso 1",

"description": "Detalhes do caso",

"value": 120

}

Dentro de header,

Tela de celular com fundo preto

Descrição gerada automaticamente

Dentro de IncidentController.js

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

*module*.*exports* = {

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

//seria possível pegar no corpo o ong\_id, mas

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

//o dado do id da ong vem no cabeçalho

//no headers vem dados da autenticação do usuario, localização

const ong\_id = request.headers.authorization;

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

title,

description,

value,

ong\_id,

});

return response.json({id});

},

}

E no routes.js

const express = require('express');

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

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

const routes = express.Router();

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

//a função anônima deve ser assíncrona, pois\*

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

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

*module*.*exports* = routes;

Criar o index dos incidentes (função para listar todos os incidentes)

Deletar um incidente

Routes.js

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

IncidentController.js

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

const { id } = request.params;

const ong\_id = request.headers.authorization;

const incident = await connection('incidents')

.where('id', id)

.select('ong\_id')

.first();

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

//código de erro não autorizado

return response.status(401).json({error: 'Operation not permitted.'});

}

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

//código de resposta: no content

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

}

Criar o controller ProfileController.js

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

*module*.*exports*={

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);

},

}

Criar um get profile no Insomnia

Criar outra rota de LOGIN – SessionController

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

*module*.*exports* = {

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

const {id} = request.body;

const ong = await connection('ongs')

.where('id', id)

.select('name')

.first();

if(!ong){

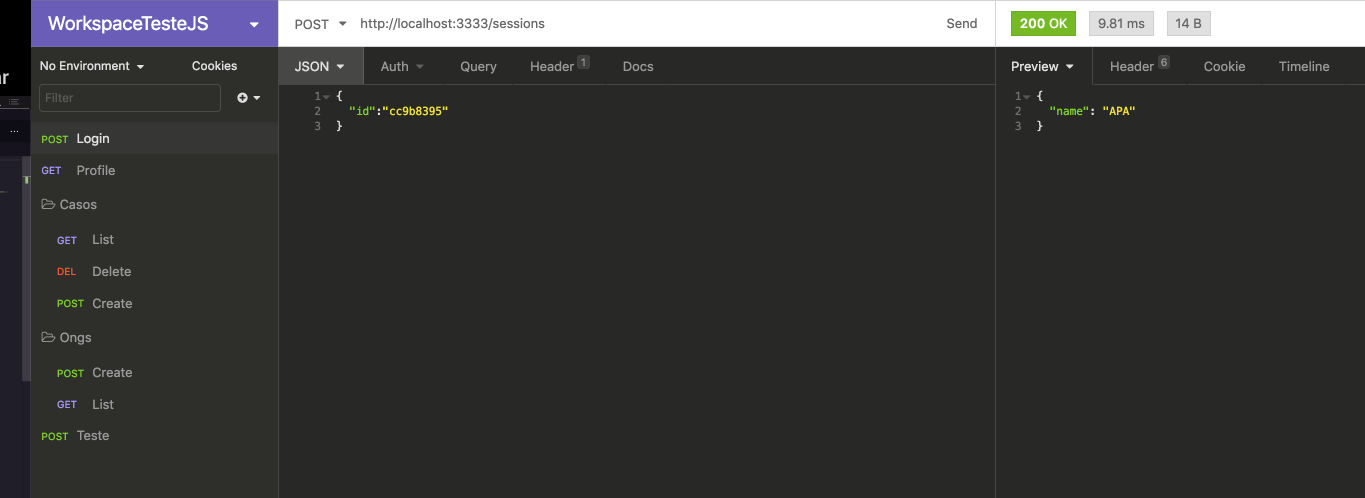
return response.status(400).json({error: 'No ong found with this ID'});

}

return response.json(ong);

}

}



Na Lista de incidentes criar a paginação dos incidentes – não retornar todos.

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

const{ page = 1 } = request.query;

//pegar o primeiro do array [ ]

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

.count();

*console*.log(count);

//pegar de 5 em 5 por página

const incidents = await connection('incidents')

.limit(5)

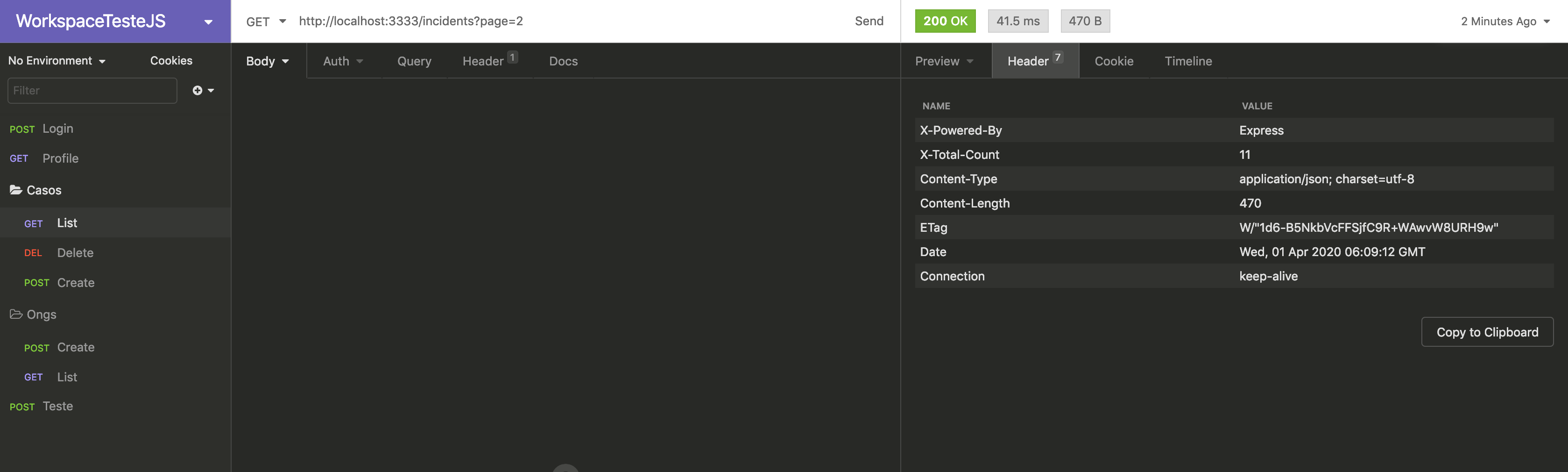
.offset( (page-1) \* 5 )

.select('\*');

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

return response.json(incidents);

},



O X-Total-Count foi inserido no header para que o front saiba quantos incidentes tem

Então agora é necessário fazer um join para que na listagem de incidentes se saiba os dados da ong e não apenas o ong\_id

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

const{ page = 1 } = request.query;

//pegar o primeiro do array [ ]

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

.count();

*console*.log(count);

//pegar de 5 em 5 por página

const incidents = await connection('incidents')

//apenas com esse join ele sobrepõe o id do incident

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

.limit(5)

.offset( (page-1) \* 5 )

//por isso o select não pode ser apenas uma string

//deve ser um array

.select(['incidents.\*',

'ongs.name',

'ongs.email',

'ongs.whatsapp',

'ongs.city',

'ongs.uf']);

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

return response.json(incidents);

},

Instalar o CORS

npm install cors

dentro do index.js

const express = require('express');

const cors = require('cors');

const routes = require('./routes');

const app = express();

//para desenvolvimento deixar vazio

//para produção inserir o origin

app.use(cors());

// app.use(cors({

// origin: 'http://site.com'

// }));

app.use(express.json());

app.use(routes)

---

Não enviar o node\_modules para o git