


Semana OmniStack 11

 Aulas realizadas em 23/03/20 - 27/03/20

Conhecendo a OmniStack

▼ Configurar ambiente de desenvolvimento

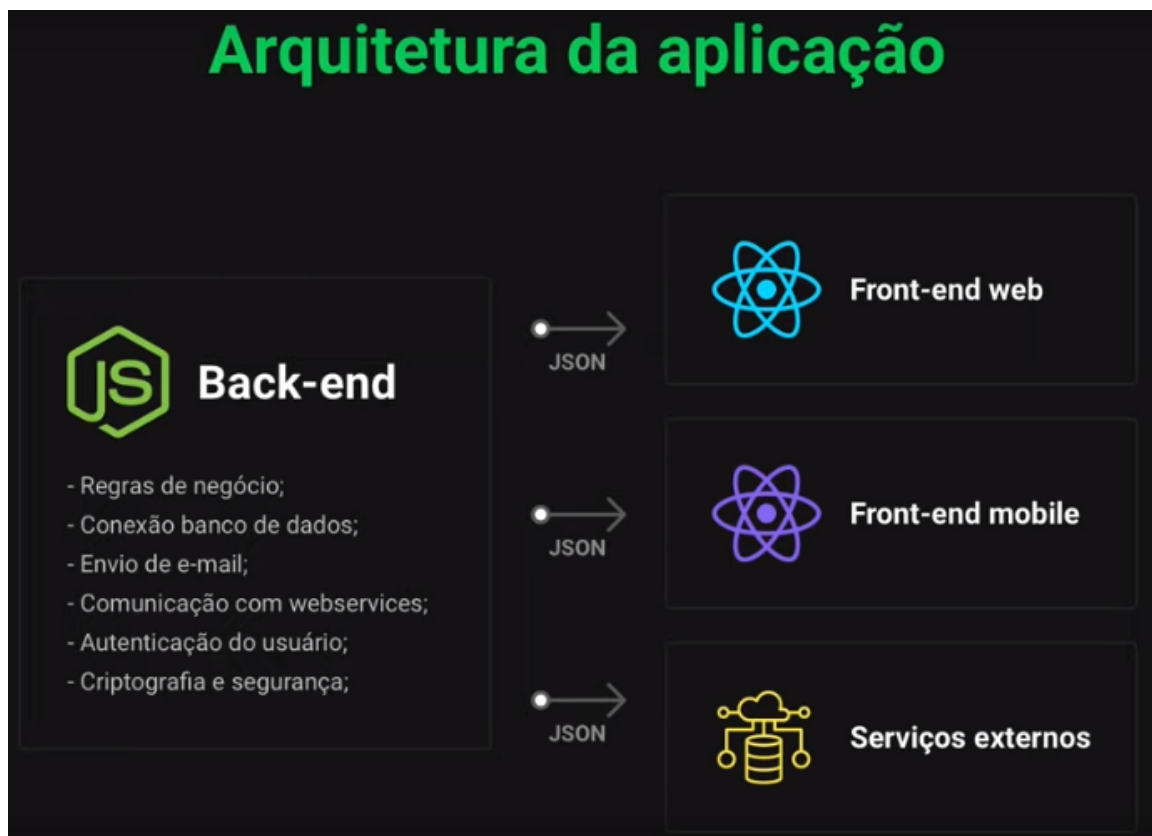
- **NodeJS e npm**

- Instalar usando package-manager Chocolatey:

<https://nodejs.org/en/download/package-manager/#windows>

- **Visual Studio Code**

▼ Back-end x Front-end



▼ Criando projeto com NodeJS

- Criar uma pasta app, diretório de toda a aplicação
- Dentro dela, criar uma pasta backend

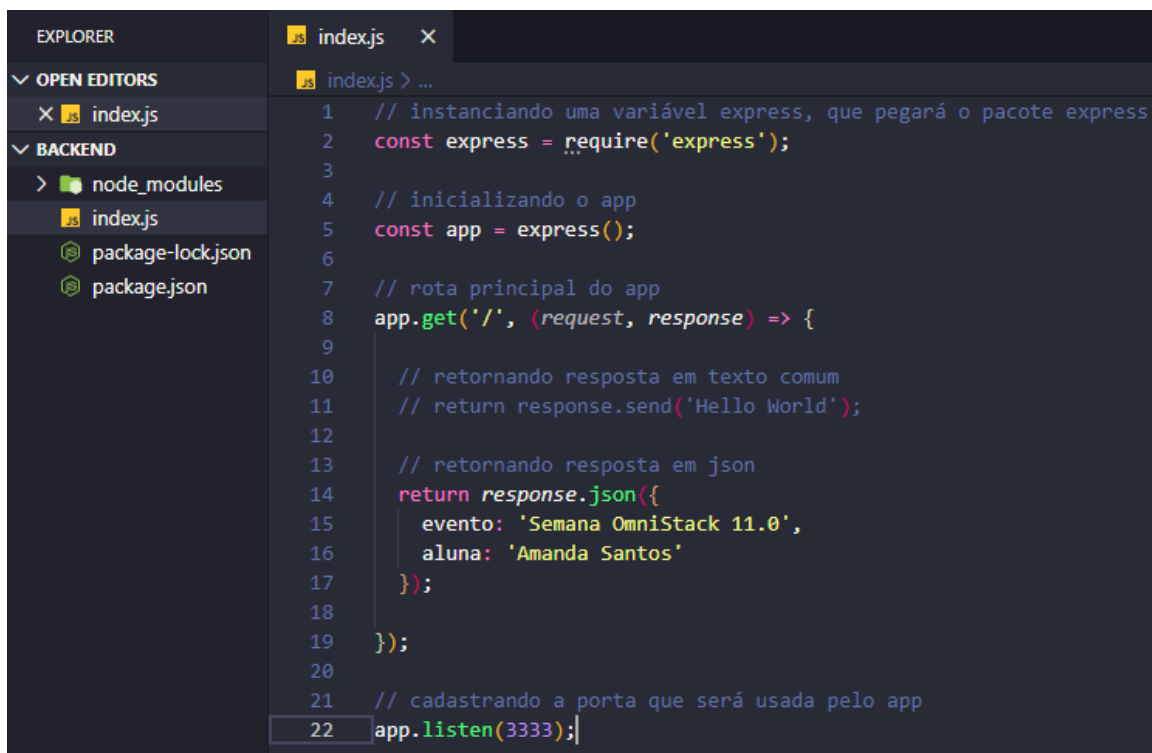
- Após criar uma pasta backend, executar nela o comando

```
npm init -y
```

- Com isso, é criado um arquivo package.json.
- Em seguida, executar dentro da pasta o comando a seguir, para instalar a pasta node_modules

```
npm install express
```

- Criando um arquivo index.js e fazendo Hello World:



The screenshot shows the VS Code interface. On the left, the Explorer sidebar shows a project structure with a 'BACKEND' folder containing 'index.js', 'package-lock.json', and 'package.json'. The 'index.js' file is open in the editor. The code in 'index.js' is as follows:

```
1 // instanciando uma variável express, que pegará o pacote express
2 const express = require('express');
3
4 // inicializando o app
5 const app = express();
6
7 // rota principal do app
8 app.get('/', (request, response) => {
9
10   // retornando resposta em texto comum
11   // return response.send('Hello World');
12
13   // retornando resposta em json
14   return response.json({
15     evento: 'Semana OmniStack 11.0',
16     aluna: 'Amanda Santos'
17   });
18
19 });
20
21 // cadastrando a porta que será usada pelo app
22 app.listen(3333);
```

- Rodar o comando a seguir na pasta do projeto:

```
node index.js
```

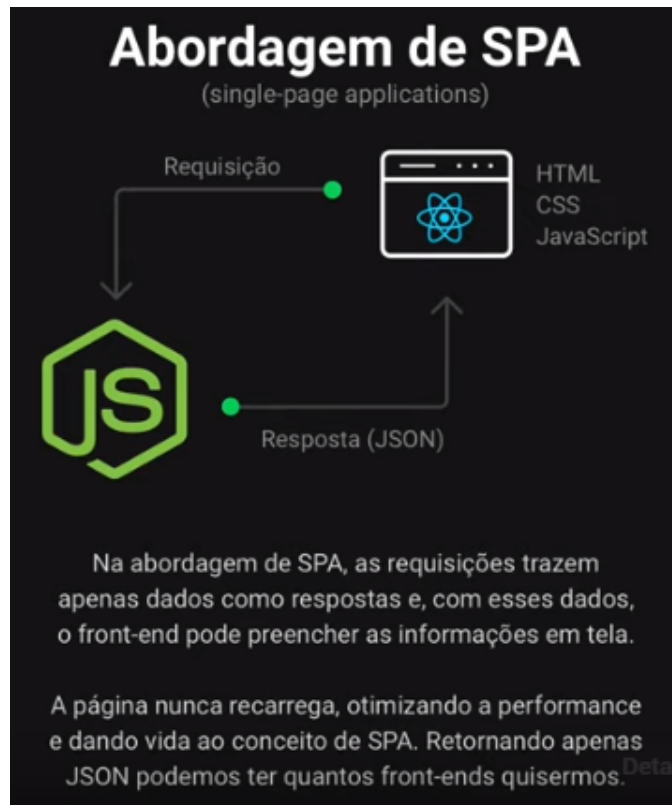
- Abrindo localhost:3333 no navegador

```
Menu | Semana OmniStack | Rocke X | Semana OmniStack 11 X | localhost:3333
localhost:3333
1 // 20200323105826
2 // http://localhost:3333/
3
4 {
5   "evento": "Semana OmniStack 11.0",
6   "aluna": "Amanda Santos"
7 }
```

▼ Criando projeto com ReactJS

- *Abordagem tradicional x Abordagem de SPA*





- Executar o comando a seguir dentro da pasta app para criar uma pasta frontend, onde ficará o projeto ReactJS:

```
npx create-react-app frontend
```

- Para executar o projeto, rodar o comando a seguir dentro de frontend

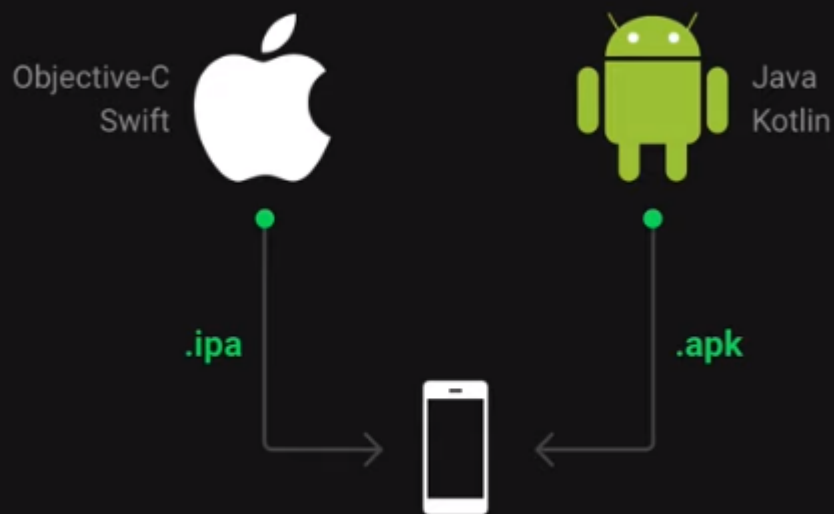
```
npm start
```

- Com isso, o projeto começa a rodar em localhost:3000

▼ Criando projeto com React Native e Expo

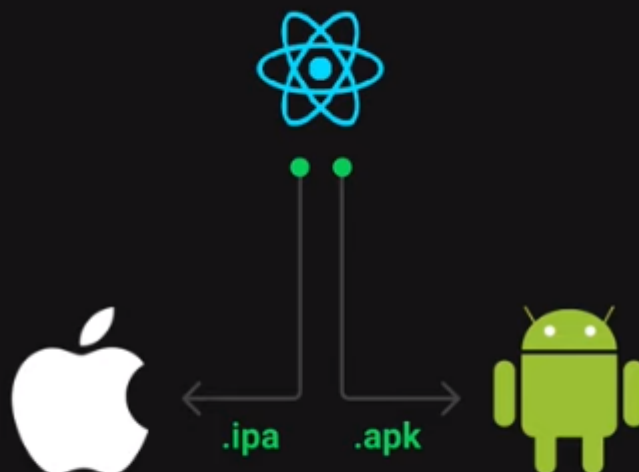
- *Abordagem tradicional x Abordagem do React Native*

Abordagem tradicional



Na abordagem tradicional, criamos uma aplicação para iOS e outra para Android, e nesses casos, o trabalho se torna repetido tanto para criação quanto para as alterações no projeto.

Abordagem do React Native



Todo código feito é em JavaScript, esse código **não é convertido em código nativo**, melhor do que isso, o dispositivo passa a entender o código JavaScript e a interface gerada é **totalmente nativa**.

- Expo



Back-End

▼ Node.js e Express

- Rotas e recursos:

- Rotas são um determinado endereço do site. Ex.: `www.meusite.com/users/1`. Recursos são uma determinada informação que eu quero acessar por meio da rota, que geralmente está relacionada a uma tabela no banco de dados. Ex.: em `www.meusite.com/users/1`, quero acessar o usuário com a identificação 1.
- Para melhor organização, criar uma pasta `src` e colocar `index.js` dentro dela. Em seguida, criar um arquivo separado `routes.js`, onde ficarão todas as rotas do projeto. Garantir a exportação das rotas em `routes.js` e a sua importação pela aplicação em `index.js`.

The screenshot shows the VS Code interface with the Explorer on the left and the Editor on the right. The Explorer shows a project structure with a `src` folder containing `index.js` and `routes.js`. The Editor shows the `routes.js` file with the following code:

```

1  const express = require('express');
2
3  const routes = express.Router();
4
5  > routes.post('/', (request, response) => { ...
16  });
17
18  module.exports = routes;

```

The screenshot shows the VS Code interface with the Explorer on the left and the Editor on the right. The Explorer shows a project structure with a `src` folder containing `index.js` and `routes.js`. The Editor shows the `index.js` file with the following code:

```

1  // instanciando uma variável express, que pegará o pacote express
2  const express = require('express');
3
4  // importando o arquivo com as rotas
5  const routes = require('./routes');
6
7  // inicializando o app
8  const app = express();
9
10 // permitindo que arquivos json sejam compreendidos
11 app.use(express.json());
12
13 // chamando o arquivo de rotas
14 app.use(routes);
15
16 // cadastrando a porta que será usada pelo app
17 app.listen(3333);

```

• Métodos HTTP:

- **GET:** buscar uma informação do back-end
- **POST:** criar uma informação no back-end
- **PUT:** atualizar uma informação no back-end
- **DELETE:** apagar uma informação do back-end
- **Tipos de parâmetros:**
 - **Query Params:** Parâmetros nomeados (você sabe qual o nome da variável enviada: nome=Maria, idade=20) enviados na rota após "?". Muito usados em filtros e paginação. Ex.:
www.meusite.com/users?name=Amanda.
 - É acessado usando *request.query*.

```
app.get('/users', (request, response) => {  
  const params = request.query;  
  
  console.log(params);  
  
  return response.json({  
    evento: 'Semana OmniStack 11.0',  
    aluno: 'Diego Fernandes'  
  });  
});  
  
app.listen(3333);
```

- **Route Params:** Parâmetros utilizados para identificar recursos. Não são nomeados, pela URL você só sabe o valor do parâmetro. Ex.:
www.meusite.com/users/1 (/users/:id) vai buscar todos os usuários de id = 1.
 - É acessado usando *request.params*.


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

  console.log(params);

  return response.json({
    evento: 'Semana OmniStack 11.0',
    aluno: 'Diego Fernandes'
  });
});

app.listen(3333);
```

- **Request Body:** é enviado o corpo da requisição (um arquivo JSON contendo os dados que vieram de um formulário, por exemplo), utilizado para inserir ou alterar uma informação/recurso.
 - É acessado com *request.body*.

```
app.post('/users', (request, response) => {
  const body = request.body;

  console.log(body);

  return response.json({
    evento: 'Semana OmniStack 11.0',
    aluno: 'Diego Fernandes'
  });
});

app.listen(3333);
```

▼ **Utilizando o Insomnia:** útil para testar as rotas de POST, PUT e DELETE, por exemplo, enquanto ainda não há um front-end.

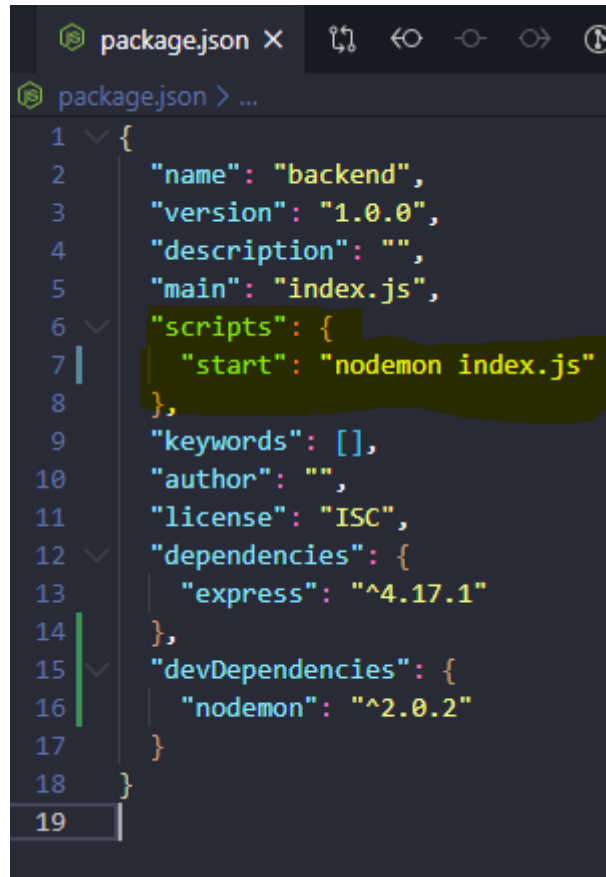
<https://insomnia.rest>

▼ **Configurando Nodemon:**

- O Nodemon permite que o servidor seja atualizado automaticamente após uma alteração no código, durante o desenvolvimento da aplicação. Para instalá-lo, basta rodar o comando a seguir (-D para instalar somente como uma dependência de desenvolvimento - *devDependencies* -, já que ele é usado somente durante o desenvolvimento).

```
npm install nodemon -D
```

- Com isso, basta definir um comando dentro de *package.json* > *scripts* para executar o Nodemon. Nesse caso, o comando definido foi *start*.



```
1 {  
2   "name": "backend",  
3   "version": "1.0.0",  
4   "description": "",  
5   "main": "index.js",  
6   "scripts": {  
7     "start": "nodemon index.js"  
8   },  
9   "keywords": [],  
10  "author": "",  
11  "license": "ISC",  
12  "dependencies": {  
13    "express": "^4.17.1"  
14  },  
15  "devDependencies": {  
16    "nodemon": "^2.0.2"  
17  }  
18 }  
19
```

- Assim, para executar o projeto basta rodar o comando a seguir:

```
npm start
```

▼ Diferenças entre banco de dados:

- SQL x NoSQL
- Neste caso, será usado o SGBD SQL **SQLite**.

▼ Configurando banco de dados:

- Pode-se acessar o banco através do código usando drivers (escrever as queries diretamente no código - ex.: *SELECT * FROM users*) ou

usando um **Query Builder**, que constrói a query usando código JavaScript (ex.: `table('users').select('*').where(...)`).

- Neste caso, será usado o Query Builder **Knex** (<http://knexjs.org>). Para instalá-lo, basta rodar

```
npm install knex
```

- Depois, rodar o comando de acordo com o banco que será instalado. Neste caso,

```
npm install sqlite3
```

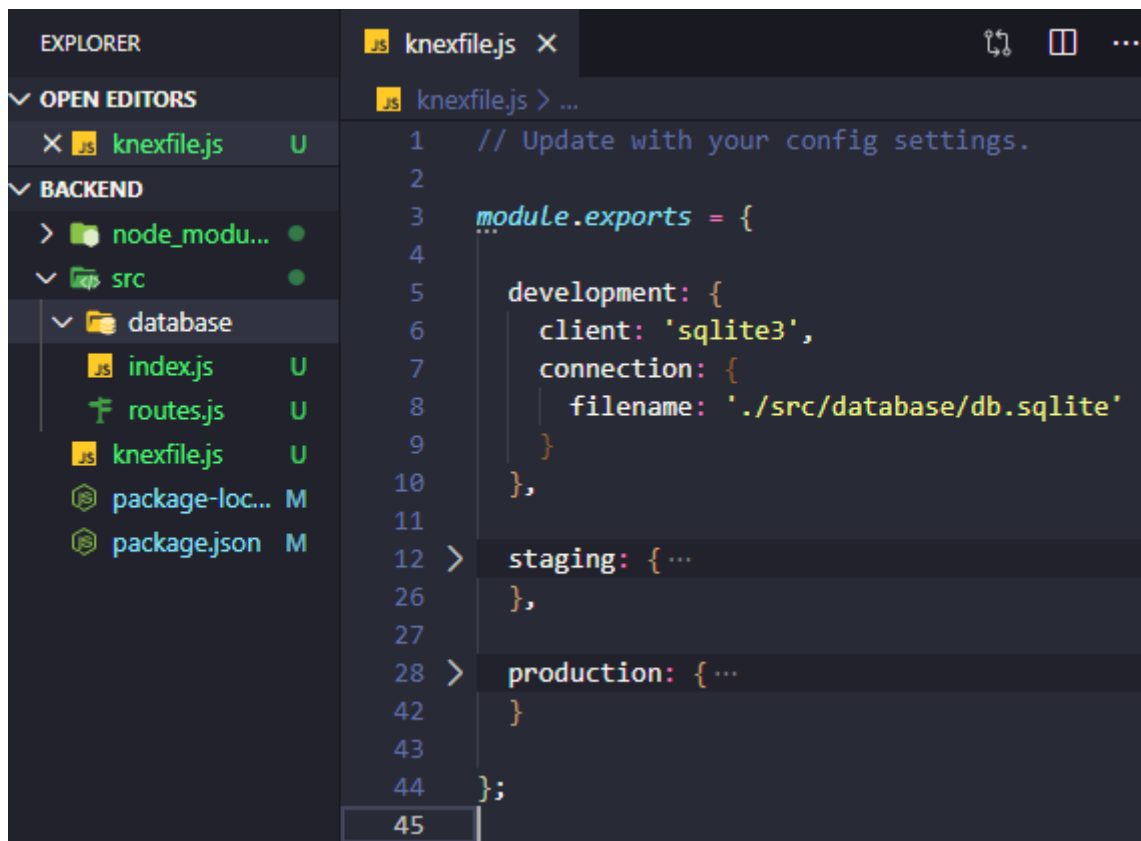
- Rodar o comando

```
npx knex init
```

para criar o arquivo que contém as configurações do banco de dados. Nele, existem as configurações de *development* (desenvolvimento), *staging* (simulação do ambiente de produção para os devs testarem) e *production* (produção).

```
knexfile.js X
knexfile.js > ...
1 // Update with your config settings.
2
3 module.exports = {
4
5   development: {
6     client: 'sqlite3',
7     connection: {
8       filename: './dev.sqlite3'
9     }
10  },
11
12 > staging: { ...
26   },
27
28 > production: { ...
42   }
43
44 };
```

- Alterar as configurações de conexão para o arquivo onde será armazenado o banco de dados. Neste caso, dentro de *src* foi criada uma pasta *database*, onde ficará o arquivo *db.sqlite*.



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the project structure: 'OPEN EDITORS' with 'knexfile.js' (U), 'BACKEND' with 'node_modu...' (●), 'src' (●) containing 'database' (●) with files 'index.js' (U), 'routes.js' (U), 'knexfile.js' (U), 'package-loc...' (M), and 'package.json' (M). The main editor window shows 'knexfile.js' with the following code:

```
1 // Update with your config settings.
2
3 module.exports = {
4
5   development: {
6     client: 'sqlite3',
7     connection: {
8       filename: './src/database/db.sqlite'
9     },
10   },
11
12   > staging: { ...
26   },
27
28   > production: { ...
42   }
43
44 };
45
```

▼ Entidades e funcionalidades da aplicação:

- **Entidades:**

- ONG
- Caso (incident)

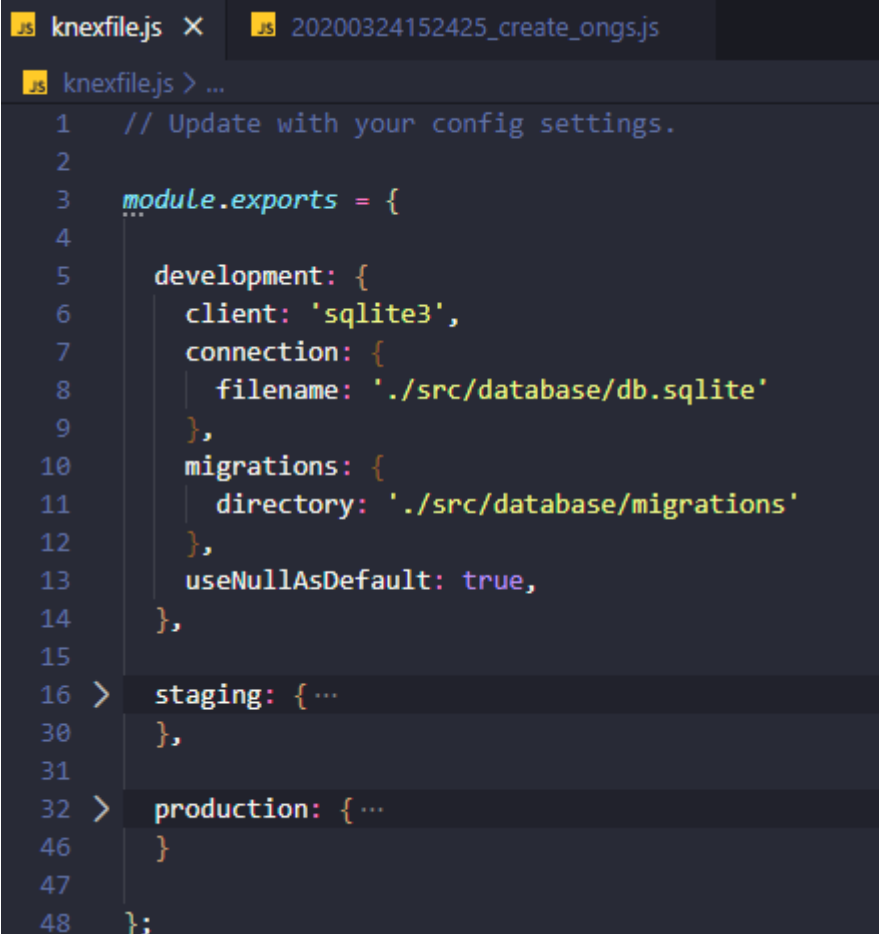
- **Funcionalidades:**

- Login de ONG
- Logout de ONG
- Cadastro de ONG
- Cadastro de novos casos
- Exclusão de casos
- Listagem de casos de uma ONG específica
- Listagem de todos os casos
- Entrar em contato com a ONG

- Para criar as tabelas rapidamente, serão usadas migrations. **Migrations** são um "controle de versão" de bancos de dados, que permitem

armazenar todos os estados do banco e as alterações que são aplicadas nele.

- Criar um diretório migrations dentro de database.
- Configurar a migration dentro de knexfile.js:

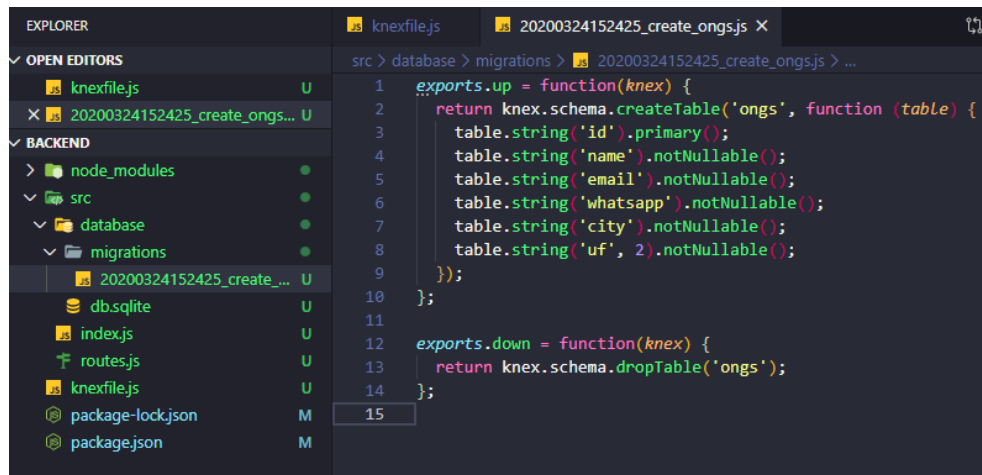


```
knexfile.js X 20200324152425_create ONGs.js
knexfile.js > ...
1 // Update with your config settings.
2
3 module.exports = {
4
5   development: {
6     client: 'sqlite3',
7     connection: {
8       filename: './src/database/db.sqlite'
9     },
10    migrations: {
11      directory: './src/database/migrations'
12    },
13    useNullAsDefault: true,
14  },
15
16 > staging: { ...
30 },
31
32 > production: { ...
46 }
47
48 };
```

- Criando uma migration para criar a tabela ONGs:

```
npx knex migrate:make create ONGs
```

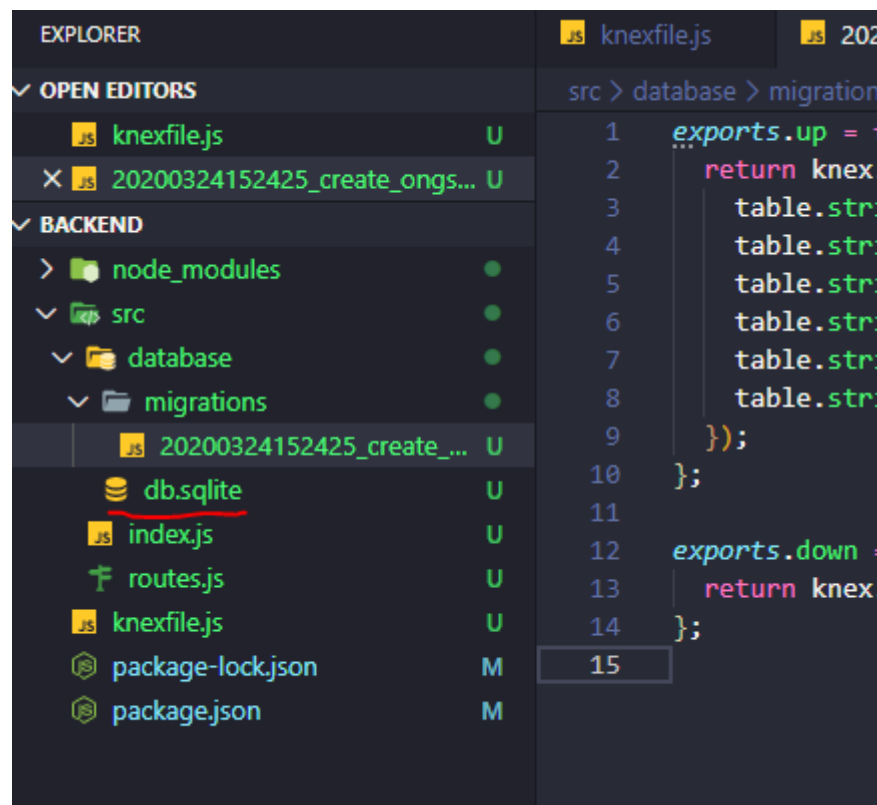
- É criado um arquivo dentro da pasta migrations. Nele, colocar o seguinte código:
 - A função *up* tem o que deve ser feito quando a migration for executada. Neste caso, é um código para criar a tabela *ongs*.
 - A função *down* tem o que deve ser feito caso a migration dê errado. Neste caso, é um código para excluir a tabela *ongs*.



- Para executar a migration:

```
npx knex migrate:latest
```

- Com isso, o banco de dados já é criado com a tabela *ongs*.



- Exemplo com a tabela *incidents*, que possui uma chave estrangeira referenciando *ongs*:

```
src > database > migrations > 20200324154342_create_incidents.js > ...
1  exports.up = function(knex) {
2    return knex.schema.createTable('incidents', function (table) {
3      table.increments();
4
5      table.string('title').nullable();
6      table.string('description').nullable();
7      table.decimal('value').nullable();
8
9      table.string('ong_id').nullable();
10
11     table.foreign('ong_id').references('id').inTable('ongs');
12   });
13 };
14
15 exports.down = function(knex) {
16   return knex.schema.dropTable('incidents');
17 };
18
```

- Para desfazer uma migration, caso tenha cometido algum erro, basta fazer um *rollback* com o comando a seguir:

```
npx knex migrate:rollback
```

- Para listar todas as migrations executadas:

```
npx knex migrate:status
```

▼ Construção do back-end

▼ Adicionando módulo CORS

- É um módulo para adicionar segurança à aplicação. Pode ser adicionado com o comando a seguir:

```
npm install cors
```

- Em *index.js*, importar o CORS:


```
index.js  X
src > index.js > ...
1 // instanciando uma variável express, que pegará o pacote express
2 const express = require('express');
3
4 // importando o módulo de segurança cors
5 const cors = require('cors');
6
7 // importando o arquivo com as rotas
8 const routes = require('./routes');
9
10 // inicializando o app
11 const app = express();
12
13 // cors
14 app.use(cors());
15
```

▼ Enviando back-end ao GitHub

Outros

📌 Site para prototipação de telas: <https://www.figma.com>

📌 Site para anotações: <https://www.notion.so>