

# TRYBE

## Modulo III – Back-end

### Bloco 30: Projeto Trybeer

Aqui listarei os principais desafios gerais de entendimento e de setting do projeto assim como resolvê-los.

#### 1) Integração front/back

##### Conectar bd

Atenção particular para:

- *models/connection.js*
- variáveis de ambiente *.env* correspondendo com o Workbench (já que bd é sql)

##### Popular bd

*sudo service mysql start* (quando não é automático no boot do compu)

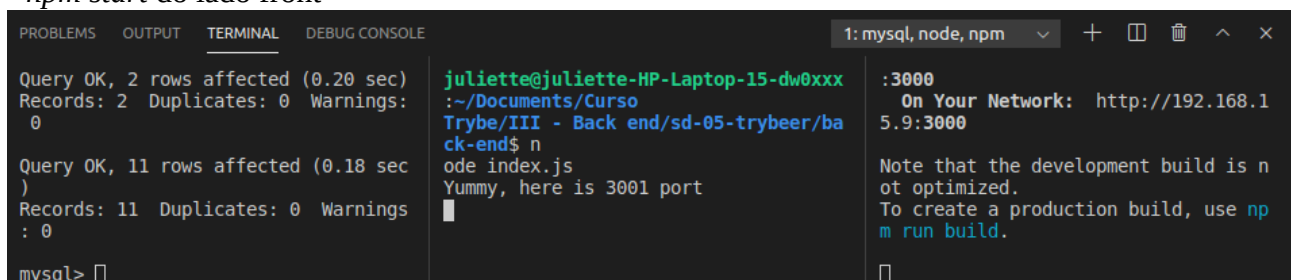
*mysql -u root -p*

> *source yourpath/script.sql*

##### Fullstack development mode

##### Rodar no terminal

- db como indicado acima e *node index.js* do lado back
- *npm start* do lado front



The screenshot shows a terminal window with three panes. The left pane shows MySQL queries and results: 'Query OK, 2 rows affected (0.20 sec) Records: 2 Duplicates: 0 Warnings: 0' and 'Query OK, 11 rows affected (0.18 sec) Records: 11 Duplicates: 0 Warnings: 0'. The middle pane shows a shell prompt 'juliette@juliette-HP-Laptop-15-dw0xxx' and the command 'node index.js' with the output 'Yummy, here is 3001 port'. The right pane shows the application's output: ':3000 On Your Network: http://192.168.1.5.9:3000' and a note about the development build.

##### Testar

Todo o anterior deve ser ativado para conseguir testar, via na raiz *npm run cypress:open*

##### Consultar no Browser

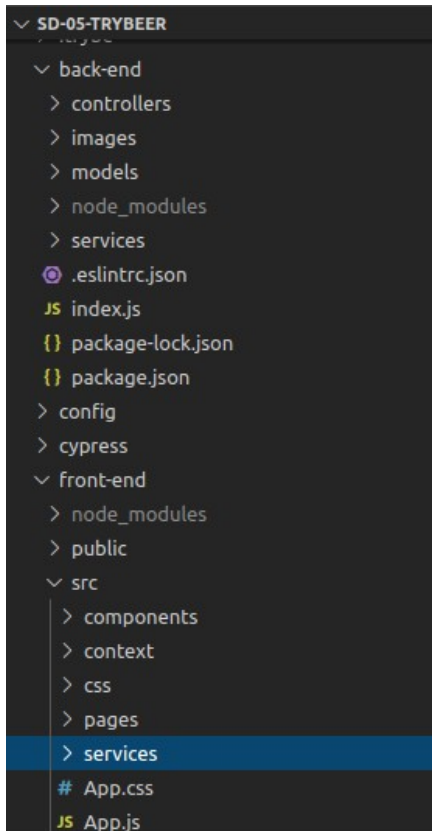
- *localhost:3000* (front)
- *localhost:3001* (back)(dica: chrome extension JSON Formatter para ver bem objetos do bd)

##### Como back e front conectam

Lembra como no front-end fazia fetch de API para depois poder manipular os dados disponibilizados?

Pois dessa vez, simplesmente sua API está no seu code de backend.

Você cria funções para pegar os dados que interessam do mesmo jeito.



→ Arquitetura do projeto.

Na pasta front-end/app/services, criar as funções de fetch como no seguinte exemplo (dica- usar axios para tratar erros facilmente):

```
import axios from 'axios';
const API_URL = 'http://localhost:3001';

export const getProducts = () => {
  const products = axios
    .get(`${API_URL}/products`, {})
    .then((response) => response.data);
  return products;
};

export const login = (email, password) => {
  const userInfo = axios
    .post(`${API_URL}/users/login`, { email, password })
    .then((response) => response.data)
    .catch((err) => err);
  return userInfo;
};
```

//etc, correspondendo a todos os endpoints criados no back-end assim chamáveis no front

## 2) ESLint

### Instalar e rodar localmente

`npm install eslint` (antes de cada push, `npm uninstall eslint`)

Rodar via comando `node_modules/.bin/eslint youroptionalpath`.

### Principais dificuldades e respostas

#### \* ObjectOf do propTypes

Solucionar: substituir por `node` ou por `shape`. Exemplos:

```
Provider.propTypes = {
  children: PropTypes.node.isRequired,
};
```

```
Profile.propTypes = {
  updateUser: PropTypes.func.isRequired,
  history: PropTypes.shape({
    location: PropTypes.shape({
      pathname: PropTypes.string,
    }).isRequired,
  }).isRequired,
}
```

#### \* missing dependencies nos useEffect

Solucionar: incluir no `[]` do segundo parâmetro. Exemplo:

```
useEffect(() => {
  getSaleById(id).then((response) => setStatus(response[0].status));
}, [id, setStatus]);
```

#### \* magic number

Solucionar: primeiro armazenar o número numa variável.