Configurações e códigos simples de backend com NodeJS no Linux Ubuntu 20.04

Instalando o NODE:

Terminal:

$ curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.0/install.sh | bash
$ source .bashrc
$ nvm install –lts

Instalando o YARN:

Terminal:

$ curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | sudo apt-key add -
$ echo "deb https://dl.yarnpkg.com/debian/ stable main" | sudo tee /etc/apt/sources.list.d/yarn.list
$ sudo apt update && sudo apt install --no-install-recommends yarn

- **Pacotes na pasta do projeto:**

1. Express
$ npm install express

2. NodeMon
$ npm install -D nodemon

3. Banco
$ npm install sqlite3

4. Knex
$ npm install knex

5. Cors
$ npm install cors

- **Passos para configurar o backend:**

1º Com a pasta desejada aberta no terminal:
$ npm init -y

- **Iniciando código**

1º Com o VSCode aberto, e importado a pasta criada, crie um arquivo server.js

2º Em server.js adicione:

```
const express = require('express');

const app = express();

app.listen(3001);
```

3º No terminal:
$ node server.js

4º Abra no navegador para conseguir ver sua aplicação no endereço: http://localhost:3001

vai ter algo como: `Cannot GET /`

5º Em server.js

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
        res.send('Hello World')

})
app.listen(3001);
```

Req = requeste → pega requisição
Res = response → envia resposta

6º No terminal: **$ node server.js.** Dando F5 no navegador o `Cannot GET /` passará a ser `Hello World`

- Configurando para que não precise dar node server.js no terminal toda vez

1º em package.json após "test": "…", adicionar:
→ **"dev": "nodemn server.js"**

Agora, para iniciar o servidor no terminal: **$ npm run dev**

Rotas

GET: utilizado para buscar/listar informações no backend.
Ex. código estrutura:

```
app.get('/users', (req, res) => {
        res.json({
                nome: "Node",
                empresa: "JS"
        });

});
```

POST: utilizado para criar/salvar informações no backend
Ex. código estrutura:

```
app.post('/users', (req, res) => {
        const params = req.params;
        res.json(params);
});
```

PUT: utilizado para alterar informações no backend
Ex. código estrutura:

```
app.put('/users', (req, res) => {
        const params = req.params;
        res.json(params);
});
```

DELETE: utilizado para deletar informações no backend
Ex. código estrutura:

```
app.delete('/users', (req, res) => {
        const params = req.params;
        res.json(params);
});
```

Parâmetros

QUERY: parâmetros nomeados enviados na rota (o link possui os elementos (ex. nome=Ana&idade=23))
Código:

```
app.get('/users', (req, res) => {
        const params = req.query;
        console.log(params);
        res.json(params);

})
```

ROUTE PARAMS: usados para identificar os recursos (a rota (link) possui o ID)
Código:

```
app.post('/users/:id', (req, res) => {
        const params = req.params;
        console.log(params); //listando os parâmetros
        res.json(params);
```

REQUEST BODY: através do corpo da requisição

```
app.use(express.json());

app.post('/users', (req, res) => {
        const params = req.body;
        console.log(params); //listando os parâmetros
        res.json(params);
```

Iniciando com o Banco de Dados

1ª Forma: Usando armazenamento em memória/arquivo
1º em server.js inicializar a variável do bd:
const sqlite3 = require('sqlite3').verbose();

2º escolher o tipo de conexão (memória ou arquivo)

```
//em memória, quando encerra, acaba perdendo todas as conexões
let db = new sqlite3.Database(':memory:', (erro)=>{
        if(erro){
                return console.error(erro.message);
        }
        console.log("Conectado com sucesso!");

});

//em arquivo, quando encerra, as conexões ficam salvas
let db = new sqlite3.Database('db.sqlite', (erro)=>{
        if(erro){
                return console.error(erro.message);
        }
        console.log("Conectado com sucesso!");

});
```

2ª Forma: Usando KNEX

- **Reestruturando**

1º Delete o arquivo db.sqlite

2º No terminal:
$ npx knex init

3º criar nova pasta src

4º transferir server.js para src

5º criar pasta database dentro de src

6º criar aquivo db.sqlite dentro de database

7ºem knexfile.js na parte de development o filename ficará:
filename: './src/database/db.sqlite'

8º em package.json modificar o dev
"dev": "nodemon ./src/server.js"

9º em src criar arquivo routes.js e adicionar o seguinte código:
const express = require('express');

const routes = express.Router();

10º em server.js manter apenas:
const express = require(‘express’);
const cors = require(‘cors’);
const app = express();

app.use(express.json());
app.use(cors());

app.listen(3001);

11º exporte o módulo, adicione no final do arquivo routes.js:
module.exports = routes;

12º importe em server.js o arquivo routes.js
const routes = require(‘./routes’);

13º adicione a linha de código em server.js
app.use(routes);

- **Configurando**

1º detro da pasta database criar pasta migrations, que é onde vão ficar os arquivos com as alterações de estado do banco.

2º dentro de knexfile.js, em development, adicionar após connection:
```
migrations: {
        directory: ‘./src/database/migrations’
},

useNullAsDefault: true
```

3º com o servidor parado, crie o seu arquivo de migração no terminal:
$ npx knex migrate:make create_users

4º no arquivo criado dentro da pasta migrations, adicionar em exports.up os dados da table
```
return knex.schema.createTable('users', function (table) {
  table.string(‘id’).primary();
  table.string('name').notNullable();
  table.string('email').notNullable();
  table.string('empresa');
  table.integer(‘idade’);
})
```

5º no arquivo criado dentro da pasta migrations, adicionar em exports.down
return knex.schema.dropTable(‘users’);

6º criar de fato a tabela, inserir no terminal:
npx knex migrate:latest

- **Conectando rotas ao usuario**

1º criar arquivo connection.js dentro da pasta database

2º em connection.js adicionar:

```
const knex = require('knex');
const config = require('../../knexfile');

connection = knex(config.development);

module.exports = connection;
```

3º Utilizando método POST para cadastrar usuário dentro de routes.js

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

app.post('/users', assync(req, res) => {
        const {name, e-mail, empresa, idade} = req.body;
        const id = crypto.randomBytes(4).toString('HEX');

        await connection('users').insert({
                id,
                name,
                e-mail,
                empresa,
                idade
        })
        res.json(id);
})
```

4º Utilizando método GET para listar usuários dentro de routes.js

```
app.get('/users', assync(req, res) => {
        const users await connection('users').select('*')
        res.json(users);
})
```

Reestruturando para ficar no padrão MVC Web

1º criar em src uma pasta chamda controller

2º criar arquivo userController.js em controller e adicionar o código:
//código que mostra todos os usuarios, apenas um usuario, adiciona novo usuario, etc...

```
const crypto = require('crypto');
const connection = require('../database/connection');
const {routes} = require('../routes');

module.exports = {
	assync create(req, res){
		const {name, email, empresa, idade} = req.body;
		const id = crypto.randomBytes(4).toString('HEX');

		await connection('users').insert({
			id,
			name,
			email,
			empresa,
			idade
		})
		return res.json(id);
	}

	assync list(req, res){
		const users = await connection('users').select('*');
		return res.json(users);
	}
	assync show(req, res){
		const {id} = req.params;
		const user = await connection('users').where('id',id);
		return res.json(user);
	}
	assync update(req, res){
		const {id} = req.params;
		const {name, email, empresa, idade} = req.body;
		await connection('users').where('id',id).update({
			id,
			name,
			email,
			empresa,
			idade
	})
		return res.status(204).send();
	}
	assync delete(req, res){
		const {id} = req.params;
		const user = await connection('users').where('id',id).delete();
		return res.status(200).send();
	}
}
```

4º Em routes.js o códio será:

```
const express = require(‘express’);
const routes = express.Router();
const userController = require(‘./controller/userController’);

routes.post(‘/users’, userController.create);

routes.get(‘/users’, userController.list);
routes.get(‘/users/:id’, userController.show);

routes.put(‘/users/:id’, userController.update);

routes.delete(‘/users/:id’, userController.delete);
```