Instituto Federal de Educação, Ciência e Tecnologia

Triângulo Mineiro – Campus Uberlândia Centro

Tecnologia em Sistemas para Internet

Projeto e Desenvolvimento de Software II

Prof. Carlos Eduardo de Carvalho Dantas

carloseduardodantas@iftm.edu.br

Parte II – Back-end

INSTITUTO FEDERAL TRIÂNGULO MINEIRO Campus Uberlândia Centro

AGENDA

2. **Desenvolvimento Back-end**
   - Nodejs
   - **Express**

- **Versão 6 – 47 slides – atualização nos slides 28 e 29**

Estudo de Caso: *NodeJS*

- NodeJS

- Plataforma altamente escalável de de baixo nível
- Trabalha no lado servidor como aplicação back-end, isto é, fornecendo objetos json via endpoints REST para aplicações clientes que possuem interfaces para o usuário final
- Programa-se em Javascript
- É single-thread – cada aplicação tem uma única thread
- Programação assíncrona i/o não bloqueante
- Event Loop – orientado a eventos de i/o – connect do banco,open de um arquivo, data de streaming de dados

Estudo de Caso: *NodeJS*

- NodeJS

- Fica em loop infinito escutando eventos. Com o evento em execução, pode-se programar um callback.

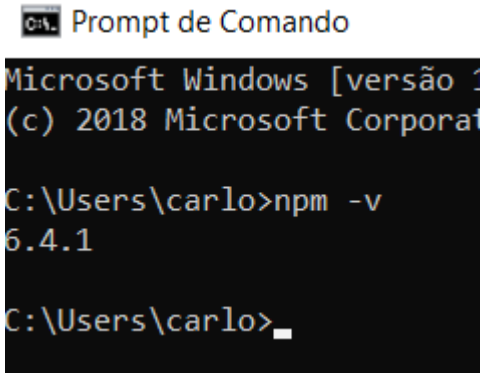
Estudo de Caso: *NodeJS*

- **Por que aprender nodejs?**
  - **Javascript everywhere** – serverside, sem serializar json
  - **Comunidade ativa**
    - **Google**:https://groups.google.com/forum/#!forum/nodebr
    - •**Facebook**:https://facebook.com/groups/nodejsbrasil
    - •**Slack**:https://nodebr.slack.com
  - **Ready for realtime** – usa websockets, que permite trafegar dados através de uma única conexão bidirecional, tratando todas as mensagens por meio de eventos javascript
  - **Big Players** – Linkedin, wallmart, microsoft, netflix, uber e paypal são algumas das grandes empresas usando node.js

Estudo de Caso: *NodeJS*

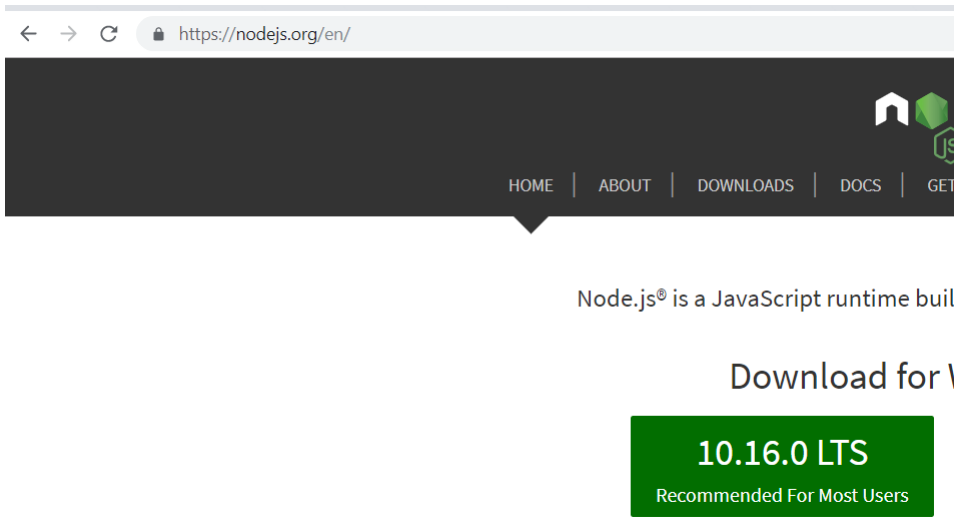
- O comando npm –v verifica se o nodejs está instalado

```
maEcommerceCLI> npm -v
6.4.1
PS D:\Magistério\IFTM\Au
SistemaEcommerceCLI>
```



- Hello world com nodejs

```
C:\Users\carlo>node
> console.log("Hello World");
Hello World
undefined
```



Estudo de Caso: *NodeJS*

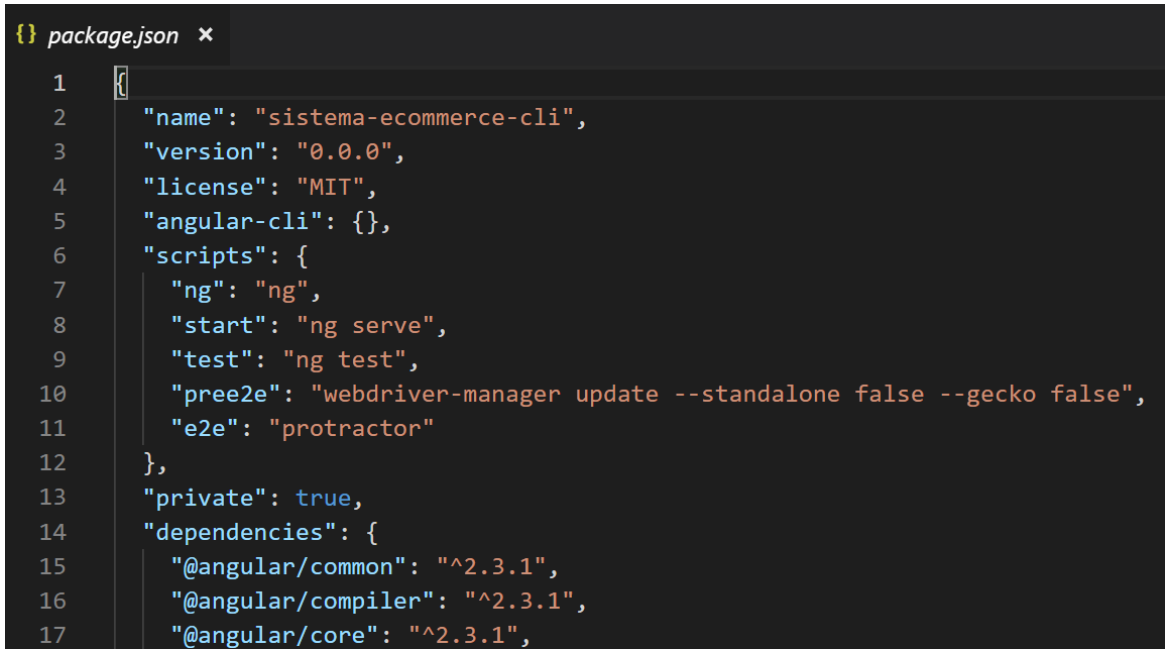
- O **NPM (Node Package Manager)** é o gerenciador de pacotes do NodeJS, assim como Maven é para o Java e RubyGems é para a comunidade Ruby.
- Está integrado à instalação do Nodejs
- O site http://npmjs.org hospeda mais de 213.000 módulos Nodejs criados por terceiros e comunidades.

Estudo de Caso: *NodeJS*

- Alguns comandos:
  - **npm init** – auxiliar na criação de um package.json do projeto;
  - **npm install nome_do_modulo** – instala um módulo no projeto;
  - **npm install –g nome_do_modulo** – instala um módulo global
  - **npm install nome_do_modulo --save** – instala o módulo e o adiciona no arquivo package.json
  - **npm list** – lista todos os módulos do projeto
  - **npm list –g** – lista todos os módulos globais do projeto
  - **npm update nome_do_modulo** – atualiza a versão de um módulo do projeto

Estudo de Caso: *NodeJS*

- Todo projeto **node.js** é chamado de módulo, pois o Javascript trabalha com uma arquitetura modular;
- Todo módulo é acompanhado de um arquivo descritor de módulos, chamado package.json
- O arquivo possui :
  - Informações básicas do módulo
  - Dependências do módulo, que são frameworks e ferramentas utilizadas no projeto.
  - Scripts para automação de Tarefas.



Estudo de Caso: *NodeJS*

- A idéia seria usar o node.js para servir os dados, isto é, ser o back-end da aplicação;
- Para construir a API, um framework bastante utilizado é o **Express**.
- **Express**
  - Código minimalista
  - Adota padrões e boas práticas de serviços REST
  - Usa rotas
  - Facilmente integrável com outros frameworks, especialmente persistência

Estudo de Caso: *NodeJS*

- Criação de um projeto **nodeBackEnd**

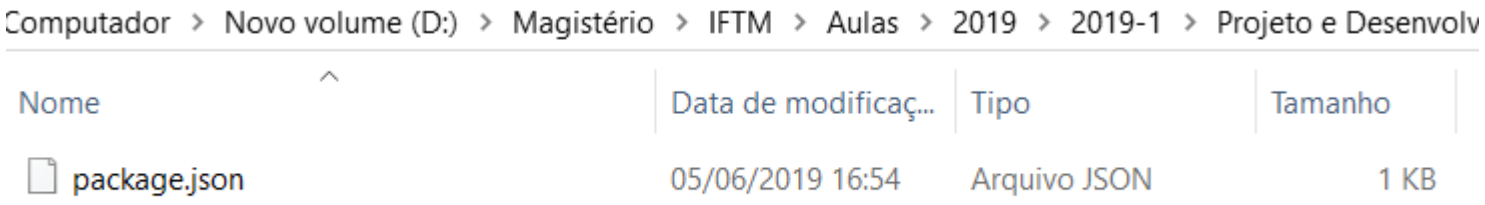
```
PS D:\Magistério\IFTM\Aul
 mkdir nodeBackEnd
```

```
PS D:\Magistério\IFTM\Aulas\20
 cd .\nodeBackEnd\
PS D:\Magistério\IFTM\Aulas\20
nodeBackEnd> npm init
This utility will walk you thr
It only covers the most commor
```

- Responda o questionário após o comando **npm init**. O preenchimento dos campos é opcional. Caso não queira preencher os campos, depois basta editar o arquivo **package.json**

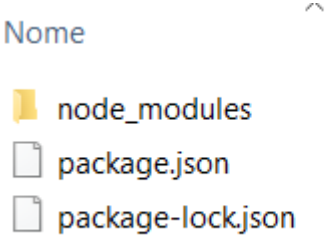
Estudo de Caso: *NodeJS*

- No diretório foi criado apenas um arquivo chamado **package.json**



- Os dois comandos abaixo irão instalar o **babel**, que fará conversão de código ES6 para ES5, caso o **node.js** não reconheça nativamente.

```
nodeBackEnd> npm install babel --save
npm WARN deprecated babel@6.23.0: In 6.x,
. Check https://opencollective.com/babel t
npm notice created a lockfile as package-l
```




```
nodeBackEnd> npm install --savedev babel-cli
[..................] / rollbackFailedOptional: verb npm-session 06ccc1d3b0e95257
```

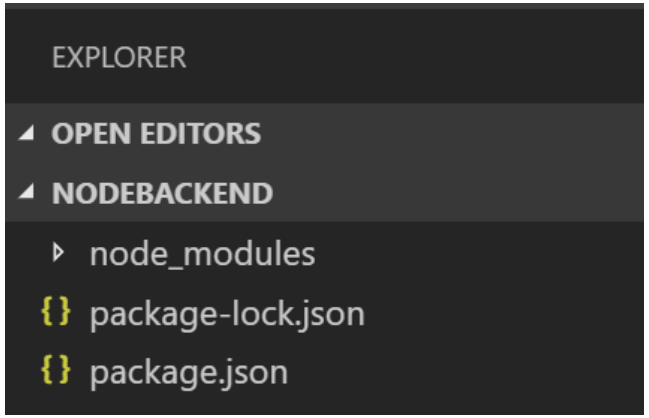
Estudo de Caso: *NodeJS*

- Instalando o **framework express** no projeto
  - Confira se as dependências foram adicionadas no arquivo **package.json**

```
PS D:\Magistério\IFTM\Aulas\2019\2019-1\Projeto e
nodeBackEnd> npm install express --save
npm WARN nodebackend@1.0.0 No repository field.
```

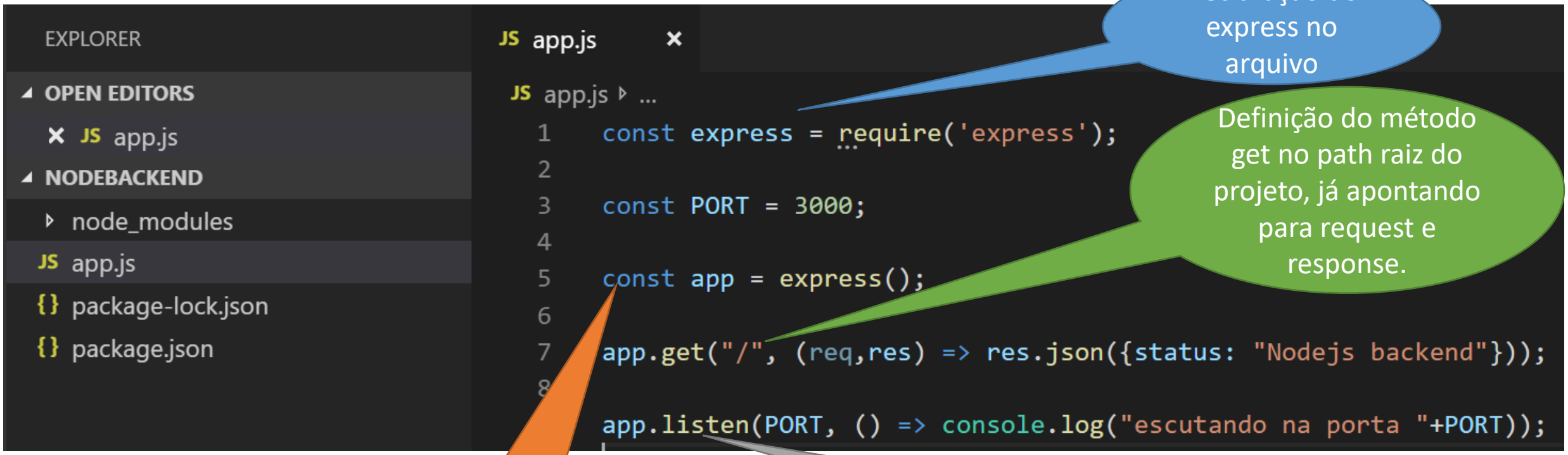
```
{} package.json ×
{} package.json ▸ ...
  8    },
  9    "author": "",
 10    "license": "ISC",
 11    "dependencies": {
 12      "babel": "^6.23.0",
 13      "babel-cli": "^6.26.0",
 14      "express": "^4.17.1"
```

- Abra o projeto no **Visual Studio Code**



Estudo de Caso: *NodeJS*

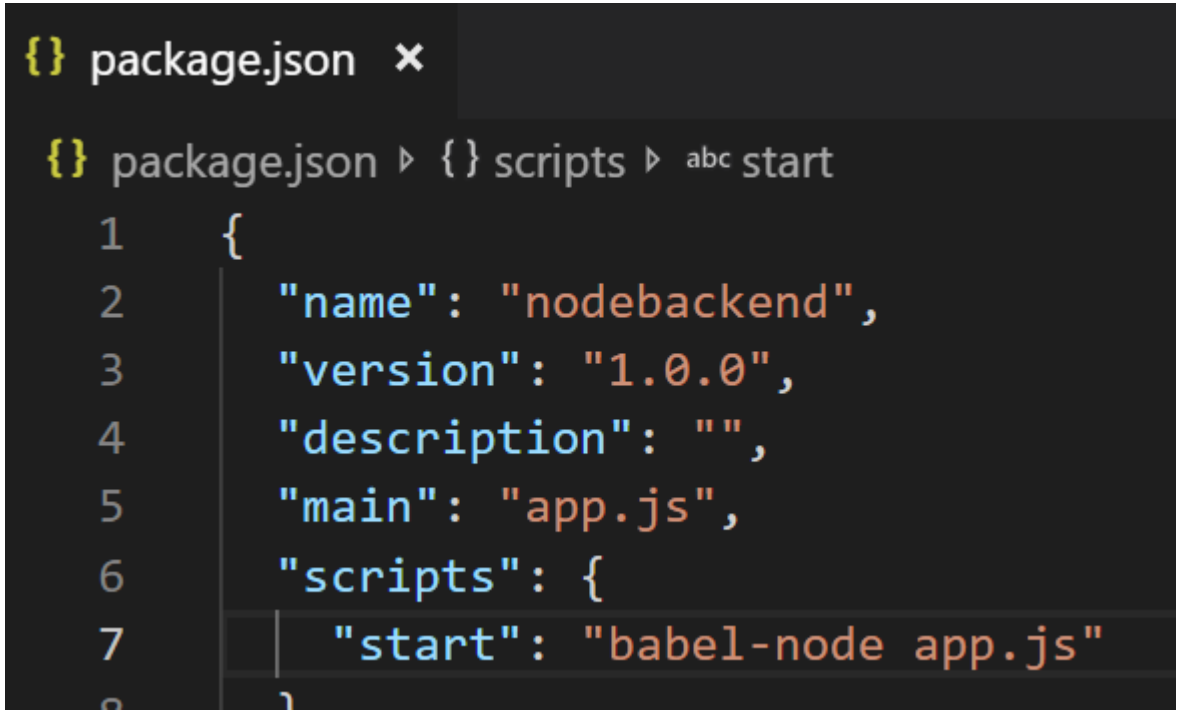
- Crie o arquivo de **app.js** e digite o código abaixo:



Estudo de Caso: *NodeJS*

- Edite o arquivo **package.json** para contemplar o script de start

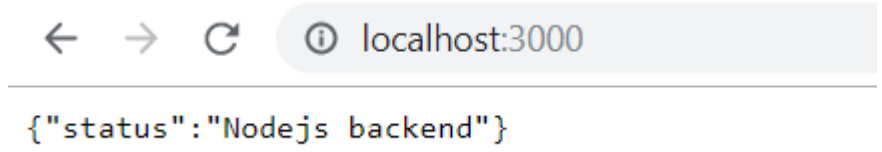
  - Quando o comando ***npm start*** for digitado na linha de comando, irá executar ***babel-node app.js***



Estudo de Caso: *NodeJS*

- Para subir a aplicação, usa-se o comando ***npm start***

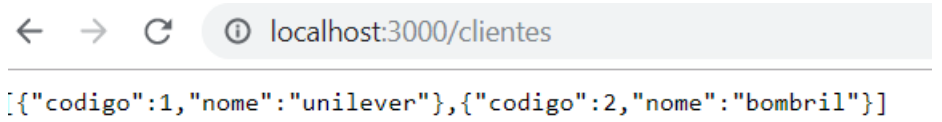
```
PS D:\Magistério\IFTM\Aulas\2019\2019-1\Pr
 npm start

> nodebackend@1.0.0 start D:\Magistério\IF
2\projetos\nodeBackEnd
> babel-node app.js

escutando na porta 3000
```



Com o nodejs, o console.log é exibido no prompt, já que neste caso, o Javascript não está executando no navegador

Estudo de Caso: *NodeJS*

- Adicionando um recurso estático de cliente



O response irá retornar um objeto json de dados estáticos ao solicitante

app.js

```
8
9   app.get("/clientes", (req,res) => {
10      res.json(
11          [
12              {'codigo':1,'nome':'unilever'},
13              {'codigo':2,'nome':'bombril'}
14          ]
15      )
16  });
17
18  app.listen(PORT, () => console.log("escutando na porta "+PORT));
```

Estudo de Caso: *NodeJS*

- Exercício:

- 1) Alterar a aplicação front-end para consumir este serviço de clientes.

Estudo de Caso: *NodeJS*

- Formatar o resultado

```
6
7   app.set("json spaces",4);
8
```

localhost:3000/clientes


```
[
    {
        "codigo": 1,
        "nome": "unilever"
    },
    {
        "codigo": 2,
        "nome": "bombril"
    }
]
```

Estudo de Caso: *NodeJS*

- Para que o conteúdo seja atualizado, se faz necessário derrubar o servidor (CTRL + C) e iniciá-lo novamente.
- Para efetuar um **hot deploy**, isto é, o servidor atualizar a cada alteração em um arquivo.js, pode-se instalar o **supervisor.** Também se faz necessário alterar o arquivo **package.json**

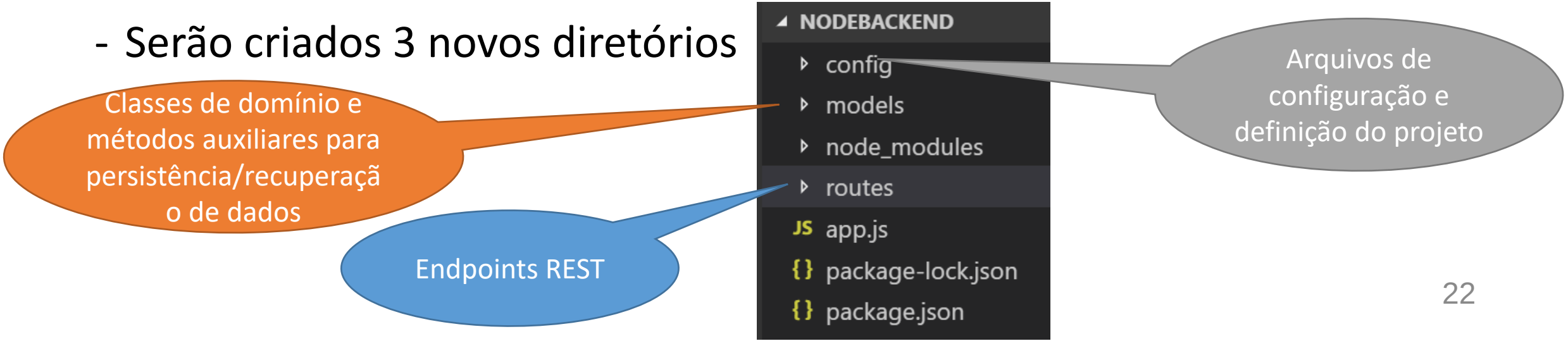
```
npm ERR! peer dep missing: rxjs@^5.0.1, required b
PS D:\Magistério\IFTM\Aulas\2019\2019-1\Projeto e
nodeBackEnd> npm install --save supervisor
```

package.json

```
{
  "name": "nodebackend",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "start": "supervisor babel-node app.js"
```

Estudo de Caso: *NodeJS*

- No exemplo utilizado, todos os endpoints REST foram colocados no arquivo **app.js,** o que evidentemente se torna inviável na medida em que forem adicionados novos endpoints.
- Para isso, será aplicado o padrão **MVR (Model View Router).** O **express** implementa **VR (View Router)**, deixando a camada **Model** livre para ser implementada por qualquer framework.
- Serão criados 3 novos diretórios



Estudo de Caso: *NodeJS*

- Dentro do diretório routes, crie os arquivos **index.js** e **clientes.js**, migrando os endpoints REST de **app.js** para estes arquivos.

JS *clientes.js* ×

routes ▸ JS clientes.js ▸ ...

```
1   const express = require('express');
2   const router = express.Router();
3
4   router.get("/", (req,res) => {
5       res.json(
6           [
7               {'codigo':1,'nome':'unilever'},
8               {'codigo':2,'nome':'bombril'}
9           ]
10      )
11  });
12
13  module.exports = router;
```

O express já trabalha nativamente com rotas

JS index.js ×

routes ▸ JS index.js ▸ router.get("/") callback

```
1   const express = require('express');
2   const router = express.Router();
3
4   router.get("/", (req,res) =>
5       res.json({status: "Nodejs backend"}));
6
7   module.exports = router;
```

Está exportando a rota que foi criada para o módulo principal do sistema

Estudo de Caso: *NodeJS*

- Dentro do diretório routes, crie os arquivos **index.js** e **clientes.js**, migrando os endpoints REST de **app.js** para estes arquivos.

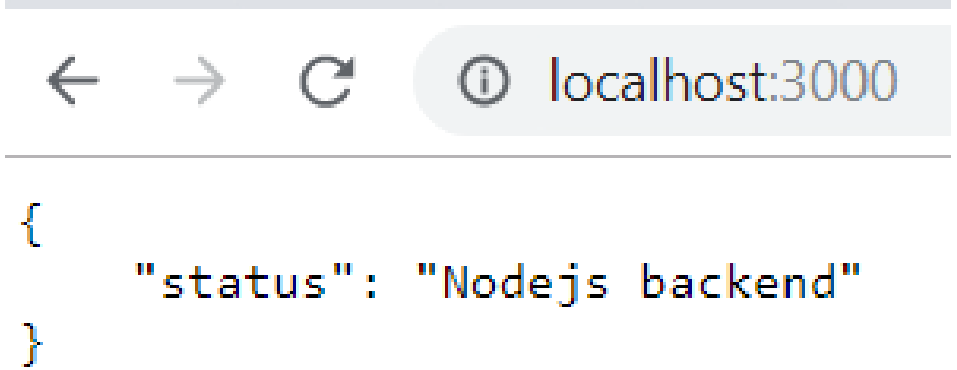
Está declarando que vai utilizar as rotas já declaradas nestes arquivos, e está definindo o path de cada

app.js

```
1   const express = require('express');
2   const PORT = 3000;
3   const app = express();
4
5   app.set("json spaces",4);
6
7   const index = require('./routes/index');
8   const clientes = require('./routes/clientes');
9
10  app.use('/', index);
11  app.use('/clientes', clientes);
12
13  app.listen(PORT, () => console.log("escutando na porta "+PORT));
```

Estudo de Caso: *NodeJS*

- Os endpoints continuam funcionando da mesma forma



Estudo de Caso: *NodeJS*

- Banco de dados relacional na camada model

  - Para efetuar o mapeamento objeto-relacional, será usado o Sequelize
  - O banco de dados será o Mysql, que já possui instalação no laboratório.
  - http://docs.sequelizejs.com/

Estudo de Caso: *NodeJS*

- Banco de dados relacional na camada model

  - Instalando o Sequelize e o mysql

```
PS D:\Magistério\IFTM\Aulas\2019\2019-1\Projeto
nodeBackEnd> npm install --save sequelize
[..................] / rollbackFailedOptional: v
```

```
PS D:\Magistério\IFTM\Aulas\2019\2019-1\Projeto e Desen
nodeBackEnd> npm install --save mysql2
```

Estudo de Caso: *NodeJS*

- Banco de dados relacional na camada model
  - Criando o arquivo de configuração do Sequelize com o mysql dentro da pasta **config**

```
JS database.js ×

config ▸ JS database.js ▸ sequelize
1   const Sequelize = require('sequelize');
2   let sequelize = new Sequelize('pds2', 'root', '', {
3     host: 'localhost',
4     dialect: 'mysql',
5     operatorsAliases: false,
6     pool: {
7       max: 20,
8       min: 5,
9       acquire: 30000,
10      idle: 10000
11    },
12  });
13  sequelize.sync();
14  module.exports = sequelize;
```

Nome do banco, usuário, senha e parâmetros

Cria as tabelas do banco automaticamente

Pool de conexões, mínimo de 5 conexões abertas, máximo 20

Estudo de Caso: *NodeJS*

- Banco de dados relacional na camada model
  - Criando o arquivo de domínio clientes.js na pasta **model**

JS clientes.js ×

models ▸ JS clientes.js ▸ Cliente

```
1   const Sequelize = require('sequelize');
2   const db = require('../config/database');
3
4   const Cliente = db.define('cliente', {
5     codigo: {
6       type: Sequelize.INTEGER,
7       primaryKey: true,
8       autoIncrement: true
9     },
10    nome: {
11      type: Sequelize.STRING,
12      allowNull: false
13    },
14  })
15
16  module.exports = Cliente;
```

Usa a referência do banco de dados para criação do objeto de domínio

Estudo de Caso: *NodeJS*

- Banco de dados relacional na camada model
  - O código adicionado em **app.js** é apenas para verificar se o banco de dados está conectado no momento em que a aplicação subir.

app.js

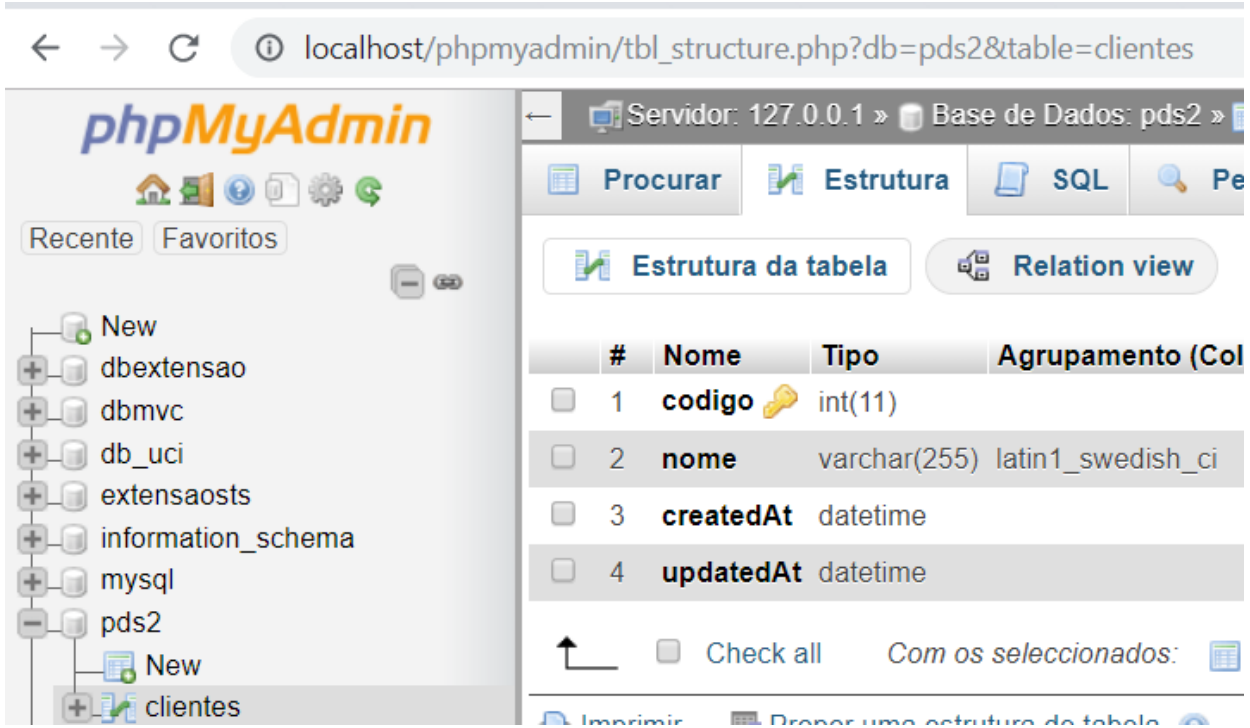
```
1  const express = require('express');
2  const PORT = 3000;
3  const app = express();
4  const db = require('./config/database');
5
6  db.authenticate()
7    .then(() => console.log('Database connected...'))
8    .catch(err => console.log('Error: ' + err))
9
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
etos\nodeBackEnd' for changes.
Press rs for restarting the process.
(node:9468) [SEQUELIZE0004] DeprecationWarning: A bo
s. This is a no-op with v5 and should be removed.
escutando na porta 3000
Executing (default): SELECT 1+1 AS result
Database connected...
```

Estudo de Caso: *NodeJS*

- Banco de dados relacional na camada model
  - No phpmyadmin observa que a tabela de clientes foi criada pelo Sequelize



Estudo de Caso: *NodeJS*

- Banco de dados relacional na camada model
  - Alterar o arquivo **clientes.js** dentro da pasta **routes** para chamar o banco

O próprio Sequelize já possui o método findAll, ou seja, somente será necessário construir consultas complexas. Todo o básico do CRUD já está pronto

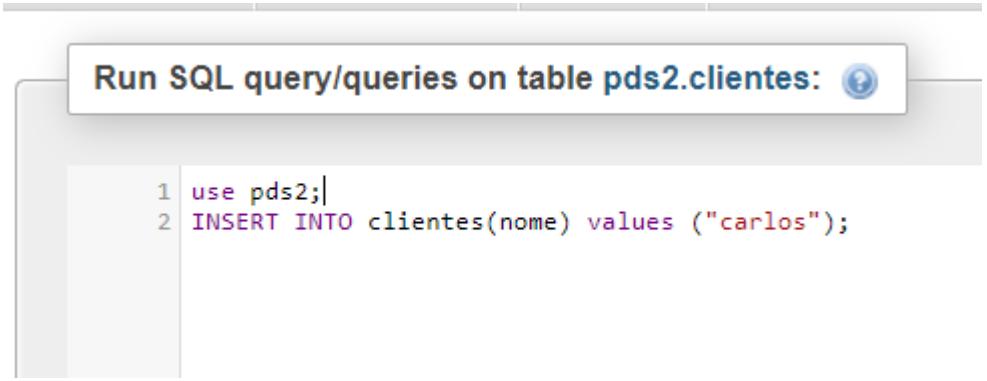
clientes.js

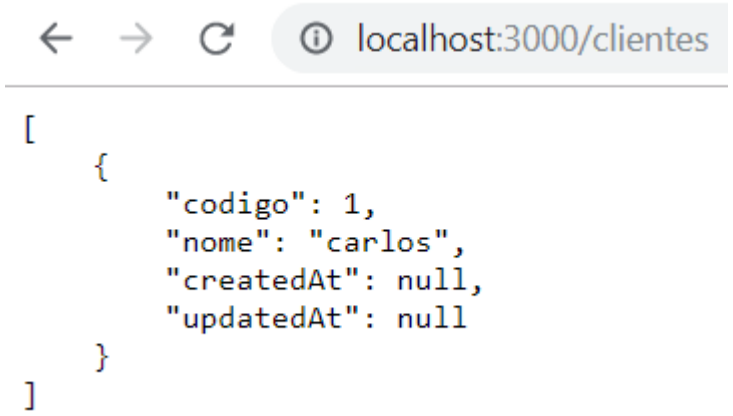
routes ▸ clientes.js ▸ ...

```
1   const express = require('express');
2   const Cliente = require('../models/clientes');
3   const router = express.Router();
4
5   router.get("/", (req,res) =>
6       Cliente.findAll()
7           .then(result => res.json(result))
8           .catch(error => {
9               res.status(412).json({msg: error.message});
10          }));
11
12  module.exports = router;
```

Estudo de Caso: *NodeJS*

- Banco de dados relacional na camada model
  - Ao inserir dados no phpmyadmin e rodar a aplicação





Estudo de Caso: *NodeJS*

- Banco de dados relacional na camada model
  - Criar a consulta pelo código

clientes.js

routes ▸ clientes.js ▸ router.get("/:id") callback

```
12  router.get("/:id", (req, res) => {
13      Cliente.findOne({
14          where: {
15              codigo: req.params.id,
16          }
17      }).then(result => {
18          if (result) {
19              res.json(result);
20          } else {
21              res.sendStatus(404);
22          }
23      }).catch(error => {
24          res.status(412).json({msg: error.message});
25      });
26  })
```

localhost:3000/clientes/1


```
{
    "codigo": 1,
    "nome": "carlos",
    "createdAt": null,
    "updatedAt": null
}
```

Estudo de Caso: *NodeJS*

- Banco de dados relacional na camada model
  - Criar a consulta pelo nome

```
28  const Sequelize = require('sequelize');
29  const Op = Sequelize.Op;
30
31  router.get('/search/params', (req, res) => {
32      var query = `%${req.query.nome}%`;
33
34      console.log(query)
35      Cliente.findAll({ where: { nome: { [Op.like]: query } } })
36        .then(clientes => res.json(clientes))
37        .catch(err => console.log(err));
38    });
```

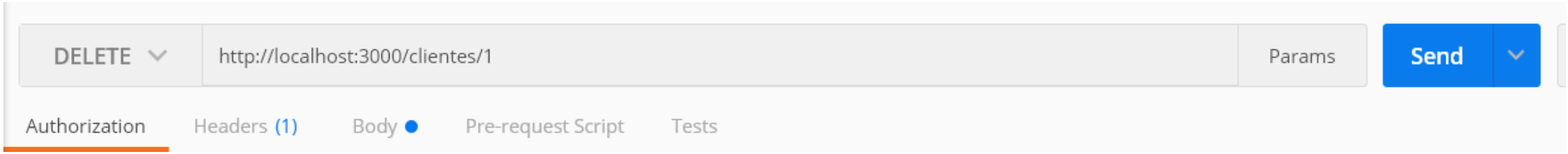
localhost:3000/clientes/search/params?nome=car

```
[
    {
        "codigo": 1,
        "nome": "carlos",
        "createdAt": null,
        "updatedAt": null
    }
]
```

Estudo de Caso: *NodeJS*

- Banco de dados relacional na camada model
  - Deletar o recurso e testando com o Postman.

```
58  router.delete("/:id", (req,res) => {
59      Cliente.destroy({
60          where: {
61              codigo: req.params.id
62          }
63      })
64      .then(result => res.sendStatus(204))
65      .catch(error => {
66                  res.status(412).json({msg: error.message});
67              });
68          });
```



Estudo de Caso: *NodeJS*

- Banco de dados relacional na camada model
  - Inserir um novo cliente

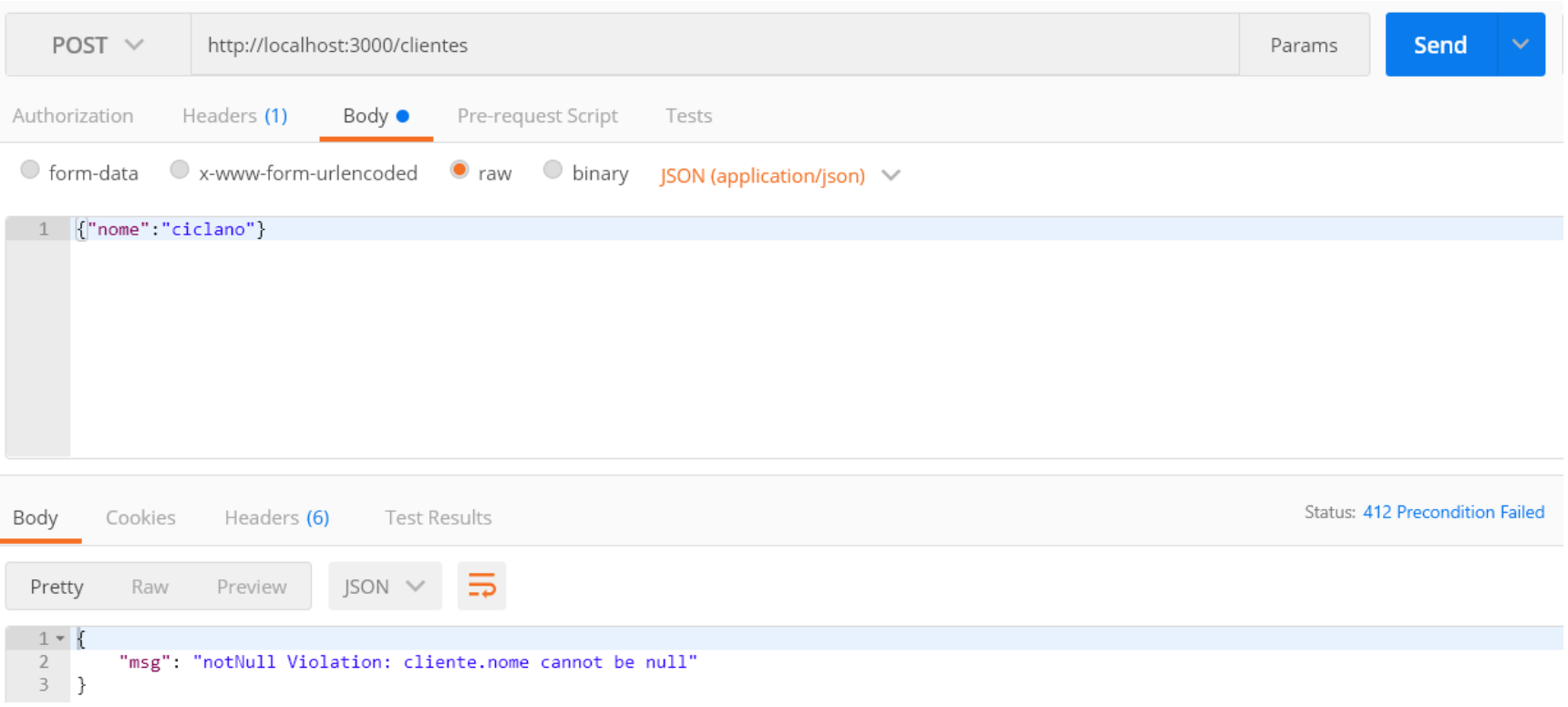
clientes.js

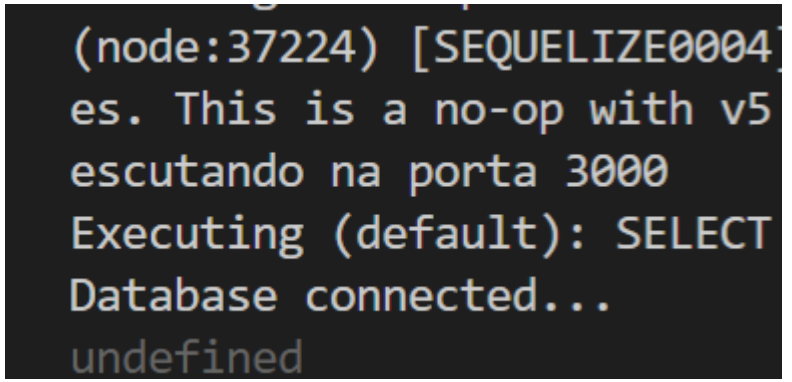
routes ▸ clientes.js ▸ router.post('/') callback

```
41  router.post('/', (req,res) => {
42      console.log(req.body);
43      Cliente.create(req.body)
44          .then(result => res.json(result))
45          .catch(error => {
46              res.status(412).json({msg: error.message});
47          });
48      });
```

Estudo de Caso: *NodeJS*

- Banco de dados relacional na camada model
  - Porém, ao executar, ocorre o retorno HTTP 412, mostrando a variável **req.body** com valor **undefined**





Estudo de Caso: *NodeJS*

- Banco de dados relacional na camada model
  - Este problema ocorre porque o nodejs não sabe converter os dados da requisição para o formato que queremos. Para isso, será instalado o módulo body-parser, que converte o body da requisição para o formato desejado.

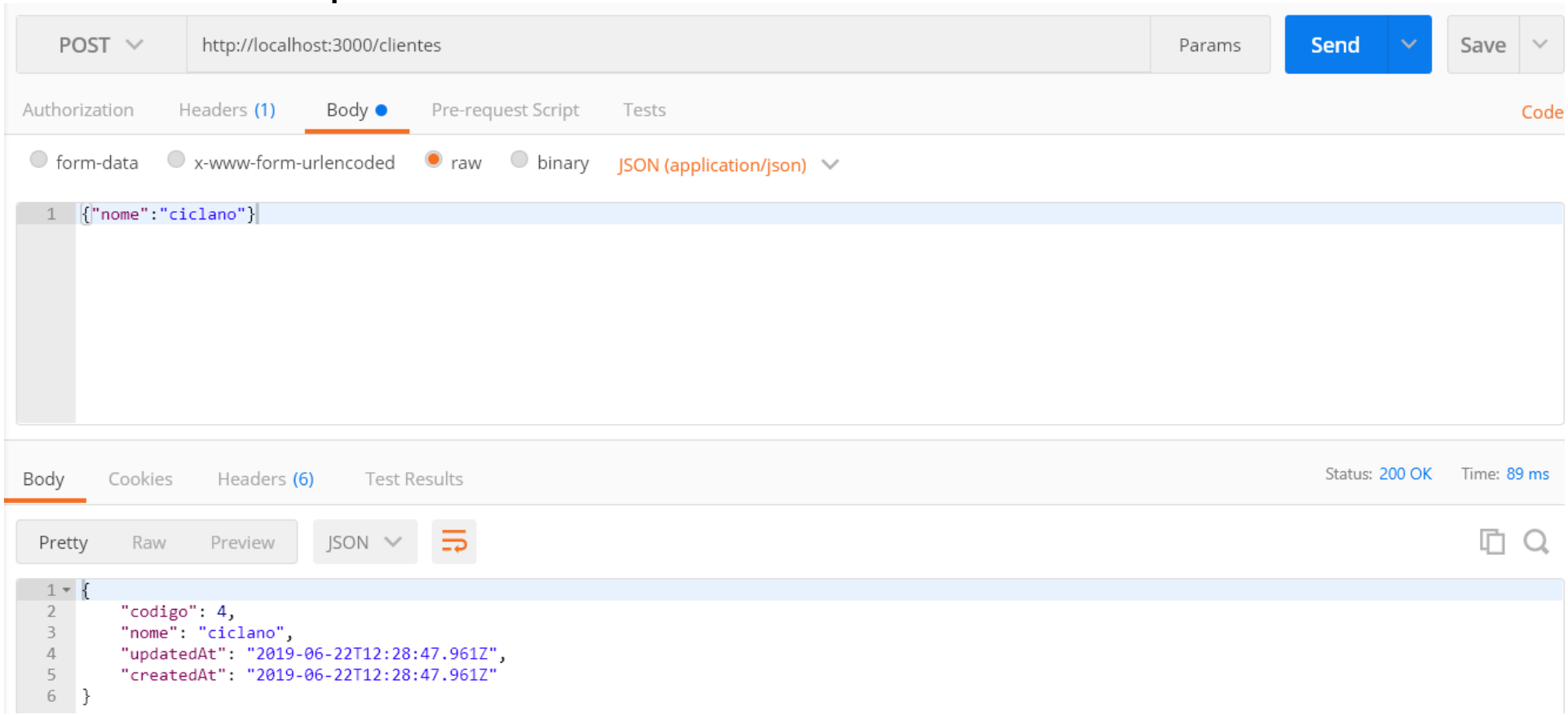
```
npm i --save body-parser
```

app.js

```
1  const express = require('express');
2  const PORT = 3000;
3  const app = express();
4  const db = require('./config/database');
5  const bodyParser = require('body-parser');
6
7  app.use(bodyParser.json());
8
```

Estudo de Caso: *NodeJS*

- Banco de dados relacional na camada model
  - Testes do postman



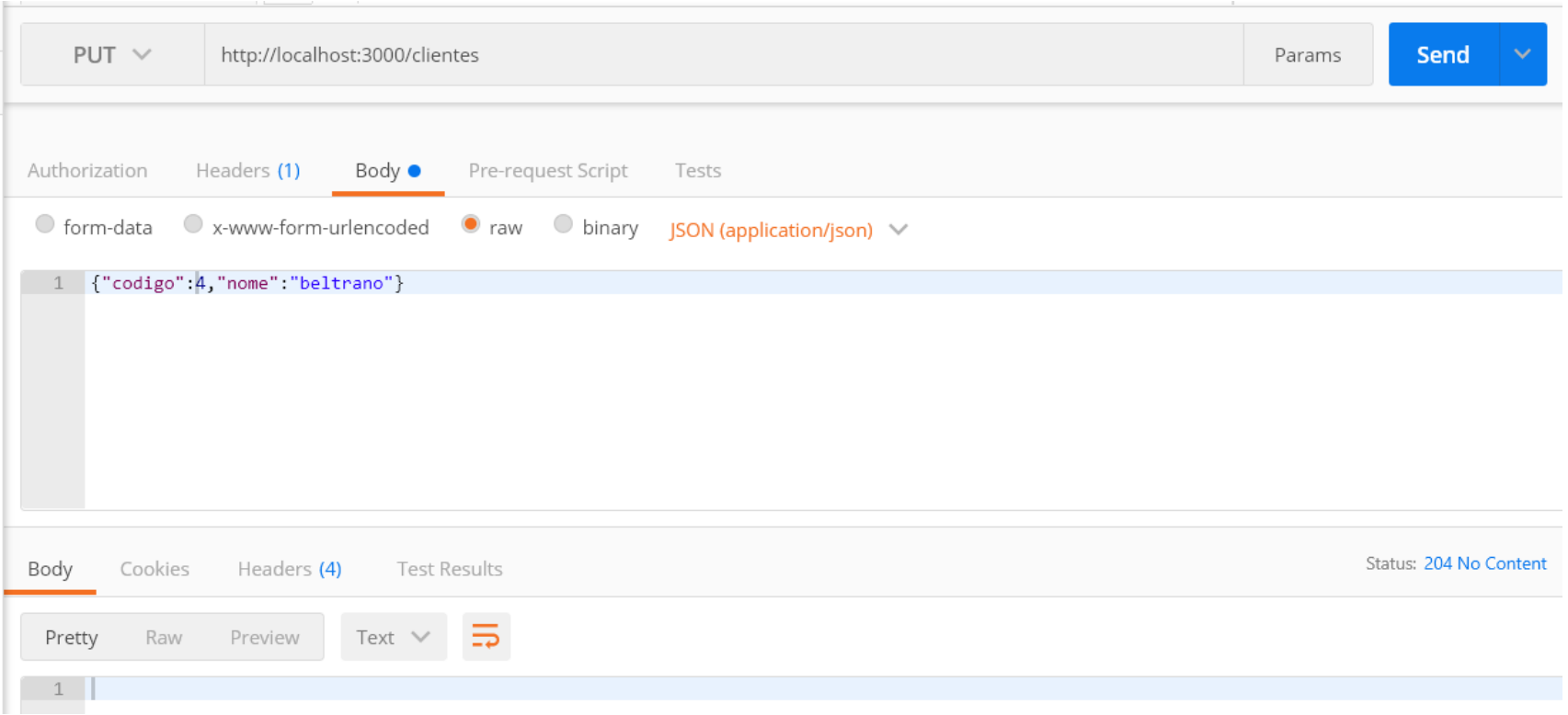
Estudo de Caso: *NodeJS*

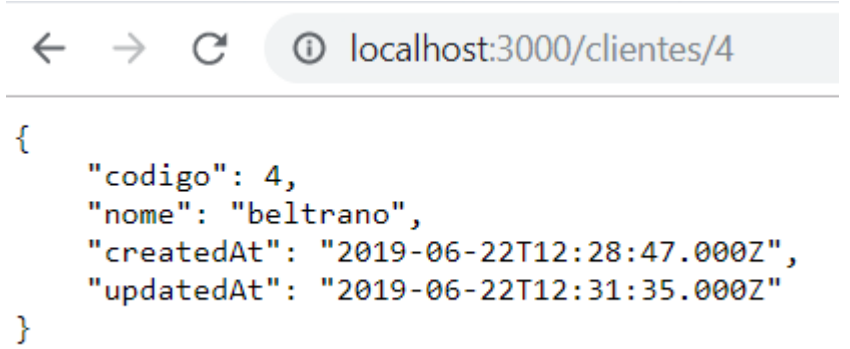
- Banco de dados relacional na camada model
  - Atualizando o cliente

```
50  router.put('/', (req,res) => {
51      Cliente.update(req.body, {
52          where: {
53          codigo: req.body.codigo
54          }
55        })
56          .then(result => res.sendStatus(204))
57          .catch(error => {
58              res.status(412).json({msg: error.message});
59          });
60      });
```

Estudo de Caso: *NodeJS*

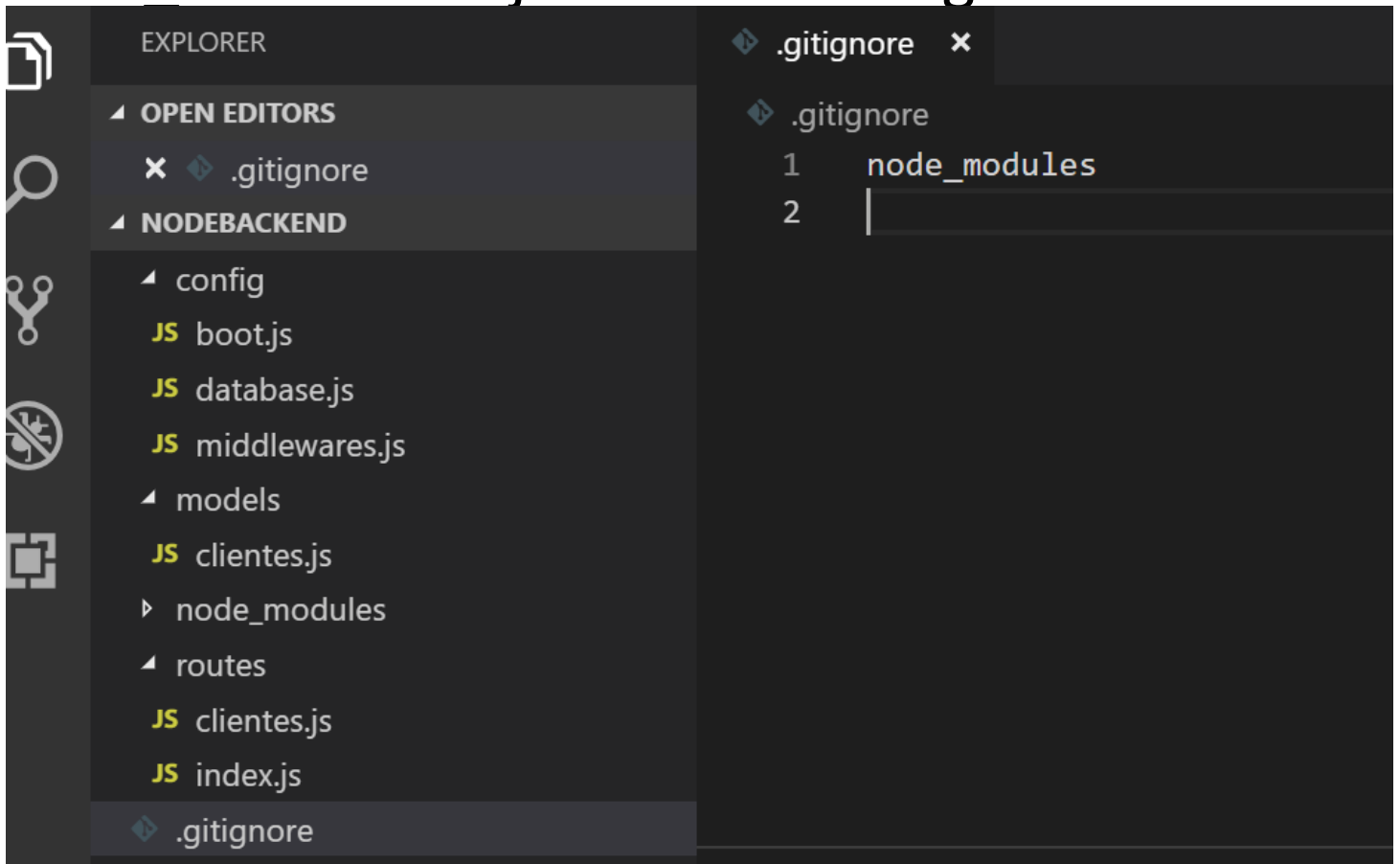
- Banco de dados relacional na camada model
  - Testando com o postman





Estudo de Caso: *NodeJS*

- Adicionando o arquivo **.gitignore**, para evitar que a pasta node_modules seja colocada no github



Estudo de Caso: *NodeJS*

- Para que o cliente se conecte ao back-end nodejs, é necessário implementar a política de CORS
- **CORS (CROSS-ORIGIN RESOURCE SHARING)** é responsável por permitir ou barrar requisições assíncronas que são realizadas por outros domínios.

• O CORS na prática são apenas headers do HTTP que são incluídos no server-side da aplicação. Tais headers podem informar qual domínio poderá consumir a API, quais métodos do HTTP serão permitidos e, principalmente, quais endpoints serão compartilhados de forma pública para outros domínios de outras aplicações consumirem.

Estudo de Caso: *NodeJS*

- Deve-se instalar o cors e habilitá-lo no arquivo **app.js**

```
npm install cors
npm WARN nodebackend@1.0.0 No description
npm WARN nodebackend@1.0.0 No repository field.
```

app.js

```
5   const bodyParser = require('body-parser');
6   const cors = require('cors');
7
8   app.use(cors({
9     origin: ['http://localhost:4200'],
10    methods: ["GET","POST","PUT","DELETE"],
11    allowedHeaders: ["Content-Type","Authorization"]
12  }));
13
14  app.use(bodyParser.json());
```

Origin são os domínios permitidos. Neste caso, a aplicação só aceitará requisições deste domínio

O domínio só poderá usar os métodos declarados neste parâmetro.

Caso qualquer aplicação cliente pudesse consumir os endpoints REST, bastaria usar app.use(cors());

LINKS DOS SOFTWARES UTILIZADOS

- **Visual Studio Code** - https://code.visualstudio.com/
- **Node JS -** https://nodejs.org/en/

- Link repositório GITHUB: https://github.com/carloseduardoxp/nodebackend

REFERÊNCIAS

- GUEDES, Thiago. Crie aplicações com Angular. Casa do Código, 2018.
- GRONER, Loiane. Curso Angular. Disponível em: https://www.youtube.com/watch?v=tPOMG0D57S0&list=PLGxZ4Rq3BOBoSRcKWEdQACbUCNWLczg2G. Acesso em 02/04/2019.