

Projeto e Desenvolvimento de Software II

Prof. Carlos Eduardo de Carvalho Dantas

carloseduardodantas@iftm.edu.br

Parte II – Back-end

AGENDA

2. Desenvolvimento Back-end

- Nodejs
- **Express**

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



- 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

- NodeJS

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



Estudo de Caso: *NodeJS*



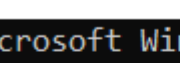
- Por que aprender nodejs?

- **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



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

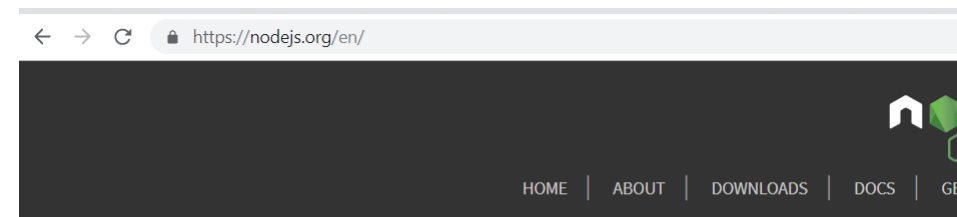
```
maEcommerceCLI> npm -v
6.4.1
PS D:\Magistério\IFTM\Automação\Projeto\maEcommerceCLI>
```



A terminal window titled 'Prompt de Comando' (Command Prompt) showing the command 'npm -v' being executed. The output is '6.4.1'. The window also displays the text 'Microsoft Windows [versão 6.0.6002.5512] (c) 2018 Microsoft Corporation'.

- Hello world com nodejs

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



Node.js® is a JavaScript runtime built

Download for \

10.16.0 LTS

Recommended For Most Users

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.



- 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



- 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.

```
{ package.json x
1 {
2   "name": "sistema-ecommerce-cli",
3   "version": "0.0.0",
4   "license": "MIT",
5   "angular-cli": {},
6   "scripts": {
7     "ng": "ng",
8     "start": "ng serve",
9     "test": "ng test",
10    "pree2e": "webdriver-manager update --standalone false --gecko false",
11    "e2e": "protractor"
12  },
13  "private": true,
14  "dependencies": {
15    "@angular/common": "^2.3.1",
16    "@angular/compiler": "^2.3.1",
17    "@angular/core": "^2.3.1",
```

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\AuI
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 common
```

- 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**



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




Computador > Novo volume (D:) > Magistério > IFTM > Aulas > 2019 > 2019-1 > Projeto e Desenvolv

Nome	Data de modificaç...	Tipo	Tamanho
 package.json	05/06/2019 16:54	Arquivo JSON	1 KB

- 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 to
npm notice created a lockfile as package-l
```

Nome

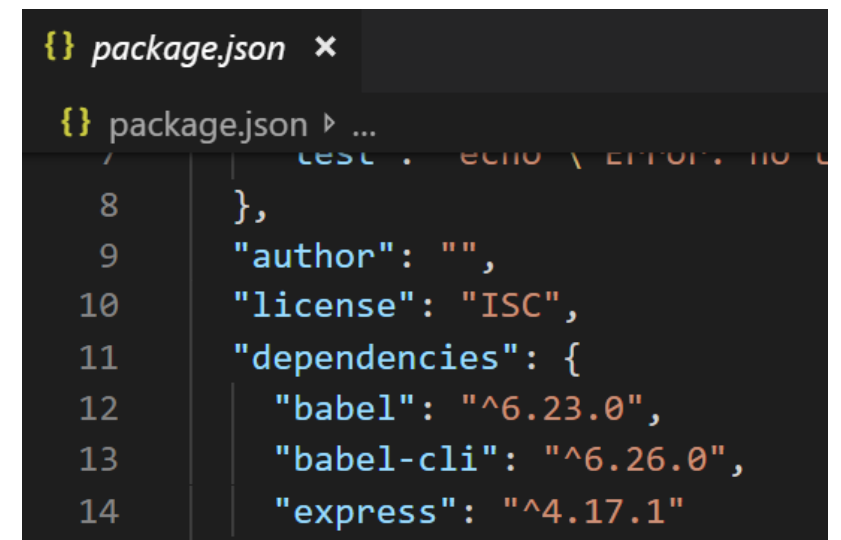
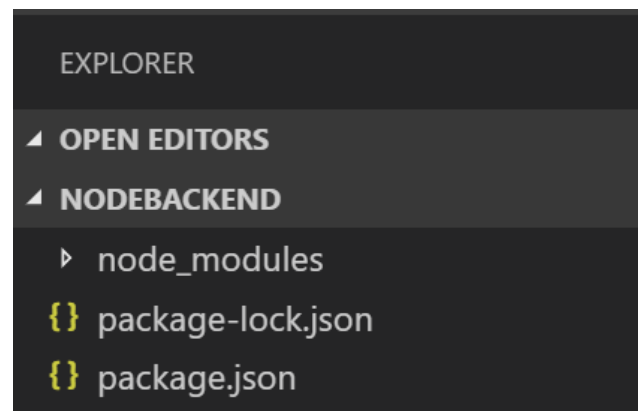
-  node_modules
-  package.json
-  package-lock.json

```
nodeBackend> npm install --save-dev babel-cli
[.....] / rollbackFailedOptional: verb npm-session 06ccc1d3b0e95257
```

- 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.
```

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



```
{
  "name": "nodebackend",
  "version": "1.0.0",
  "author": "",
  "license": "ISC",
  "dependencies": {
    "babel": "^6.23.0",
    "babel-cli": "^6.26.0",
    "express": "^4.17.1"
  }
}
```



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

The screenshot shows the VS Code interface. On the left, the Explorer sidebar shows the file structure: 'node_modules', 'app.js', 'package-lock.json', and 'package.json'. The main editor area shows the content of 'app.js' with the following code:

```
1  const express = require('express');
2
3  const PORT = 3000;
4
5  const app = express();
6
7  app.get("/", (req, res) => res.json({status: "Nodejs backend"}));
8
9  app.listen(PORT, () => console.log("escutando na porta "+PORT));
```

Four callouts provide explanations for specific parts of the code:

- Declaração do express no arquivo** (blue bubble) points to line 1: `const express = require('express');`
- Definição do método get no path raiz do projeto, já apontando para request e response.** (green bubble) points to line 7: `app.get("/", (req, res) => res.json({status: "Nodejs backend"}));`
- Variável app que representa o express** (orange bubble) points to line 5: `const app = express();`
- O método listen vai iniciar o servidor, escutando na porta 3000** (grey bubble) points to line 9: `app.listen(PORT, () => console.log("escutando na porta "+PORT));`



- 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**

```
{ } package.json x
{ } package.json ▶ { } scripts ▶ abc start
1  {
2    "name": "nodebackend",
3    "version": "1.0.0",
4    "description": "",
5    "main": "app.js",
6    "scripts": {
7      "start": "babel-node app.js"
8    }
9  }
```


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
```

← → ↻ ⓘ localhost:3000

```
{"status": "Nodejs backend"}
```

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



- Adicionando um recurso estático de cliente

← → ↻ ⓘ localhost:3000/clientes
[{"codigo":1,"nome":"unilever"}, {"codigo":2,"nome":"bombril"}]

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

```
JS app.js x
JS 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.

- Formatar o resultado

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



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

- 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**

`{}` package.json ✕

`{}` package.json ▶ ...

```
1  {
2    "name": "nodebackend",
3    "version": "1.0.0",
4    "description": "",
5    "main": "app.js",
6    "scripts": {
7      "start": "supervisor babel-node app.js"
```

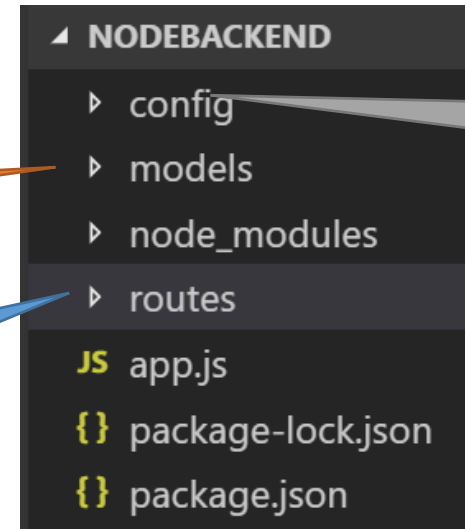
```
npm ERR! peer dep missing: rxjs@^5.0.1, required by
PS D:\Magistério\IFTM\Aulas\2019\2019-1\Projeto e
nodeBackend> npm install --save supervisor
```



- 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

Classes de domínio e métodos auxiliares para persistência/recuperação de dados

Endpoints REST



Arquivos de configuração e definição do projeto

- 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 x

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 x

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

- 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

```
JS app.js x
JS 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));
```




- Os endpoints continuam funcionando da mesma forma

← → ↻ ⓘ localhost:3000/clientes

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

← → ↻ ⓘ localhost:3000

```
{  
  "status": "Nodejs backend"  
}
```

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/>



- Banco de dados relacional na camada model
- Instalando o Sequelize e o mysql

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

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

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

JS database.js x

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;
```

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

Nome do banco,
usuário, senha e
parâmetros

Cria as tabelas do
banco
automaticamente

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

JS clientes.js x

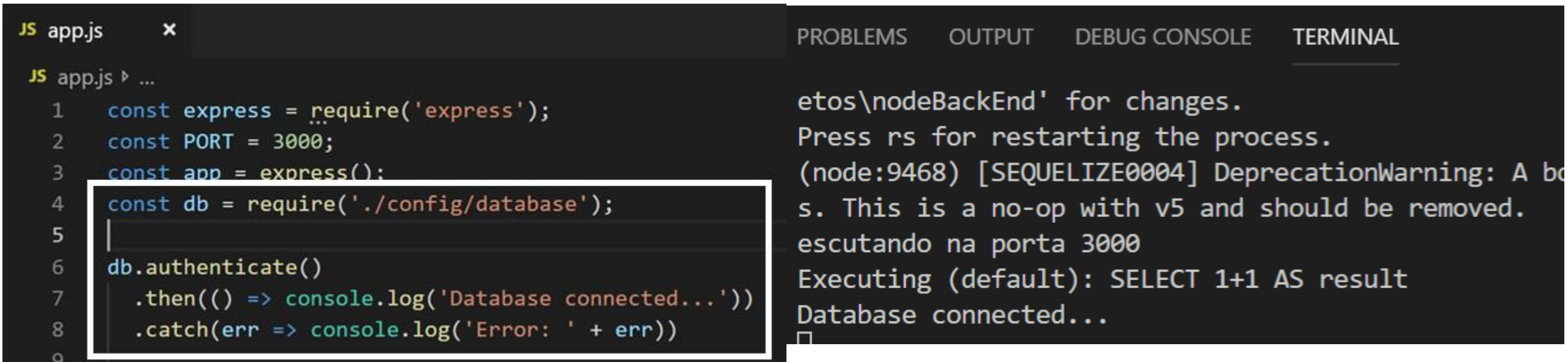
models ▸ JS clientes.js ▸ [🔗] Cliente

```
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



- 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.



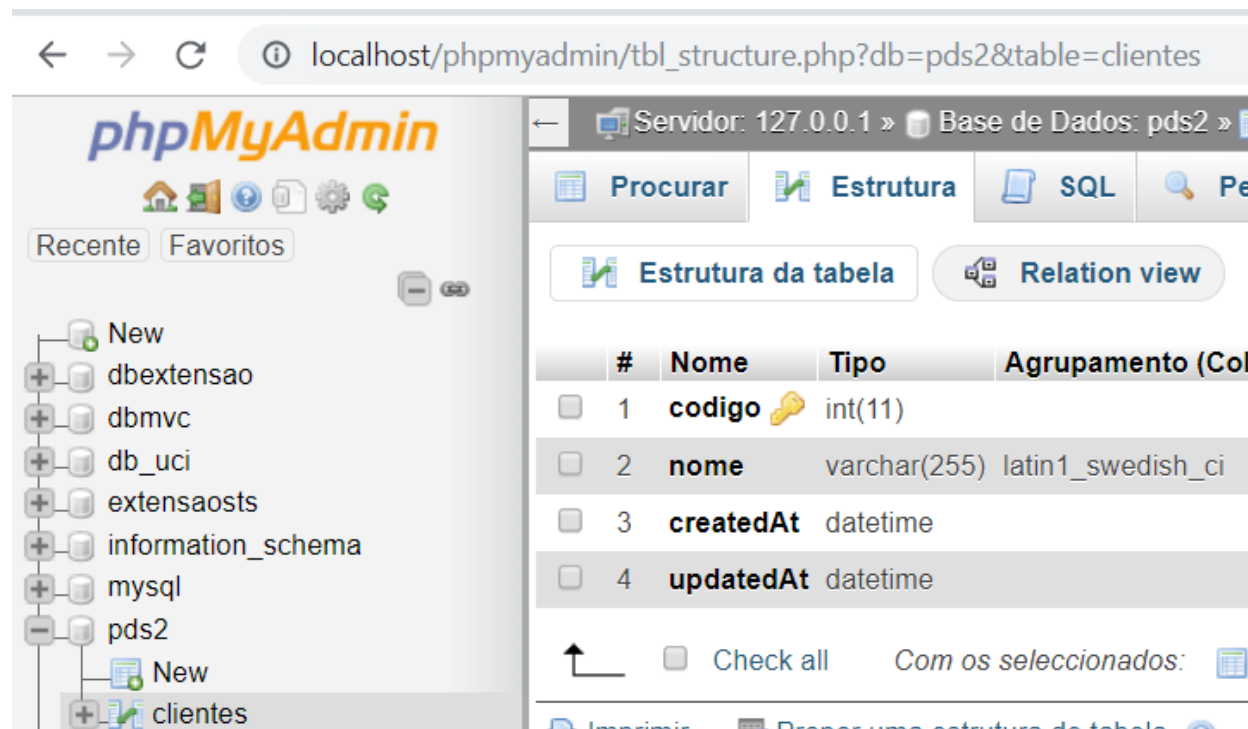
```
JS app.js x
```

```
JS 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...  
□
```

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



- 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

JS clientes.js x

routes ▸ JS 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;
```


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

```
Run SQL query/queries on table pds2.clientes: ⓘ  
  
1 use pds2;  
2 INSERT INTO clientes(nome) values ("carlos");
```

```
← → ↻ ⓘ localhost:3000/clientes  
  
[  
  {  
    "codigo": 1,  
    "nome": "carlos",  
    "createdAt": null,  
    "updatedAt": null  
  }  
]
```



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

JS clientes.js x

routes ▸ JS 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 })
```



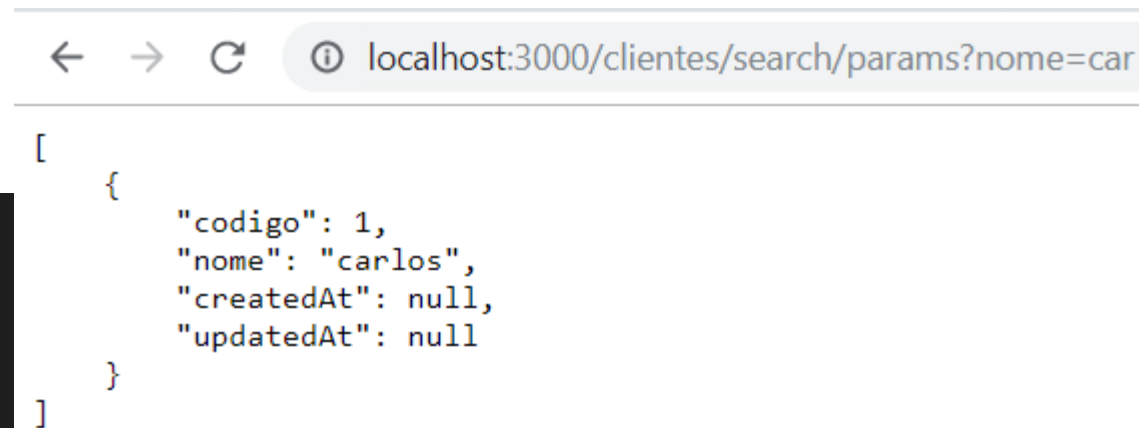
localhost:3000/clientes/1

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



- 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
  }
]
```



- 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 });
```

DELETE ▾

http://localhost:3000/clientes/1

Params

Send ▾



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

```
JS clientes.js x
routes ▸ JS 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 });
49
```

Estudo de Caso: *NodeJS*



Express



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

POST http://localhost:3000/clientes

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json)

1 {"nome": "ciclano"}

Body Cookies Headers (6) Test Results Status: 412 Precondition Failed

Pretty Raw Preview JSON

1 {
2 "msg": "notNull Violation: cliente.nome cannot be null"
3 }

```
(node:37224) [SEQUELIZE0004]  
es. This is a no-op with v5  
escutando na porta 3000  
Executing (default): SELECT  
Database connected...  
undefined
```

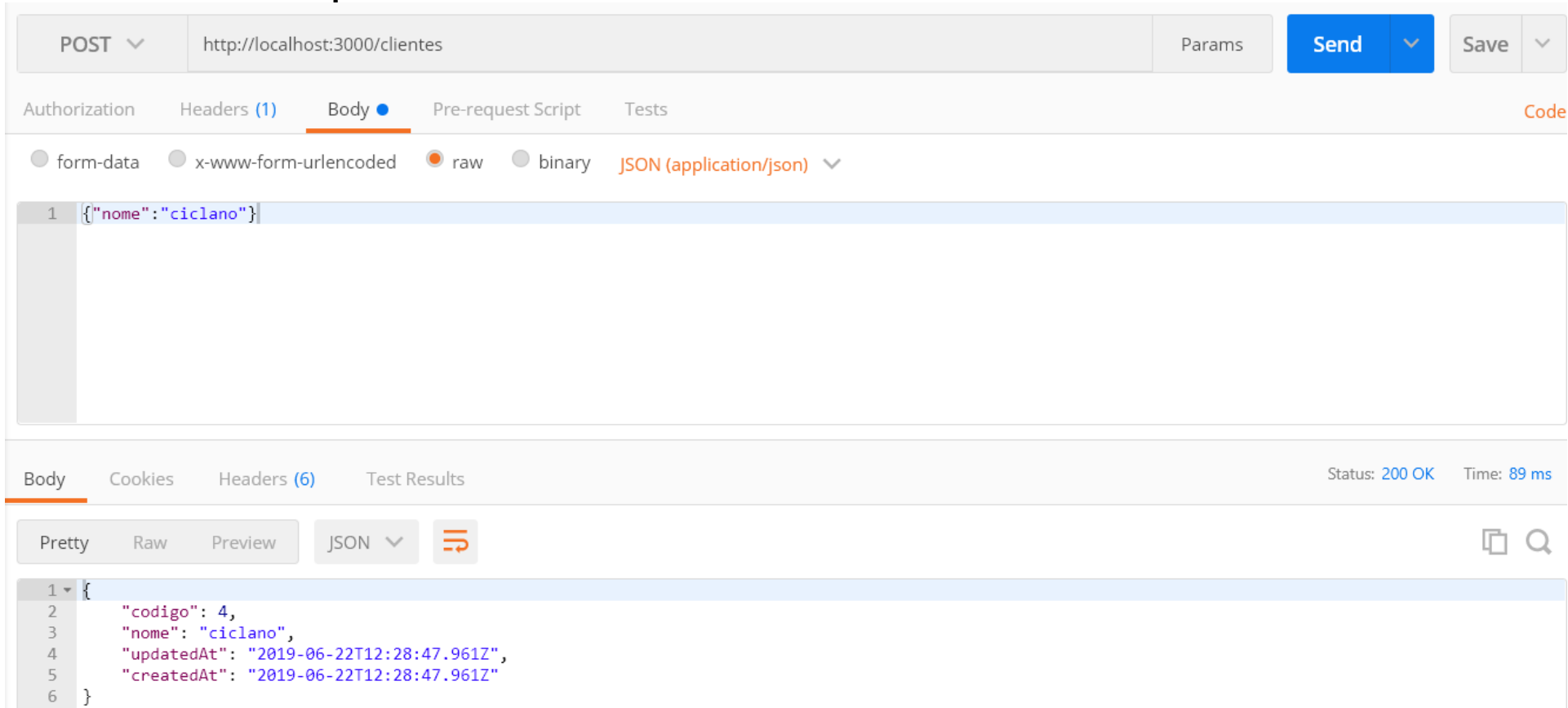
- Banco de dados relacional na camada model
 - Este problema ocorre porque o nodejs não sabe converter os dados da requisição para o formato que queremos. Para isso, será instalado o módulo body-parser, que converte o body da requisição para o formato desejado.

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

```
JS app.js ×  
JS 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
```



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



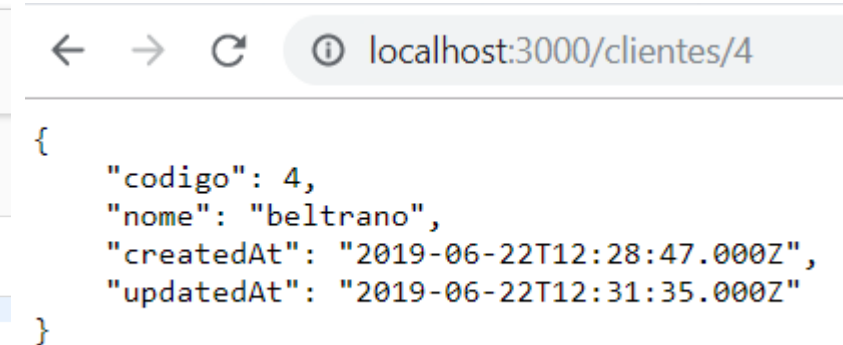
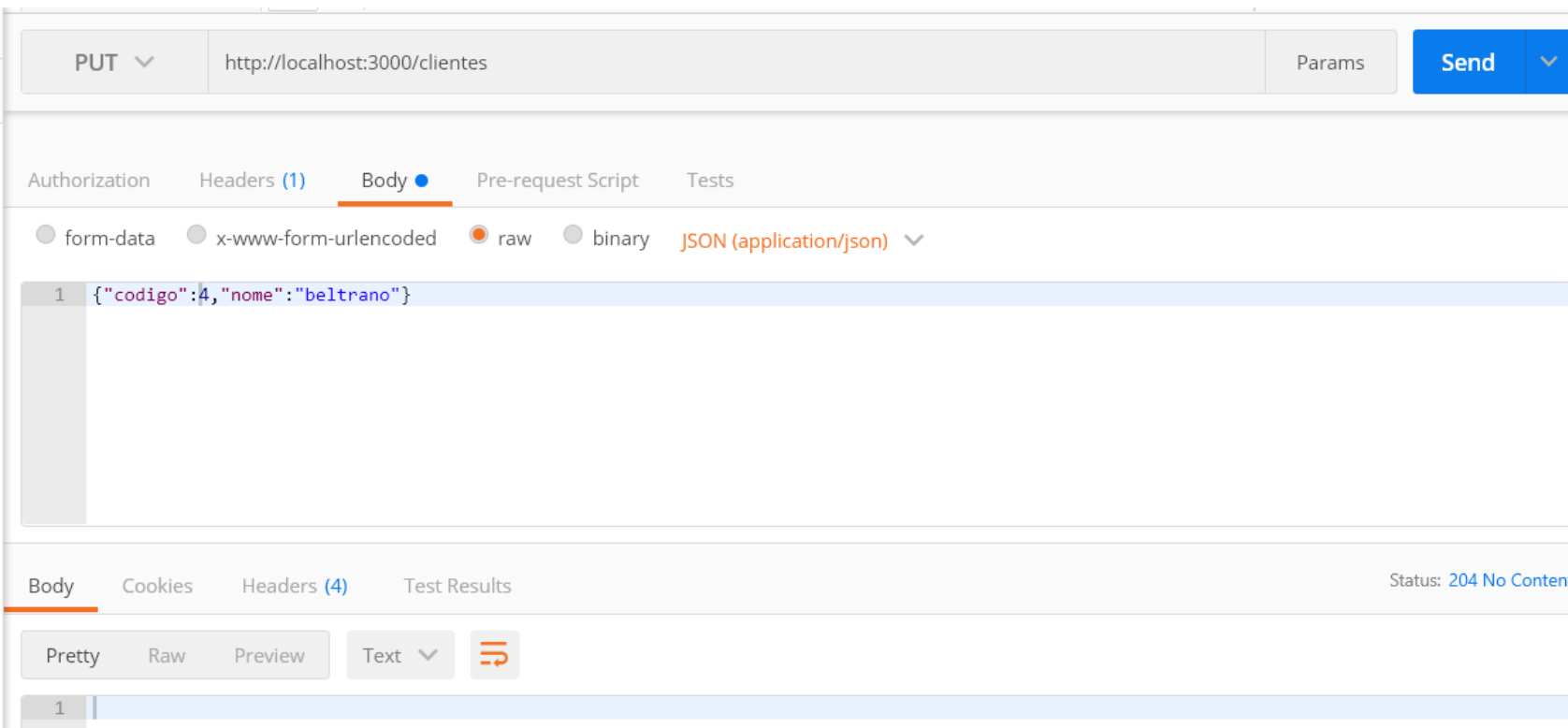


- Banco de dados relacional na camada model
 - Atualizando o cliente

```
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  });  
61
```

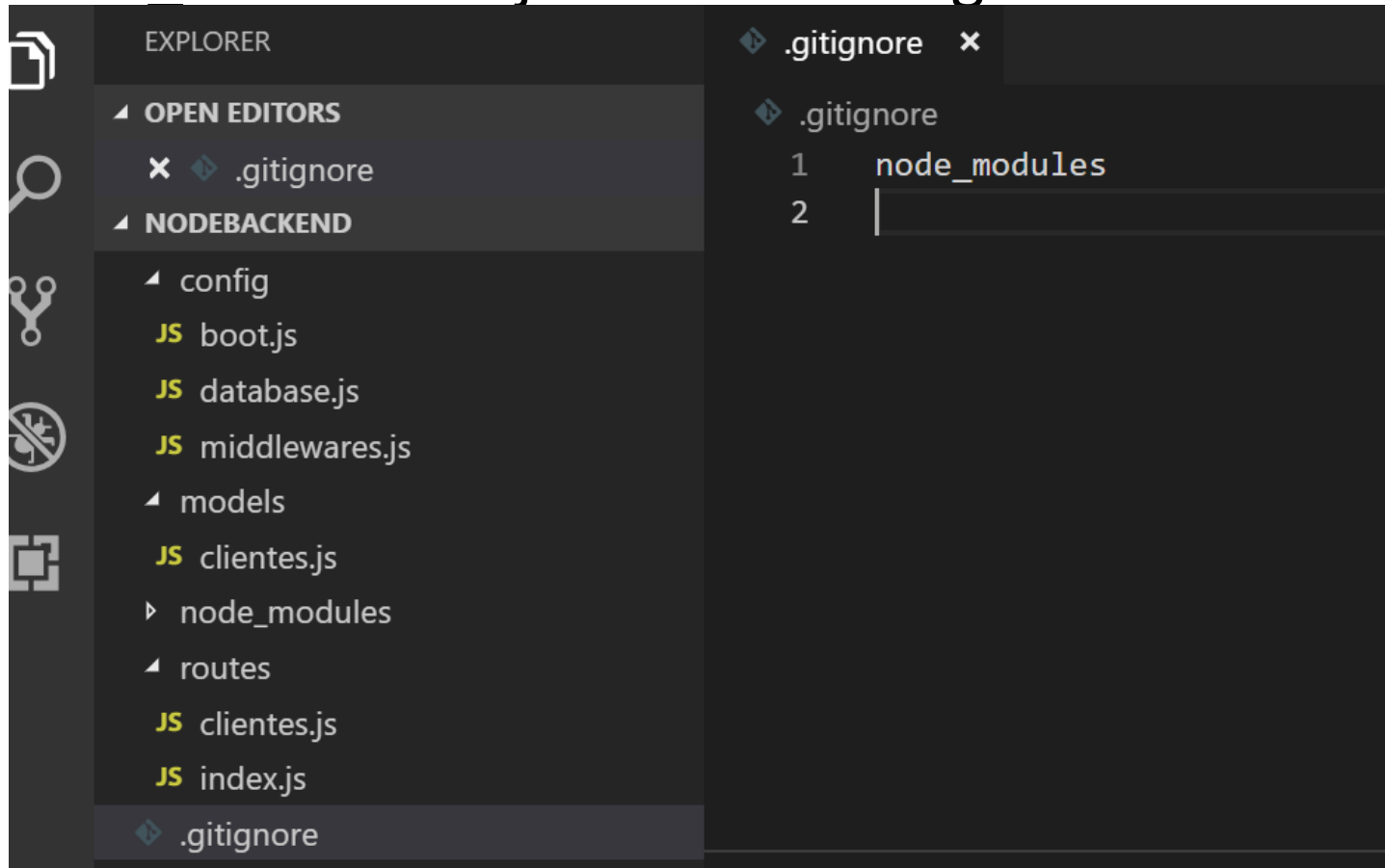


- Banco de dados relacional na camada model
- Testando com o postman





- 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.

- 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.
```

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

```
JS app.js x
JS 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());
15
```

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.