

Criando API's com Node.JS balta.io (André Baltieri)

<https://www.youtube.com/watch?v=wDWdqIYxfcw&list=PLHIHvK2InJndvvycjBqQAbgEDqXxKLogn>

Resumo do curso feito por Roberto Pinheiro

Github - Código fonte:

<https://github.com/balta-io/1972>

Aula 01 - Instalação Node, NPM e VS Code

- Dentro de C:\ crie uma pasta chamada **balta** e dentro dela crie uma subpasta chamada **nodejs**. Dentro dela crie uma subpasta chamada **node-str**. Essa será a pasta do nosso projeto inicial.

- Baixe e instale o Node:

<https://nodejs.org/pt-br/download/>

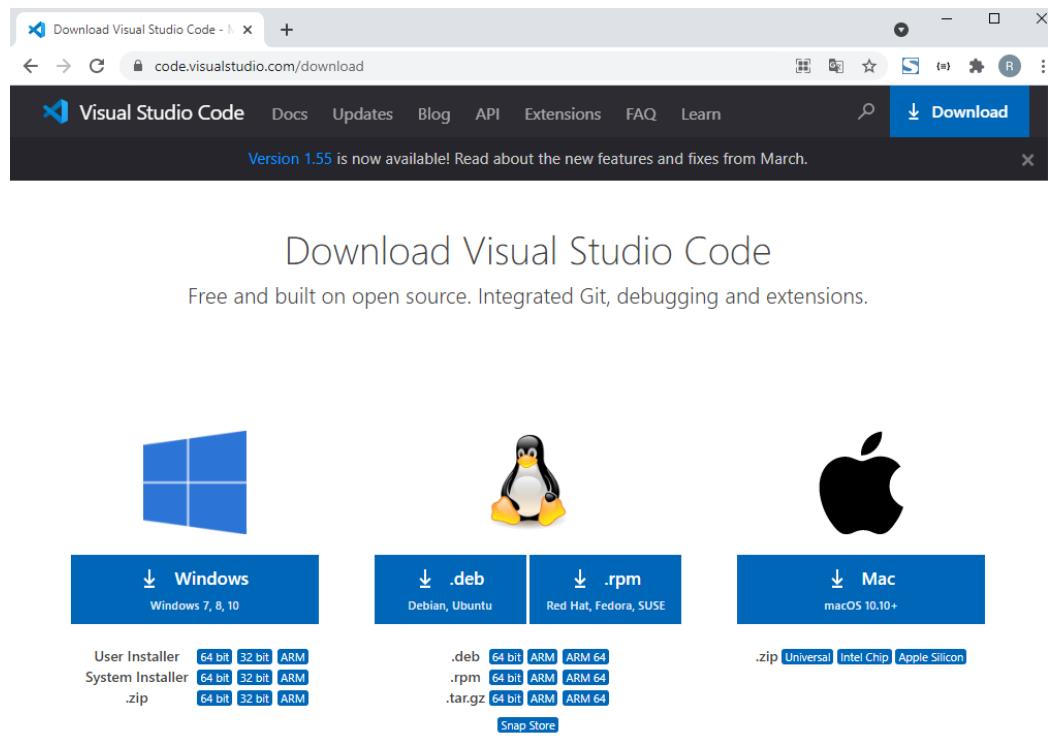
The screenshot shows the Node.js download page. At the top, there are two main sections: 'LTS' (Recommended for most users) and 'Atual' (Latest resources). Under 'LTS', there are links for 'Instalador Windows (.msi)', 'Binário para Windows (.zip)', 'Instalador macOS (.pkg)', 'Binário para macOS (.tar.gz)', 'Binários para Linux (x64)', 'Binários para Linux (ARM)', and 'Código fonte'. Under 'Atual', there are links for 'Instalador Windows (.msi)', 'Binário para Windows (.zip)', 'Instalador macOS (.pkg)', 'Binário para macOS (.tar.gz)', 'Binários para Linux (x64)', 'Binários para Linux (ARM)', and 'Código fonte'. A table below shows binary distributions for 32-bit and 64-bit architectures across different platforms.

	32-bit	64-bit
Windows (.msi)	node-v14.16.1.msi	
Windows (.zip)	node-v14.16.1.zip	
macOS (.pkg)		node-v14.16.1.pkg
macOS (.tar.gz)		node-v14.16.1.tar.gz
Linux (x64)		node-v14.16.1-x64.tar.gz
Linux (ARM)		node-v14.16.1-arm.tar.gz
Código fonte		node-v14.16.1.tar.gz

- Baixe a versão LTS

- Baixe e instale o Visual Studio Code

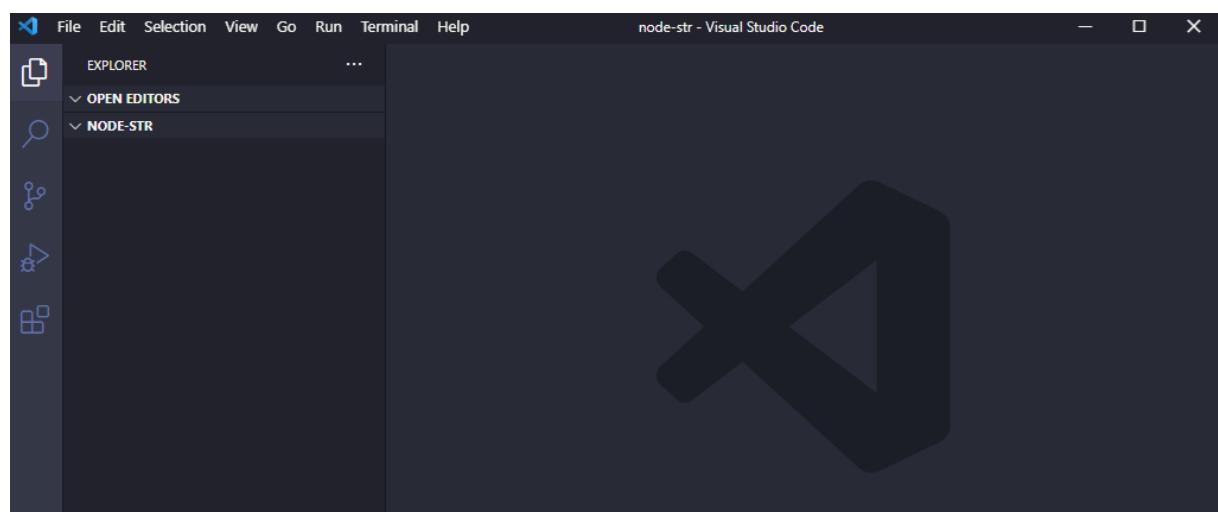
<https://code.visualstudio.com/download>



- Na pasta do projeto, entre com:

code .

```
C:\ Prompt de Comando  
C:\balta\nodejs\node-str>code .
```



- Abra um terminal **Powershell** e entre com os seguintes comandos:

```
node --version
```

```
PS C:\balta\nodejs\node-str> node --version
v14.15.3
```

```
npm --version
```

```
PS C:\balta\nodejs\node-str> npm --version
6.14.9
```

Aula 02 - npm init e instalação dos pacotes

Inicializando uma aplicação do Node

- No terminal, na pasta do projeto, entre com o comando:

```
npm init -y
```

The screenshot shows a terminal window in a dark-themed code editor. The tabs at the top are PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL, with TERMINAL selected. The title bar says "1: powershell". The terminal content is as follows:

```
PS C:\balta\nodejs\node-str> npm init -y
Wrote to C:\balta\nodejs\node-str\package.json:

{
  "name": "node-str",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

- É criado um arquivo chamado **package.json**

package.json

```
{
  "name": "node-str",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Instalando pacotes básicos

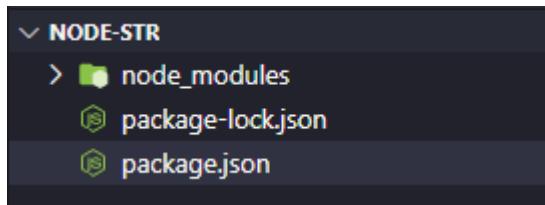
O comando a seguir irá instalar os pacotes:

- http
- express
- debug

```
npm install http express debug --save
```

```
PS C:\balta\nodejs\node-str> npm install http express debug --save
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN node-str@1.0.0 No description
npm WARN node-str@1.0.0 No repository field.

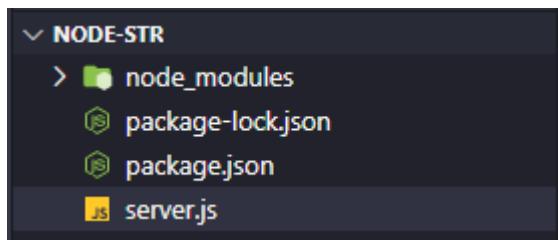
+ express@4.17.1
+ debug@4.3.1
+ http@0.0.1-security
added 59 packages from 38 contributors and audited 59 packages in 23.813s
found 0 vulnerabilities
```



package.json

```
{
  "name": "node-str",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "debug": "^4.3.1",
    "express": "^4.17.1",
    "http": "0.0.1-security"
  }
}
```

- Crie um arquivo chamado **server.js**



server.js

```
'use strict'  
  
console.log('Testando...');
```

```
PS C:\balta\nodejs\node-str> node ./server.js  
Testando...
```

Aula 03 - Criando um servidor web

server.js

```
'use strict'

const http = require('http');
const debug = require('debug')('nodestr: server');
const express = require('express');

const app = express();
const port = 3000;
app.set('port', port);

const server = http.createServer(app);
const router = express.Router();

const route = router.get('/', (req, res, next) => {
  res.status(200).send({
    title: "Node Store API",
    version: "0.0.1"
  });
});

app.use('/', route);

server.listen(port);
console.log('API rodando na porta ' + port);
```

- Rode o servidor:

```
node ./server.js
```

```
PS C:\balta\nodejs\node-str> node ./server.js
API rodando na porta 3000
|
```

- Instale a extensão **JSON Viewer** para Google Chrome.

- Selecione o tema **Dracula**:

{ ≡ } Options page

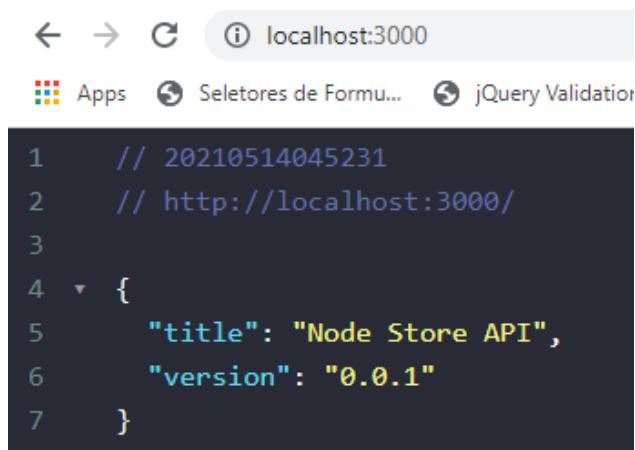
Theme

dracula ▾

```
1  {
2      "title": "JSON Example",
3  {
4      "nested": {
5          "someInteger": 7,
6          "someBoolean": true,
7          "someArray": [
8              "list of",
9              "fake strings",
10             "and fake keys"
11         ]
12     }
13 }
```

- No browser:

<http://localhost:3000/>



A screenshot of a web browser window. The address bar shows 'localhost:3000'. Below the address bar, there are tabs for 'Apps', 'Seletores de Formu...', and 'jQuery Validation'. The main content area displays the following JSON code:

```
1 // 20210514045231
2 // http://localhost:3000/
3
4 {
5     "title": "Node Store API",
6     "version": "0.0.1"
7 }
```

- Baixe e instale o **Postman**. Ele será utilizado para simular requisições.

Usando o Postman:

The screenshot shows the Postman application interface. On the left, there's a sidebar with sections for Collections, APIs, Environments, Mock Servers, Monitors, and History. The main area displays a collection titled "Testes - Node.JS - Balta". Inside this collection, there is a single API endpoint named "GET Testes - Node.JS - Balta". The request method is set to GET, and the URL is localhost:3000. The "Tests" tab is selected, containing a single test script line: "1". Below the request details, the response status is shown as 200 OK with a response time of 607 ms and a size of 279 B. The response body is displayed in Pretty JSON format, showing the following data:

```
1
2   "title": "Node Store API",
3   "version": "0.0.1"
4 }
```

Aula 04 - Normalizando a porta

server.js

```
'use strict'

const http = require('http');
const debug = require('debug')('nodestr: server');
const express = require('express');

const app = express();
const port = normalizePort(process.env.PORT) || '3000';
app.set('port', port);

const server = http.createServer(app);
const router = express.Router();

const route = router.get('/', (req, res, next) => {
  res.status(200).send({
    title: "Node Store API",
    version: "0.0.1"
  });
});

app.use('/', route);

server.listen(port);
console.log('API rodando na porta ' + port);

function normalizePort(val){
  const port = parseInt(val, 10);

  if(isNaN(port)){
    return val;
  }

  if(port >= 0){
    return port;
  }

  return false;
}
```

```
node ./server.js
```

```
PS C:\balta\nodejs\node-str> node ./server.js
API rodando na porta 3000
[
```

The screenshot shows the Postman application interface. On the left, there's a sidebar with icons for Collections, APIs, Environments, Mock Servers, Monitors, and History. The main area has tabs for Home, Workspaces, Reports, and Explore. A search bar at the top right says "Search Postman". Below it, there's a "New Collection" button and a collection named "Testes - Node.JS - Balta". Under this collection, there's a "GET Testes - Node.JS - Balta" request. The request details show a "GET" method, URL "localhost:3000", and a "Tests" tab selected. The "Tests" section contains a single line of JavaScript code: "1". Below the request, the response status is "200 OK", time "750 ms", size "279 B", and a "Save Response" button. The response body is displayed in "Pretty" format, showing the JSON object: { "title": "Node Store API", "version": "0.0.1" }.

Aula 05 - Gerenciando erros do servidor

server.js

```
'use strict'

const http = require('http');
const debug = require('debug')('nodestr: server');
const express = require('express');

const app = express();
const port = normalizePort(process.env.PORT) || '3000';
app.set('port', port);

const server = http.createServer(app);
const router = express.Router();

const route = router.get('/', (req, res, next) => {
  res.status(200).send({
    title: "Node Store API",
    version: "0.0.1"
  });
});

app.use('/', route);

server.listen(port);
server.on('error', onError);
console.log('API rodando na porta ' + port);

function normalizePort(val){
  const port = parseInt(val, 10);

  if(isNaN(port)){
    return val;
  }

  if(port >= 0){
    return port;
  }

  return false;
}

function onError(error) {
  if (error.syscall !== 'listen') {
    throw error;
  }
}
```

```
const bind = typeof port === 'string'  
  ? 'Pipe ' + port  
  : 'Port ' + port;  
  
switch (error.code) {  
  case 'EACCES':  
    console.error(bind + ' requires elevated privileges');  
    process.exit(1);  
    break;  
  case 'EADDRINUSE':  
    console.error(bind + ' is already in use');  
    process.exit(1);  
    break;  
  default:  
    throw error;  
}  
}
```

Criando um erro

- No Visual Studio Code, abra e execute o servidor em dois terminais:

```
node ./server.js
```

```
PS C:\balta\nodejs\node-str> node ./server.js  
API rodando na porta 3000  
Pipe 3000 is already in use
```

Aula 06 - Iniciando o Debug

server.js

```
'use strict'

const http = require('http');
const debug = require('debug')('nodestr: server');
const express = require('express');

const app = express();
const port = normalizePort(process.env.PORT) || '3000';
app.set('port', port);

const server = http.createServer(app);
const router = express.Router();

const route = router.get('/', (req, res, next) => {
  res.status(200).send({
    title: "Node Store API",
    version: "0.0.1"
  });
});

app.use('/', route);

server.listen(port);
server.on('error', onError);
server.on('listening', onListening);
console.log('API rodando na porta ' + port);

function normalizePort(val){
  const port = parseInt(val, 10);

  if(isNaN(port)){
    return val;
  }

  if(port >= 0){
    return port;
  }

  return false;
}
```

```
function onError(error) {
  if (error.syscall !== 'listen') {
    throw error;
  }

  const bind = typeof port === 'string'
    ? 'Pipe ' + port
    : 'Port ' + port;

  switch (error.code) {
    case 'EACCES':
      console.error(bind + ' requires elevated privileges');
      process.exit(1);
      break;
    case 'EADDRINUSE':
      console.error(bind + ' is already in use');
      process.exit(1);
      break;
    default:
      throw error;
  }
}

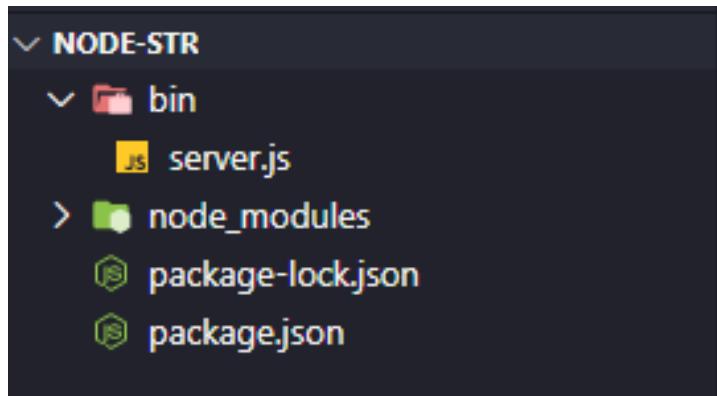
function onListening() {
  const addr = server.address();
  const bind = typeof addr === 'string'
    ? 'pipe ' + addr
    : 'port ' + addr.port;
  debug('Listening on ' + bind);
}
```

node ./server.js

```
PS C:\balta\nodejs\node-str> node ./server.js
API rodando na porta 3000
```

Aula 07 - Separando o servidor

- No diretório raiz da aplicação, crie uma pasta chamada **bin**. E mova o arquivo **server.js** para dentro dela.



- No arquivo **bin/server.js** altere a linha da **const app** e remova o trecho de código especificado:

bin/server.js

```
const app = require('../src/app');
const http = require('http');
const debug = require('debug')('balta:server');

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

const port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

const server = http.createServer(app);
const router = express.Router();

const route = router.get('/', (req, res, next) => {
  res.status(200).send({
    title: "Node Store API",
    version: "0.0.1"
  });
});

app.use('/', route);

server.listen(port);
```

```

server.on('error', onError);
server.on('listening', onListening);
console.log('API rodando na porta ' + port);

function normalizePort(val) {
  const port = parseInt(val, 10);

  if (isNaN(port)) {
    return val;
  }

  if (port >= 0) {
    return port;
  }

  return false;
}

function onError(error) {
  if (error.syscall !== 'listen') {
    throw error;
  }

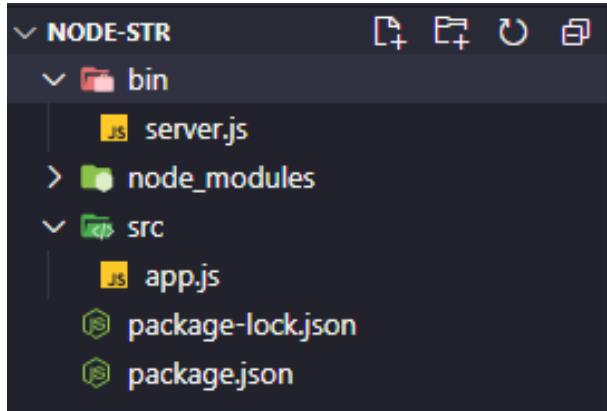
  const bind = typeof port === 'string'
    ? 'Pipe ' + port
    : 'Port ' + port;

  switch (error.code) {
    case 'EACCES':
      console.error(bind + ' requires elevated privileges');
      process.exit(1);
      break;
    case 'EADDRINUSE':
      console.error(bind + ' is already in use');
      process.exit(1);
      break;
    default:
      throw error;
  }
}

function onListening() {
  const addr = server.address();
  const bind = typeof addr === 'string'
    ? 'pipe ' + addr
    : 'port ' + addr.port;
  debug('Listening on ' + bind);
}

```

- No diretório raiz da aplicação, crie uma nova pasta chamada **src** e dentro dela insira um arquivo chamado **app.js**:



src/app.js

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

const app = express();
const router = express.Router();

const route = router.get('/', (req, res, next) => {
  res.status(200).send({
    title: "Node Store API",
    version: "0.0.1"
  });
});

app.use('/', route);

module.exports = app;
```

[node ./bin/server.js](#)

```
PS C:\balta\nodejs\node-str> node ./bin/server.js
API rodando na porta 3000
```

Aula 08 - Configurando o npm start

package.json

```
{  
  "name": "node-str",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1",  
    "start": "node ./bin/server.js"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC",  
  "dependencies": {  
    "debug": "^4.1.1",  
    "express": "^4.17.1",  
    "http": "0.0.0"  
  }  
}
```

npm start

```
PS C:\balta\nodejs\node-str> npm start  
> node-str@1.0.0 start C:\balta\nodejs\node-str  
> node ./bin/server.js  
  
API rodando na porta 3000  
|
```

Aula 09 - Nodemon

Instalando o pacote nodemon

```
npm install nodemon --save-dev
```

```
PS C:\balta\nodejs\node-str> npm install nodemon --save-dev
> nodemon@2.0.7 postinstall C:\balta\nodejs\node-str\node_modules\nodemon
> node bin/postinstall || exit 0

Love nodemon? You can now support the project via the open collective:
> https://opencollective.com/nodemon/donate

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@~2.3.1 (node_modules\chokidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN node-str@1.0.0 No description
npm WARN node-str@1.0.0 No repository field.

+ nodemon@2.0.7
added 118 packages from 53 contributors and audited 178 packages in 73.399s

11 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

package.json

```
{
  "name": "node-str",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1",
    "start": "node ./bin/server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "debug": "^4.3.1",
    "express": "^4.17.1",
    "http": "0.0.1-security"
  },
  "devDependencies": {
    "nodemon": "^2.0.7"
  }
}
```

```
nodemon ./bin/server.js
```

```
C:\balta\nodejs\node-str>nodemon ./bin/server.js
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node ./bin/server.js`
API rodando na porta 3000
█
```

- Fazendo uma alteração no arquivo src/app.js, ao salvá-lo:

src/app.js

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

const app = express();
const router = express.Router();

const route = router.get('/', (req, res, next) => {
  res.status(200).send({
    title: "Node Store API",
    version: "0.0.2"
  });
});

app.use('/', route);

module.exports = app;
```

```
C:\balta\nodejs\node-str>nodemon ./bin/server.js
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node ./bin/server.js`
API rodando na porta 3000
[nodemon] restarting due to changes...
[nodemon] starting `node ./bin/server.js`
API rodando na porta 3000
█
```

Aula 10 - CRUD Rest

Instalando o pacote body-parser

```
npm install body-parser --save
```

```
PS C:\balta\nodejs\node-str> npm install body-parser --save
npm WARN node-str@1.0.0 No description
npm WARN node-str@1.0.0 No repository field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

+ body-parser@1.19.0
updated 1 package and audited 179 packages in 10.138s

11 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

src/app.js

```
const express = require('express');
const bodyParser = require('body-parser');

const app = express();
const router = express.Router();

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({
  extended: false
}));

const route = router.get('/', (req, res, next) => {
  res.status(200).send({
    title: "Node Store API",
    version: "0.0.2"
  });
});

const create = router.post('/', (req, res, next) => {
  res.status(201).send(req.body);
});

app.use('/', route);
app.use('/products', create);

module.exports = app;
```

src/app.js

```
const express = require('express');
const bodyParser = require('body-parser');

const app = express();
const router = express.Router();

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({
  extended: false
}));

const route = router.get('/', (req, res, next) => {
  res.status(200).send({
    title: "Node Store API",
    version: "0.0.1"
  });
});

const create = router.post('/', (req, res, next) => {
  res.status(201).send(req.body);
});

const put = router.put('/:id', (req, res, next) => {
  const id = req.params.id;
  res.status(200).send({
    id: id,
    item: req.body
  });
});

const del = router.delete('/', (req, res, next) => {
  res.status(200).send(req.body);
});

app.use('/', route);
app.use('/products', create);
app.use('/products', put);
app.use('/products', del);

module.exports = app;
```

create → método POST

The screenshot shows the Postman interface with a collection named "Testes - Node JS - Balta". A POST request is being made to `localhost:3000/products`. The "Body" tab is selected, showing a JSON payload:

```
1
2   ...
3     "title": "Teste"
```

The response section shows a 201 Created status with 18 ms and 257 B. The response body is also displayed:

```
1
2   "title": "Teste"
3
```

put → método PUT

The screenshot shows the Postman interface with the same collection. A PUT request is being made to `localhost:3000/products/123`. The "Body" tab is selected, showing a JSON payload:

```
1
2   ...
3     "title": "Teste"
```

The response section shows a 200 OK status with 17 ms and 272 B. The response body is displayed:

```
1
2   "id": "123",
3   "item": {
4     "title": "Teste"
5   }
6
```

del → método DELETE

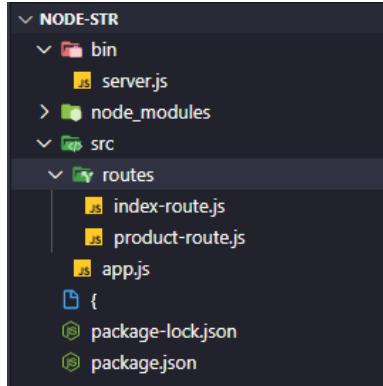
The screenshot shows the Postman interface with the same collection. A DELETE request is being made to `localhost:3000/products`. The "Body" tab is selected, showing a JSON payload:

```
1
2   ...
3     "title": "Teste"
```

The response section shows a 200 OK status with 13 ms and 252 B. The response body is displayed:

```
1
2   "title": "Teste"
3
```

Aula 11 - Rotas



src/routes/index-route.js

```
const express = require('express');
const router = express.Router();

router.get('/', (req, res, next) => {
    res.status(200).send({
        title: "Node Store API",
        version: "0.0.2"
    });
});

module.exports = router;
```

src/routes/product-route.js

```
const express = require('express');
const router = express.Router();

router.post('/', (req, res, next) => {
    res.status(201).send(req.body);
});

router.put('/:id', (req, res, next) => {
    const id = req.params.id;
    res.status(200).send({
        id: id,
        item: req.body
    });
});

router.delete('/', (req, res, next) => {
    res.status(200).send(req.body);
});

module.exports = router;
```

src/app.js

```
const express = require('express');
const bodyParser = require('body-parser');

const app = express();
const router = express.Router();

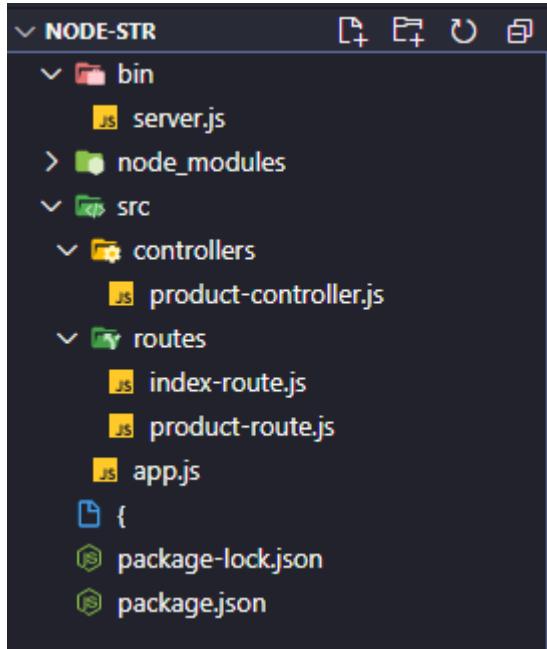
// Carrega as rotas
const indexRoute = require('./routes/index-route');
const productRoute = require('./routes/product-route');

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({
  extended: false
}));

app.use('/', indexRoute);
app.use('/products', productRoute);

module.exports = app;
```

Aula 12 - Controllers



src/controllers/product-controller.js

```
'use strict';

exports.post = (req, res, next) => {
    res.status(201).send(req.body);
};

exports.put = (req, res, next) => {
    const id = req.params.id;
    res.status(200).send({
        id: id,
        item: req.body
    });
};

exports.delete = (req, res, next) => {
    res.status(200).send(req.body);
};
```

src/routes/product-route.js

```
const express = require('express');
const router = express.Router();
const controller = require("../controllers/product-controller");

router.post('/', controller.post);
router.put('/:id', controller.put);
router.delete('/', controller.delete);

module.exports = router;
```

[nodemon ./bin/server.js](#)

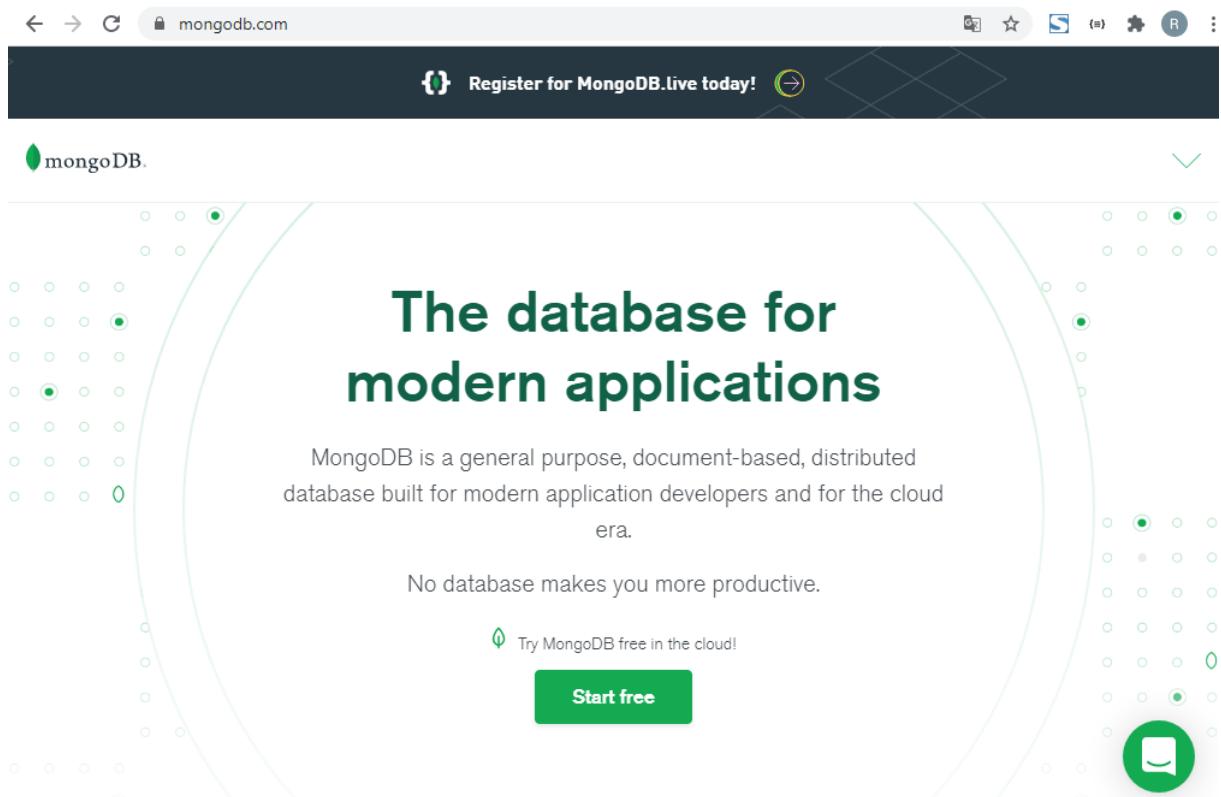
```
C:\balta\nodejs\node-str>nodemon ./bin/server.js
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node ./bin/server.js`
API rodando na porta 3000
|
```

- Faça os testes com o Postman para verificar se tudo continua funcionando normalmente.

Aula 13 - MongoDB Setup

- Baixe e instale o MongoDB.

www.mongodb.com



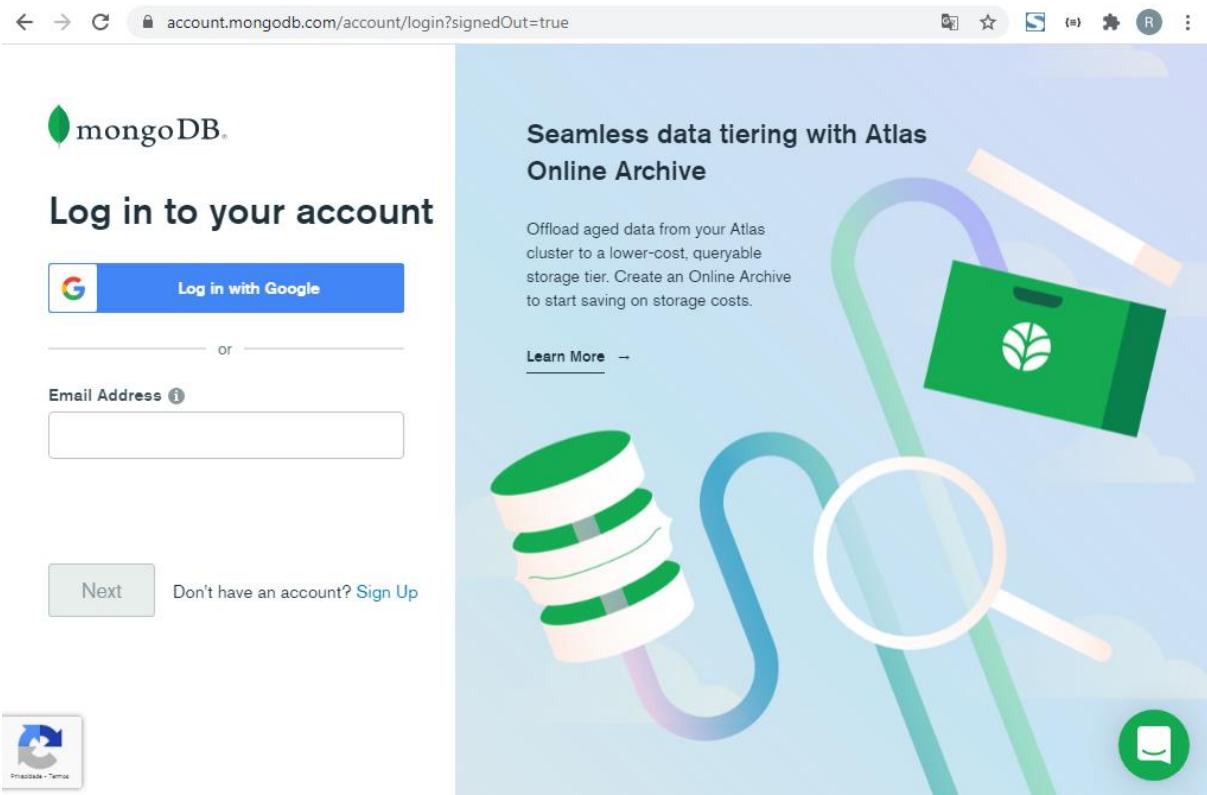
- Acesse <https://mlab.com> e abra uma conta (SIGN UP).

The screenshot shows the mLab website homepage. At the top, it says "mLab is now part of MongoDB, Inc." with a link to learn about migration. The mLab logo is on the left, and navigation links for "PLANS & PRICING" and "DOCUMENTATION" are at the top right, along with "SIGN UP" and "LOG IN" buttons. The main heading "Database-as-a-Service for MongoDB" is prominently displayed in the center, with a subtext "Now part of the MongoDB family, powering over 1 million deployments worldwide." Below this is a large button labeled "GET STARTED INSTANTLY with 500 MB FREE!".

The screenshot shows the MongoDB Atlas sign-up page from mLab. On the left, there's a section titled "mLab is now part of the MongoDB family" with a description of MongoDB Atlas as a fully-managed database-as-a-service. It lists steps for creating an account, including picking a cloud provider and selecting cluster tier. On the right, there's a section titled "Try MongoDB Atlas" with a subtext "Used by millions of developers around the world." Below this is a form for entering company information, work email, first name, last name, and password, followed by a checkbox for agreeing to terms and a "Get Started with 512 MB Free" button.

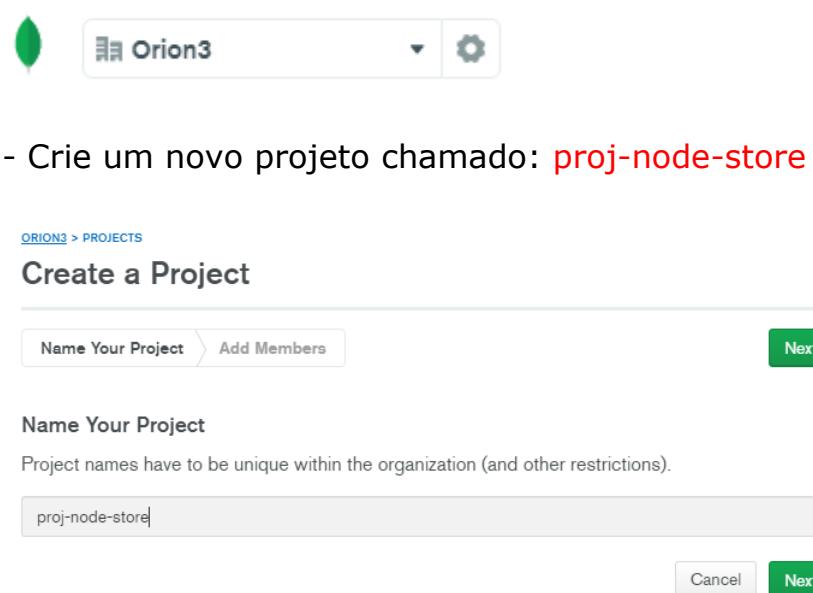
- Faça o login:

<https://account.mongodb.com/account/login>



The screenshot shows the MongoDB account login interface. On the left, there's a logo and a "Log in to your account" button. Below it are two login options: "Log in with Google" and a "Log in with Email Address" field. To the right, there's a promotional banner for "Seamless data tiering with Atlas Online Archive". The banner features a magnifying glass over a database icon and text about offloading aged data from an Atlas cluster.

- Crie uma organização com nome: **Orion3**



The screenshot shows the "Create a Project" step in the MongoDB organization creation process. It includes fields for "Name Your Project" (set to "proj-node-store") and "Add Members". A "Next" button is visible at the bottom right. The top navigation bar shows the organization name "Orion3".

- Crie um cluster para este projeto:

The screenshot shows the MongoDB Atlas Clusters page. On the left, there's a sidebar with sections for DATA STORAGE (Clusters, Triggers, Data Lake), SECURITY (Database Access, Network Access, Advanced), and Feature Requests. The main area is titled 'Clusters' with a search bar. It features a large green 'Create a cluster' button with a plus sign icon. Below it, a sub-section says 'Choose your cloud provider, region, and specs.' and contains a 'Build a Cluster' button. A note at the bottom says 'Once your cluster is up and running, live migrate an existing MongoDB database into Atlas with our Live Migration Service.' There's also a small circular icon with a mail symbol.

- Crie um cluster com nome: **node-store-cluster**

The screenshot shows the 'Create a Shared Cluster' wizard. At the top, it says 'CLUSTERS > CREATE A SHARED CLUSTER'. The first step, 'Create a Shared Cluster', is shown. It has a 'Cloud Provider & Region' section where 'AWS, N. Virginia (us-east-1)' is selected. Below this are dropdown menus for 'Cloud Provider' (AWS, Google Cloud, Azure) and 'Region' (Asia, North America, Europe, Australia). Under 'Cloud Provider & Region', there's a '★ Recommended region' section with 'Singapore (ap-southeast-1)', 'N. Virginia (us-east-1)' (selected), 'Oregon (us-west-2)', 'Mumbai (ap-south-1)', 'Frankfurt (eu-central-1)', and 'Ireland (eu-west-1)'. The next steps are 'Cluster Tier' (M0 Sandbox (Shared RAM, 512 MB Storage)) and 'Additional Settings' (MongoDB 4.4, No Backup). The final step is 'Cluster Name' (set to 'node-store-cluster'). A note below says 'One time only: once your cluster is created, you won't be able to change its name.' and a placeholder 'node-store-cluster' with a note that cluster names can only contain ASCII letters, numbers, and hyphens.

- Clique no botão "Create Cluster" e aguarde o cluster ser criado (leva um bom tempo):

The screenshot shows the MongoDB Cloud interface. The top navigation bar includes links for Orion3, Access Manager, Support, Billing, All Clusters, and a user account. Below the navigation is a header with 'proj-node-store', 'Atlas' (highlighted in green), 'Realm', and 'Charts'. On the left, a sidebar under 'DATA STORAGE' has 'Clusters' selected (highlighted in green). Other options include Triggers, Data Lake, SECURITY (Database Access, Network Access, Advanced), and Feature Requests. The main content area is titled 'Clusters' and shows 'ORION3 > PROJ-NODE-STORE'. It displays a cluster named 'node-store-cluster' (Version 4.4.5) in a 'SANDBOX' tier. The cluster details include: CLUSTER TIER (MO Sandbox (General)), REGION (AWS / N. Virginia (us-east-1)), TYPE (Replica Set - 3 nodes), and LINKED REALM APP (None Linked). A 'CONNECT' button, 'METRICS' tab (selected), 'COLLECTIONS' tab, and a three-dot menu are visible. To the right, there's a summary card for a 'Shared Tier Cluster' with metrics like Operations (R: 0 W: 0), Logical Size (0.0 B), Connections (0), and a note about upgrading to a dedicated cluster. A 'Create a New Cluster' button is in the top right corner.

- Crie um usuário para acessar o banco de dados.

The screenshot shows the MongoDB Cloud interface. The top navigation bar includes links for Orion3, Access Manager, Support, Billing, All Clusters, and a user account. Below the navigation is a header with 'proj-node-store', 'Atlas' (highlighted in green), 'Realm', and 'Charts'. On the left, a sidebar under 'SECURITY' has 'Database Access' selected (highlighted in green). Other options include Network Access and Advanced. The main content area is titled 'Database Access' and shows 'ORION3 > PROJ-NODE-STORE'. It displays a 'Database Users' section with a 'Custom Roles' tab. A large green '+' icon with a user icon is centered. Below it is a section titled 'Create a Database User' with the sub-instruction: 'Set up database users, permissions, and authentication credentials in order to connect to your clusters.' A 'Add New Database User' button is at the bottom, along with a 'Learn more' link. A 'Feature Requests' button is at the bottom left, and a green speech bubble icon is at the bottom right.

Add New Database User

Create a database user to grant an application or user, access to databases and collections in your clusters in this Atlas project. Granular access control can be configured with default privileges or custom roles. You can grant access to an Atlas project or organization using the corresponding Access Manager [Access Manager](#)

Authentication Method

- Password
- Certificate
- AWS IAM
(MongoDB 4.4 and up)

MongoDB uses **SCRAM** as its default authentication method.

Password Authentication

.....
SHOW

Database User Privileges

Select a [built-in role or privileges](#) for this user.

Read and write to any database

Restrict Access to Specific Clusters/Data Lakes

Enable to specify the resources this user can access. By default, all resources in this project are accessible. OFF

Temporary User

This user is temporary and will be deleted after your specified duration of 6 hours, 1 day, or 1 week. OFF

order to connect to

[Cancel](#) [Add User](#)

Learn more

We are deploying your changes (current action: configuring MongoDB)

ORION3 > PROJ-NODE-STORE

Database Access

[Database Users](#) [Custom Roles](#) [+ ADD NEW DATABASE USER](#)

User Name	Authentication Method	MongoDB Roles	Resources	Actions
betopinheiro1005	SCRAM	readWriteAnyDatabase@admin	All Resources	EDIT DELETE

- Em **node-store-cluster**, clique na aba "Collections"

The screenshot shows the MongoDB Atlas Cluster Overview page for the 'node-store-cluster'. At the top, it displays the cluster name, version (4.0.12), and region (N. Virginia (us-east-1)). Below this, there are tabs for Overview, Real Time, Metrics, and Collections, with 'Collections' being the active tab. It shows 0 databases and 0 collections. A central section titled 'Interact with your data' encourages running queries, viewing metadata, managing indexes, and interacting with data using full CRUD functionality. It includes buttons to 'Load a Sample Dataset' or 'Add my own data'.

- Crie o database **node-store-db** com a collection **products**.

The screenshot shows the 'Create Database' dialog box. It has fields for 'DATABASE NAME' (containing 'node-store-db') and 'COLLECTION NAME' (containing 'products'). There is a checkbox for 'Capped Collection' which is unchecked. A note below states: 'Before MongoDB can save your new database, a collection name must be specified at the time of creation.' At the bottom are 'Cancel' and 'Create' buttons.

The screenshot shows the MongoDB Atlas Cluster Overview page for the 'node-store-cluster'. The left sidebar shows 'Clusters' selected under 'DATA STORAGE'. In the main area, it shows 1 database ('node-store-db') and 1 collection ('products'). The 'Collections' tab is active. The 'products' collection details are shown on the right, including its size (0B), document count (0), and index size (4KB). It includes tabs for Find, Indexes, Schema Anti-Patterns, Aggregation, and Search Indexes. A search bar at the bottom contains the filter '("filter": "example")'.

- Em **node-store-cluster**, clique no botão "Connect":

Connect to node-store-cluster

The screenshot shows a step in the MongoDB connection wizard titled 'Choose a connection method'. It includes a breadcrumb navigation bar with three items: 'Setup connection security' (with a green checkmark), 'Choose a connection method' (with a green checkmark), and 'Connect' (unselected). Below the breadcrumb is a sub-header 'Choose a connection method' with a 'View documentation' link. A note says 'Get your pre-formatted connection string by selecting your tool below.' Three options are listed: 'Connect with the mongo shell' (using the mongo shell icon), 'Connect your application' (using the application icon), and 'Connect using MongoDB Compass' (using the compass icon). Each option has a brief description and a right-pointing arrow. At the bottom are 'Go Back' and 'Close' buttons.

- Escolha o método de conexão: "**Connect Your Application**":

Connect to node-store-cluster

The screenshot shows the 'Choose a connection method' step with the 'Connect Your Application' option selected. The breadcrumb now shows 'Setup connection security' (green checkmark), 'Choose a connection method' (green checkmark), and 'Connect' (selected). Step 1, 'Select your driver and version', shows dropdown menus for 'DRIVER' (Node.js) and 'VERSION' (3.6 or later). Step 2, 'Add your connection string into your application code', contains a code input field with the connection string: `mongodb+srv://betopinheiro1005:<password>@node-store-cluster.14jj0.mongodb.net/myFirstDatabase?retryWrites=true&w=majority`. A note below says to replace <password> with the password for the betopinheiro1005 user. Step 3, 'Include full driver code example', has an unchecked checkbox. At the bottom are 'Go Back' and 'Close' buttons.

- Copie a string de conexão:

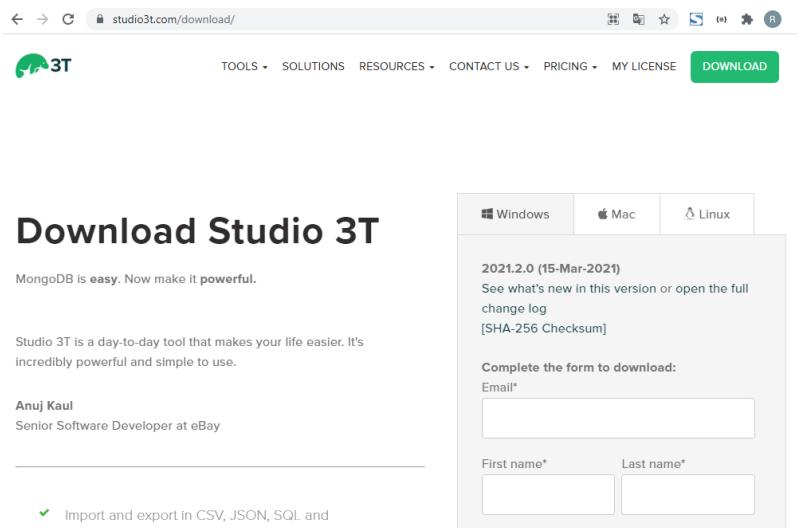
`mongodb+srv://betopinheiro1005:<password>@node-store-cluster.l4jj0.mongodb.net/myFirstDatabase?retryWrites=true&w=majority`

- Substitua myFirstDatabase por node-store-db
- Substitua <password> pela senha do usuário.

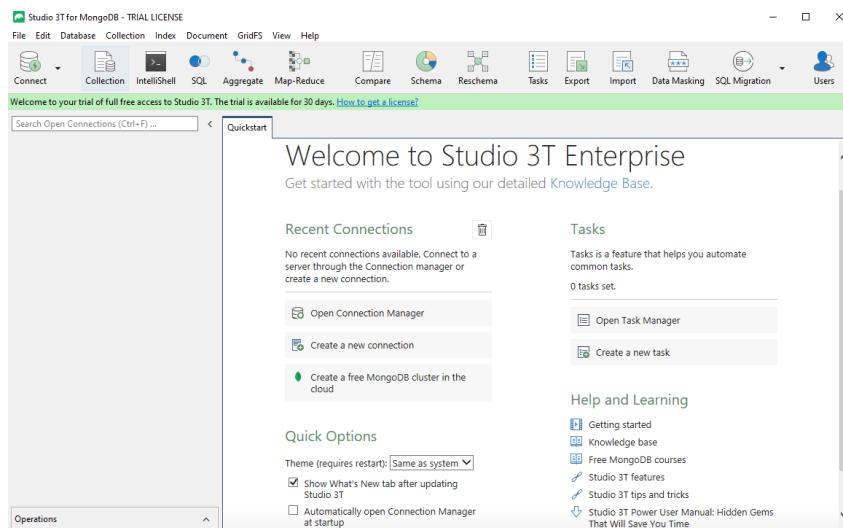
`mongodb+srv://betopinheiro1005:angstron1005@node-store-cluster.l4jj0.mongodb.net/node-store-db?retryWrites=true&w=majority`

-
- Baixe e instale o programa **Studio 3T**.

<https://studio3t.com/download/>

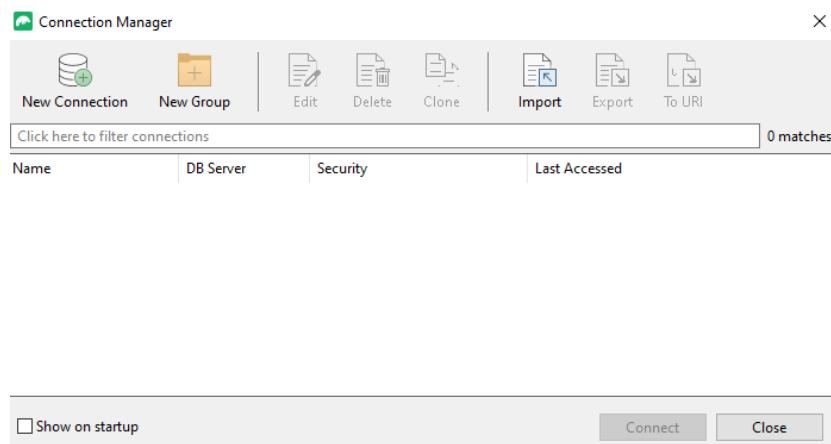


The screenshot shows the official website for Studio 3T. At the top, there's a navigation bar with links for 'TOOLS', 'SOLUTIONS', 'RESOURCES', 'CONTACT US', 'PRICING', and 'MY LICENSE'. A prominent green 'DOWNLOAD' button is located on the right side of the header. Below the header, there's a section titled 'Download Studio 3T' with a sub-section for 'MongoDB is easy. Now make it powerful.' It features a 'Windows' button, an 'Apple Mac' button, and a 'Linux' button. To the right, there's a '2021.2.0 (15-Mar-2021)' release note with a link to the 'change log' and 'SHA-256 Checksum'. Below this, there's a form to 'Complete the form to download:' with fields for 'Email*', 'First name*', and 'Last name*'. At the bottom left, there's a note about importing and exporting data in CSV, JSON, and SQL formats.

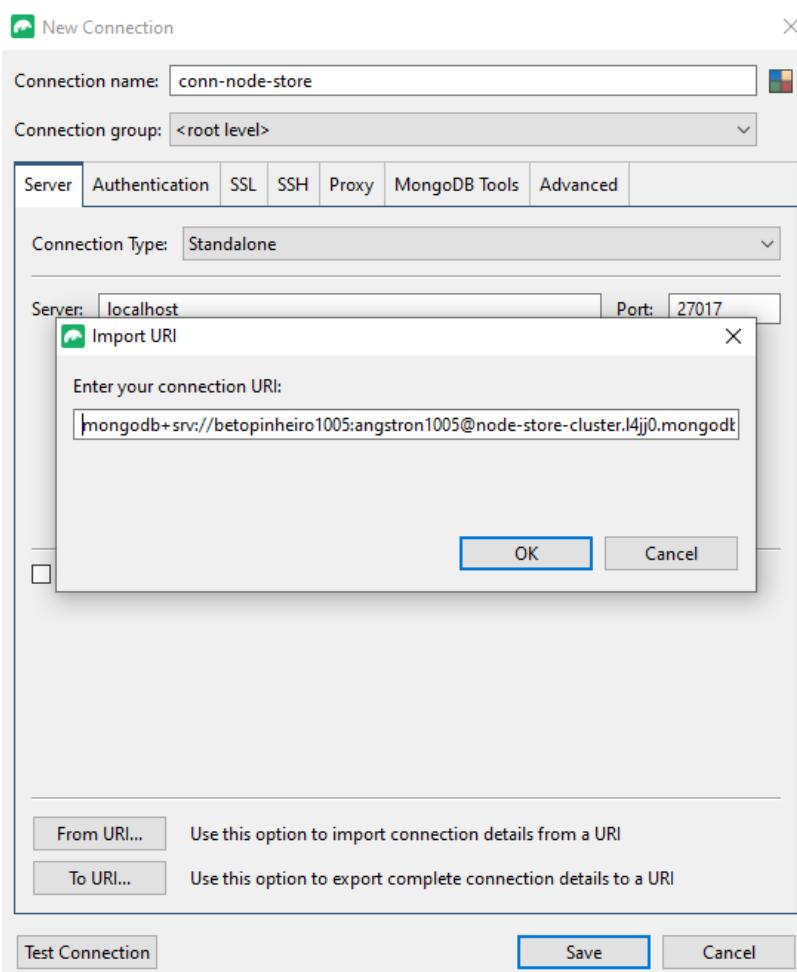


The screenshot shows the Studio 3T Enterprise application window. The title bar says 'Studio 3T for MongoDB - TRIAL LICENSE'. The main interface has several sections: 'Recent Connections' (empty), 'Tasks' (empty), 'Help and Learning' (links to 'Getting started', 'Knowledge base', 'Free MongoDB courses', 'Studio 3T features', 'Studio 3T tips and tricks', and 'Studio 3T Power User Manual: Hidden Gems That Will Save You Time'), and 'Quick Options' (checkboxes for 'Show What's New tab after updating Studio 3T' and 'Automatically open Connection Manager at startup'). A message at the top of the main area says 'Welcome to your trial of full free access to Studio 3T. The trial is available for 30 days. How to get a license?'.

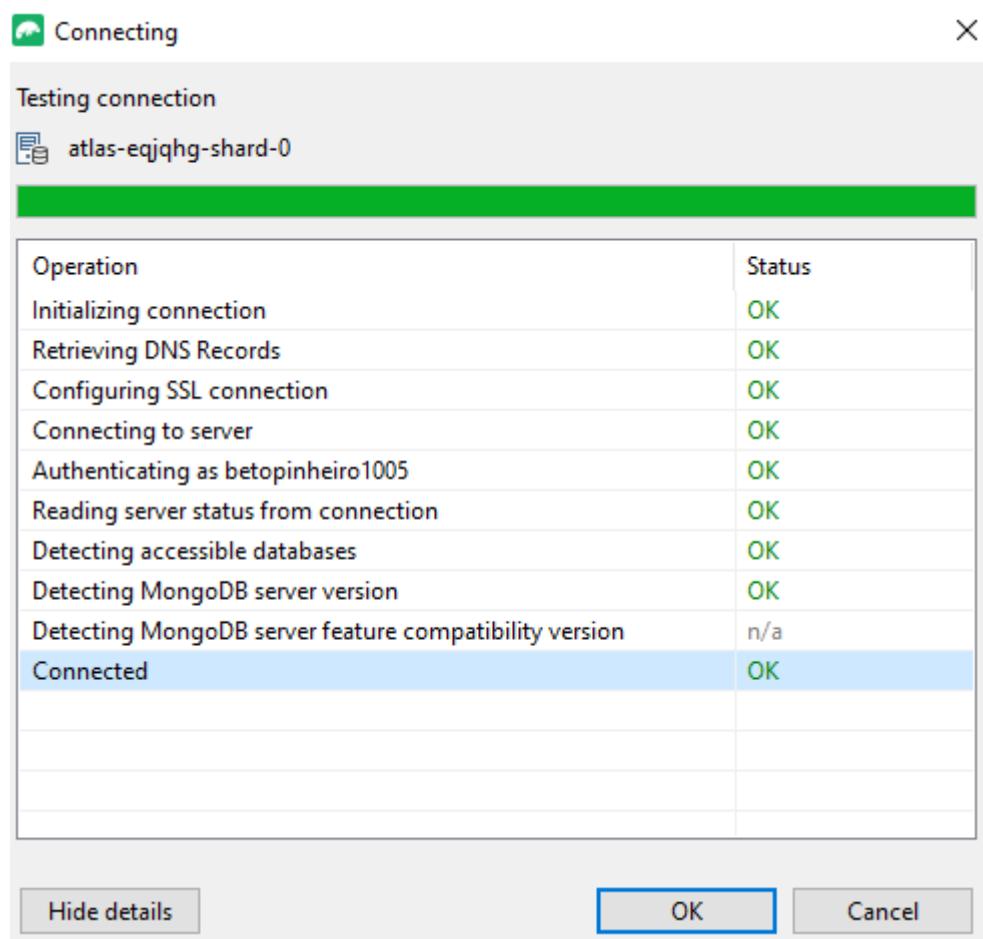
- Clique no botão **Connect**.
- Clique em **New Connection**.



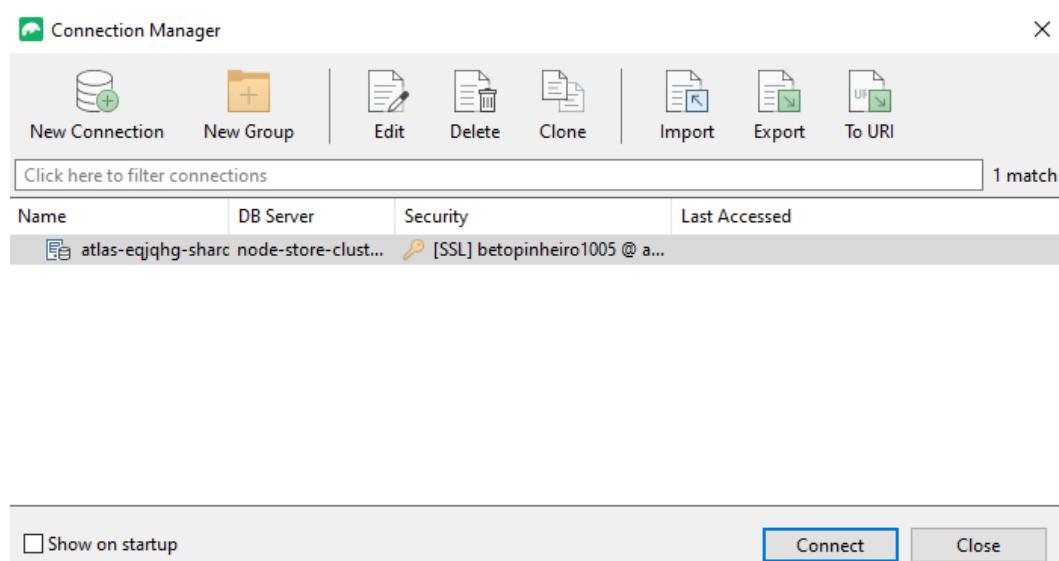
- Dê o nome para a conexão: **conn-node-store**.
- Clique no botão **From URI** e cole a string de conexão.

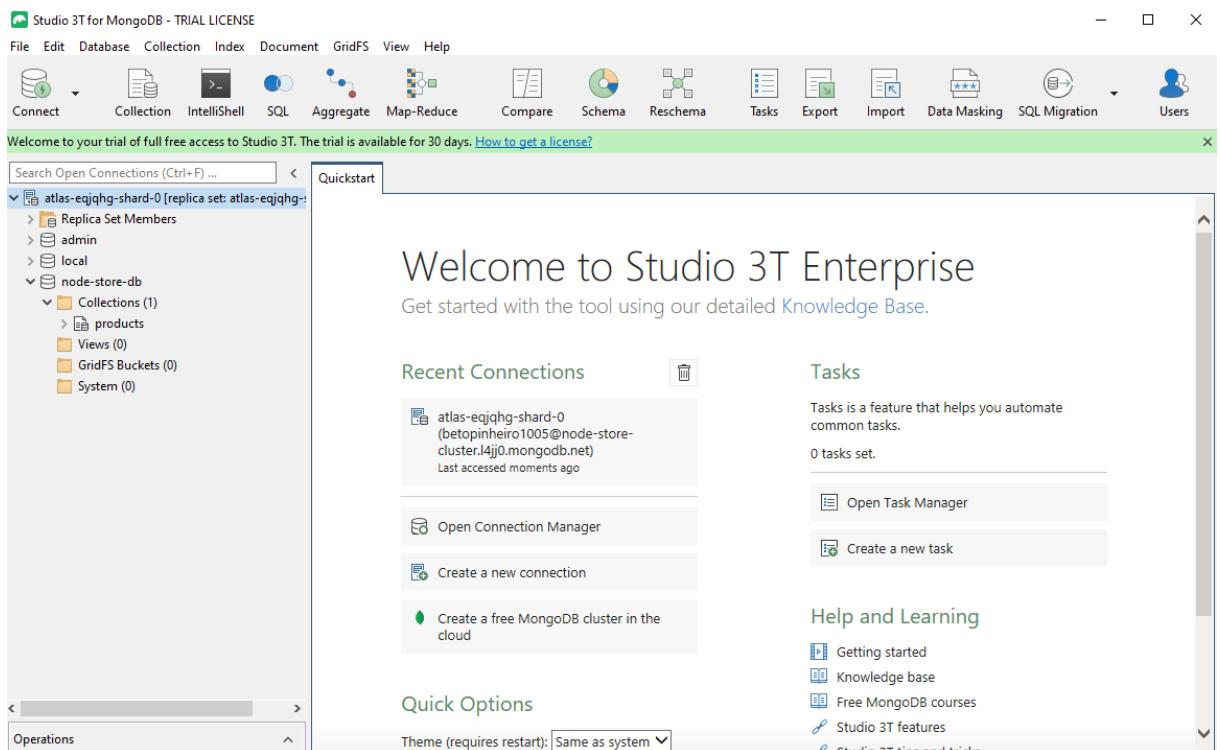


- Faça o teste de conexão:



- Se estiver tudo ok, clique no botão "**Save**" para salvar a configuração.
- Clique no botão "**Connect**"





- Outro possível serviço

<https://robomongo.org/download>

The screenshot shows the Robo 3T website. At the top, there's a navigation bar with back, forward, and search icons, followed by the URL 'robomongo.org/download'. To the right are links for 'Download', 'Blog', and 'Account'. Below the navigation is a large green header with the text 'Simplicity Meets Power'. Underneath it, a sub-header says 'Download the latest version of Robo 3T'. A paragraph below that states 'A free 30-day trial of the full access edition of Studio 3T is included with your double-pack download of Robo 3T.' It also encourages users to 'Try it out and see how much more you can do.' At the bottom is a green button labeled 'Download your Double Pack'.

Aula 14 - Mongoose

Instalação do Mongoose

- Para fazer a conexão com o banco de dados vamos utilizar um pacote chamado Mongoose. Para instalá-lo use o comando:

```
npm install mongoose --save
```

```
C:\balta\nodejs\node-str>npm install mongoose --save
npm WARN node-str@1.0.0 No description
npm WARN node-str@1.0.0 No repository field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
+ mongoose@5.12.7
added 29 packages from 92 contributors and audited 211 packages in 33.806s

13 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

package.json

```
{
  "name": "node-str",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1",
    "start": "node ./bin/server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.19.0",
    "debug": "^4.3.1",
    "express": "^4.17.1",
    "http": "0.0.1-security",
    "mongoose": "^5.12.7"
  },
  "devDependencies": {
    "nodemon": "^2.0.7"
  }
}
```

src/app.js

```
const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');

const app = express();
const router = express.Router();

// Conecta ao banco
mongoose.connect("mongodb+srv://betopinheiro1005:angstron1005@node-store-cluster.l4jj0.mongodb.net/node-store-db?retryWrites=true&w=majority");

// Carrega as rotas
const indexRoute = require('./routes/index-route');
const productRoute = require('./routes/product-route');

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({
    extended: false
}));

app.use('/', indexRoute);
app.use('/products', productRoute);

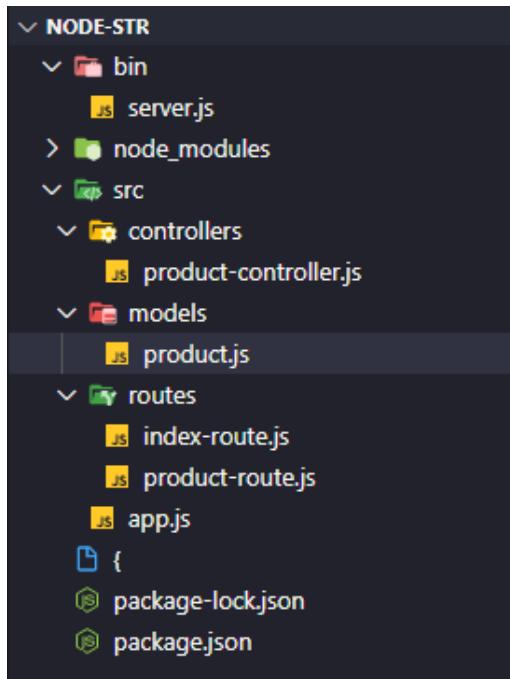
module.exports = app;
```

- Se estiver tudo ok, o servidor irá rodar normalmente:

nodemon ./bin/server.js

```
C:\balta\nodejs\node-str>nodemon ./bin/server.js
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node ./bin/server.js`
API rodando na porta 3000
(node:3484) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:3484) [MONGODB DRIVER] Warning: Top-level use of w, wtimeout, j, and fsync is deprecated. Use writeConcern instead.
(node:3484) [MONGODB DRIVER] Warning: Current Server Discovery and Monitoring engine is deprecated, and will be removed in a future version. To use the new Server Discover and Monitoring engine, pass option { useUnifiedTopology: true } to the MongoClient constructor.
```

Aula 15 - Models



src/models/product.js

```
'use strict';

const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const schema = new Schema({
  title: {
    type: String,
    required: true,
    trim: true
  },
  slug: {
    type: String,
    required: [true, 'O slug é obrigatório'],
    trim: true,
    index: true,
    unique: true
  },
  description: {
    type: String,
    required: true
  },
  price: {
    type: Number,
    required: true
  },
})
```

```
active: {
  type: Boolean,
  required: true,
  default: true
},
tags: [
  {
    type: String,
    required: true
  }
]);
module.exports = mongoose.model('Product', schema);
```

Aula 16 - Criando um produto

src\controllers\product-controller.js

```
'use strict';

const mongoose = require('mongoose');
const Product = mongoose.model('Product');

exports.post = (req, res, next) => {
  var product = new Product(req.body);
  product.save().then(x => {
    res.status(201).send({ message: "Produto cadastrado com sucesso!" });
  }).catch(e => {
    res.status(400).send({ message: "Falha ao cadastrar o produto!", data: e });
  });
};

exports.put = (req, res, next) => {
  const id = req.params.id;
  res.status(200).send({
    id: id,
    item: req.body
  });
};

exports.delete = (req, res, next) => {
  res.status(200).send(req.body);
};
```

src\app.js

```
const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');

const app = express();
const router = express.Router();

// Conecta ao banco
mongoose.connect("mongodb+srv://betopinheiro1005:angstron1005@node-str-f9kvu.mongodb.net/test?retryWrites=true&w=majority");

// Carrega os models
const Product = require('./models/product');

// Carrega as rotas
const indexRoute = require('./routes/index-route');
const productRoute = require('./routes/product-route');

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({
  extended: false
}));

app.use('/', indexRoute);
app.use('/products', productRoute);

module.exports = app;
```

Usando o Postman para cadastrar um produto

The screenshot shows the Postman interface. On the left sidebar, there are sections for Collections, APIs, Environments, Mock Servers, Monitors, and History. Under APIs, there is a tree view with nodes like 'Postman Echo', 'Testes - Node JS - Balta', and 'Cadastrando um produto'. The main workspace shows a POST request to 'localhost:3000/products'. The 'Body' tab is selected, displaying the following JSON payload:

```
1 {
2   "title": "Mouse Gamer",
3   "description": "Mouse Gamer",
4   "slug": "mouse-gamer",
5   "price": 299,
6   "active": true,
7   "tags": [
8     "informática", "mouse", "games"
9   ]
10 }
```

Below the body, the response status is 201 Created with a message: "Produto cadastrado com sucesso!"

No Studio 3T:

The screenshot shows the Studio 3T interface. On the left, a tree view of the database structure is visible, including 'atlas-ejqqhg-shard-0' (replica set), 'Replica Set Members', 'admin', 'local', and 'node-store-db'. Under 'node-store-db', there is a 'Collections (1)' folder containing a single entry 'products'. Other items under 'node-store-db' include 'Views (0)', 'GridFS Buckets (0)', and 'System (0)'.

The screenshot shows the Studio 3T interface with a query results table. The table has columns for 'Key' and 'Value'. The 'Key' column lists fields like '_id', 'active', 'tags', 'title', 'description', 'slug', 'price', and '__v'. The 'Value' column displays their corresponding values. A 'Type' column indicates the data type for each field. The table shows one document with the following data:

Key	Type
0 _id	Document
0 __v	Int32
0 active	Bool
0 description	String
0 slug	String
0 title	String
0 tags	Array
0 price	Int32

No mlab:

The screenshot shows the MongoDB Atlas interface. In the top navigation bar, the cluster name 'Orion3' is selected. The main menu includes 'Access Manager', 'Support', 'Billing', 'All Clusters', and 'Roberto'. Below the menu, there are tabs for 'Atlas', 'Realm', 'Charts', 'Collections', 'Search', 'Profiler', 'Performance Advisor', 'Online Archive', and 'Command Line Tools'. The 'Collections' tab is active, showing 'node-store-db.products'. The left sidebar has sections for 'Data Lake', 'SECURITY', 'Database Access', 'Network Access', and 'Advanced'. Under 'Database Access', there is a '+ Create Database' button and a 'NAMESPACES' search bar. The 'node-store-db' section is expanded, showing the 'products' collection. The 'products' collection details are displayed: 'COLLECTION SIZE: 182B', 'TOTAL DOCUMENTS: 1', 'INDEXES TOTAL SIZE: 40KB'. Below this, there are buttons for 'Find', 'Indexes', 'Schema Anti-Patterns', 'Aggregation', and 'Search Indexes'. A search bar contains the filter '{ "filter": "example" }'. At the bottom, there are 'Find' and 'Reset' buttons, and a 'QUERY RESULTS 1-1 OF 1' section showing a single document:

```
_id: ObjectId("8e95aa9fbcdcc36e01267ce")
active: true
tags: Array
  0: "informática"
  1: "mouse"
  2: "games"
title: "Mouse Gamer"
description: "Mouse Gamer"
slug: "mouse-gamer"
price: 299
__v: 0
```

Aula 17 - Listando os produtos

src/controllers/product-controller.js

```
'use strict';

const mongoose = require('mongoose');
const Product = mongoose.model('Product');

exports.get = (req, res, next) => {
  Product.find({}).then(data => {
    res.status(200).send(data);
  }).catch(e => {
    res.status(400).send(e);
  });
};

exports.post = (req, res, next) => {
  var product = new Product(req.body);
  product.save().then(x => {
    res.status(201).send({message: 'Produto cadastrado com sucesso!'});
  }).catch(e => {
    res.status(400).send({message: 'Falha ao cadastrar o produto!', data: e});
  });
};

exports.put = (req, res, next) => {
  const id = req.params.id;
  res.status(200).send({
    id: id,
    item: req.body
  });
};

exports.delete = (req, res, next) => {
  res.status(200).send(req.body);
};
```

src/routes/product-route.js

```
const express = require('express');
const router = express.Router();
const controller = require("../controllers/product-controller");

router.get('/', controller.get);
router.post('/', controller.post);
router.put('/:id', controller.put);
router.delete('/', controller.delete);

module.exports = router;
```

Testando no Postman

GET - localhost:3000/products

The screenshot shows the Postman application interface. On the left, there's a sidebar with sections for Collections, APIs, Environments, Mock Servers, and Monitors. A collection named 'Testes - Node JS - Balta' is selected, and it contains a test script named 'Listando produtos'. The test script has one test case for a GET request to 'localhost:3000/products'. The response body is shown as JSON:

```
1  {
2   "active": true,
3   "tags": [
4     "informática",
5     "mouse",
6     "games"
7   ],
8   "_id": "6095a69fbedcdc36e81267c0",
9   "title": "Mouse Gamer",
10  "description": "Mouse Gamer",
11  "slug": "mouse-gamer",
12  "price": 299,
13  "__v": 0
14}
```

No navegador

<http://localhost:3000/products>



The screenshot shows a browser window with the address bar containing 'localhost:3000/products'. The main content area displays a JSON array of products. The first product in the array is shown in detail:

```
1 // 20210507180924
2 // http://localhost:3000/products
3
4 [
5   {
6     "active": true,
7     "tags": [
8       "informática",
9       "mouse",
10      "games"
11    ],
12    "_id": "6095a69fbedcdc36e01267c0",
13    "title": "Mouse Gamer",
14    "description": "Mouse Gamer",
15    "slug": "mouse-gamer",
16    "price": 299,
17    "__v": 0
18  }
19]
```

Exibindo apenas alguns campos

src/controllers/product-controller.js

```
'use strict';

const mongoose = require('mongoose');
const Product = mongoose.model('Product');

exports.get = (req, res, next) => {
  Product.find({ active: true }, 'title price slug').then(data => {
    res.status(200).send(data);
  }).catch(e => {
    res.status(400).send(e);
  });
};

exports.post = (req, res, next) => {
  var product = new Product(req.body);
  product.save().then(x => {
    res.status(201).send({message: 'Produto cadastrado com sucesso!'});
  }).catch(e => {
    res.status(400).send({message: 'Falha ao cadastrar o produto!', data: e});
  });
};
```

```

exports.put = (req, res, next) => {
  const id = req.params.id;
  res.status(200).send({
    id: id,
    item: req.body
  });
};

exports.delete = (req, res, next) => {
  res.status(200).send(req.body);
};

```

- No Postman:

The screenshot shows the Postman application interface. On the left, there's a sidebar with sections for Collections, APIs, Environments, Mock Servers, Monitors, and History. A collection named 'Testes - Node JS - Balta' is expanded, showing five endpoints: 'GET route - GET - localhost:3000', 'GET create - POST - localhost:3000/products', 'GET put - PUT - localhost:3000/products/:id', 'GET del - DELETE - localhost:3000/products', and 'GET Cadastrando um produto'. The 'GET Listando produtos' endpoint is currently selected. The main panel shows a request card for a 'GET' request to 'localhost:3000/products'. Below it, the 'Tests' tab is active, containing a single test script: '1'. The 'Body' tab is selected, showing the response body in a JSON tree view. The response is a single object with properties: '_id', 'title', 'slug', and 'price'. The status bar at the bottom indicates a 200 OK response with a 232 ms duration and 326 B size.

- No browser:

The screenshot shows a browser developer tools Network tab. A request to 'localhost:3000/products' is listed with a status of '200 OK'. The response body is displayed as a JSON object. The object has properties: '_id', 'title', 'slug', and 'price'. The value for '_id' is '6095a69fbedcdc36e01267c0', 'title' is 'Mouse Gamer', 'slug' is 'mouse-gamer', and 'price' is 299.

Aula 18 - Listando um produto pelo slug

src/controllers/product-controller.js

```
'use strict';

const mongoose = require('mongoose');
const Product = mongoose.model('Product');

exports.get = (req, res, next) => {
  Product.find({ active: true }, 'title price slug').then(data => {
    res.status(200).send(data);
  }).catch(e => {
    res.status(400).send(e);
  });
};

exports.getBySlug = (req, res, next) => {
  Product.findOne({ slug: req.params.slug, active: true }, 'title description price slug
tags').then(data => {
    res.status(200).send(data);
  }).catch(e => {
    res.status(400).send(e);
  });
};

exports.post = (req, res, next) => {
  var product = new Product(req.body);
  product.save().then(x => {
    res.status(201).send({message: 'Produto cadastrado com sucesso!'});
  }).catch(e => {
    res.status(400).send({message: 'Falha ao cadastrar o produto!', data: e});
  });
};

exports.put = (req, res, next) => {
  const id = req.params.id;
  res.status(200).send({
    id: id,
    item: req.body
  });
};

exports.delete = (req, res, next) => {
  res.status(200).send(req.body);
};
```

src/routes/product-route.js

```
const express = require('express');
const router = express.Router();
const controller = require("../controllers/product-controller");

router.get('/', controller.get);
router.get('/:slug', controller.getBySlug);
router.post('/', controller.post);
router.put('/:id', controller.put);
router.delete('/', controller.delete);

module.exports = router;
```

Testando no Postman

GET - localhost:3000/products/mouse-gamer

The screenshot shows the Postman application interface. On the left, there's a sidebar with sections for Collections, APIs, Environments, Mock Servers, Monitors, and History. A collection named 'Testes - Node JS - Balta' is selected. Inside this collection, there are several items: 'GET route - GET - localhost:3000', 'GET create - POST - localhost:3000/products', 'GET put - PUT - localhost:3000/products/:id', 'GET del - DELETE - localhost:3000/products', 'GET Cadastrando um produto', 'GET Listando produtos', and 'GET Busca de produto por slug'. The 'GET Busca de produto por slug' item is currently expanded. In the main workspace, there's a search bar at the top with 'Search Postman'. Below it, a header bar shows 'Testes - Node JS - Balta / Busca de produto por slug'. A 'Send' button is visible. The main area shows a 'GET' request to 'localhost:3000/products/mouse-gamer'. The 'Tests' tab is selected. The response body is shown as JSON:

```
1 {
2   "tags": [
3     "informática",
4     "mouse",
5     "games"
6   ],
7   "_id": "6095a69fbedcdc36e01267c0",
8   "title": "Mouse Gamer",
9   "description": "Mouse Gamer",
10  "slug": "mouse-gamer",
11  "price": 299
12 }
```

No navegador

```
// 20210507183225
// http://localhost:3000/products/mouse-gamer

{
  "tags": [
    "informática",
    "mouse",
    "games"
  ],
  "_id": "6095a69fbedcdc36e01267c0",
  "title": "Mouse Gamer",
  "description": "Mouse Gamer",
  "slug": "mouse-gamer",
  "price": 299
}
```

Aula 19 - Listando um produto pelo id

src/controllers/product-controller.js

```
'use strict';

const mongoose = require('mongoose');
const Product = mongoose.model('Product');

exports.get = (req, res, next) => {
  Product.find({ active: true }, 'title price slug').then(data => {
    res.status(200).send(data);
  }).catch(e => {
    res.status(400).send(e);
  });
};

exports.getBySlug = (req, res, next) => {
  Product.findOne({ slug: req.params.slug, active: true }, 'title description price slug
tags').then(data => {
    res.status(200).send(data);
  }).catch(e => {
    res.status(400).send(e);
  });
};

exports.getById = (req, res, next) => {
  Product.findById({_id: req.params.id}).then(data => {
    res.status(200).send(data);
  }).catch(e => {
    res.status(400).send(e);
  });
};

exports.post = (req, res, next) => {
  var product = new Product(req.body);
  product.save().then(x => {
    res.status(201).send({message: 'Produto cadastrado com sucesso!'});
  }).catch(e => {
    res.status(400).send({message: 'Falha ao cadastrar o produto!', data: e });
  });
};

exports.put = (req, res, next) => {
  const id = req.params.id;
  res.status(200).send({
    id: id,
    item: req.body
  });
};

exports.delete = (req, res, next) => {
  res.status(200).send(req.body);
};
```

src/routes/product-route.js

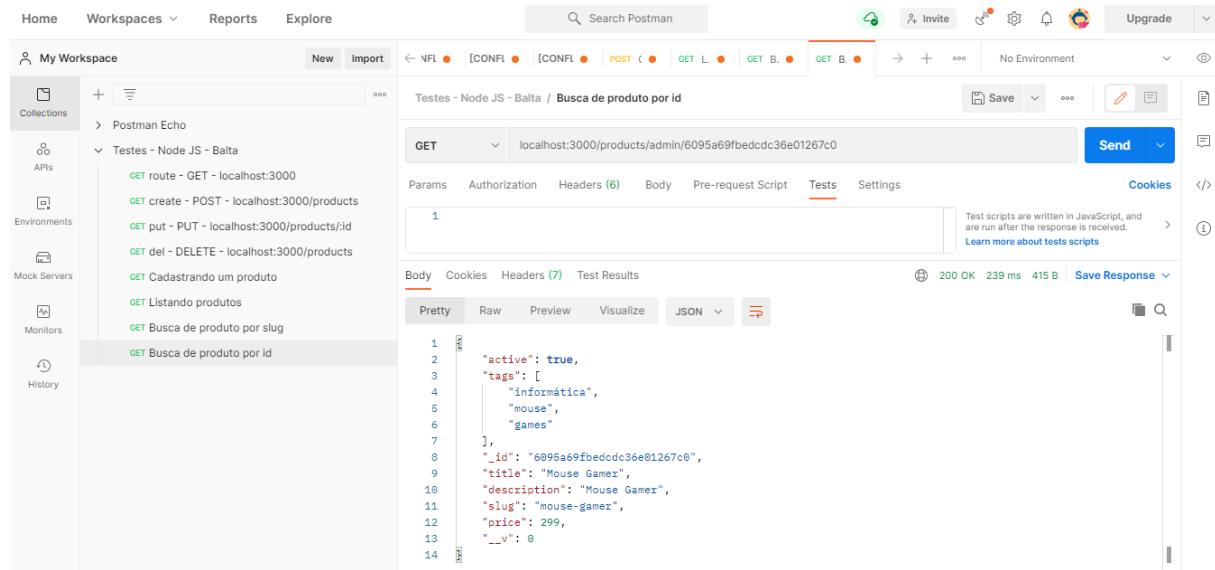
```
const express = require('express');
const router = express.Router();
const controller = require("../controllers/product-controller");

router.get('/', controller.get);
router.get('/:slug', controller.getBySlug);
router.get('/admin/:id', controller.getId);
router.post('/', controller.post);
router.put('/:id', controller.put);
router.delete('/', controller.delete);

module.exports = router;
```

Testando no Postman

localhost:3000/products/admin/6095a69fbedcdc36e01267c0



The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing collections like 'Postman Echo' and 'Testes - Node JS - Balta'. Under 'Testes - Node JS - Balta', several API endpoints are listed: 'GET route - GET - localhost:3000', 'GET create - POST - localhost:3000/products', 'GET put - PUT - localhost:3000/products/:id', 'GET del - DELETE - localhost:3000/products', 'GET Cadastrando um produto', 'GET Listando produtos', 'GET Busca de produto por slug', and 'GET Busca de produto por id'. The main area shows a 'Testes - Node JS - Balta / Busca de produto por id' collection. A 'GET' request is selected with the URL 'localhost:3000/products/admin/6095a69fbedcdc36e01267c0'. The 'Tests' tab is active, showing a single test script:

```
1
```

Test scripts are written in JavaScript, and are run after the response is received.[Learn more about tests scripts](#)

The 'Body' tab shows the response body in JSON format:

```
1
2   "active": true,
3   "tags": [
4     "informática",
5     "mouse",
6     "games"
7   ],
8   "_id": "6095a69fbedcdc36e01267c0",
9   "title": "Mouse Gamer",
10  "description": "Mouse Gamer",
11  "slug": "mouse-gamer",
12  "price": 299,
13  "__v": 0
14 }
```

Testando no navegador

http://localhost:3000/products/admin/6095a69fbedcdc36e01267c0



The screenshot shows a browser window with the address bar containing 'localhost:3000/products/admin/6095a69fbedcdc36e01267c0'. The page content is a JSON object, identical to the one shown in the Postman screenshot:

```
1   // 20210507184428
2   // http://localhost:3000/products/admin/6095a69fbedcdc36e01267c0
3
4   {
5     "active": true,
6     "tags": [
7       "informática",
8       "mouse",
9       "games"
10    ],
11    "_id": "6095a69fbedcdc36e01267c0",
12    "title": "Mouse Gamer",
13    "description": "Mouse Gamer",
14    "slug": "mouse-gamer",
15    "price": 299,
16    "__v": 0
17  }
```

Aula 20 - Listando os produtos de uma tag

src/controllers/product-controller.js

```
'use strict';

const mongoose = require('mongoose');
const Product = mongoose.model('Product');

exports.get = (req, res, next) => {
  Product.find({ active: true }, 'title price slug').then(data => {
    res.status(200).send(data);
  }).catch(e => {
    res.status(400).send(e);
  });
};

exports.getBySlug = (req, res, next) => {
  Product.findOne({ slug: req.params.slug, active: true }, 'title description price
slug tags').then(data => {
    res.status(200).send(data);
  }).catch(e => {
    res.status(400).send(e);
  });
};

exports.getById = (req, res, next) => {
  Product.findById({ _id: req.params.id }).then(data => {
    res.status(200).send(data);
  }).catch(e => {
    res.status(400).send(e);
  });
};

exports.getByTag = (req, res, next) => {
  Product.find({tags: req.params.tag, active: true}, 'title description price slug
tags').then(data => {
    res.status(200).send(data);
  }).catch(e => {
    res.status(400).send(e);
  });
};

exports.post = (req, res, next) => {
  var product = new Product(req.body);
  product.save().then(x => {
    res.status(201).send({message: 'Produto cadastrado com sucesso!'});
  }).catch(e => {
    res.status(400).send({message: 'Falha ao cadastrar o produto!', data: e});
  });
};
```

```

    });

};

exports.put = (req, res, next) => {
  const id = req.params.id;
  res.status(200).send({
    id: id,
    item: req.body
  });
};

exports.delete = (req, res, next) => {
  res.status(200).send(req.body);
};

```

src/routes/product-route.js

```

const express = require('express');
const router = express.Router();
const controller = require("../controllers/product-controller");

router.get('/', controller.get);
router.get('/:slug', controller.getBySlug);
router.get('/admin/:id', controller.getId);
router.get('/tags/:tag', controller.getByTag);
router.post('/', controller.post);
router.put('/:id', controller.put);
router.delete('/', controller.delete);

module.exports = router;

```

Testando no Postman

GET - localhost:3000/products/tags/games

The screenshot shows the Postman application interface. On the left, there's a sidebar with sections for Collections, APIs, Environments, Mock Servers, Monitors, and History. In the main area, a collection named "Tests - Node JS - Balta" is selected. Under it, a specific test case is shown with the URL "localhost:3000/products/tags/games". The "Body" tab is selected, displaying the JSON response from the API. The response object has a "tags" key with an array value: ["informática", "mouse", "games"]. Other keys in the object include "_id", "title", "description", "slug", and "price". The status bar at the bottom indicates a 200 OK status with a time of 516 ms.

```

{
  "tags": [
    "informática",
    "mouse",
    "games"
  ],
  "_id": "605a69fbedcdc36e01267c0",
  "title": "Mouse Gamer",
  "description": "Mouse Gamer",
  "slug": "mouse-gamer",
  "price": 299
}

```

Aula 21 - Atualizando um produto

src/controllers/product-controller.js

```
'use strict';

const mongoose = require('mongoose');
const Product = mongoose.model('Product');

exports.get = (req, res, next) => {
  Product.find({ active: true }, 'title price slug').then(data => {
    res.status(200).send(data);
  }).catch(e => {
    res.status(400).send(e);
  });
};

exports.getBySlug = (req, res, next) => {
  Product.findOne({ slug: req.params.slug, active: true }, 'title description price slug
tags').then(data => {
    res.status(200).send(data);
  }).catch(e => {
    res.status(400).send(e);
  });
};

exports.getById = (req, res, next) => {
  Product.findById({_id: req.params.id}).then(data => {
    res.status(200).send(data);
  }).catch(e => {
    res.status(400).send(e);
  });
};

exports.getTag = (req, res, next) => {
  Product.find({tags: req.params.tag, active: true}, 'title description price slug
tags').then(data => {
    res.status(200).send(data);
  }).catch(e => {
    res.status(400).send(e);
  });
};

exports.post = (req, res, next) => {
  var product = new Product(req.body);
  product.save().then(x => {
    res.status(201).send({message: 'Produto cadastrado com sucesso!'});
  }).catch(e => {
    res.status(400).send({message: 'Falha ao cadastrar o produto!', data: e });
  });
};
```

```

exports.put = (req, res, next) => {
  Product.findByIdAndUpdate(req.params.id, {
    $set: {
      title: req.body.title,
      description: req.body.description,
      slug: req.body.slug,
      price: req.body.price
    }
  }).then(x => {
    res.status(200).send({
      message: "Produto atualizado com sucesso!"
    });
  }).catch(e => {
    res.status(400).send({
      message: "Falha ao atualizar produto!", data: e
    });
  });
};

exports.delete = (req, res, next) => {
  res.status(200).send(req.body);
};

```

Testando no Postman

PUT - localhost:3000/products/6095a69fbedcdc36e01267c0

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing collections like 'Postman Echo' and 'Testes - Node JS - Balta'. Under 'Testes - Node JS - Balta', there are several test cases: 'GET route - GET - localhost:3000', 'GET create - POST - localhost:3000...', 'GET put - PUT - localhost:3000/pro...', 'GET del - DELETE - localhost:3000/...', 'GET Cadastrando um produto', 'GET Listando produtos', 'GET Busca de produto por slug', 'GET Busca de produto por id', 'GET Busca de produto por tag', and 'GET Atualizando produto'. The 'GET Atualizando produto' test case is currently selected.

The main workspace shows a 'PUT' request to 'localhost:3000/products/6095a69fbedcdc36e01267c0'. The 'Body' tab is selected, showing a JSON payload:

```

1   {
2     ... "title": "Cadeira Gamer",
3     ... "description": "Cadeira Gamer",
4     ... "slug": "cadeira-gamer",
5     ... "price": 1299
6   }

```

Below the body, the response is shown in 'Pretty' format:

```

1   {
2     ... "message": "Produto atualizado com sucesso!"
3   }

```

The status bar at the bottom indicates 'Status: 200 OK Time: 265 ms Size: 280 B'.

No Studio 3T

Result		
Query Code		
Explain		
< < > >> 50 Documents 1 to 1		
Key	Value	Type
↳ (1) {_id : 5d5bb12ffc3f890d988dcb1c}	{ 8 fields }	Document
↳ _id	5d5bb12ffc3f890d988dcb1c	ObjectId
↳ active	true	Bool
↳ tags	[3 elements]	Array
↳ title	Cadeira Gamer	String
↳ description	Cadeira Gamer	String
↳ slug	cadeira-gamer	String
↳ price	1299	Int32
↳ __v	0	Int32

No mlab

DATABASES: 1 COLLECTIONS: 1

+ Create Database

node-store-db.products

COLLECTION SIZE: 188B TOTAL DOCUMENTS: 1 INDEXES TOTAL SIZE: 56KB

Find Indexes Schema Anti-Patterns Search Indexes

FILTER {"filter": "example"}

QUERY RESULTS 1-1 OF 1

```
_id: ObjectId("6095a69fbedcdc36e01267c0")
active: true
tags: Array
title: "Cadeira Gamer"
description: "Cadeira Gamer"
slug: "cadeira-gamer"
price: 1299
__v: 0
```

Aula 22 - Excluindo um produto

src/controllers/product-controller.js

```
'use strict';

const mongoose = require('mongoose');
const Product = mongoose.model('Product');

exports.get = (req, res, next) => {
  Product.find({ active: true }, 'title price slug').then(data => {
    res.status(200).send(data);
  }).catch(e => {
    res.status(400).send(e);
  });
};

exports.getBySlug = (req, res, next) => {
  Product.findOne({ slug: req.params.slug, active: true }, 'title description price slug
tags').then(data => {
    res.status(200).send(data);
  }).catch(e => {
    res.status(400).send(e);
  });
};

exports.getById = (req, res, next) => {
  Product.findById({_id: req.params.id}).then(data => {
    res.status(200).send(data);
  }).catch(e => {
    res.status(400).send(e);
  });
};

exports.getTag = (req, res, next) => {
  Product.find({tags: req.params.tag, active: true}, 'title description price slug
tags').then(data => {
    res.status(200).send(data);
  }).catch(e => {
    res.status(400).send(e);
  });
};

exports.post = (req, res, next) => {
  var product = new Product(req.body);
  product.save().then(x => {
    res.status(201).send({message: 'Produto cadastrado com sucesso!'});
  }).catch(e => {
    res.status(400).send({message: 'Falha ao cadastrar o produto!', data: e });
  });
};
```

```

exports.put = (req, res, next) => {
  Product.findByIdAndUpdate(req.params.id, {
    $set: {
      title: req.body.title,
      description: req.body.description,
      slug: req.body.slug,
      price: req.body.price
    }
  }).then(x => {
    res.status(200).send({
      message: "Produto atualizado com sucesso!"
    });
  }).catch(e => {
    res.status(400).send({
      message: "Falha ao atualizar produto!", data: e
    });
  });
};

exports.delete = (req, res, next) => {
  Product.findOneAndRemove(req.params.id).then(x => {
    res.status(200).send({
      message: "Produto removido com sucesso!"
    });
  }).catch(e => {
    res.status(400).send({
      message: "Falha ao remover produto!", data: e
    });
  });
};

```

src/routes/product-route.js

```

const express = require('express');
const router = express.Router();
const controller = require("../controllers/product-controller");

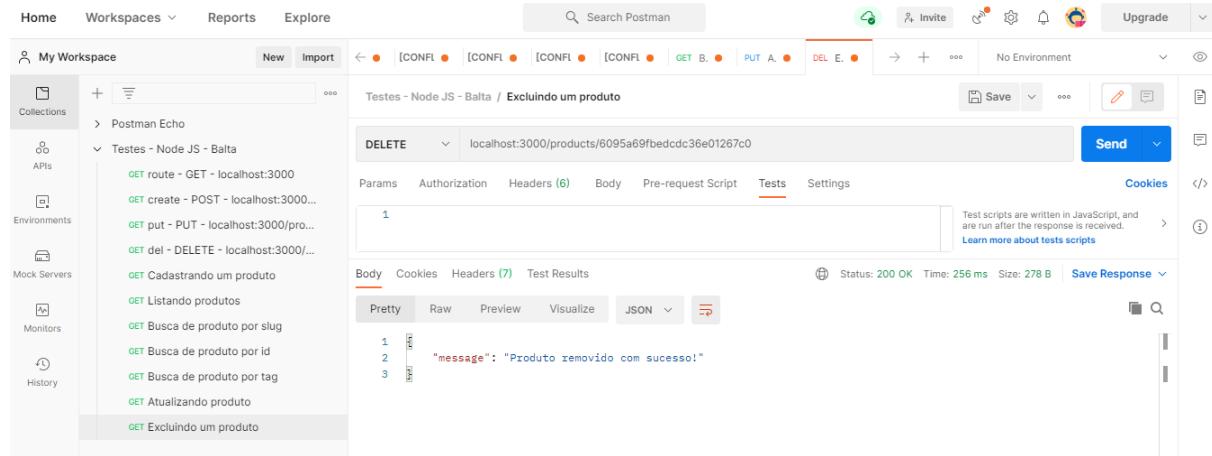
router.get('/', controller.get);
router.get('/:slug', controller.getBySlug);
router.get('/admin/:id', controller.getId);
router.get('/tags/:tag', controller.getTag);
router.post('/', controller.post);
router.put('/:id', controller.put);
router.delete('/:id', controller.delete);

module.exports = router;

```

Testando no Postman

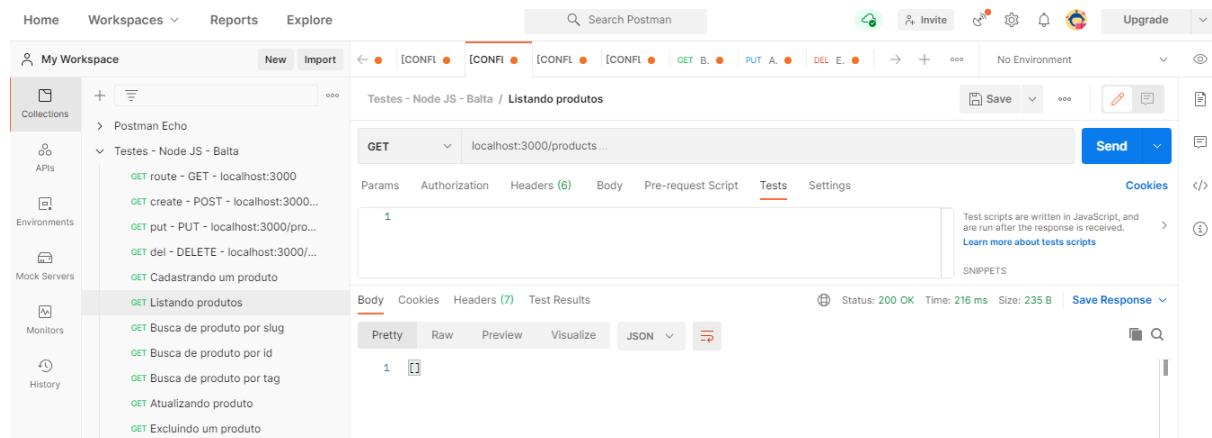
localhost:3000/products/6095a69fbedcdc36e01267c0



The screenshot shows the Postman interface with a collection named "Testes - Node JS - Balta". A DELETE request is made to `localhost:3000/products/6095a69fbedcdc36e01267c0`. The response body contains the message: "message": "Produto removido com sucesso!".

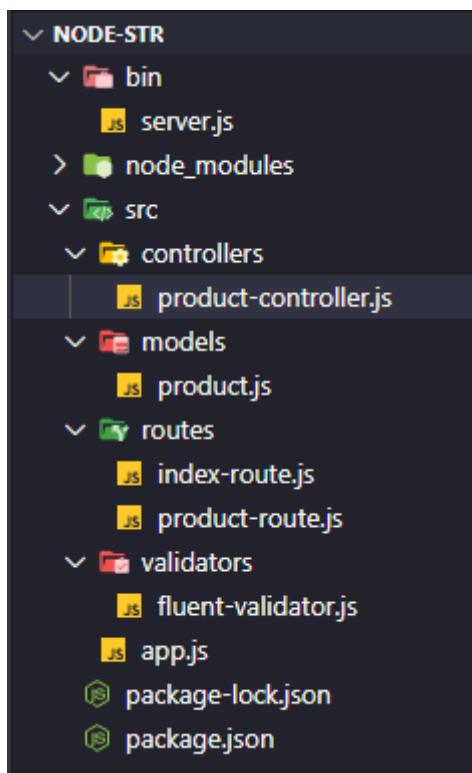
- Agora, ao listar os produtos:

GET - localhost:3000/products



The screenshot shows the Postman interface with a collection named "Testes - Node JS - Balta". A GET request is made to `localhost:3000/products`. The response body is an empty JSON array: `[]`.

Aula 23 - Validações



srcValidators/fluent-validator.js

```
'use strict';

let errors = [];

function ValidationContract() {
    errors = [];
}

ValidationContract.prototype.isRequired = (value, message) => {
    if (!value || value.length <= 0)
        errors.push({ message: message });
}

ValidationContract.prototype.hasMinLen = (value, min, message) => {
    if (!value || value.length < min)
        errors.push({ message: message });
}

ValidationContract.prototype.hasMaxLen = (value, max, message) => {
    if (!value || value.length > max)
        errors.push({ message: message });
}

ValidationContract.prototype.isFixedLen = (value, len, message) => {
    if (value.length != len)
        errors.push({ message: message });
}

ValidationContract.prototype.isEmail = (value, message) => {
    var reg = new RegExp(/^\w+([-+.']\w+)*@\w+([.-]\w+)*\.\w+([.-]\w+)*$/);
    if (!reg.test(value))
        errors.push({ message: message });
}

ValidationContract.prototype.errors = () => {
    return errors;
}

ValidationContract.prototype.clear = () => {
    errors = [];
}

ValidationContract.prototype.isValid = () => {
    return errors.length == 0;
}

module.exports = ValidationContract;
```

src/controllers/product-controller.js

```
'use strict';

const mongoose = require('mongoose');
const Product = mongoose.model('Product');
const ValidationContract = require('../validators/fluent-validator');

exports.get = (req, res, next) => {
    Product.find({ active: true }, 'title price slug').then(data => {
        res.status(200).send(data);
    }).catch(e => {
        res.status(400).send(e);
    });
};

exports.getBySlug = (req, res, next) => {
    Product.findOne({ slug: req.params.slug, active: true }, 'title description price slug
tags').then(data => {
        res.status(200).send(data);
    }).catch(e => {
        res.status(400).send(e);
    });
};

exports.getById = (req, res, next) => {
    Product.findById({_id: req.params.id}).then(data => {
        res.status(200).send(data);
    }).catch(e => {
        res.status(400).send(e);
    });
};

exports getByTag = (req, res, next) => {
    Product.find({tags: req.params.tag, active: true}, 'title description price slug
tags').then(data => {
        res.status(200).send(data);
    }).catch(e => {
        res.status(400).send(e);
    });
};

exports.post = (req, res, next) => {

    let contract = new ValidationContract();
    contract.hasMinLen(req.body.title, 3, 'O título deve conter pelo menos 3 caracteres');
    contract.hasMinLen(req.body.slug, 3, 'O slug deve conter pelo menos 3 caracteres');
    contract.hasMinLen(req.body.description, 3, 'A descrição deve conter pelo menos 3
caracteres');

    // Se os dados forem inválidos
    if (!contract.isValid()) {
        res.status(400).send(contract.errors()).end();
        return;
    }
}
```

```

var product = new Product(req.body);
product.save().then(x => {
    res.status(201).send({message: 'Produto cadastrado com sucesso!'});
}).catch(e => {
    res.status(400).send({message: 'Falha ao cadastrar o produto!', data: e });
});
};

exports.put = (req, res, next) => {
    Product.findByIdAndUpdate(req.params.id, {
        $set: {
            title: req.body.title,
            description: req.body.description,
            slug: req.body.slug,
            price: req.body.price
        }
    }).then(x => {
        res.status(200).send({
            message: "Produto atualizado com sucesso!"
        });
    }).catch(e => {
        res.status(400).send({
            message: "Falha ao atualizar produto!", data: e
        });
    });
};

exports.delete = (req, res, next) => {
    Product.findOneAndRemove(req.body.id).then(x => {
        res.status(200).send({
            message: "Produto removido com sucesso!"
        });
    }).catch(e => {
        res.status(400).send({
            message: "Falha ao remover produto!", data: e
        });
    });
};

```

nodemon ./bin/server.js

```

C:\balta\nodejs\node-str>nodemon ./bin/server.js
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node ./bin/server.js`
API rodando na porta 3000
(node:9048) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the
new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:9048) [MONGODB DRIVER] Warning: Top-level use of w, wtimeout, j, and fsync is deprecated. Use writeConcern instead.
(node:9048) [MONGODB DRIVER] Warning: Current Server Discovery and Monitoring engine is deprecated, and will be removed in a
future version. To use the new Server Discover and Monitoring engine, pass option { useUnifiedTopology: true } to the MongoCl
ient constructor.
(node:9048) DeprecationWarning: collection.ensureIndex is deprecated. Use createIndexes instead.

```

Testando no Postman

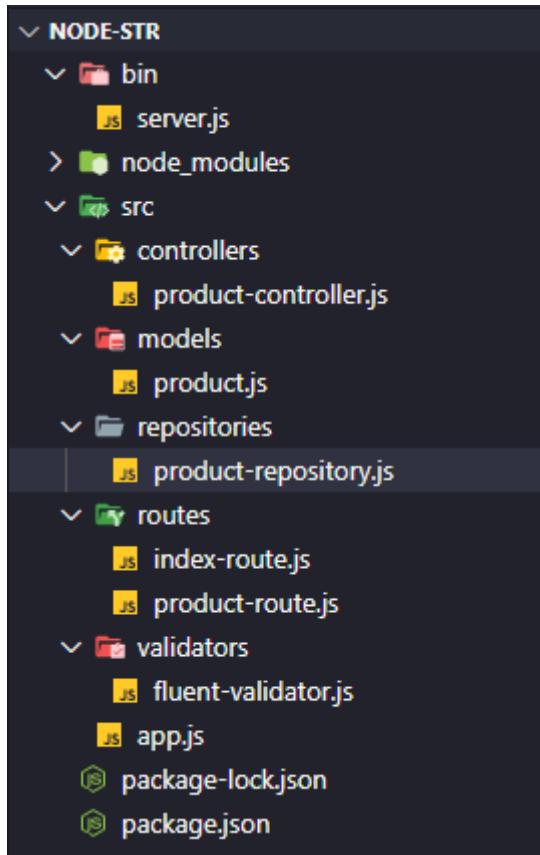
The screenshot shows the Postman application interface. On the left, there's a sidebar with options like Home, Workspaces, Reports, Explore, Collections, APIs, Environments, Mock Servers, Monitors, and History. The main area displays a collection named "Testes - Node JS - Balta". Inside this collection, there are several test cases: "Postman Echo", "Testes - Node JS - Balta", and others related to product management. The "Testes - Node JS - Balta" test case is currently selected and expanded. It contains a POST request to "localhost:3000/products". The request body is set to "Body" (JSON) and contains the following JSON payload:

```
1 [ 2   { 3     "title": "", 4     "description": "ex", 5     "slug": "", 6     "price": 299, 7   }]
```

The response tab shows a 400 Bad Request status with the following error messages:

```
1 [ 2   { 3     "message": "O título deve conter pelo menos 3 caracteres", 4   }, 5   { 6     "message": "O slug deve conter pelo menos 3 caracteres", 7   }, 8   { 9     "message": "A descrição deve conter pelo menos 3 caracteres", 10    }]
```

Aula 24 - Reppositórios



src/repositories/product-repository.js

```
'use strict';
const mongoose = require('mongoose');
const Product = mongoose.model('Product');

exports.get = () => {
  return Product.find({
    active: true
  }, 'title price slug');
}

exports.getBySlug = (slug) => {
  return Product
    .findOne({
      slug: slug,
      active: true
    }, 'title description price slug tags');
}

exports.getById = (id) => {
  return Product
    .findById(id);
}
```

```
exports.getTag = (tag) => {
  return Product
    .find({
      tags: tag,
      active: true
    }, 'title description price slug tags');
}

exports.create = (data) => {
  var product = new Product(data);
  return product.save();
}

exports.update = (id, data) => {
  return Product
    .findByIdAndUpdate(id, {
      $set: {
        title: data.title,
        description: data.description,
        price: data.price,
        slug: data.slug
      }
    });
}

exports.delete = (id) => {
  return Product
    .findOneAndRemove(id);
}
```

src/controllers/product-controller.js

```
'use strict';

const mongoose = require('mongoose');
const Product = mongoose.model('Product');
const ValidationContract = require('../validators/fluent-validator');
const repository = require('../repositories/product-repository');

exports.get = (req, res, next) => {
    repository
        .get()
        .then(data => {
            res.status(200).send(data);
        }).catch(e => {
            res.status(400).send(e);
        });
};

exports.getBySlug = (req, res, next) => {
    repository
        .getBySlug(req.params.slug)
        .then(data => {
            res.status(200).send(data);
        }).catch(e => {
            res.status(400).send(e);
        });
};

exports.getById = (req, res, next) => {
    repository
        .getById(req.params.id)
        .then(data => {
            res.status(200).send(data);
        }).catch(e => {
            res.status(400).send(e);
        });
};

exports getByTag = (req, res, next) => {
    repository
        .getByTag(req.params.tag)
        .then(data => {
            res.status(200).send(data);
        }).catch(e => {
            res.status(400).send(e);
        });
};
```

```
exports.post = (req, res, next) => {

  let contract = new ValidationContract();
  contract.hasMinLen(req.body.title, 3, 'O título deve conter pelo menos 3 caracteres');
  contract.hasMinLen(req.body.slug, 3, 'O slug deve conter pelo menos 3 caracteres');
  contract.hasMinLen(req.body.description, 3, 'A descrição deve conter pelo menos 3 caracteres');

  // Se os dados forem inválidos
  if (!contract.isValid()) {
    res.status(400).send(contract.errors()).end();
    return;
  }

  repository
  .create(req.body)
  .then(x => {
    res.status(201).send({message: 'Produto cadastrado com sucesso!'});
  }).catch(e => {
    res.status(400).send({message: 'Falha ao cadastrar o produto!', data: e});
  });
};

exports.put = (req, res, next) => {
  repository
  .update(req.params.id, req.body)
  .then(x => {
    res.status(200).send({
      message: "Produto atualizado com sucesso!"
    });
  }).catch(e => {
    res.status(400).send({
      message: "Falha ao atualizar produto!", data: e
    });
  });
};

exports.delete = (req, res, next) => {
  repository
  .delete(req.body.id)
  .then(x => {
    res.status(200).send({
      message: "Produto removido com sucesso!"
    });
  }).catch(e => {
    res.status(400).send({
      message: "Falha ao remover produto!", data: e
    });
  });
};
```

```
nodemon ./bin/server.js
```

```
C:\balta\nodejs\node-str>nodemon ./bin/server.js
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node ./bin/server.js`
API rodando na porta 3000
(node:2428) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:2428) [MONGODB DRIVER] Warning: Top-level use of w, wtimeout, j, and fsync is deprecated. Use writeConcern instead.
(node:2428) [MONGODB DRIVER] Warning: Current Server Discovery and Monitoring engine is deprecated, and will be removed in a future version. To use the new Server Discover and Monitoring engine, pass option { useUnifiedTopology: true } to the MongoClient constructor.
(node:2428) DeprecationWarning: collection.ensureIndex is deprecated. Use createIndexes instead.
```

Cadastrando um produto

POST - localhost:3000/products

The screenshot shows the Postman interface with the following details:

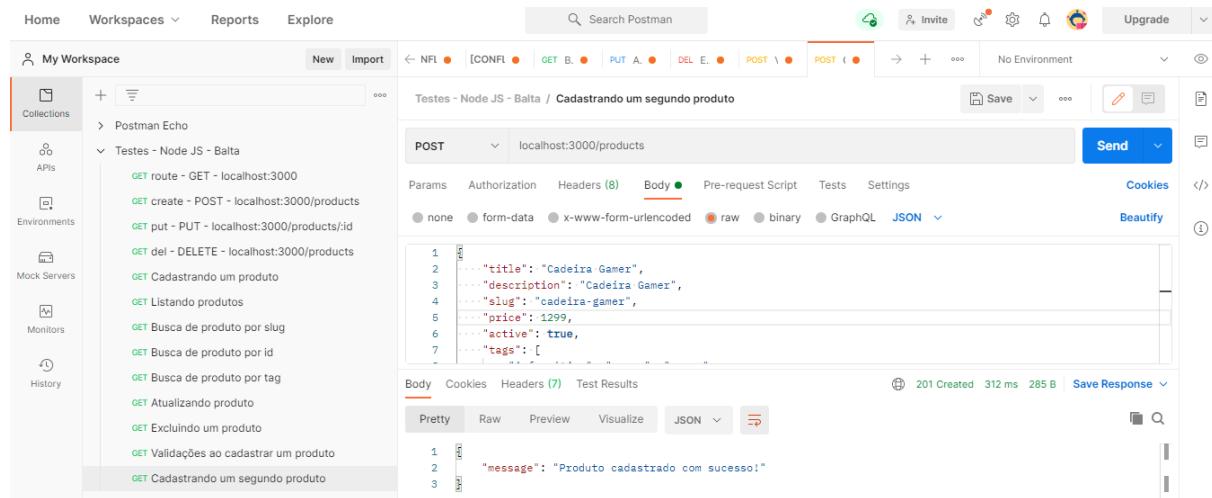
- Collection:** Testes - Node JS - Balta / Cadastrando um produto
- Method:** POST
- URL:** localhost:3000/products
- Body (JSON):**

```
1 {
2   "title": "Mouse Gamer",
3   "description": "Mouse Gamer",
4   "slug": "mouse-gamer",
5   "price": 299,
6   "active": true,
7   "tags": [
8     "informática", "mouse", "games"
9   ]
10 }
```
- Response Headers:** 201 Created, 901 ms, 285 B
- Response Body (Pretty):**

```
1   "message": "Produto cadastrado com sucesso!"
```

Cadastrando um segundo produto

POST - localhost:3000/products



POST localhost:3000/products

```
1
2 ...
3   "title": "Cadeira Gamer",
4   "description": "Cadeira Gamer",
5   "slug": "cadeira-gamer",
6   "price": 1299,
7   "active": true,
8   "tags": [ ]
```

Body Cookies Headers (7) Test Results

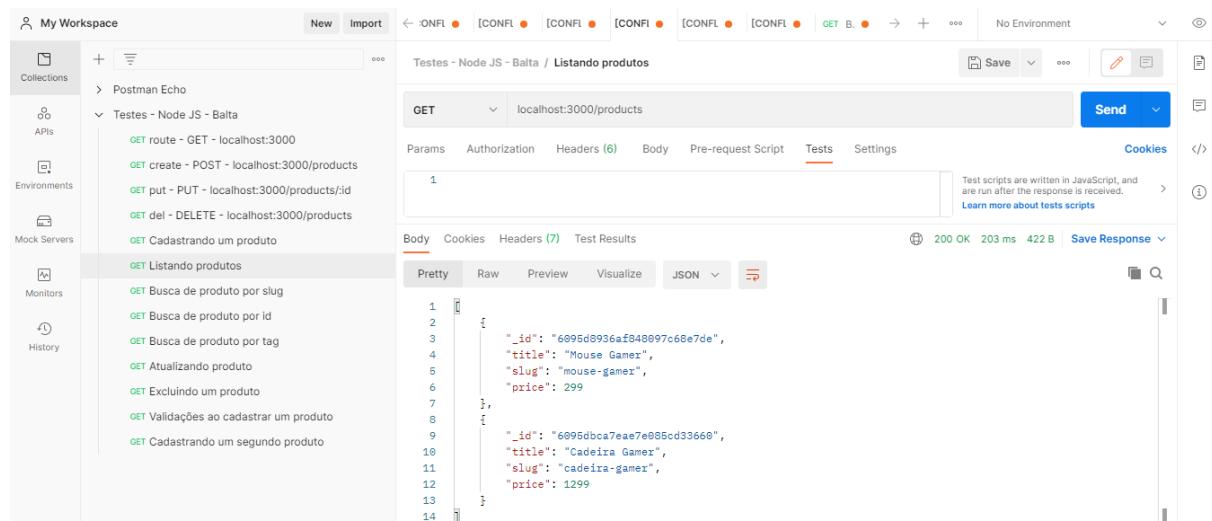
Pretty Raw Preview Visualize JSON

1 201 Created 312 ms 285 B Save Response

```
1 ...
2   "message": "Produto cadastrado com sucesso!"
```

Listando os produtos

GET - localhost:3000/products



GET localhost:3000/products

```
1
```

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

1 200 OK 203 ms 422 B Save Response

```
1 ...
2   {
3     "_id": "6095d8936af848097c68e7de",
4     "title": "Mouse Gamer",
5     "slug": "mouse-gamer",
6     "price": 299
7   },
8   {
9     "_id": "6095dbca7eae7e085cd33660",
10    "title": "Cadeira Gamer",
11    "slug": "cadeira-gamer",
12    "price": 1299
13 }
```

Exibindo dados de um produto por slug

GET - localhost:3000/products/mouse-gamer

The screenshot shows the Postman interface with a collection named 'Testes - Node JS - Balta'. A specific test case is selected, showing a GET request to 'localhost:3000/products/mouse-gamer'. The response body is displayed in JSON format:

```
1
2   "tags": [
3     "informática",
4     "mouse",
5     "games"
6   ],
7   "_id": "6095d8936af848097c68e7de",
8   "title": "Mouse Gamer",
9   "description": "Mouse Gamer",
10  "slug": "mouse-gamer",
11  "price": 299
```

Exibindo dados de um produto por id

localhost:3000/products/admin/6095d8936af848097c68e7de

The screenshot shows the Postman interface with the same collection. A different test case is selected, showing a GET request to 'localhost:3000/products/admin/6095d8936af848097c68e7de'. The response body is displayed in JSON format:

```
1
2   "active": true,
3   "tags": [
4     "informática",
5     "mouse",
6     "games"
7   ],
8   "_id": "6095d8936af848097c68e7de",
9   "title": "Mouse Gamer",
10  "description": "Mouse Gamer",
11  "slug": "mouse-gamer",
12  "price": 299,
13  "__v": 0
```

Exibindo produtos por tag

GET - localhost:3000/products/tags/games

```
[{"_id": "6095d8936af848097c68e7de", "title": "Mouse Gamer", "description": "Mouse Gamer", "slug": "mouse-gamer", "price": 299, "tags": ["inform\u00e1tica", "mouse", "games"]}, {"_id": "6095dbca7eae7e085cd33660", "title": "Cadeira Gamer", "description": "Cadeira Gamer", "slug": "cadeira-gamer", "price": 1299, "tags": ["inform\u00e1tica", "mouse", "games"]}]
```

Atualizando dados de um produto

localhost:3000/products/6095dbca7eae7e085cd33660

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing collections like 'Postman Echo' and 'Testes - Node JS - Balta'. The main area shows a 'PUT' request to 'localhost:3000/products/6095dbca7eae7e085cd33660'. The 'Body' tab is selected, displaying the following JSON payload:

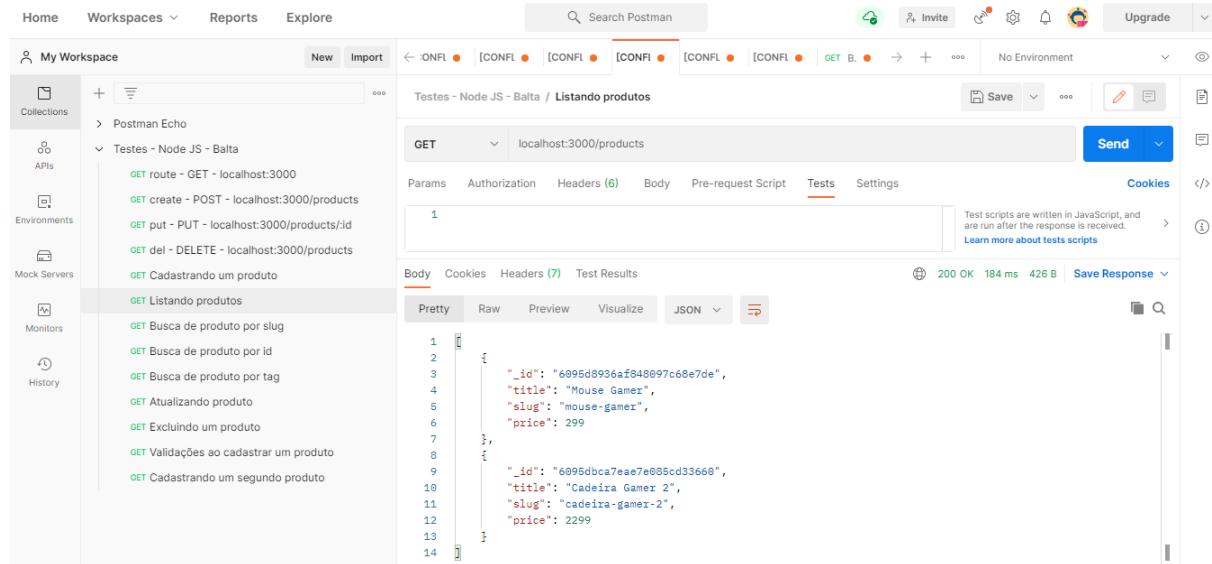
```
1 ... "title": "Cadeira Gamer 2",
2 ... "description": "Cadeira Gamer 2",
3 ... "slug": "cadeira-gamer-2",
4 ... "price": 2299
```

Below the body, the response status is 200 OK with a response time of 196 ms and a size of 280 B. The response body is shown as:

```
1 ...
2 ...
3 ... "message": "Produto atualizado com sucesso!"
```

Listando os produtos

GET - localhost:3000/products

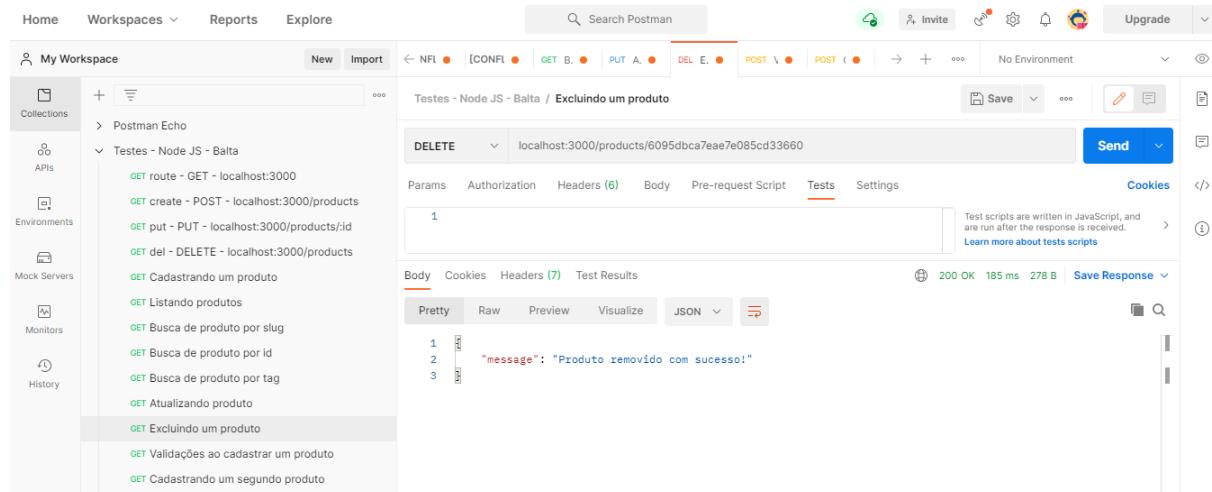


The screenshot shows the Postman interface with a collection named 'Testes - Node JS - Balta'. A test case titled 'GET Listando produtos' is selected. The request method is 'GET' and the URL is 'localhost:3000/products'. The response status is 200 OK, and the response body is a JSON array containing two products:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
{
  "_id": "6095d8936af848097c68e7de",
  "title": "Mouse Gamer",
  "slug": "mouse-gamer",
  "price": 299
},
{
  "_id": "6095dbca7eae7e085cd33660",
  "title": "Cadeira Gamer 2",
  "slug": "cadeira-gamer-2",
  "price": 2299
}
```

Excluindo um produto

DELETE - localhost:3000/products/5d5c6627cc81b832f8f37dd9



The screenshot shows the Postman interface with the same collection and test case. The request method is 'DELETE' and the URL is 'localhost:3000/products/6095dbca7eae7e085cd33660'. The response status is 200 OK, and the response body is a JSON object with a single key 'message': "Produto removido com sucesso!"

```
1
2
3
{
  "message": "Produto removido com sucesso!"
}
```

Listando os produtos

GET - localhost:3000/products

The screenshot shows the Postman application interface. On the left, there's a sidebar with sections for Collections, APIs, Environments, Mock Servers, Monitors, and History. The main area displays a collection named "Testes - Node JS - Balta". Inside this collection, there is a folder named "Listando produtos" which contains a single item: "GET Listando produtos". This item has a status of "200 OK" and a response time of "179 ms". The response body is shown in JSON format:

```
1
2
3
4
5
6
7
8
{
  "_id": "6095dbca7eae7e085cd33660",
  "title": "Cadeira Gamer 2",
  "slug": "cadeira-gamer-2",
  "price": 2299
}
```

Aula 25 - Async / Await

src/repositories/product-repository.js

```
'use strict';
const mongoose = require('mongoose');
const Product = mongoose.model('Product');

exports.get = async() => {
  const res = await Product.find({
    active: true
  }, 'title price slug');
  return res;
}

exports.getBySlug = async(slug) => {
  const res = await Product
    .findOne({
      slug: slug,
      active: true
    }, 'title description price slug tags');
  return res;
}

exports.getById = async(id) => {
  const res = await Product
    .findById(id);
  return res;
}

exports.getByTag = async(tag) => {
  const res = Product
    .find({
      tags: tag,
      active: true
    }, 'title description price slug tags');
  return res;
}

exports.create = async(data) => {
  var product = new Product(data);
  await product.save();
}

exports.update = async(id, data) => {
  await Product
    .findByIdAndUpdate(id, {
      $set: {
        title: data.title,
        description: data.description,
        price: data.price,
        slug: data.slug
      }
    });
}
```

```
exports.delete = async(id) => {
  await Product
    .findOneAndRemove(id);
}
```

src/controllers/product-controller.js

```
'use strict';

const ValidationContract = require('../validators/fluent-validator');
const repository = require('../repositories/product-repository');

exports.get = async(req, res, next) => {
  try {
    var data = await repository.get();
    res.status(200).send(data);
  } catch (e) {
    res.status(500).send({
      message: 'Falha ao processar sua requisição'
    });
  }
}

exports.getBySlug = async(req, res, next) => {
  try {
    var data = await repository.getBySlug(req.params.slug);
    res.status(200).send(data);
  } catch (e) {
    res.status(500).send({
      message: 'Falha ao processar sua requisição'
    });
  }
}

exports.getById = async(req, res, next) => {
  try {
    var data = await repository.getById(req.params.id);
    res.status(200).send(data);
  } catch (e) {
    res.status(500).send({
      message: 'Falha ao processar sua requisição'
    });
  }
}

exports.getTag = async(req, res, next) => {
  try {
    const data = await repository.getTag(req.params.tag);
    res.status(200).send(data);
  } catch (e) {
    res.status(500).send({
      message: 'Falha ao processar sua requisição'
    });
  }
}
```

```
}

exports.post = async(req, res, next) => {
  let contract = new ValidationContract();
  contract.hasMinLen(req.body.title, 3, 'O título deve conter pelo menos 3 caracteres');
  contract.hasMinLen(req.body.slug, 3, 'O título deve conter pelo menos 3 caracteres');
  contract.hasMinLen(req.body.description, 3, 'O título deve conter pelo menos 3
caracteres');

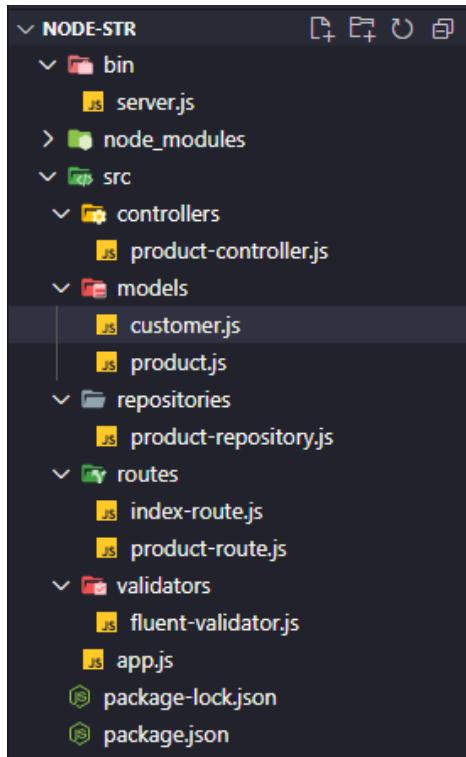
  // Se os dados forem inválidos
  if (!contract.isValid()) {
    res.status(400).send(contract.errors()).end();
    return;
  }

  try {
    await repository.create(req.body);
    res.status(201).send({
      message: 'Produto cadastrado com sucesso!'
    });
  } catch (e) {
    console.log(e);
    res.status(500).send({
      message: 'Falha ao processar sua requisição'
    });
  }
};

exports.put = async(req, res, next) => {
  try {
    await repository.update(req.params.id, req.body);
    res.status(200).send({
      message: 'Produto atualizado com sucesso!'
    });
  } catch (e) {
    res.status(500).send({
      message: 'Falha ao processar sua requisição'
    });
  }
};

exports.delete = async(req, res, next) => {
  try {
    await repository.delete(req.body.id)
    res.status(200).send({
      message: 'Produto removido com sucesso!'
    });
  } catch (e) {
    res.status(500).send({
      message: 'Falha ao processar sua requisição'
    });
  }
};
```

Aula 26 - Revisitando os Models: Customer



src/models/customer.js

```
'use strict';

const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const schema = new Schema({
  name: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true
  },
  password: {
    type: String,
    required: true
  }
});

module.exports = mongoose.model('Customer', schema);
```

src/app.js

```
const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');

const app = express();
const router = express.Router();

// Conecta ao banco
mongoose.connect("mongodb+srv://betopinheiro1005:angstron1005@node-
store-cluster-nlcnv.mongodb.net/node-str-db?retryWrites=true&w=majority", {
useNewUrlParser: true });

// Carrega os models
const Product = require('./models/product');
const Customer = require('./models/customer');

// Carrega as rotas
const indexRoute = require('./routes/index-route');
const productRoute = require('./routes/product-route');

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({
  extended: false
}));

app.use('/', indexRoute);
app.use('/products', productRoute);

module.exports = app;
```

Aula 27 - Revisitando os Models: Order

src/models/order.js

```
'use strict';

const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const schema = new Schema({
  customer: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Customer'
  },
  number: {
    type: String,
    required: true
  },
  createDate: {
    type: Date,
    required: true,
    default: Date.now
  },
  status: {
    type: String,
    required: true,
    enum: ['created', 'done'],
    default: 'created'
  },
  items: [
    {
      quantity: {
        type: Number,
        required: true,
        default: 1
      },
      price: {
        type: Number,
        required: true
      },
      product: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Product'
      }
    }
  ],
});

module.exports = mongoose.model('Order', schema);
```

src/app.js

```
const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');

const app = express();
const router = express.Router();

// Conecta ao banco
mongoose.connect("mongodb+srv://betopinheiro1005:angstron1005@node-store-cluster-nlcnv.mongodb.net/node-str-db?retryWrites=true&w=majority", {
useNewUrlParser: true });

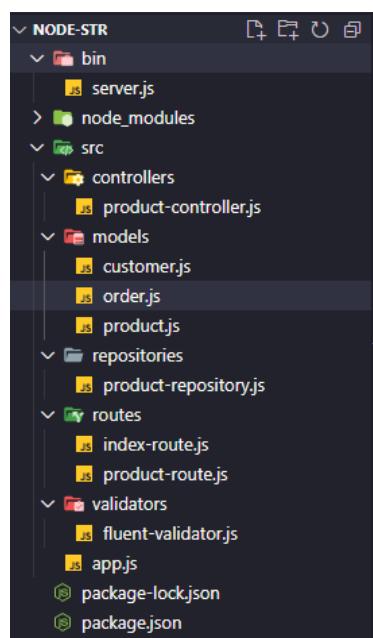
// Carrega os models
const Product = require('./models/product');
const Customer = require('./models/customer');
const Order = require('./models/order');

// Carrega as rotas
const indexRoute = require('./routes/index-route');
const productRoute = require('./routes/product-route');

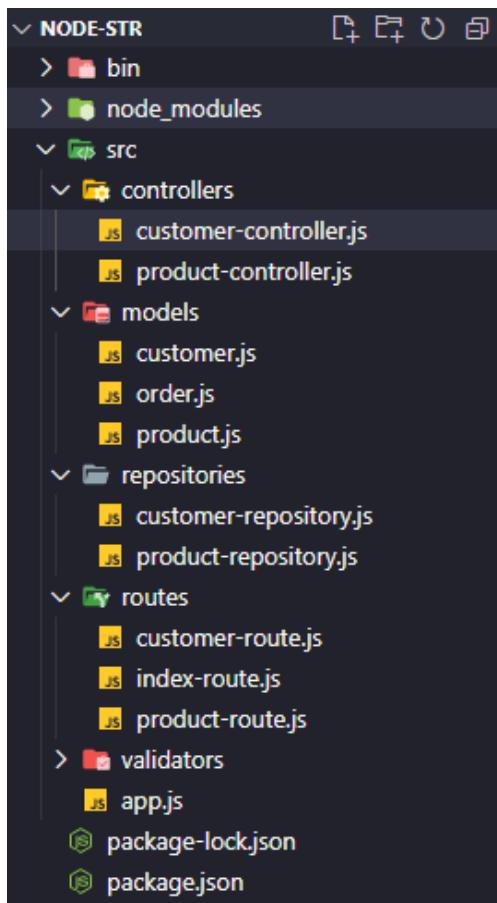
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({
    extended: false
}));

app.use('/', indexRoute);
app.use('/products', productRoute);

module.exports = app;
```



Aula 28 - Revisitando os Controllers: Customer



src/controllers/customer-controller.js

```
'use strict';

const ValidationContract = require('../validators/fluent-validator');
const repository = require('../repositories/customer-repository');

exports.post = async(req, res, next) => {
    let contract = new ValidationContract();
    contract.hasMinLen(req.body.name, 3, 'O nome deve conter pelo menos 3
caracteres');
    contract.isEmail(req.body.email, 'Email inválido');
    contract.hasMinLen(req.body.password, 6, 'A senha deve conter pelo menos 6
caracteres');

    // Se os dados forem inválidos
    if (!contract.isValid()) {
        res.status(400).send(contract.errors()).end();
        return;
    }
}
```

```
try {
    await repository.create(req.body);
    res.status(201).send({
        message: 'Cliente cadastrado com sucesso!'
    });
} catch (e) {
    console.log(e);
    res.status(500).send({
        message: 'Falha ao processar sua requisição'
    });
}
};
```

src/repositories/customer-repository.js

```
'use strict';
const mongoose = require('mongoose');
const Customer = mongoose.model('Customer');

exports.create = async(data) => {
    var customer = new Customer(data);
    await customer.save();
}
```

src/routes/customer-route.js

```
const express = require('express');
const router = express.Router();
const controller = require("../controllers/customer-controller");

router.post('/', controller.post);

module.exports = router;
```

src/app.js

```
const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');

const app = express();
const router = express.Router();

// Conecta ao banco
mongoose.connect("mongodb+srv://betopinheiro1005:angstron1005@node-store-cluster-nlcnv.mongodb.net/node-str-db?retryWrites=true&w=majority", {
useNewUrlParser: true });

// Carrega os models
const Product = require('./models/product');
const Customer = require('./models/customer');
const Order = require('./models/order');

// Carrega as rotas
const indexRoute = require('./routes/index-route');
const productRoute = require('./routes/product-route');
const customerRoute = require('./routes/customer-route');

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({
  extended: false
}));

app.use('/', indexRoute);
app.use('/products', productRoute);
app.use('/customers', customerRoute);

module.exports = app;
```

nodemon ./bin/server.js

```
C:\balta\nodejs\node-str>nodemon ./bin/server.js
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node ./bin/server.js`
API rodando na porta 3000
(node:9060) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:9060) [MONGODB DRIVER] Warning: Top-level use of w, wtimeout, j, and fsync is deprecated. Use writeConcern instead.
(node:9060) [MONGODB DRIVER] Warning: Current Server Discovery and Monitoring engine is deprecated, and will be removed in a future version. To use the new Server Discover and Monitoring engine, pass option { useUnifiedTopology: true } to the MongoClient constructor.
(node:9060) DeprecationWarning: collection.ensureIndex is deprecated. Use createIndexes instead.
```

Criando um cliente (customer)

POST - localhost:3000/customers

The screenshot shows the Postman interface. On the left, the 'My Workspace' sidebar lists various collections, APIs, environments, mock servers, monitors, and history. The main workspace shows a 'Testes - Node JS ~ Balta / Criando um cliente (customer)' collection. A POST request is selected with the URL 'localhost:3000/customers'. The 'Body' tab is active, showing a JSON payload:

```
1 {  
2   ... "name": "Roberto Pinheiro",  
3   ... "email": "betopinheiro1005@yahoo.com.br",  
4   ... "password": "12345678"  
5 }
```

Below the body, the response status is 201 Created with a response time of 389 ms and a size of 285 B. The response body contains the message: "message": "Cliente cadastrado com sucesso!"

No Studio 3T:

The screenshot shows the Studio 3T interface for MongoDB. The top menu includes File, Edit, Database, Collection, Index, Document, GridFS, View, and Help. The toolbar includes Connect, Collection, IntelliShell, SQL, Aggregate, Map-Reduce, Compare, Schema, Reschema, Tasks, Export, Import, Data Masking, SQL Migration, Users, Roles, Server 3T, and Feedback. The left sidebar shows open connections: 'atlas-ejqhqg-shard-0 [replica set: atlas-ejqhqg-shard-0]'. The 'customers' collection is selected. The right panel displays the results of a query on the 'customers' collection, showing one document with the following fields and values:

Key	Value	Type
\$_id	6095f4259795c2364cb104a	Document
name	Roberto Pinheiro	String
email	betopinheiro1005@yahoo.com.br	String
password	12345678	String
_v	0	Int32

Aula 29 - Revisitando os Controllers: Order

src/repositories/order-repository.js

```
'use strict';
const mongoose = require('mongoose');
const Order = mongoose.model('Order');

exports.get = async(data) => {
  var res = await Order.find({}, 'name status customer items')
    .populate('customer', 'name')
    .populate('items.product', 'title');
  return res;
}

exports.create = async(data) => {
  var order = new Order(data);
  await order.save();
}
```

Instalação do pacote guid

```
npm install guid --save
```

```
C:\balta\nodejs\node-str>npm install guid --save
npm WARN deprecated guid@0.0.12: Please use node-uuid instead. It is much better.
npm WARN node-str@1.0.0 No description
npm WARN node-str@1.0.0 No repository field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
+ guid@0.0.12
added 1 package from 2 contributors and audited 212 packages in 9.7s

13 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

src/controllers/order-controller.js

```
'use strict';

const ValidationContract = require('../validators/fluent-validator');
const repository = require('../repositories/order-repository');
const guid = require('guid');

exports.get = async(req, res, next) => {
    try {
        var data = await repository.get();
        res.status(200).send(data);
    } catch (e) {
        res.status(500).send({
            message: 'Falha ao processar sua requisição'
        });
    }
}

exports.post = async(req, res, next) => {
    try {
        await repository.create({
            customer: req.body.customer,
            number: guid.raw().substring(0, 6),
            items: req.body.items
        });
        res.status(201).send({
            message: 'Pedido cadastrado com sucesso!'
        });
    } catch (e) {
        console.log(e);
        res.status(500).send({
            message: 'Falha ao processar sua requisição'
        });
    }
};
```

src/routes/order-route.js

```
const express = require('express');
const router = express.Router();
const controller = require("../controllers/order-controller");

router.get('/', controller.get);
router.post('/', controller.post);

module.exports = router;
```

src/app.js

```
const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');

const app = express();
const router = express.Router();

// Conecta ao banco
mongoose.connect("mongodb+srv://betopinheiro1005:angstron1005@node-store-cluster-nlcnv.mongodb.net/node-str-db?retryWrites=true&w=majority", {
useNewUrlParser: true });

// Carrega os models
const Product = require('./models/product');
const Customer = require('./models/customer');
const Order = require('./models/order');

// Carrega as rotas
const indexRoute = require('./routes/index-route');
const productRoute = require('./routes/product-route');
const customerRoute = require('./routes/customer-route');
const orderRoute = require('./routes/order-route');

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({
  extended: false
}));

app.use('/', indexRoute);
app.use('/products', productRoute);
app.use('/customers', customerRoute);
app.use('/orders', orderRoute);

module.exports = app;
```

nodemon ./bin/server.js

```
C:\balta\nodejs\node-str>nodemon ./bin/server.js
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node ./bin/server.js`
API rodando na porta 3000
(node:10124) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the
new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:10124) [MONGODB DRIVER] Warning: Top-level use of w, wtimeout, j, and fsync is deprecated. Use writeConcern instead.
(node:10124) [MONGODB DRIVER] Warning: Current Server Discovery and Monitoring engine is deprecated, and will be removed in a
future version. To use the new Server Discover and Monitoring engine, pass option { useUnifiedTopology: true } to the MongoC
lient constructor.
(node:10124) DeprecationWarning: collection.ensureIndex is deprecated. Use createIndexes instead.
```

Testando no Postman

POST - localhost:3000/orders

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing collections like 'Postman Echo' and 'Testes - Node JS - Balta'. The main area shows a 'POST' request to 'localhost:3000/orders'. The 'Body' tab is selected, displaying the following JSON payload:

```
1 {
2   ... "customer": "6095f42597f95c2364cb104a",
3   ... "items": [
4     ... { "quantity": 1, "price": 2299, "product": "6095dbca7eae7e085cd33660" }
5   ...
6 }
```

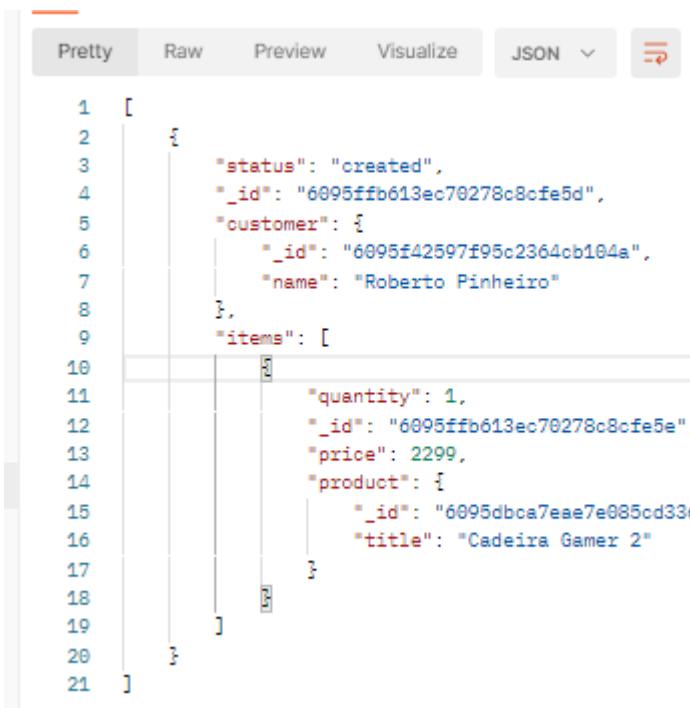
Below the body, the response section shows a status of 201 Created with a response message: "Pedido cadastrado com sucesso!".

No Studio 3T:

Key	Value	Type
↙ (1) {_id : 6095ffb613ec70278c8cf5d}	{ 7 fields }	Document
↳ _id	6095ffb613ec70278c8cf5d	ObjectId
↳ status	created	String
↳ customer	6095f42597f95c2364cb104a	ObjectId
↳ number	b790cc	String
↳ items	[1 elements]	Array
↳ 0	{ 4 fields }	Object
↳ quantity	1	Int32
↳ _id	6095ffb613ec70278c8cf5e	ObjectId
↳ price	2299	Int32
↳ product	6095dbca7eae7e085cd33660	ObjectId
↳ createDate	2021-05-08T03:04:22.091Z	Date
↳ __v	0	Int32

Listando os pedidos no Postman

GET - localhost:3000/orders



The screenshot shows the Postman application interface with a JSON response. The response is displayed in 'Pretty' mode, which shows the JSON structure with line numbers and indentation. The JSON data represents a single order document.

```
1 [  
2 {  
3     "status": "created",  
4     "_id": "6095ffb613ec70278c8cfef5d",  
5     "customer": {  
6         "_id": "6095f42597f95c2364cb104a",  
7         "name": "Roberto Pinheiro"  
8     },  
9     "items": [  
10        {  
11            "quantity": 1,  
12            "_id": "6095ffb613ec70278c8cfef5e",  
13            "price": 2299,  
14            "product": {  
15                "_id": "6095dbca7eae7e085cd33660",  
16                "title": "Cadeira Gamer 2"  
17            }  
18        }  
19    ]  
20}  
21]
```

Aula 30 - Arquivo de configurações

src/config.js

```
global.SALT_KEY = 'f5b99242-6504-4ca3-90f2-05e78e5761ef';
global.EMAIL_TMPL = 'Olá, <strong>{0}</strong>, seja bem vindo à Node Store!';

module.exports = {
  connectionString: 'mongodb+srv://betopinheiro1005:angstron1005@node-store-cluster.l4jj0.mongodb.net/node-store-db?retryWrites=true&w=majority',
  sendgridKey: 'TBD',
  containerConnectionString: 'TBD'
}
```

src/app.js

```
const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const config = require('./config');

const app = express();
const router = express.Router();

// Conecta ao banco
mongoose.connect(config.connectionString);

// Carrega os models
const Product = require('./models/product');
const Customer = require('./models/customer');
const Order = require('./models/order');

// Carrega as rotas
const indexRoute = require('./routes/index-route');
const productRoute = require('./routes/product-route');
const customerRoute = require('./routes/customer-route');
const orderRoute = require('./routes/order-route');

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({
  extended: false
}));

app.use('/', indexRoute);
app.use('/products', productRoute);
app.use('/customers', customerRoute);
app.use('/orders', orderRoute);

module.exports = app;
```

Aula 31 - Encriptando a senha

Instalação do md5

```
npm install md5 --save
```

```
C:\balta\nodejs\node-str>npm install md5 --save
npm WARN node-str@1.0.0 No description
npm WARN node-str@1.0.0 No repository field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
+ md5@2.3.0
added 4 packages from 4 contributors and audited 216 packages in 6.473s

13 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

src/controllers/customer-controller.js

```
'use strict';

const ValidationContract = require('../validators/fluent-validator');
const repository = require('../repositories/customer-repository');
const md5 = require('md5');

exports.post = async(req, res, next) => {
  let contract = new ValidationContract();
  contract.hasMinLen(req.body.name, 3, 'O título deve conter pelo menos 3 caracteres');
  contract.isEmail(req.body.email, 'Email inválido');
  contract.hasMinLen(req.body.password, 6, 'A senha deve conter pelo menos 6 caracteres');

  // Se os dados forem inválidos
  if (!contract.isValid()) {
    res.status(400).send(contract.errors()).end();
    return;
  }

  try {
    await repository.create({
      name: req.body.name,
      email: req.body.email,
      password: md5(req.body.password + global.SALT_KEY)
    });
    res.status(201).send({
      message: 'Cliente cadastrado com sucesso!'
    });
  } catch (e) {
    console.log(e);
  }
}
```

```

        res.status(500).send({
            message: 'Falha ao processar sua requisição'
        });
    }
};

```

Criando um segundo cliente

POST - localhost:3000/customers

The screenshot shows a Postman interface with a POST request to `localhost:3000/customers`. The Body tab is selected and contains the following JSON:

```

1 {
2   "name": "André Baltieri",
3   "email": "andrebaltieri@gmail.com",
4   "password": "andrebaltieri"
5 }

```

The response tab shows the following JSON:

```

1 {
2   "message": "Cliente cadastrado com sucesso!"
3 }

```

No Studio 3T

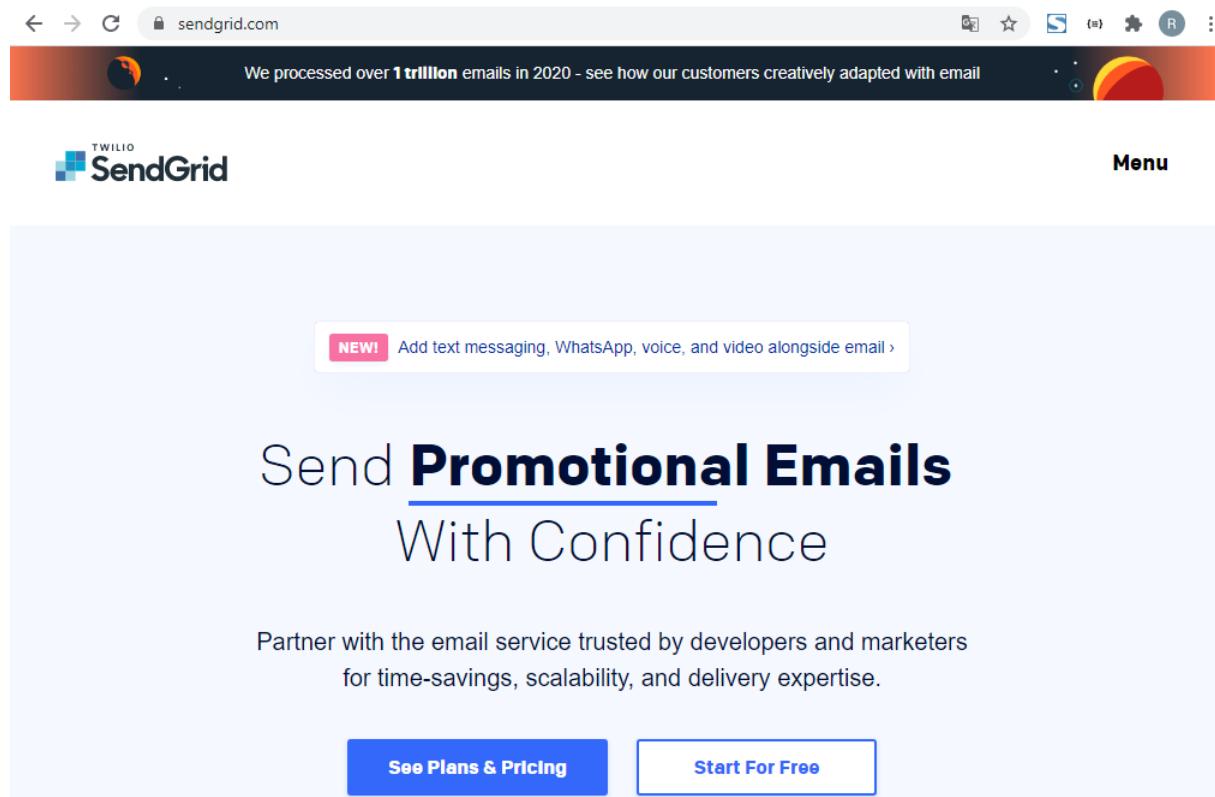
Key	Value	Type
① (1) <code>{ _id : 609e5d023fa4711440213558}</code>	{ 5 fields }	Document
`_id`	609e5d023fa4711440213558	ObjectId
`name`	Roberto Pinheiro	String
`email`	betopinheiro1005@yahoo.com.br	String
`password`	12345678	String
`__v`	0	Int32
② (2) <code>{ _id : 609e8053a8e0b53cd40564ef}</code>	{ 5 fields }	Document
`_id`	609e8053a8e0b53cd40564ef	ObjectId
`name`	André Baltieri	String
`email`	andrebaltieri@gmail.com	String
`password`	08a6bccb1bcf2e49d089863cc3e4b96c	String
`__v`	0	Int32

- Repare que a senha do segundo cliente está encriptada.

Aula 32 - Enviando email de boas vindas

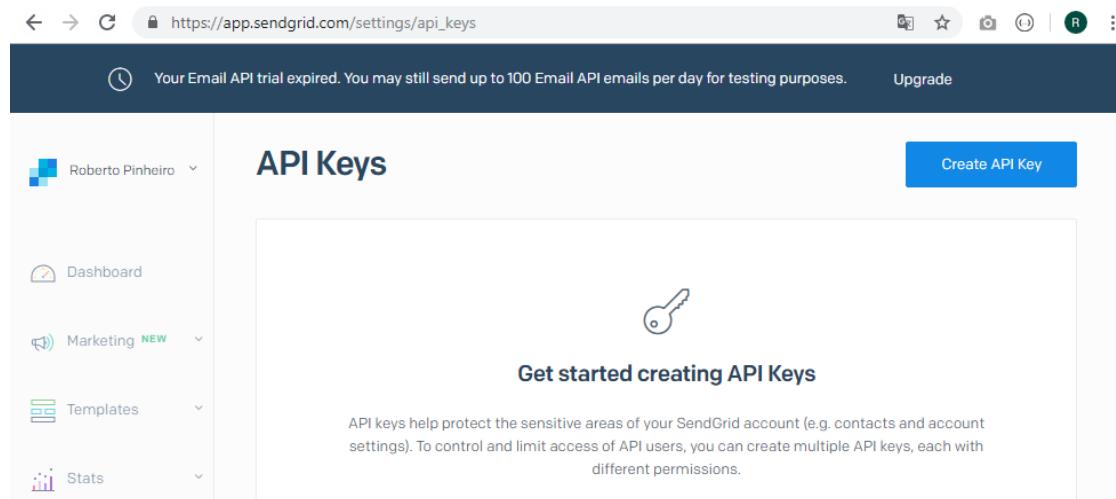
SendGrid

- Acesse <https://sendgrid.com/> e abra uma conta.



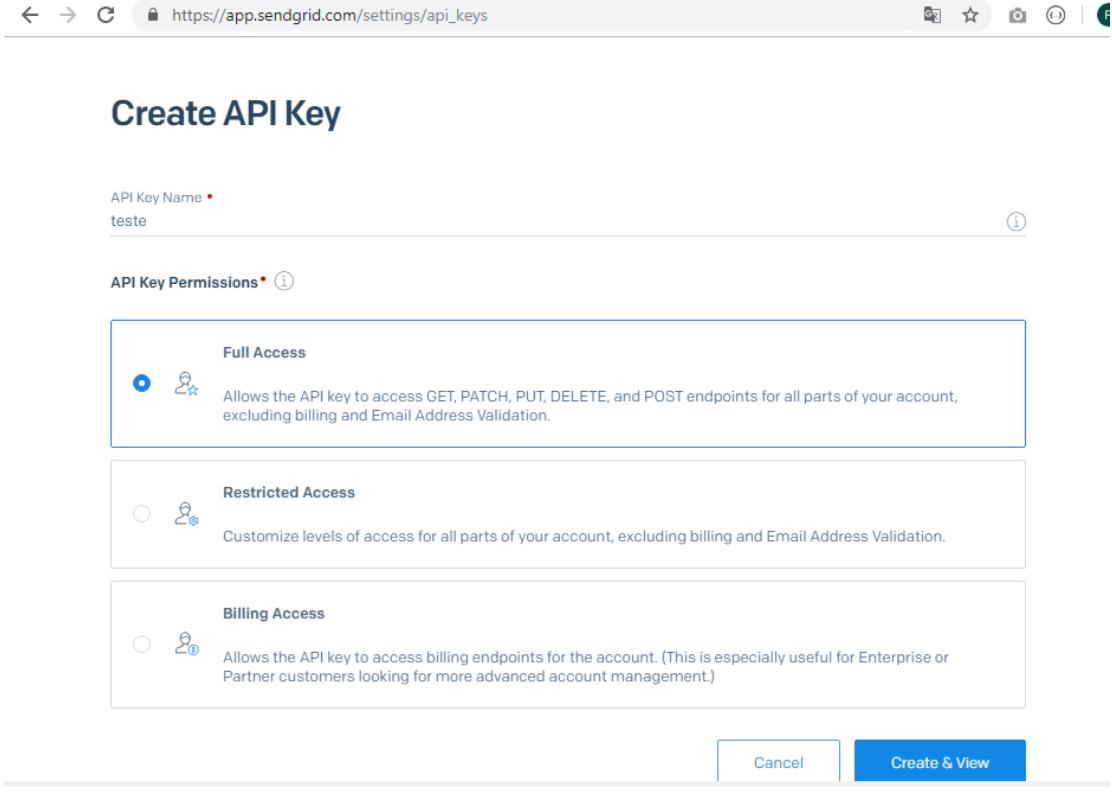
The screenshot shows the SendGrid homepage. At the top, there's a banner stating "We processed over 1 trillion emails in 2020 - see how our customers creatively adapted with email". Below the banner, the SendGrid logo is visible, along with a "Menu" button. The main headline reads "Send Promotional Emails With Confidence". A subtext below it says "Partner with the email service trusted by developers and marketers for time-savings, scalability, and delivery expertise." Two prominent buttons are at the bottom: "See Plans & Pricing" and "Start For Free".

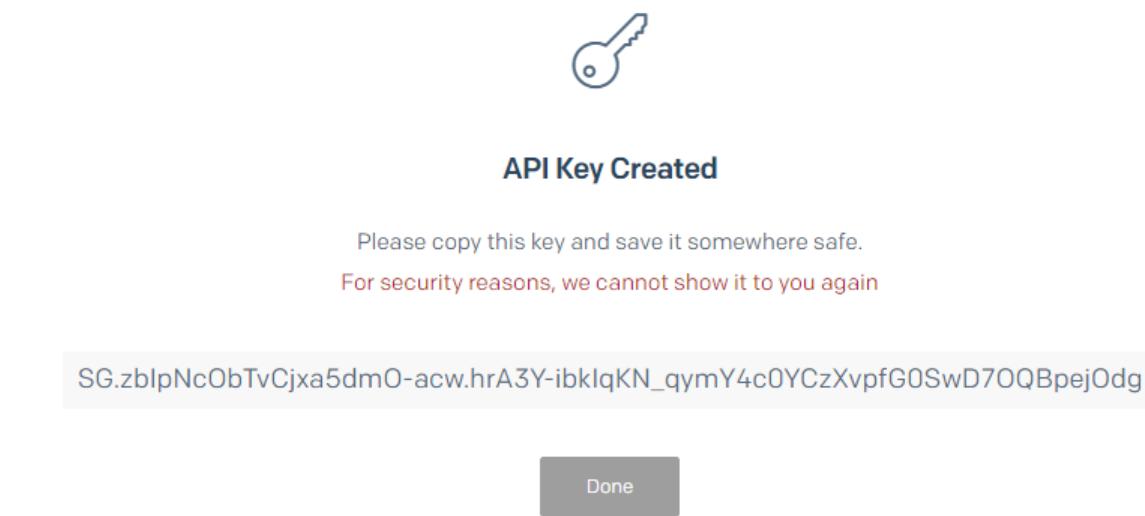
- Crie uma API Key

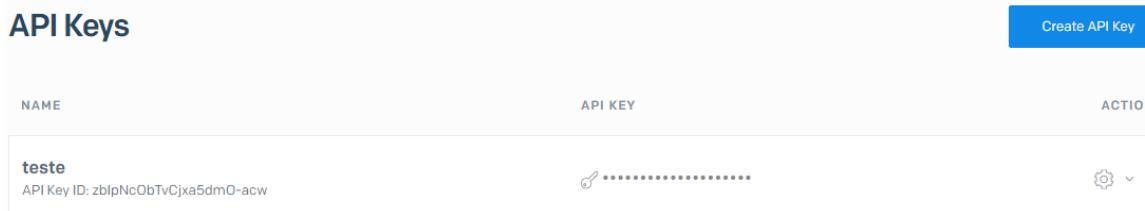


The screenshot shows the "API Keys" page in the SendGrid dashboard. A banner at the top indicates that the Email API trial has expired, allowing 100 emails per day for testing purposes, with an "Upgrade" link. On the left, a sidebar menu includes "Dashboard", "Marketing NEW", "Templates", and "Stats". The main content area features a "Create API Key" button and a section titled "Get started creating API Keys" with a key icon. Below this, a paragraph explains that API keys protect sensitive account areas and allow for controlled access.

- Nomeie a API de **teste** e dê acesso total (full access):

A screenshot of the SendGrid 'Create API Key' page. At the top, the URL is https://app.sendgrid.com/settings/api_keys. The main title is 'Create API Key'. Below it, 'API Key Name' is set to 'teste'. Under 'API Key Permissions', 'Full Access' is selected, which allows the API key to access all endpoints except billing and email validation. There are also options for 'Restricted Access' and 'Billing Access'. At the bottom right are 'Cancel' and 'Create & View' buttons.

A screenshot of the 'API Key Created' page. It features a large key icon and the heading 'API Key Created'. A message says 'Please copy this key and save it somewhere safe.' and 'For security reasons, we cannot show it to you again'. The API key itself is displayed as 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. A 'Done' button is at the bottom.

A screenshot of the 'API Keys' list page. It shows a table with columns: NAME, API KEY, and ACTION. One row is shown for the 'teste' key, with its API Key ID and a copy icon. A 'Create API Key' button is in the top right.

- Copie a chave e cole-a no arquivo **config.js**:

src/config.js

```
global.SALT_KEY = 'f5b99242-6504-4ca3-90f2-05e78e5761ef';
global.EMAIL_TMPL = 'Olá, <strong>{0}</strong>, seja bem vindo à Node Store!';

module.exports = {
  connectionString: 'mongodb+srv://betopinheiro1005:angstron1005@node-store-cluster.l4jj0.mongodb.net/node-store-db?retryWrites=true&w=majority',
  sendgridKey: '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',
  containerConnectionString: 'TBD'
}
```

Instalação do SendGrid

npm install sendgrid@2.0.0 --save

```
C:\balta\nodejs\node-str>npm install sendgrid@2.0.0 --save
npm WARN deprecated sendgrid@2.0.0: Please see v6.X+ at https://www.npmjs.com/org/sendgrid
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/request/request/issues/3142
npm WARN deprecated har-validator@5.1.5: this library is no longer supported
npm WARN deprecated core-js@2.6.12: core-js@<3 is no longer maintained and not recommended for usage due to the number of issues. Please, upgrade your dependencies to the actual version of core-js@3.
> core-js@2.6.12 postinstall C:\balta\nodejs\node-str\node_modules\core-js
> node -e "try{require('./postinstall')}catch(e){}"
Thank you for using core-js ( https://github.com/zloirock/core-js ) for polyfilling JavaScript standard library!

The project needs your help! Please consider supporting of core-js on Open Collective or Patreon:
> https://opencollective.com/core-js
> https://www.patreon.com/zloirock

Also, the author of core-js ( https://github.com/zloirock ) is looking for a good job -)

npm WARN node-str@1.0.0 No description
npm WARN node-str@1.0.0 No repository field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
+ sendgrid@2.0.0
added 141 packages from 200 contributors and audited 357 packages in 190.45s

15 packages are looking for funding
  run `npm fund` for details

found 7 vulnerabilities (3 low, 1 moderate, 3 high)
```

package.json

```
{  
  "name": "node-str",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1",  
    "start": "node ./bin/server.js"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC",  
  "dependencies": {  
    "body-parser": "^1.19.0",  
    "debug": "^4.3.1",  
    "express": "^4.17.1",  
    "guid": "0.0.12",  
    "http": "0.0.1-security",  
    "md5": "^2.3.0",  
    "mongoose": "^5.12.7",  
    "sendgrid": "^2.0.0"  
  },  
  "devDependencies": {  
    "nodemon": "^2.0.7"  
  }  
}
```

src/services/email-service.js

```
'use strict';  
var config = require('../config');  
var sendgrid = require('sendgrid')(config.sendgridKey);  
  
exports.send = async (to, subject, body) => {  
  sendgrid.send({  
    to: to,  
    from: 'andrebaltieri@balta.io',  
    subject: subject,  
    html: body  
  });  
}
```

src/controllers/customer-controller.js

```
'use strict';

const ValidationContract = require('../validators/fluent-validator');
const repository = require('../repositories/customer-repository');
const md5 = require('md5');
const emailService = require('../services/email-service');

exports.post = async(req, res, next) => {
    let contract = new ValidationContract();
    contract.hasMinLen(req.body.name, 3, 'O título deve conter pelo menos 3 caracteres');
    contract.isEmail(req.body.email, 'Email inválido');
    contract.hasMinLen(req.body.password, 6, 'A senha deve conter pelo menos 6 caracteres');

    // Se os dados forem inválidos
    if (!contract.isValid()) {
        res.status(400).send(contract.errors()).end();
        return;
    }

    try {
        await repository.create({
            name: req.body.name,
            email: req.body.email,
            password: md5(req.body.password + global.SALT_KEY)
        });

        emailService.send(
            req.body.email,
            'Bem vindo ao Node Store',
            global.EMAIL_TMPL.replace('{0}', req.body.name)
        );

        res.status(201).send({
            message: 'Cliente cadastrado com sucesso!'
        });
    } catch (e) {
        console.log(e);
        res.status(500).send({
            message: 'Falha ao processar sua requisição'
        });
    }
};
```

- No Studio 3T, apague os clientes cadastrados.
- Crie um novo cliente.

No Postman

POST <localhost:3000/customers>

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```

1 {
2   "name": "Roberto Pinheiro",
3   "email": "betopinheiro1005@yahoo.com.br",
4   "password": "robertopinheiro"
5 }
```

Body Cookies Headers (7) Test Results Status: 201 Created Time: 430 ms Size: 285 B Save Response

Pretty Raw Preview Visualize JSON

```

1 {
2   "message": "Cliente cadastrado com sucesso!"
3 }
```

Key	Value	Type
« ↴ (1) { _id : 5d5c6d8bcc81b832f8f37ddd } »	{ 5 fields }	Document
↳ _id	5d5c6d8bcc81b832f8f37ddd	ObjectId
↳ name	Roberto Pinheiro	String
↳ email	betopinheiro1005@yahoo.com.br	String
↳ password	891f095be7ed455ec084e1fc23a3bb21	String
↳ __v	0	Int32

- Ao realizar o cadastro, o cliente irá receber em seu email a seguinte mensagem:

Hoje

Land Rover ANÚNCIO A aventura está em nosso DNA. Capacidade e versatilidade para enfrentar qual...

andrebaltieri@balta... Bem vindo ao Node Store Olá, Roberto Pinheiro, seja bem vindo à Node ...

Bem vindo ao Node Store Yahoo/Spam

andrebaltieri@balta.io 20 de ago às 03:05

Para: betopinheiro1005@yahoo.com.br

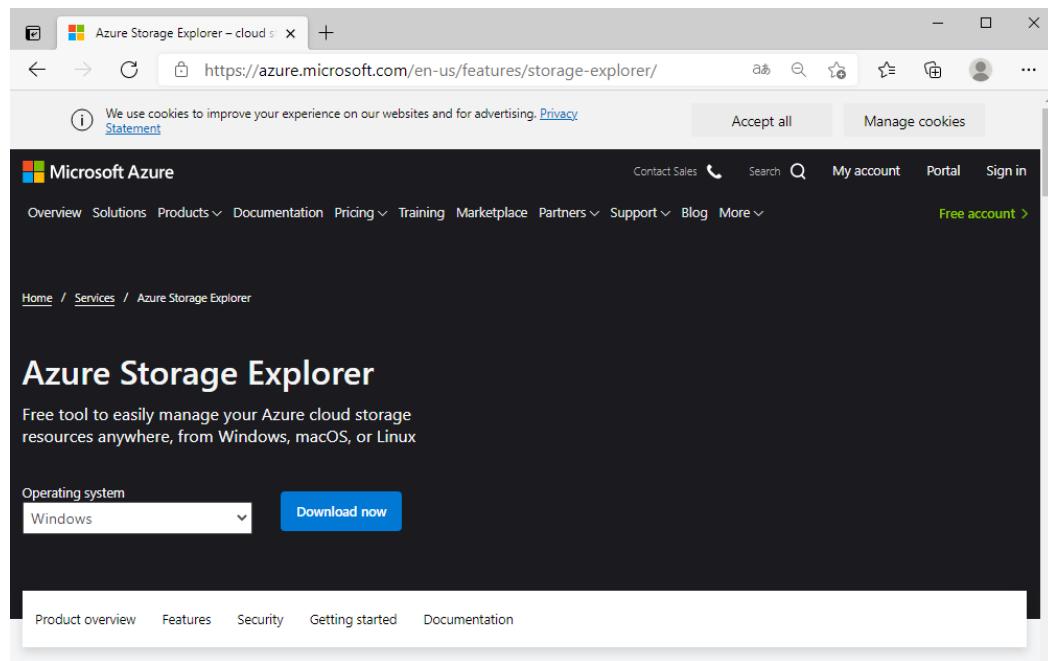
Olá, Roberto Pinheiro, seja bem vindo à Node Store!

Responder, Responder a todos ou Encaminhar

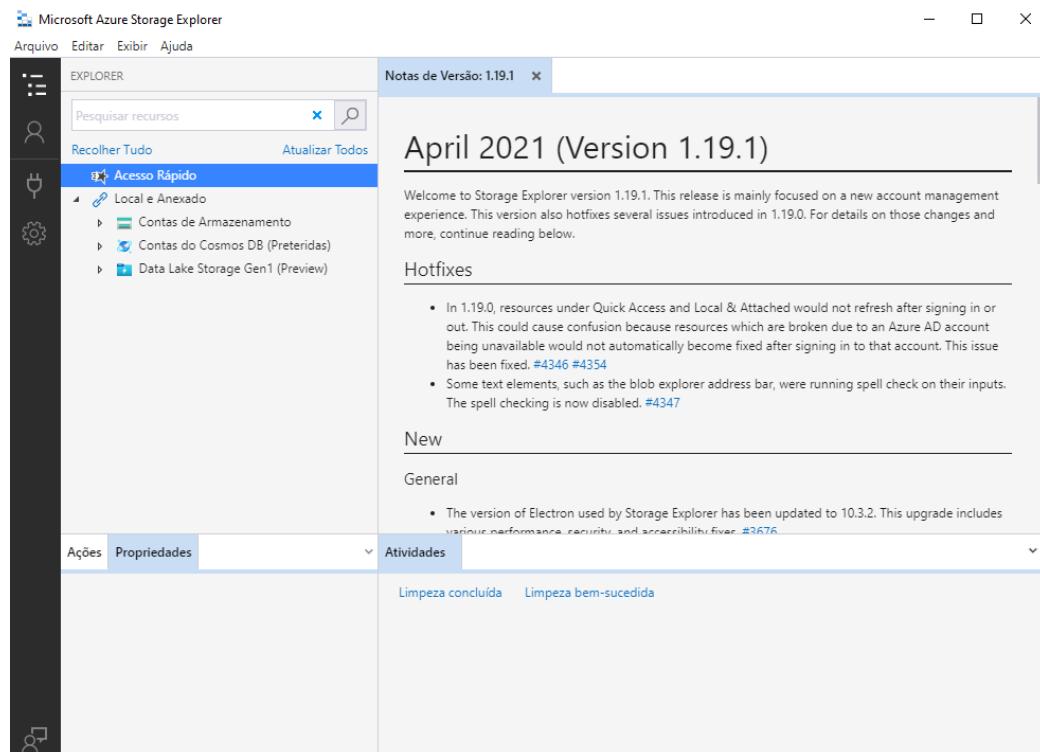
Aula 33 - Upload da imagem do produto

Acesse a URL:

<https://azure.microsoft.com/en-us/features/storage-explorer/>



- Baixe e instale o programa **Microsoft Azure Storage Explorer free**.



- Crie sua conta no Azzure:

<https://azure.microsoft.com/pt-br/free/>

The screenshot shows the Microsoft Azure free account creation landing page. At the top, there's a message about cookie usage and links to accept all or manage cookies. Below that is the Microsoft Azure logo and navigation links for Contatar Vendas, Pesquisa, Minha conta, Portal, and Entrar. A main heading says "Crie sua conta gratuita do Azure hoje mesmo" and "Comece com 12 meses de serviços gratuitos". There are two prominent buttons: "Início gratuito" (Get started) and "Ou compre agora" (Or buy now). Below these buttons is a preview of the Azure portal interface showing recent resources like "amr", "BuildApp", "AI-Downloader-3c93", and "adventure-vm-3-ip". A "Chat com Vendas" button is also visible.

- Acesse o portal.

The screenshot shows the Microsoft Azure portal home page. It features a welcome message "Bem-vindo(a) ao Azure." and a note about not having an account yet. It highlights three main sections: "Começar com uma avaliação gratuita do Azure" (with a "Iniciar" button), "Gerenciar o Azure Active Directory" (with a "Exibir" button), and "Acessar os benefícios do aluno" (with a "Explorar" button). Below these are "Serviços do Azure" icons for various services like Máquinas virtuais, Serviços de Aplicativos, and Banco de dados SQL. A "Navegar" section is also present.

- Crie um Storage Account.

Aula 34 - Autenticação

- Vamos utilizar autenticação via token com JWT.

Instalando o pacote JWT

```
npm install jsonwebtoken@7.4.0 --save
```

```
C:\balta\nodejs\node-str>npm install jsonwebtoken@7.4.0 --save
npm WARN deprecated joi@6.10.1: This version has been deprecated in accordance with the hapi support policy (hapi.im/support)
. Please upgrade to the latest version to get the best features, bug fixes, and security patches. If you are unable to upgrade at this time, paid support is available for older versions (hapi.im/commercial).
npm WARN deprecated topo@1.1.0: This version has been deprecated in accordance with the hapi support policy (hapi.im/support)
. Please upgrade to the latest version to get the best features, bug fixes, and security patches. If you are unable to upgrade at this time, paid support is available for older versions (hapi.im/commercial).
npm WARN deprecated hoek@2.16.3: This version has been deprecated in accordance with the hapi support policy (hapi.im/support)
. Please upgrade to the latest version to get the best features, bug fixes, and security patches. If you are unable to upgrade at this time, paid support is available for older versions (hapi.im/commercial).
npm WARN node-str@1.0.0 No description
npm WARN node-str@1.0.0 No repository field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

+ jsonwebtoken@7.4.0
added 13 packages from 18 contributors and audited 370 packages in 43.908s

15 packages are looking for funding
  run `npm fund` for details

found 9 vulnerabilities (3 low, 3 moderate, 3 high)
  run `npm audit fix` to fix them, or `npm audit` for details
```

src\services\auth-service.js

```
'use strict';
const jwt = require('jsonwebtoken');

exports.generateToken = async (data) => {
    return jwt.sign(data, global.SALT_KEY, { expiresIn: '1d' });
}

exports.decodeToken = async (token) => {
    var data = await jwt.verify(token, global.SALT_KEY);
    return data;
}

exports.authorize = function (req, res, next) {
    var token = req.body.token || req.query.token || req.headers['x-access-token'];

    if (!token) {
        res.status(401).json({
            message: 'Acesso Restrito'
        });
    } else {
        jwt.verify(token, global.SALT_KEY, function (error, decoded) {
            if (error) {
                res.status(401).json({
                    message: 'Token Inválido'
                });
            } else {
                next();
            }
        });
    }
};
```

src\routes\product-route.js

```
const express = require('express');
const router = express.Router();
const controller = require("../controllers/product-controller");
const authService = require('../services/auth-service');

router.get('/', controller.get);
router.get('/:slug', controller.getBySlug);
router.get('/admin/:id', controller.getId);
router.get('/tags/:tag', controller.getTag);
router.post('/', authService.authorize, controller.post);
router.put('/:id', authService.authorize, controller.put);
router.delete('/:id', authService.authorize, controller.delete);

module.exports = router;
```

```
nodemon ./bin/server.js
```

```
C:\balta\nodejs\node-str>nodemon ./bin/server.js
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node ./bin/server.js`
API rodando na porta 3000
(node:14384) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:14384) [MONGODB DRIVER] Warning: Top-level use of w, wtimeout, j, and fsync is deprecated. Use writeConcern instead.
(node:14384) [MONGODB DRIVER] Warning: Current Server Discovery and Monitoring engine is deprecated, and will be removed in a future version. To use the new Server Discover and Monitoring engine, pass option { useUnifiedTopology: true } to the MongoClient constructor.
(node:14384) DeprecationWarning: collection.ensureIndex is deprecated. Use createIndexes instead.
```

- Ao tentar criar um produto:

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing various test collections and environments. The main area shows a 'Testes - Node JS - Balta / Cadastrando um produto' collection. A 'POST' request is selected, pointing to 'localhost:3000/products'. The 'Body' tab is active, showing a JSON payload:

```
1 {
2   "title": "Mouse Gamez",
3   "description": "Mouse Gamez",
4   "slug": "mouse-gamez",
5   "price": 200,
6   "active": true,
7   "tags": []
```

The response pane shows a status of 'Status: 401 Unauthorized' with a 'Save Response' button. The bottom navigation bar includes 'Pretty', 'Raw', 'Preview', 'Visualize', and 'JSON' tabs.

src\repositories\customer-repository.js

```
'use strict';
const mongoose = require('mongoose');
const Customer = mongoose.model('Customer');

exports.create = async(data) => {
  var customer = new Customer(data);
  await customer.save();
}

exports.authenticate = async(data) => {
  const res = await Customer.findOne({
    email: data.email,
    password: data.password
  });
  return res;
}
```

src\controllers\customer-controller.js

```
'use strict';

const ValidationContract = require('../validators/fluent-validator');
const repository = require('../repositories/customer-repository');
const md5 = require('md5');
const authService = require('../services/auth-service');

exports.post = async(req, res, next) => {
    let contract = new ValidationContract();
    contract.hasMinLen(req.body.name, 3, 'O nome deve conter pelo menos 3 caracteres');
    contract.isEmail(req.body.email, 'Email inválido');
    contract.hasMinLen(req.body.password, 6, 'A senha deve conter pelo menos 6 caracteres');

    // Se os dados forem inválidos
    if (!contract.isValid()) {
        res.status(400).send(contract.errors()).end();
        return;
    }

    try {
        await repository.create({
            name: req.body.name,
            email: req.body.email,
            password: md5(req.body.password + global.SALT_KEY)
        });
        res.status(201).send({
            message: 'Cliente cadastrado com sucesso!'
        });
    } catch (e) {
        console.log(e);
        res.status(500).send({
            message: 'Falha ao processar sua requisição'
        });
    }
};

exports.authenticate = async(req, res, next) => {

    try {
        const customer = await repository.authenticate({
            email: req.body.email,
            password: md5(req.body.password + global.SALT_KEY)
        });

        if(!customer){
            res.status(404).send({
                message: 'Usuário ou senha inválido(s)!'
```

```
        });
        return;
    }

const token = await authService.generateToken({
    email: customer.email,
    name: customer.name
});

res.status(201).send({
    token: token,
    data: {
        email: customer.email,
        name: customer.name
    }
});
} catch (e) {
    console.log(e);
    res.status(500).send({
        message: 'Falha ao processar sua requisição'
    });
}
};
```

- No Postman, passando senha errada:

The screenshot shows the Postman interface with the following details:

- Testes - Node JS - Balta / Customer - authenticate**
- Method:** POST
- URL:** localhost:3000/customers/authenticate
- Body (JSON):**

```
1 "email": "betopinheiro1005@yahoo.com.br",
2 "password": "robertopinheiro2"
```
- Response Status:** 404 Not Found
- Response Time:** 176 ms
- Response Size:** 285 B
- Response Body (Pretty JSON):**

```
1 "message": "Usuário ou senha inválidos!"
```

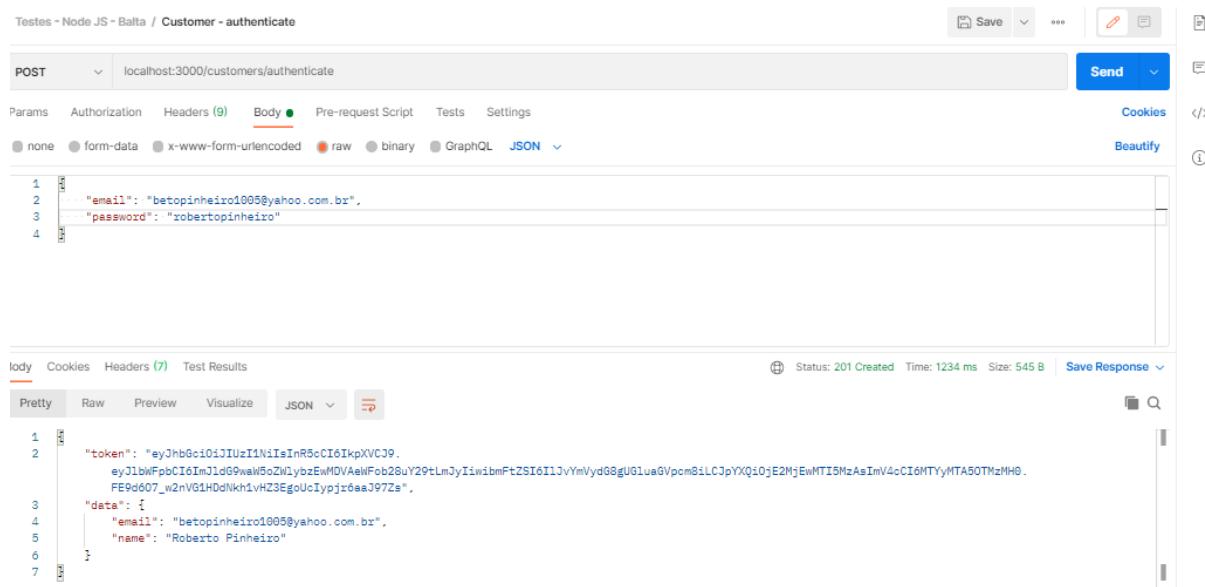
src\routes\customer-route.js

```
const express = require('express');
const router = express.Router();
const controller = require("../controllers/customer-controller");

router.post('/', controller.post);
router.post('/authenticate', controller.authenticate);

module.exports = router;
```

No Postman, passando a senha correta:



The screenshot shows a Postman test named "Customer - authenticate". The request method is POST to "localhost:3000/customers/authenticate". The body is set to "raw" and contains the following JSON:

```
1
2   "email": "betopinheiro1005@yahoo.com.br",
3   "password": "robertopinheiro"
4 }
```

The response status is 201 Created, with a token and user data returned in the body:

```
1
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImJldG9waW5oZWlybzEwMDVAeWFob28uY29tLmJyIiwibmFtZSI6IjJvYmVydG8gUGluaGVpcm8iLCJpYXQiOjE2MjEwMTI5MzAsImV4cCI6MTYyMTA5OTMzMH0.FE9d607_w2nVG1HDdNkh1vHZ3EgoUcipjr6aaJ97Zs",
3   "data": {
4     "email": "betopinheiro1005@yahoo.com.br",
5     "name": "Roberto Pinheiro"
6   }
7 }
```

- No Postman, na requisição que cadastra um produto, na aba **Header**, insira a **key**:

x-access-token

- Em **value**, cole o valor do token obtido ao cadastrar o cliente.

Aula 35 - Recuperando dados do usuário logado

src\routes\order-route.js

```
const express = require('express');
const router = express.Router();
const controller = require("../controllers/order-controller");
const authService = require('../services/auth-service');

router.get('/', authService.authorize, controller.get);
router.post('/', authService.authorize, controller.post);

module.exports = router;
```

src\controllers\customer-controller.js

```
'use strict';

const ValidationContract = require('../validators/fluent-validator');
const repository = require('../repositories/customer-repository');
const md5 = require('md5');
const emailService = require('../services/email-service');

const authService = require('../services/auth-service');

exports.get = async(req, res, next) => {
    try {
        var data = await repository.get();
        res.status(200).send(data);
    } catch (e) {
        res.status(500).send({
            message: 'Falha ao processar sua requisição'
        });
    }
}

exports.post = async(req, res, next) => {
    let contract = new ValidationContract();
    contract.hasMinLen(req.body.name, 3, 'O nome deve conter pelo menos 3 caracteres');
    contract.isEmail(req.body.email, 'Email inválido');
    contract.hasMinLen(req.body.password, 6, 'A senha deve conter pelo menos 6 caracteres');

    // Se os dados forem inválidos
    if (!contract.isValid()) {
        res.status(400).send(contract.errors()).end();
        return;
    }

    try {
        await repository.create({
            name: req.body.name,
            email: req.body.email,
            // password: req.body.password
    }
}
```

```

        password: md5(req.body.password + global.SALT_KEY)
    });

    emailService.send(
        req.body.email,
        'Bem vindo ao Node Store',
        global.EMAIL_TMPL.replace('{0}', req.body.name)
    );

    res.status(201).send({
        message: 'Cliente cadastrado com sucesso!'
    });
} catch (e) {
    console.log(e);
    res.status(500).send({
        message: 'Falha ao processar sua requisição'
    });
}
};

exports.authenticate = async(req, res, next) => {

try {
    const customer = await repository.authenticate({
        email: req.body.email,
        password: md5(req.body.password + global.SALT_KEY)
    });

    if(!customer){
        res.status(404).send({
            message: 'Usuário ou senha inválidos!'
        });
        return;
    }

    const token = await authService.generateToken({
        id: customer._id,
        email: customer.email,
        name: customer.name
    });

    res.status(201).send({
        token: token,
        data: {
            email: customer.email,
            name: customer.name
        }
    });
} catch (e) {
    console.log(e);
    res.status(500).send({
        message: 'Falha ao processar sua requisição'
    });
}
};

```

src\controllers\order-controller.js

```
'use strict';

const ValidationContract = require('../validators/fluent-validator');
const repository = require('../repositories/order-repository');
const guid = require('guid');
const authService = require('../services/auth-service');

exports.get = async(req, res, next) => {
    try {
        var data = await repository.get();
        res.status(200).send(data);
    } catch (e) {
        res.status(500).send({
            message: 'Falha ao processar sua requisição'
        });
    }
}

exports.post = async(req, res, next) => {
    try {
        const token = req.body.token || req.query.token || req.headers['x-access-token'];
        const data = await authService.decodeToken(token);

        await repository.create({
            customer: data.id,
            number: guid.raw().substring(0, 6),
            items: req.body.items
        });
        res.status(201).send({
            message: 'Pedido cadastrado com sucesso!'
        });
    } catch (e) {
        console.log(e);
        res.status(500).send({
            message: 'Falha ao processar sua requisição'
        });
    }
};
```

- No Postman:

POST localhost:3000/orders

Headers (9)

Key	Value	Description
Host	<calculated when request is sent>	
User-Agent	PostmanRuntime/7.28.0	
Accept	/*	
Accept-Encoding	gzip, deflate, br	
Connection	keep-alive	
x-access-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJtWFpbCl6ImJldG9w...	
Key	Value	Description

Testes - Node JS - Balta / Criando uma ordem de pedido (order)

POST localhost:3000/orders

Body (9)

```

1 {
2   "items": [
3     {"quantity": 1, "price": 299, "product": "6095dbca7eae7e085cd33660"}
4   ]
5 }

```

Body Cookies Headers (7) Test Results

Status: 201 Created Time: 196 ms Size: 284 B Save Response

Pretty Raw Preview Visualize JSON

```

1 "message": "Pedido cadastrado com sucesso!"
2
3

```

Key	Value	Type
> ⚡ (1) {_id : 6095ffb613ec70278c8cfef5d}	{ 7 fields }	Document
✓ ⚡ (2) {_id : 609ec3dcee14102584139e92}	{ 6 fields }	Document
▀ _id	609ec3dcee14102584139e92	ObjectId
▀ status	created	String
▀ number	5ac399	String
▼ items	[1 elements]	Array
▼ ⚡ 0	{ 4 fields }	Object
▀ quantity	1	Int32
▀ _id	609ec3dcee14102584139e93	ObjectId
▀ price	299	Int32
▀ product	6095dbca7eae7e085cd33660	ObjectId
▀ createDate	2021-05-14T18:39:24.387Z	Date
▀ __v	0	Int32

Aula 36 - Refresh Token

src\controllers\customer-controller.js

```
'use strict';

const ValidationContract = require('../validators/fluent-validator');
const repository = require('../repositories/customer-repository');
const md5 = require('md5');
const emailService = require('../services/email-service');

const authService = require('../services/auth-service');

exports.get = async(req, res, next) => {
  try {
    var data = await repository.get();
    res.status(200).send(data);
  } catch (e) {
    res.status(500).send({
      message: 'Falha ao processar sua requisição'
    });
  }
}

exports.post = async(req, res, next) => {
  let contract = new ValidationContract();
  contract.hasMinLen(req.body.name, 3, 'O nome deve conter pelo menos 3 caracteres');
  contract.isEmail(req.body.email, 'Email inválido');
  contract.hasMinLen(req.body.password, 6, 'A senha deve conter pelo menos 6 caracteres');

  // Se os dados forem inválidos
  if (!contract.isValid()) {
    res.status(400).send(contract.errors()).end();
    return;
  }

  try {
    await repository.create({
      name: req.body.name,
      email: req.body.email,
      // password: req.body.password
      password: md5(req.body.password + global.SALT_KEY)
    });

    emailService.send(
      req.body.email,
      'Bem vindo ao Node Store',
      global.EMAIL_TEMPL.replace('{0}', req.body.name)
    );

    res.status(201).send({
      message: 'Cliente cadastrado com sucesso!'
    });
  }
}
```

```

    } catch (e) {
        console.log(e);
        res.status(500).send({
            message: 'Falha ao processar sua requisição'
        });
    }
};

exports.authenticate = async(req, res, next) => {

    try {
        const customer = await repository.authenticate({
            email: req.body.email,
            password: md5(req.body.password + global.SALT_KEY)
        });

        if(!customer){
            res.status(404).send({
                message: 'Usuário ou senha inválidos!'
            });
            return;
        }

        const token = await authService.generateToken({
            id: customer._id,
            email: customer.email,
            name: customer.name
        });

        res.status(201).send({
            token: token,
            data: {
                email: customer.email,
                name: customer.name
            }
        });
    } catch (e) {
        console.log(e);
        res.status(500).send({
            message: 'Falha ao processar sua requisição'
        });
    }
};

exports.refreshToken = async(req, res, next) => {
    try {
        const token = req.body.token || req.query.token || req.headers['x-access-token'];
        const data = await authService.decodeToken(token);

        const customer = await repository.getById(data.id);

        if (!customer) {
            res.status(404).send({
                message: 'Cliente não encontrado'
            });
            return;
        }
    }
};

```

```

const tokenData = await authService.generateToken({
  id: customer._id,
  email: customer.email,
  name: customer.name,
  roles: customer.roles
});

res.status(201).send({
  token: token,
  data: {
    email: customer.email,
    name: customer.name
  }
});
} catch (e) {
  res.status(500).send({
    message: 'Falha ao processar sua requisição'
  });
}
};


```

src\repositories\customer-repository.js

```

'use strict';
const mongoose = require('mongoose');
const Customer = mongoose.model('Customer');

exports.get = async() => {
  const res = await Customer.find({
    }, 'name email password');
  return res;
}

exports.create = async(data) => {
  var customer = new Customer(data);
  await customer.save();
}

exports.authenticate = async(data) => {
  const res = await Customer.findOne({
    email: data.email,
    password: data.password
  });
  return res;
}

exports.getById = async(id) => {
  const res = await Customer.findById(id);
  return res;
}


```

src\routes\customer-route.js

```
'use strict';

const express = require('express');
const router = express.Router();
const controller = require("../controllers/customer-controller");
const authService = require('../services/auth-service');

router.get('/', controller.get);
router.post('/', controller.post);
router.post('/authenticate', controller.authenticate);
router.post('/refresh-token', authService.authorize, controller.refreshToken);

module.exports = router;
```

nodemon ./bin/server.js

```
C:\balta\nodejs\node-str>nodemon ./bin/server.js
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node ./bin/server.js`
API rodando na porta 3000
(node:7280) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:7280) [MONGODB DRIVER] Warning: Top-level use of w, wtimeout, j, and fsync is deprecated. Use writeConcern instead.
(node:7280) [MONGODB DRIVER] Warning: Current Server Discovery and Monitoring engine is deprecated, and will be removed in a future version. To use the new Server Discover and Monitoring engine, pass option { useUnifiedTopology: true } to the MongoClient constructor.
(node:7280) DeprecationWarning: collection.ensureIndex is deprecated. Use createIndexes instead.
```

- No Postman, crie a requisição localhost:3000/customers/refresh-token:
- Sem o token na key **x-access-token** da aba Headers:

The screenshot shows a Postman test environment. At the top, it says "Testes - Node JS - Balta / Refresh Token". Below that, a "POST" method is selected, and the URL is "localhost:3000/customers/refresh-token". The "Headers" tab is active, showing the following headers:

Key	Value	Description
Host	<calculated when request is sent>	
User-Agent	PostmanRuntime/7.28.0	
Accept	*	
Accept-Encoding	gzip, deflate, br	
Connection	keep-alive	
x-access-token		

The "Body" tab is also visible at the bottom, showing a JSON response:

```
1   {
2     "message": "Acesso Restrito"
3 }
```

- Com o token ainda válido, na key **x-access-token** da aba **Headers**, ao clicar no botão **Send** é gerado um novo token:

The screenshot shows the Postman interface with the following details:

- Testes - Node JS - Balta / Refresh Token**
- POST** to **localhost:3000/customers/refresh-token**
- Headers (8)** tab selected, showing:
 - Host: <calculated when request is sent>
 - User-Agent: PostmanRuntime/7.28.0
 - Accept: */*
 - Accept-Encoding: gzip, deflate, br
 - Connection: keep-alive
 - x-access-token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
- Body** tab selected, showing the response body in **Pretty** format:

```

1
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJpZCI6IjYwOWVhZWM2ODcyYmVkJCJyYw1IjoiUm9iZXJ0byBQaW5oZWlybyIsImIhdCI6MTYyMTAxOTgxMSwiZXhwIjoxNjIxMTA2MjExfQ.4UaQISndRTxXM6HLzyr-TtsRFXtxfPiiUEAgG-_Eaw",
3   "data": {
4     "email": "betopinheiro1005@yahoo.com.br",
5     "name": "Roberto Pinheiro"
6   }
7 
```

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjYwOWVhZWM2ODcyYmVkJCJyYw1IjoiUm9iZXJ0byBQaW5oZWlybyIsImIhdCI6MTYyMTAxOTgxMSwiZXhwIjoxNjIxMTA2MjExfQ.4UaQISndRTxXM6HLzyr-TtsRFXtxfPiiUEAgG-_Eaw",
  "data": {
    "email": "betopinheiro1005@yahoo.com.br",
    "name": "Roberto Pinheiro"
  }
}
```

Aula 37 - Autorização

src\models\customer.js

```
'use strict';

const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const schema = new Schema({
  name: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true
  },
  password: {
    type: String,
    required: true
  },
  roles: [
    {
      type: String,
      required: true,
      enum: ['user', 'admin'],
      default: 'user'
    }
  ]
});

module.exports = mongoose.model('Customer', schema);
```

src\controllers\customer-controller.js

```
'use strict';

const ValidationContract = require('../validators/fluent-validator');
const repository = require('../repositories/customer-repository');
const md5 = require('md5');
const emailService = require('../services/email-service');

const authService = require('../services/auth-service');

exports.get = async(req, res, next) => {
  try {
    var data = await repository.get();
    res.status(200).send(data);
  } catch (e) {
    res.status(500).send({
      message: 'Falha ao processar sua requisição'
    });
  }
};
```

```

        }
    }

exports.post = async(req, res, next) => {
    let contract = new ValidationContract();
    contract.hasMinLen(req.body.name, 3, 'O nome deve conter pelo menos 3
caracteres');
    contract.isEmail(req.body.email, 'Email inválido');
    contract.hasMinLen(req.body.password, 6, 'A senha deve conter pelo menos 6
caracteres');

    // Se os dados forem inválidos
    if (!contract.isValid()) {
        res.status(400).send(contract.errors()).end();
        return;
    }

    try {
        await repository.create({
            name: req.body.name,
            email: req.body.email,
            password: md5(req.body.password + global.SALT_KEY),
            roles: ["user"]
        });

        emailService.send(
            req.body.email,
            'Bem vindo ao Node Store',
            global.EMAIL_TMPL.replace('{0}', req.body.name)
        );

        res.status(201).send({
            message: 'Cliente cadastrado com sucesso!'
        });
    } catch (e) {
        console.log(e);
        res.status(500).send({
            message: 'Falha ao processar sua requisição'
        });
    }
};

exports.authenticate = async(req, res, next) => {

    try {
        const customer = await repository.authenticate({
            email: req.body.email,
            password: md5(req.body.password + global.SALT_KEY)
        });

        if(!customer){
            res.status(404).send({
                message: 'Usuário ou senha inválidos!'
            });
            return;
        }
    }
};

```

```

const token = await authService.generateToken({
  id: customer._id,
  email: customer.email,
  name: customer.name
});

res.status(201).send({
  token: token,
  data: {
    email: customer.email,
    name: customer.name
  }
});
} catch (e) {
  console.log(e);
  res.status(500).send({
    message: 'Falha ao processar sua requisição'
  });
}
};

exports.refreshToken = async(req, res, next) => {
  try {
    const token = req.body.token || req.query.token || req.headers['x-access-token'];
    const data = await authService.decodeToken(token);

    const customer = await repository.getById(data.id);

    if (!customer) {
      res.status(404).send({
        message: 'Cliente não encontrado'
      });
      return;
    }

    const tokenData = await authService.generateToken({
      id: customer._id,
      email: customer.email,
      name: customer.name
    });

    res.status(201).send({
      token: token,
      data: {
        email: customer.email,
        name: customer.name
      }
    });
  } catch (e) {
    res.status(500).send({
      message: 'Falha ao processar sua requisição'
    });
  }
};

```

- No Studio 3T, exclua o cliente.

- Rode o servidor:

`nodemon ./bin/server.js`

- No Postman, crie um novo cliente(customer).

The screenshot shows the Postman interface with a POST request to `localhost:3000/customers`. The request body is set to `JSON` and contains the following data:

```
1 {  
2   "name": "Roberto Pinheiro",  
3   "email": "betopinheiro1005@yahoo.com.br",  
4   "password": "robertopinheiro"  
5 }
```

The response tab shows the following details:

- Status: 201 Created
- Time: 1720 ms
- Size: 285 B
- Save Response

The response body is:

```
1 {  
2   "message": "Cliente cadastrado com sucesso!"  
3 }
```

- No 3T Studio:

Key	Value	Type
<code>{} (1) {_id: 609fa26eb0d07e3930bea3fe}</code>	{ 6 fields }	Document
<code>__id</code>	609fa26eb0d07e3930bea3fe	ObjectId
<code>roles</code>	[1 elements]	Array
<code>0</code>	user	String
<code>name</code>	Roberto Pinheiro	String
<code>email</code>	betopinheiro1005@yahoo.com.br	String
<code>password</code>	891f095be7ed455ec084e1fc23a3bb21	String
<code>_v</code>	0	Int32

- Repare que o customer foi criado com role **user** (valor default).

- Para permitir que somente as pessoas com o role **admin** possam criar, editar ou excluir produtos, é necessário executar alguns passos a seguir:

src\services\auth-service.js

```
'use strict';
const jwt = require('jsonwebtoken');

exports.generateToken = async (data) => {
    return jwt.sign(data, global.SALT_KEY, { expiresIn: '1d' });
}

exports.decodeToken = async (token) => {
    var data = await jwt.verify(token, global.SALT_KEY);
    return data;
}

exports.authorize = function (req, res, next) {
    var token = req.body.token || req.query.token || req.headers['x-access-token'];

    if (!token) {
        res.status(401).json({
            message: 'Acesso Restrito'
        });
    } else {
        jwt.verify(token, global.SALT_KEY, function (error, decoded) {
            if (error) {
                res.status(401).json({
                    message: 'Token Inválido'
                });
            } else {
                next();
            }
        });
    }
};

exports.isAdmin = function (req, res, next) {
    var token = req.body.token || req.query.token || req.headers['x-access-token'];

    if (!token) {
        res.status(401).json({
            message: 'Token Inválido'
        });
    } else {
        jwt.verify(token, global.SALT_KEY, function (error, decoded) {
            if (error) {
                res.status(401).json({
                    message: 'Token Inválido'
                });
            } else {
                if (decoded.roles.includes('admin')) {
                    next();
                } else {
                    res.status(403).json({
                        message: 'Esta funcionalidade é restrita para administradores'
                    });
                }
            }
        });
    }
};
```

```
}; }
```

src\controllers\customer-controller.js

```
'use strict';

const ValidationContract = require('../validators/fluent-validator');
const repository = require('../repositories/customer-repository');
const md5 = require('md5');
const emailService = require('../services/email-service');

const authService = require('../services/auth-service');

exports.get = async(req, res, next) => {
    try {
        var data = await repository.get();
        res.status(200).send(data);
    } catch (e) {
        res.status(500).send({
            message: 'Falha ao processar sua requisição'
        });
    }
}

exports.post = async(req, res, next) => {
    let contract = new ValidationContract();
    contract.hasMinLen(req.body.name, 3, 'O nome deve conter pelo menos 3 caracteres');
    contract.isEmail(req.body.email, 'Email inválido');
    contract.hasMinLen(req.body.password, 6, 'A senha deve conter pelo menos 6 caracteres');

    // Se os dados forem inválidos
    if (!contract.isValid()) {
        res.status(400).send(contract.errors()).end();
        return;
    }

    try {
        await repository.create({
            name: req.body.name,
            email: req.body.email,
            password: md5(req.body.password + global.SALT_KEY),
            roles: ["user"]
        });

        emailService.send(
            req.body.email,
            'Bem vindo ao Node Store',
            global.EMAIL_TMPL.replace('{0}', req.body.name)
        );

        res.status(201).send({
            message: 'Cliente cadastrado com sucesso!'
        });
    }
}
```

```

} catch (e) {
    console.log(e);
    res.status(500).send({
        message: 'Falha ao processar sua requisição'
    });
}
};

exports.authenticate = async(req, res, next) => {

try {
    const customer = await repository.authenticate({
        email: req.body.email,
        password: md5(req.body.password + global.SALT_KEY)
    });

    if(!customer){
        res.status(404).send({
            message: 'Usuário ou senha inválidos!'
        });
        return;
    }

    const token = await authService.generateToken({
        id: customer._id,
        email: customer.email,
        name: customer.name,
        roles: customer.roles
    });

    res.status(201).send({
        token: token,
        data: {
            email: customer.email,
            name: customer.name
        }
    });
} catch (e) {
    console.log(e);
    res.status(500).send({
        message: 'Falha ao processar sua requisição'
    });
}
};

exports.refreshToken = async(req, res, next) => {
try {
    const token = req.body.token || req.query.token || req.headers['x-access-token'];
    const data = await authService.decodeToken(token);

    const customer = await repository.getById(data.id);

    if (!customer) {
        res.status(404).send({
            message: 'Cliente não encontrado'
        });
        return;
    }
}

```

```

    }

    const tokenData = await authService.generateToken({
      id: customer._id,
      email: customer.email,
      name: customer.name,
      roles: customer.roles
    });

    res.status(201).send({
      token: token,
      data: {
        email: customer.email,
        name: customer.name
      }
    });
  } catch (e) {
    res.status(500).send({
      message: 'Falha ao processar sua requisição'
    });
  }
}

```

src\routes\product-route.js

```

const express = require('express');
const router = express.Router();
const controller = require("../controllers/product-controller");
const authService = require('../services/auth-service');

router.get('/', controller.get);
router.get('/:slug', controller.getBySlug);
router.get('/admin/:id', controller.getId);
router.get('/tags/:tag', controller.getTag);
router.post('/', authService.isAdmin, controller.post);
router.put('/:id', authService.isAdmin, controller.put);
router.delete('/:id', authService.isAdmin, controller.delete);

module.exports = router;

```

- No Postman, crie um novo authenticate, para gerar um novo token:

The screenshot shows the Postman interface with a successful response. The request URL is `localhost:3000/customers/authenticate`. The body contains a JSON object with `"email": "betopinheiro1005@yahoo.com.br"` and `"password": "robertopinheiro"`. The response status is `201 Created`, and the response body is a JSON object with a `token` field and a `data` field containing the user's email and name.

```
1 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjYwOWZhMjZlYjBkMDdIMzkzMGIYTNmZSIsImVtYWlsIjoiYmV0b3BpbmhlaXJvMTAwNUB5YWhvby5jb20uYnIiLCJuYW1IjoiUm9iZXJ0byBQaW5oZWlybyIsInJvbGVzIjpbInVzZXIiXSwiaWF0IjoxNjIxMDc2MDE3LCJleHAiOjE2MjExNjI0MTd9.swywO9wsTEwvSyR0Ax9cSZPRCxzFEUPM5QLJZTbjAMU",
2 "data": {
3     "email": "betopinheiro1005@yahoo.com.br",
4     "name": "Roberto Pinheiro"
5 }
```

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjYwOWZhMjZlYjBkMDdIMzkzMGIYTNmZSIsImVtYWlsIjoiYmV0b3BpbmhlaXJvMTAwNUB5YWhvby5jb20uYnIiLCJuYW1IjoiUm9iZXJ0byBQaW5oZWlybyIsInJvbGVzIjpbInVzZXIiXSwiaWF0IjoxNjIxMDc2MDE3LCJleHAiOjE2MjExNjI0MTd9.swywO9wsTEwvSyR0Ax9cSZPRCxzFEUPM5QLJZTbjAMU",
  "data": {
    "email": "betopinheiro1005@yahoo.com.br",
    "name": "Roberto Pinheiro"
  }
}
```

- Tente cadastrar um novo produto. Copie e cole o token acima na key **x-access-token** (aba Headers) e em seguida clique no botão **Send**.

The screenshot shows the Postman interface with a failed response. The request URL is `localhost:3000/products`. The headers section includes a `x-access-token` header with the copied token value. The response status is `403 Forbidden`, and the response body is a JSON object with a `message` field containing the error message: `"Esta funcionalidade é restrita para administradores"`.

```
1 "message": "Esta funcionalidade é restrita para administradores"
```

- No 3T Studio edite o role do cliente (customer). Altere de **user** para **admin**:

Key	Value	Type
1 { _id : 609fbdee12d8e93dac35c2ee }	{ 6 fields }	Document
_id	609fbdee12d8e93dac35c2ee	ObjectId
roles	[1 elements]	Array
0	admin	String
name	Roberto Pinheiro	String
email	betopinheiro1005@yahoo.com.br	String
password	891f095be7ed455ec084e1fc23a3bb21	String
__v	0	Int32

- Na autenticação atual o customer é apenas user. Portanto, agora é necessário fazer uma nova autenticação para tratá-lo como admin:

```

POST /customers/authenticate
{
  "email": "betopinheiro1005@yahoo.com.br",
  "password": "robertopinheiro"
}
  
```

Body	Cookies	Headers (7)	Test Results
Pretty		Status: 201 Created Time: 278 ms Size: 612 B Save Response	
Raw			
JSON			
1 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjYwOWZiZGVIMTjkOGU5M2RhYzM1YzJlZSIIsImVtYWlsIjoiYmV0b3BpbmhlaXJvMTAwNUB5YWhvby5jb20uYnIiLCJuYW1IjoiUm9iZXJ0byBQaW5oZWlybyIsInJvbGVzIjpbImFkbWluIl0sImlhdCI6MTYyMTA4MTg5NSwiZXhwIjoxNjIxMTY4Mjk1fQ.rBOUV9GGF14NIqqMSWyWsWJV7ka0fuCWyVKYlhAuTdU",			
2 "data": {			
3 "email": "betopinheiro1005@yahoo.com.br",			
4 "name": "Roberto Pinheiro"			
5 }			
6 }			
7 }			

- Tente cadastrar um novo produto. Copie e cole o token acima na key **x-access-token** (aba **Headers**) e em seguida clique no botão **Send**.

POST localhost:3000/products

Body (JSON)

```

1 {
2   "title": "Mouse Gamer",
3   "description": "Mouse Gamer",
4   "slug": "mouse-gamer",
5   "price": 299,
6   "active": true,
7   "tags": [
8     "informática", "mouse", "games"
9   ]
10

```

POST localhost:3000/products

Headers

Key	Description
Content-Type	application/json
Content-Length	<calculated when request is sent>
Host	<calculated when request is sent>
User-Agent	PostmanRuntime/7.28.0
Accept	/*
Accept-Encoding	gzip, deflate, br
Connection	keep-alive
x-access-token	eyJhbGciOiJIUzI1NltsInR5cCI6IkpXVCJ9.eyJpZCI6IjYwOWZlZGVIM...

Body

```

1 {
2   "message": "Produto cadastrado com sucesso!"
3

```

Key	Value	Type
❶ (1) {_id : 609fc22f3d2e2b2d605b8ce3}	{ 8 fields }	Document
❷ _id	609fc22f3d2e2b2d605b8ce3	ObjectId
❸ active	true	Bool
❹ tags	[3 elements]	Array
❺ title	Mouse Gamer	String
❻ description	Mouse Gamer	String
❼ slug	mouse-gamer	String
❼ price	299	Int32
❼ __v	0	Int32

Aula 38 - Outros

src\app.js

```
const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');

const app = express();
const router = express.Router();

// Conecta ao banco
mongoose.connect("mongodb+srv://betopinheiro1005:angstron1005@node-store-cluster.l4jj0.mongodb.net/node-store-db?retryWrites=true&w=majority");

// Carrega os models
const Product = require('./models/product');
const Customer = require('./models/customer');
const Order = require('./models/order');

// Carrega as rotas
const indexRoute = require('./routes/index-route');
const productRoute = require('./routes/product-route');
const customerRoute = require('./routes/customer-route');
const orderRoute = require('./routes/order-route');

app.use(bodyParser.json({
  limit: '5mb'
}));

app.use(bodyParser.urlencoded({
  extended: false
}));

// Habilita o CORS
app.use(function (req, res, next) {
  res.header('Access-Control-Allow-Origin', '*');
  res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content-Type, Accept, x-access-token');
  res.header('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE, OPTIONS');
  next();
});

app.use('/', indexRoute);
app.use('/products', productRoute);
app.use('/customers', customerRoute);
app.use('/orders', orderRoute);

module.exports = app;
```