

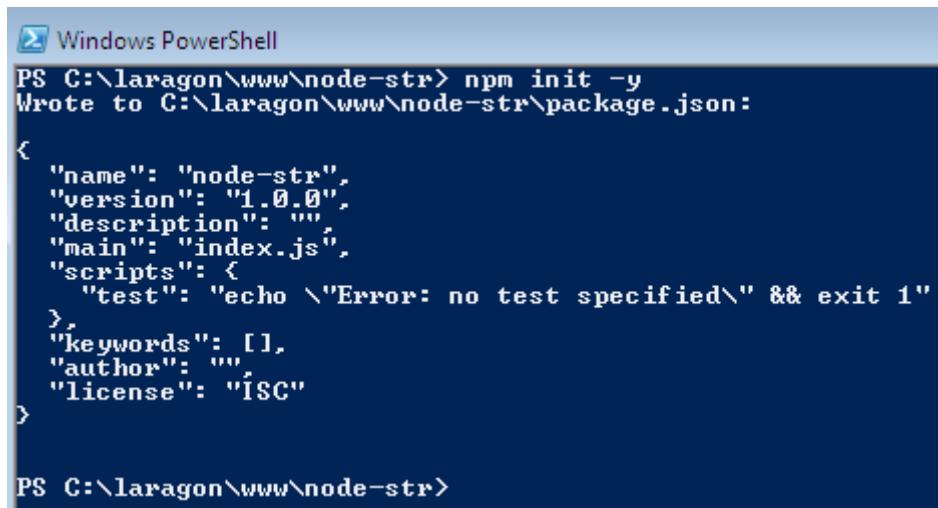
# Criando API's com Node.JS - Aulas 1 a 33

## balta.io (André Baltieri)

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

Resumo do curso feito por Roberto Pinheiro

## Aula 02 - npm init e instalação dos pacotes



```
Windows PowerShell
PS C:\laragon\www\node-str> npm init -y
Wrote to C:\laragon\www\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"
}

PS C:\laragon\www\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"
}
```

## Instalando pacotes básicos

O comando a seguir irá instalar os pacotes:

- http
- express
- debug

`npm install http express debug --save`

```
C:\laragon\www\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.

+ http@0.0.0
+ debug@4.1.1
+ express@4.17.1
added 59 packages from 37 contributors and audited 129 packages in 22.898s
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": "^4.1.1",
    "express": "^4.17.1",
    "http": "0.0.0"
  }
}
```

## Aula 03 - Criando um servidor web

### server.js

```
'use strict'

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

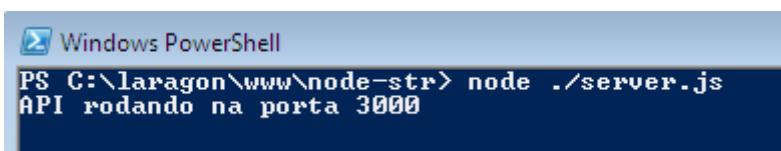
const app = express();
const port = 3000;
app.set('port', port);

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

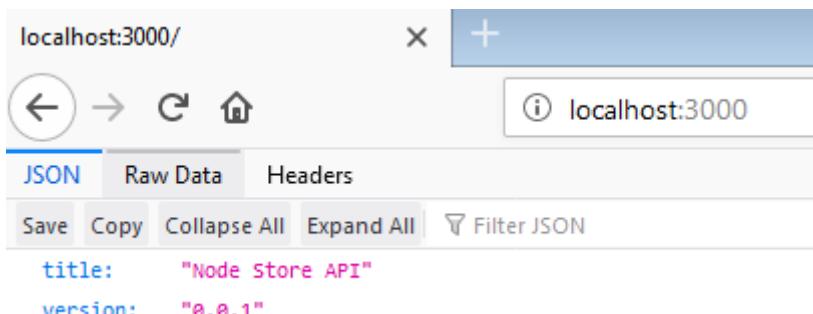
const route = router.get('/', (req, res, next) => {
  res.status(200).send({
    title: "Node Store API",
    version: "0.0.1"
  });
});

app.use('/', route);

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



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



## Usando o Postman:

The screenshot shows the Postman application interface. At the top, there's a menu bar with File, Edit, View, Help, and a toolbar with New, Import, Runner, My Workspace, Invite, and other icons. The main workspace shows a request URL of GET localhost:3000. Below the URL, the method is set to GET and the target is localhost:3000. The Params tab is selected, showing a single parameter: Key (Value) and Description (Description). The status bar at the bottom indicates a 200 OK response with 110 ms time and 256 B size.

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Status: 200 OK Time: 110 ms Size: 256 B

```
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);

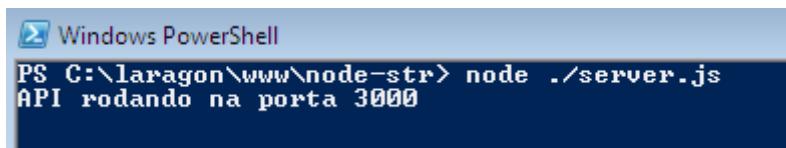
server.listen(port);
console.log('API rodando na porta ' + port);

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

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

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

  return false;
}
```



```
Windows PowerShell
PS C:\laragon\www\node-str> node ./server.js
API rodando na porta 3000
```

The screenshot shows the Postman application interface. At the top, there's a header bar with a green 'GET' button, the URL 'localhost:3000', and three small icons. To the right of the URL is a dropdown menu set to 'No Environment'. Below the header is a toolbar with a blue 'Send' button, a 'Save' button, and other icons.

The main workspace has a title 'localhost:3000'. Underneath it, a request card is displayed with 'GET' selected, the URL 'localhost:3000', and a 'Send' button. Below the request card is a table titled 'Params' with columns for KEY, VALUE, and DESCRIPTION. A single row is present with 'Key' in the KEY column and 'Value' in the VALUE column. There are also 'Cookies', 'Code', and 'Comments (0)' tabs to the right of the table.

Below the table, there are tabs for 'Body', 'Cookies', 'Headers (6)', and 'Test Results'. The 'Body' tab is active, showing a status of 'Status: 200 OK', a time of 'Time: 114 ms', and a size of 'Size: 256 B'. To the right of these status details are 'Download' and search/filter icons.

The 'Body' section contains a code editor with the following JSON response:

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

## 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;  
}  
}
```

## 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);
}
```

## Aula 07 - Separando o servidor

### bin/server.js

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

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

const server = http.createServer(app);

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);
}
```

### **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;
```

```
C:\laragon\www\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

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

# Aula 09 - Nodemon

## Instalando o pacote nodemon

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

```
C:\laragon\www\node-str>npm install nodemon --save-dev
> nodemon@1.19.1 postinstall C:\laragon\www\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 node-str@1.0.0 No description
npm WARN node-str@1.0.0 No repository.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"ia32"})
+ nodemon@1.19.1
added 220 packages from 128 contributors and audited 2398 packages in 84.533s
found 0 vulnerabilities
```

## package.json

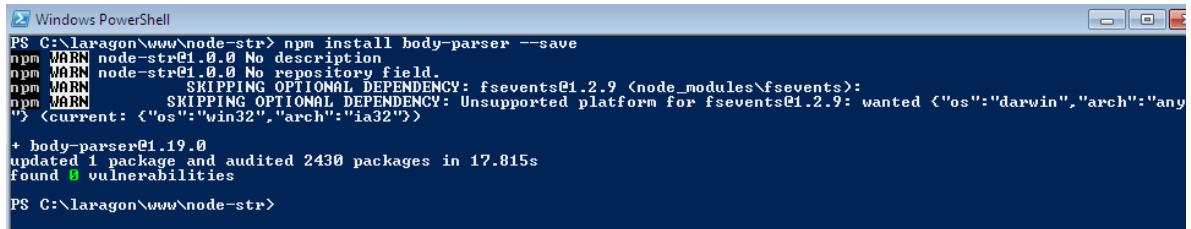
```
{
  "name": "node-str",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node ./bin/server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "debug": "^4.1.1",
    "express": "^4.17.1",
    "http": "0.0.0"
  },
  "devDependencies": {
    "nodemon": "^1.19.1"
  }
}
```

```
C:\laragon\www\node-str>nodemon ./bin/server.js
[nodemon] 1.19.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: ***!
[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:\laragon\www\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           SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\fsevents):
npm WARN             SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":any}
" > (current: {"os":"win32","arch":"ia32"})
+ body-parser@1.19.0
updated 1 package and audited 2430 packages in 17.815s
found 0 vulnerabilities

PS C:\laragon\www\node-str>
```

## 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);
});

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 GET

The screenshot shows the Postman application interface. At the top, there are buttons for 'New', 'Import', 'Runner', and 'My Workspace'. Below the header, there are tabs for 'GET Untitled Request' and 'POST localhost:3000/products'. The 'POST' tab is selected. To the right of the tabs are buttons for '+', '...', 'No Environment', and 'Sign In'. The main area shows the URL 'localhost:3000/products'. Below the URL, there is a dropdown for 'POST' and a field for 'localhost:3000/products'. On the right, there are 'Send' and 'Save' buttons. Under the 'Body' tab, the 'JSON (application/json)' option is selected. The JSON payload is defined as:

```
1 [{}  
2 "title": "teste"  
3]}
```

The screenshot shows the Postman application interface after a POST request has been sent. The interface is identical to the previous screenshot, but the status bar at the bottom now displays the results of the request. The status is 'Status: 201 Created', the time taken is 'Time: 604 ms', and the size of the response is 'Size: 234 B'. Below the status bar, there is a 'Download' button. The 'Body' tab is selected, showing the JSON response:

```
1 [{}  
2 "title": "teste"  
3]}
```

## put → método PUT

The screenshot shows the Postman application interface. At the top, there are tabs for 'GET Untitled Request' and 'PUT localhost:3000/products/123'. The 'PUT' tab is selected. To the right of the tabs are buttons for '+', '...', 'No Environment', and settings. Below the tabs, the URL 'localhost:3000/products/123' is displayed. On the left, there are dropdown menus for 'PUT' and 'localhost:3000/products/123'. To the right of these are 'Send' and 'Save' buttons. Below the URL, there are tabs for 'Params', 'Authorization', 'Headers (1)', 'Body (1)', 'Pre-request Script', and 'Tests'. The 'Body (1)' tab is selected and highlighted in red. Under 'Body' options, 'raw' is selected and highlighted in green. The 'JSON (application/json)' type is chosen. To the right of the body tab are 'Cookies', 'Code', and 'Comments (0)'. Below the tabs, the JSON payload is shown in a code editor:

```
1 [{}  
2   "title": "teste"  
3 ]|
```

At the bottom of the interface, there are tabs for 'Body', 'Cookies', 'Headers (6)', and 'Test Results'. The 'Body' tab is selected. To the right, status information is displayed: 'Status: 200 OK', 'Time: 135 ms', 'Size: 249 B', and a 'Download' button. Below these tabs, there are buttons for 'Pretty', 'Raw', 'Preview', and 'JSON'. The 'JSON' button is selected and highlighted in green. To the right of these buttons are icons for copy and search. The JSON response is displayed in a code editor:

```
1 [{}  
2   "id": "123",  
3   "item": {  
4     "title": "teste"  
5   }  
6 ]|
```

## delete → método DELETE

The screenshot shows the Postman application interface. At the top, there are three tabs: 'GET Untitled Request' (disabled), 'DEL localhost:3000/products/' (selected), and 'No Environment'. Below the tabs, the URL 'localhost:3000/products/' is displayed. The method dropdown shows 'DELETE'. The 'Send' and 'Save' buttons are visible. The 'Body' tab is selected, showing a JSON payload:

```
1 [{}]
2   "title": "teste"
3 }
```

Below the body, the status bar shows 'Status: 200 OK' and 'Time: 117 ms'. The 'Body' tab is active, and the JSON response 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: "0.0.1"
  });
});

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;
```

## Aula 13 - MongoDB Setup

- Acesse <https://mlab.com> e abra uma conta.
- Crie uma organização com nome: **Orion3**
- Crie um novo projeto chamado: **proj-node-store**

ORION3 > PROJECTS

### Create a Project

Name Your Project > Add Members Next

#### Name Your Project

Project names have to be unique within the organization (and other restrictions).

CancelNext

### Create a cluster

Choose your cloud provider, region, and specs.

Build a Cluster

Once your cluster is up and running, live migrate an existing MongoDB database into Atlas with our [Live Migration Service](#).

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

# Configuração

Cloud Provider & Region

Azure, Virginia (eastus2) ▾



Create a **free tier cluster** by selecting a region with **FREE TIER AVAILABLE** and choosing the M0 cluster tier below.

★ Recommended region ⓘ

NORTH AMERICA		EUROPE		ASIA	
Iowa (centralus)	★	Ireland (northeurope)	★	Pune (centralindia)	
Virginia (eastus)	★	Netherlands (westeurope)	★	Mumbai (westindia)	
Virginia (eastus2)	★	London (uksouth)		Chennai (southindia)	
Illinois (northcentralus)	★	Cardiff (ukwest)		Hong Kong (eastasia)	<b>FREE TIER AVAILABLE</b>
California (westus)	★	Paris (francecentral)		Tokyo (japaneast)	
Texas (southcentralus)	★	SOUTH AMERICA		Osaka (japanwest)	
Washington (westus2)		Sao Paulo (brazilsouth)		Singapore (southeastasia)	

Cluster Tier

M0 Sandbox (Shared RAM, 512 MB Storage)

Encrypted

Base hourly rate is for a MongoDB replica set with **3 data bearing servers**.

**Shared** Clusters for development environments and low-traffic applications

Tier	RAM	Storage	vCPU	Base Price
M0 Sandbox	Shared	512 MB	Shared	<b>Free forever</b>
M0 clusters are best for getting started, and are not suitable for production environments.				
100 max connections	Low network performance	100 max databases	500 max collections	
M2	Shared	2 GB	Shared	\$9 / MONTH
M5	Shared	5 GB	Shared	\$25 / MONTH

**Dedicated** Clusters for development environments and low-traffic applications

Tier	RAM	Storage	vCPU	Base Price
M10	2 GB	32 GB	1 vCPU	from \$0.11/hr
M20	4 GB	32 GB	2 vCPUs	from \$0.22/hr

**Additional Settings**

MongoDB 4.0, No Backup ▾

Select a Version

All clusters launch with the WiredTiger™ storage engine.

MongoDB 4.0

Turn on Backup (M2 and up)

See Backup Solutions for Paid Clusters (M2+)

Advanced Settings

Shard your cluster (M30 and up)

Sharding supports high throughput and large datasets, and can be increased as data requirements grow. Sharded clusters cannot be converted to replica sets.

Enable Business Intelligence Connector (M10 and up)

The BI Connector allows you to visualize your data on relational business intelligence tools (e.g. Tableau, MicroStrategy, Qlik).

**Cluster Name** node-store-cluster

One time only: once your cluster is created, you won't be able to change its name.

node-store-cluster

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):

ORIONS > PROJ-NODE-STORE

**Clusters**

Build a New Cluster

Find a cluster...

**SANDBOX**

node-store-cluster

Version 4.0.12

CONNECT METRICS COLLECTIONS ...

CLUSTER TIER

M0 Sandbox (General)

REGION

Azure / Virginia (eastus2)

TYPE

Replica Set - 3 nodes

LINKED STITCH APP

None Linked

Operations R: 0 W: 0 100.0/s Last 6 Hours

Logical Size 0.0 B 512.0 MB max 0.0 B Last 6 Hours

Connections 0 100 max Last 6 Hours

Enhance Your Experience

For dedicated throughput, richer metrics and enterprise security options, upgrade your cluster now!

Upgrade

- Crie um usuário para acessar o banco de dados.

### Add New User

**SCRAM Authentication**  
SCRAM is MongoDB's default authentication method.

betopinheiro1005	
e.g. new-user_31	
*****	SHOW
<input type="button" value="Autogenerate Secure Password"/>	

**User Privileges**

Atlas admin	Read and write to any database	Only read any database	Select Custom Role
-------------	--------------------------------	------------------------	--------------------

[Add Default Privileges](#)

Save as temporary user Cancel Add User

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

ORION2 > PROJ-NODE-STORE > CLUSTERS

**node-store-cluster**

Overview	Real Time	Metrics	Collections	Command Line Tools
----------	-----------	---------	-------------	--------------------

DATABASES: 0 COLLECTIONS: 0

 Interact with your data  
Run queries, view metadata about your collections, manage indexes, and interact with your data with full CRUD functionality.

Load a Sample Dataset   
More information

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

ORION2 > PROJ-NODE-STORE > CLUSTERS

**node-store-cluster**

Overview	Real Time	Metrics	Collections	Command Line Tools
----------	-----------	---------	-------------	--------------------

DATABASES: 1 COLLECTIONS: 1

NAMESPACES

**node-store-db.products**  
COLLECTION SIZE: 0B TOTAL DOCUMENTS: 0 INDEXES TOTAL SIZE: 4KB

[Find](#) [Indexes](#) [Aggregation](#)

{"filter": "example"}

QUERY RESULTS 0

- 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'. At the top, there is a navigation bar with three items: 'Setup connection security' (with a green checkmark), 'Choose a connection method' (the current active step), and 'Connect'. Below the navigation bar, the title 'Choose a connection method' is followed by a link 'View documentation'. A descriptive text states: 'See methods to add data and diagnostics in the Command Line Tools shortcut from within your cluster.' Three connection options are listed in cards:

- Connect with the Mongo Shell**: 'Mongo Shell with TLS/SSL support is required'.
- Connect Your Application**: 'Get a connection string and view driver connection examples'.
- Connect with MongoDB Compass**: 'Download Compass to explore, visualize, and manipulate your data'.

At the bottom of the screen are two buttons: 'Go Back' on the left and 'Close' on the right.

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

- Copie a string de conexão:

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

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

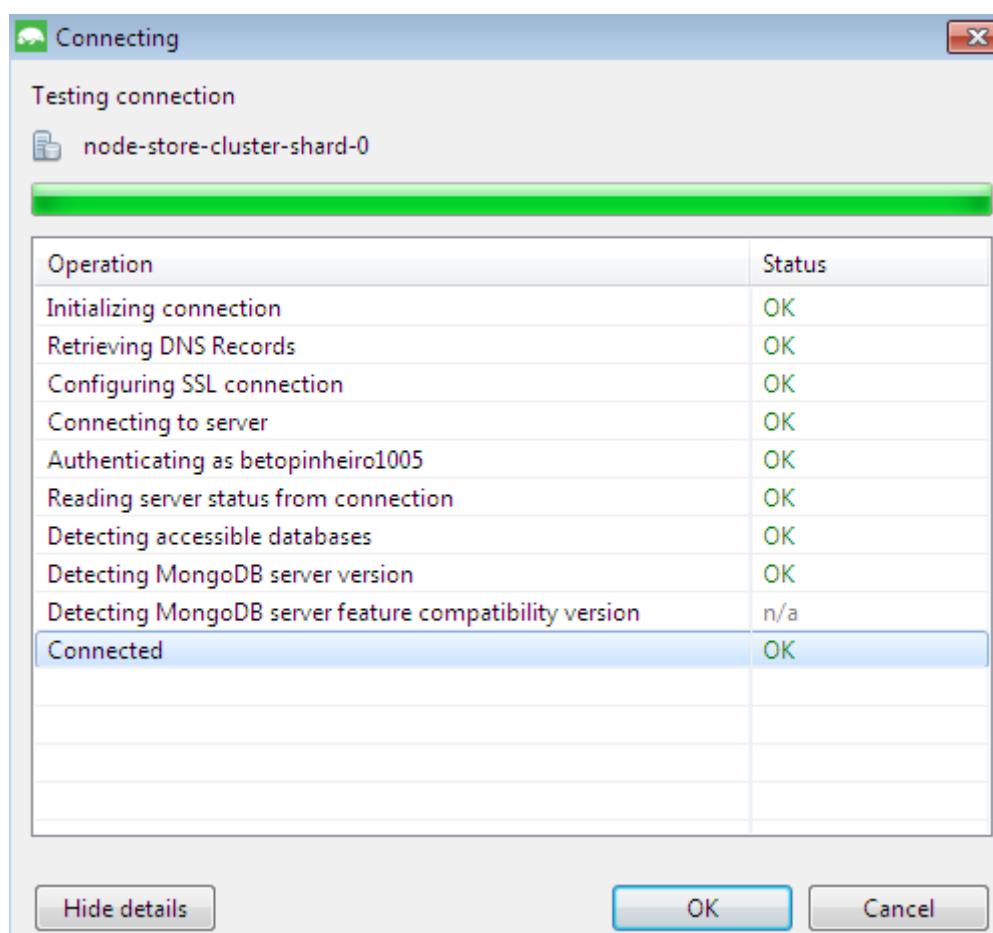
- 
- Baixe e instale o programa **Studio 3T**.

- Opção de outra interface gráfica:

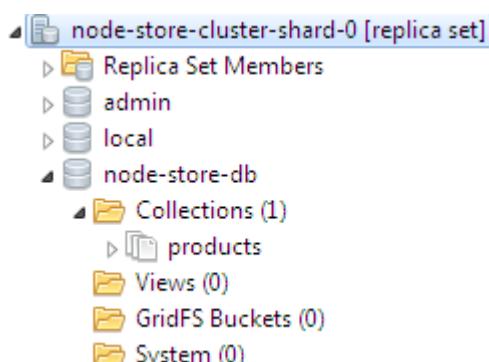
RoboMongo

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

- Dê o nome para a conexão: **conn-node-store**.
- Clique no botão **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**"



# Aula 14 - Mongoose

## Instalação do Mongoose

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

```
C:\laragon\www\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@1.2.9 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"ia32"})
+ mongoose@5.6.9
added 21 packages from 17 contributors and audited 2469 packages in 47.395s
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": "^4.1.1",
    "express": "^4.17.1",
    "http": "0.0.0",
    "mongoose": "^5.6.9"
  },
  "devDependencies": {
    "nodemon": "^1.19.1"
  }
}
```

## 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:<password>@node-store-cluster-nlcnv.azure.mongodb.net/node-store-db?retryWrites=true&w=majority", {
useNewUrlParser: true });

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

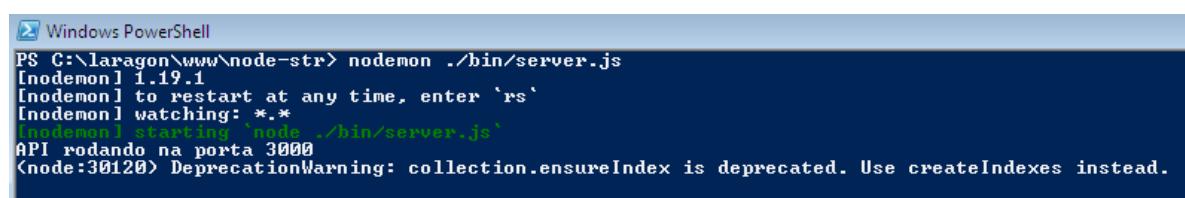
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({
  extended: false
}));

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

module.exports = app;
```

**Atenção:** Na string de conexão, substitua <password> pela senha cadastrada na mlab para esse banco de dados.

Se estiver tudo ok, o servidor irá rodar normalmente:



```
PS C:\laragon\www\node-str> nodemon ./bin/server.js
[nodemon] 1.19.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting 'node ./bin/server.js'
API rodando na porta 3000
(node:30120) DeprecationWarning: collection.ensureIndex is deprecated. Use createIndexes instead.
```

## 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/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-9kvu.mongodb.net/test?retryWrites=true&w=majority", { useNewUrlParser: true });

// 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;
```

POST localhost:3000/products

Params Authorization Headers (1) **Body** Pre-request Script Tests

none form-data x-www-form-urlencoded raw binary **JSON (application/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 }

```

Body Cookies Headers (6) Test Results

Pretty Raw Preview JSON

```

1 [
2   "message": "Produto cadastrado com sucesso!"
3 ]

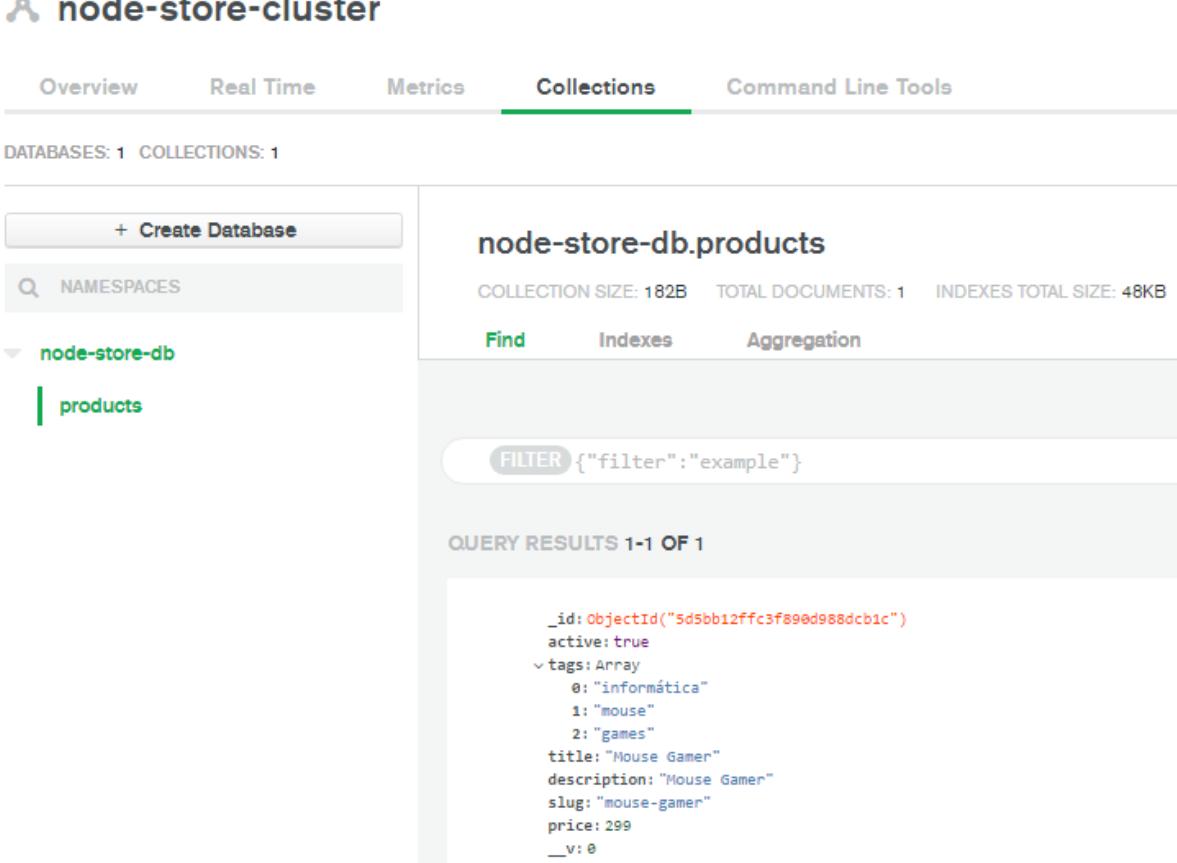
```

## No Studio 3T:

- node-store-cluster-shard-0 [replica set]
  - Replica Set Members
  - admin
  - local
  - node-store-db
    - Collections (1)
      - products
    - Views (0)
    - GridFS Buckets (0)
    - System (0)

Result	Query	Code	Explain
Documents 1 to 1			
Key	Value	Type	
(1) {_id : 5d5bb12ffc3f890d988dc1c}	{ 8 fields }	Document	
_id	5d5bb12ffc3f890d988dc1c	ObjectId	
active	true	Bool	
tags	[ 3 elements ]	Array	
title	Mouse Gamer	String	
description	Mouse Gamer	String	
slug	mouse-gamer	String	
price	299	Int32	
__v	0	Int32	

No mlab:

The screenshot shows the MongoDB Compass interface. At the top, there are tabs for Overview, Real Time, Metrics, Collections (which is highlighted in green), and Command Line Tools. Below the tabs, it says "DATABASES: 1" and "COLLECTIONS: 1". On the left sidebar, there is a "+ Create Database" button and a "NAMESPACES" search bar. Under "NAMESPACES", there is a tree view with a node labeled "node-store-db" expanded, showing a single collection named "products". In the main panel, the title is "node-store-db.products". It displays "COLLECTION SIZE: 182B", "TOTAL DOCUMENTS: 1", and "INDEXES TOTAL SIZE: 48KB". Below this, there are three tabs: "Find" (which is green), "Indexes", and "Aggregation". A "FILTER" button with the value {"filter": "example"} is present. The "QUERY RESULTS 1-1 OF 1" section shows a single document:

```
_id: ObjectId("5d5bb12ffc3f890d988dcb1c")
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

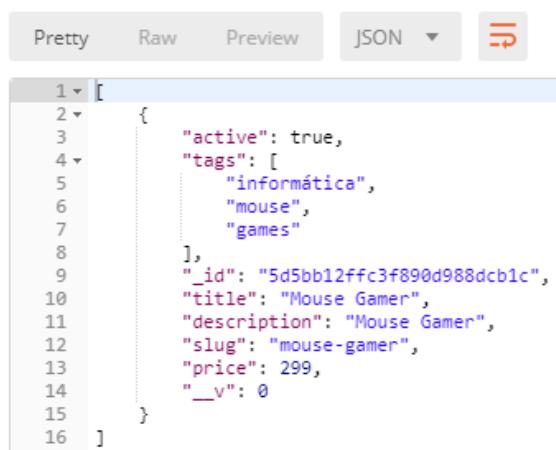
```
const express = require('express');
const router = express.Router();
const controller = require("../controllers/product-controller");

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

module.exports = router;
```

## Testando no Postman

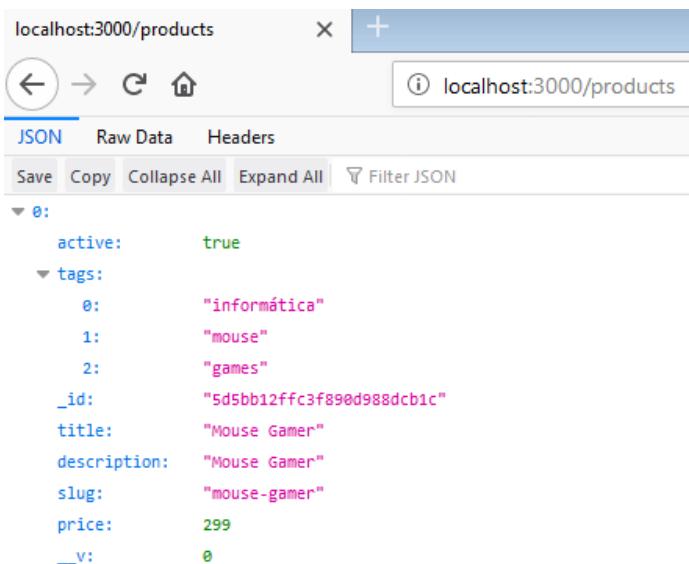
GET - localhost:3000/products



The screenshot shows the Postman application interface. At the top, there are tabs for 'Pretty', 'Raw', 'Preview', 'JSON' (which is selected), and a copy icon. Below the tabs, the JSON response is displayed in a code editor-like format with line numbers from 1 to 16. The response is an array containing a single object representing a product.

```
1 [
2   {
3     "active": true,
4     "tags": [
5       "informática",
6       "mouse",
7       "games"
8     ],
9     "_id": "5d5bb12fffc3f890d988dcb1c",
10    "title": "Mouse Gamer",
11    "description": "Mouse Gamer",
12    "slug": "mouse-gamer",
13    "price": 299,
14    "__v": 0
15  }
16 ]
```

## No navegador



The screenshot shows a browser window with the URL 'localhost:3000/products'. The page title is 'localhost:3000/products'. Below the address bar, there are buttons for back, forward, refresh, and home. The main content area shows the JSON response in a collapsible tree view. The response is identical to the one shown in the Postman screenshot.

```
localhost:3000/products
```

```
0:
  active: true
  tags:
    0: "informática"
    1: "mouse"
    2: "games"
  _id: "5d5bb12fffc3f890d988dcb1c"
  title: "Mouse Gamer"
  description: "Mouse Gamer"
  slug: "mouse-gamer"
  price: 299
  __v: 0
```

## 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);
};
```

The screenshot shows a JSON viewer with the following interface elements:

- Top navigation: Pretty, Raw, Preview, JSON (selected), and a copy icon.
- Document structure:
  - Line 1: [
  - Line 2: {
  - Line 3: "id": "5d5bb12ffc3f890d988dcb1c",
  - Line 4: "title": "Mouse Gamer",
  - Line 5: "slug": "mouse-gamer",
  - Line 6: "price": 299
  - Line 7: }
  - Line 8: ]

localhost:3000/products      X      +

← → ⌂ ⌂

localhost:3000/products

JSON Raw Data Headers

Save Copy Collapse All Expand All Filter JSON

▼ 0:

```
_id: "5d5bb12fffc3f890d988dcb1c"
title: "Mouse Gamer"
slug: "mouse-gamer"
price: 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

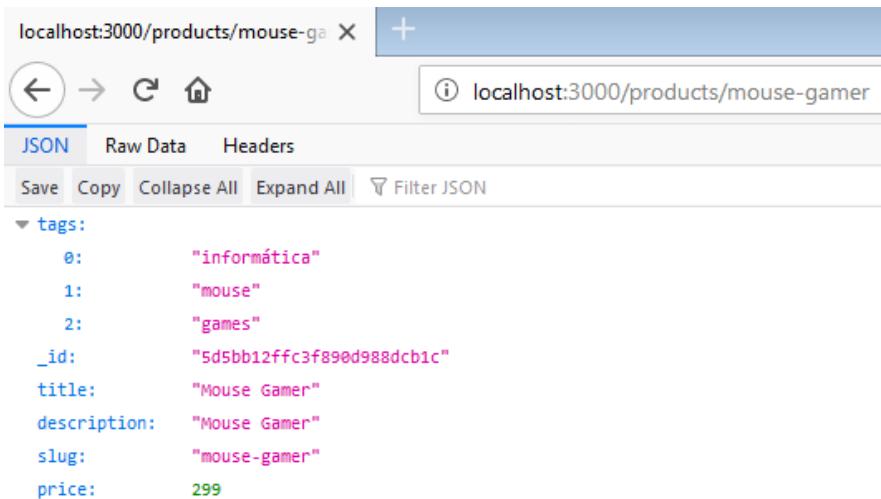
GET - localhost:3000/products/mouse-gamer



The screenshot shows the Postman application interface. At the top, there are tabs for 'Pretty', 'Raw', 'Preview', 'JSON' (which is selected), and a share icon. Below the tabs, the JSON response is displayed in a code editor-like format:

```
1  {
2    "tags": [
3      "informática",
4      "mouse",
5      "games"
6    ],
7    "_id": "5d5bb12ffc3f890d988dcb1c",
8    "title": "Mouse Gamer",
9    "description": "Mouse Gamer",
10   "slug": "mouse-gamer",
11   "price": 299
12 }
```

## No navegador



The screenshot shows a browser window with the URL 'localhost:3000/products/mouse-gamer'. The page content is identical to the Postman response, displayed as JSON:

```
localhost:3000/products/mouse-gamer
+  

← → ⌂ ⌄ ① localhost:3000/products/mouse-gamer  

JSON Raw Data Headers  

Save Copy Collapse All Expand All Filter JSON  

tags:  

  0: "informática"  

  1: "mouse"  

  2: "games"  

_id: "5d5bb12ffc3f890d988dcb1c"  

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

module.exports = router;

```

## **Testando no Postman**

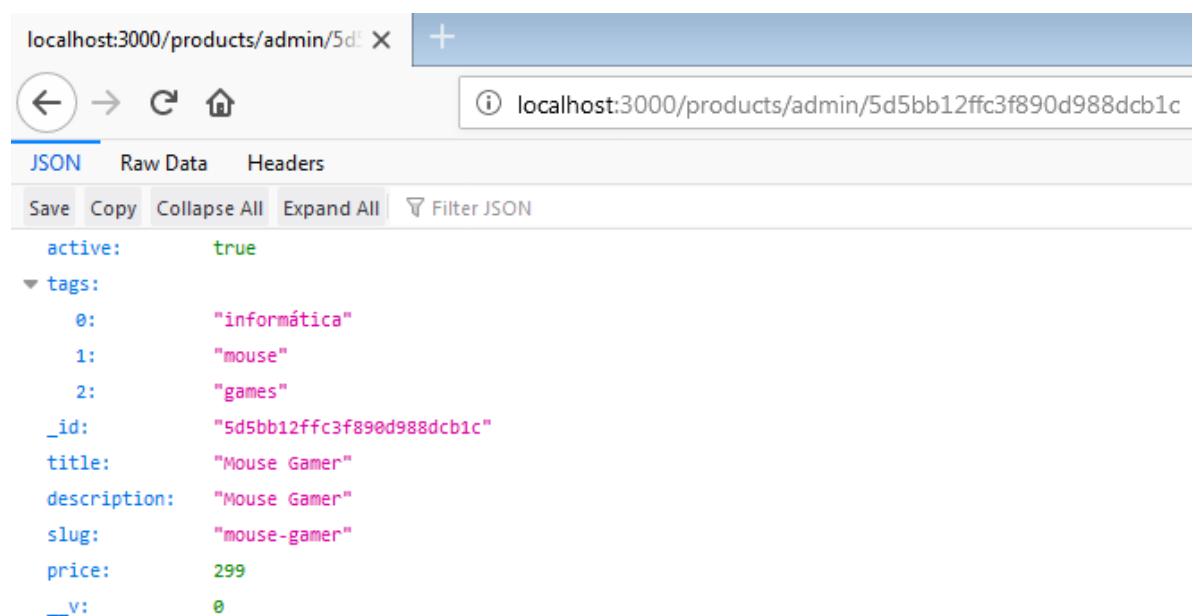
GET - localhost:3000/products/admin/5d5bb12ffc3f890d988dcb1c

```

Pretty Raw Preview JSON ↗
1 [
2   "active": true,
3   "tags": [
4     "informática",
5     "mouse",
6     "games"
7   ],
8   "_id": "5d5bb12ffc3f890d988dcb1c",
9   "title": "Mouse Gamer",
10  "description": "Mouse Gamer",
11  "slug": "mouse-gamer",
12  "price": 299,
13  "__v": 0
14 ]

```

## Testando no navegador



A screenshot of a web browser window displaying a JSON object. The URL in the address bar is `localhost:3000/products/admin/5d5bb12ffc3f890d988dcb1c`. The browser interface includes standard navigation buttons (back, forward, refresh, home) and a toolbar with `Save`, `Copy`, `Collapse All`, `Expand All`, and `Filter JSON` buttons. The JSON data is shown in a hierarchical tree view:

```
active: true
tags:
  0: "informática"
  1: "mouse"
  2: "games"
_id: "5d5bb12ffc3f890d988dcb1c"
title: "Mouse Gamer"
description: "Mouse Gamer"
slug: "mouse-gamer"
price: 299
__v: 0
```

## 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**



```

Pretty Raw Preview JSON ↗
1 [ 
2   { 
3     "tags": [
4       "informática",
5       "mouse",
6       "games"
7     ],
8     "_id": "5d5bb12fffc3f890d988dcb1c",
9     "title": "Mouse Gamer",
10    "description": "Mouse Gamer",
11    "slug": "mouse-gamer",
12    "price": 299
13  }
14 ]

```

## 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 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) => {
  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/

The screenshot shows the Postman application interface. At the top, there's a header bar with the URL "localhost:3000/products/5d58f19dfcfa037570f86962". Below it, the main interface has several tabs: "PUT", "localhost:3000/products/5d58f19dfcfa037570f86962", "Send", "Save", "Params", "Authorization", "Headers (1)", "Body", "Pre-request Script", "Tests", "Cookies", "Code", "Comments (0)". The "Body" tab is selected and contains a JSON editor. The JSON payload is:

```

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

```

Below the JSON editor, there are tabs for "Body", "Cookies", "Headers (6)", and "Test Results". The "Test Results" tab is selected and shows the response details: "Status: 200 OK", "Time: 1232 ms", "Size: 257 B", and a "Download" button. The response body is displayed in a JSON viewer:

```

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

```

## No Studio 3T

The screenshot shows the MongoDB Studio 3T interface. At the top, there are tabs for 'Result', 'Query Code', and 'Explain'. Below the tabs, there are navigation icons for back, forward, search, and other database operations. A status bar indicates '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

The screenshot shows the 'node-store-cluster' interface. At the top, there are tabs for 'Overview', 'Real Time', 'Metrics', 'Collections' (which is highlighted in green), and 'Command Line Tools'. Below the tabs, it displays 'DATABASES: 1' and 'COLLECTIONS: 1'.

On the left, there is a sidebar with a 'Create Database' button, a 'Namespaces' search input, and a tree view showing a database named 'node-store-db' with a collection named 'products' selected.

The main area is titled 'node-store-db.products' and shows collection details: 'COLLECTION SIZE: 193B', 'TOTAL DOCUMENTS: 1', and 'INDEXES TOTAL SIZE: 60KB'. It has tabs for 'Find', 'Indexes', and 'Aggregation'. A 'FILTER' button with the value '{ "filter": "example" }' is present.

The 'QUERY RESULTS 1-1 OF 1' section displays the document structure:

```
_id: ObjectId("5d5bb12ffc3f890d988dcb1c")
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.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) => {
  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");

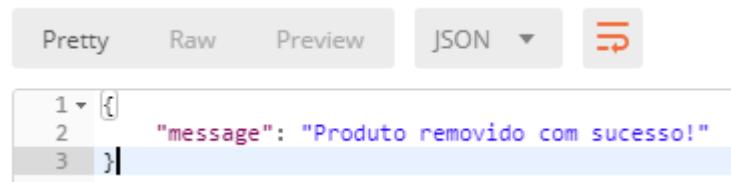
router.get('/', controller.get);
router.get('/:slug', controller.getBySlug);
router.get('/admin/:id', controller.getId);
router.get('/tags/:tag', controller.getTag);
router.post('/', controller.post);
router.put('/:id', controller.put);
router.delete('/:id', controller.delete);

module.exports = router;

```

## Testando no Postman

DELETE - localhost:3000/products/5d5bb12ffc3f890d988dcb1c



The screenshot shows the Postman interface with a JSON response. The response is a single object with a 'message' key containing the value 'Produto removido com sucesso!'. The JSON tab is selected at the top.

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

## 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 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;
}

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
        });
    });
};
```

## Testando no Postman

The screenshot shows the Postman application interface. At the top, there's a header with 'POST' selected, the URL 'localhost:3000/products', and buttons for 'Send' and 'Save'. Below the header, the 'Body' tab is active, showing a JSON payload:

```
1 ▾ {  
2   "title": "",  
3   "description": "er",  
4   "slug": "#",  
5   "price": 299,  
6   "active": true,  
7   "tags": ["informática", "mouse", "games"]  
8 }
```

Below the body, the status bar indicates 'Status: 400 Bad Request' and 'Time: 936 ms'. The main content area displays the error response in JSON format:

```
1 ▾ [  
2   {  
3     "message": "O titulo deve conter pelo menos 3 caracteres"  
4   },  
5   {  
6     "message": "O titulo deve conter pelo menos 3 caracteres"  
7   },  
8   {  
9     "message": "O titulo deve conter pelo menos 3 caracteres"  
10  }  
11 ]
```

## Aula 24 - Repositórios

### 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 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 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;
  }

  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
      });
    });
};
```

## Cadastrando um produto

POST - localhost:3000/products

The screenshot shows the Postman application interface. At the top, there is a header bar with the method 'POST' and the URL 'localhost:3000/products'. Below the header, there are tabs for 'Params', 'Authorization', 'Headers (1)', 'Body', 'Pre-request Script', and 'Tests'. The 'Body' tab is currently selected, indicated by a red underline. Under the 'Body' tab, there are options for 'none', 'form-data', 'x-www-form-urlencoded', 'raw', 'binary', and 'JSON (application/json)', with 'JSON (application/json)' being selected. The 'raw' section contains a JSON object representing a product:

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

At the bottom of the 'Body' tab, there are buttons for 'Pretty', 'Raw', 'Preview', and 'JSON'. The 'JSON' button is selected, indicated by a red underline. The 'Raw' section displays the same JSON object:

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

## Cadastrando um segundo produto

POST - localhost:3000/products

The screenshot shows the Postman interface for a POST request to `localhost:3000/products`. 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   "active": true,  
7   "tags": [  
8     "informatica", "mouse", "games"  
9   ]  
10 }
```

The response body is:

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

## Listando os produtos

GET - localhost:3000/products

The screenshot shows the Postman interface for a GET request to `localhost:3000/products`. The 'JSON' tab is selected, displaying the following response:

```
1 [  
2   {  
3     "_id": "5d5c65f7cc81b832f8f37dd8",  
4     "title": "Mouse Gamer",  
5     "slug": "mouse-gamer",  
6     "price": 299  
7   },  
8   {  
9     "_id": "5d5c6627cc81b832f8f37dd9",  
10    "title": "Cadeira Gamer",  
11    "slug": "cadeira-gamer",  
12    "price": 1299  
13  }  
14 ]
```

Exibindo dados de um produto por slug

GET - localhost:3000/products/mouse-gamer

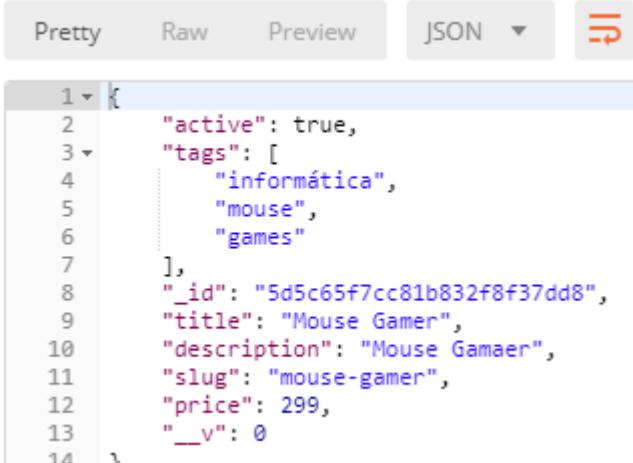


The screenshot shows a JSON viewer interface with tabs for 'Pretty', 'Raw', 'Preview', and 'JSON'. The 'JSON' tab is selected. The data is presented in a numbered, indented JSON format:

```
1 {
2   "tags": [
3     "informática",
4     "mouse",
5     "games"
6   ],
7   "_id": "5d5c65f7cc81b832f8f37dd8",
8   "title": "Mouse Gamer",
9   "description": "Mouse Gamaer",
10  "slug": "mouse-gamer",
11  "price": 299
12 }
```

Exibindo dados de um produto por id

localhost:3000/products/admin/ 5d5c65f7cc81b832f8f37dd8



The screenshot shows a JSON viewer interface with tabs for 'Pretty', 'Raw', 'Preview', and 'JSON'. The 'JSON' tab is selected. The data is presented in a numbered, indented JSON format:

```
1 {
2   "active": true,
3   "tags": [
4     "informática",
5     "mouse",
6     "games"
7   ],
8   "_id": "5d5c65f7cc81b832f8f37dd8",
9   "title": "Mouse Gamer",
10  "description": "Mouse Gamaer",
11  "slug": "mouse-gamer",
12  "price": 299,
13  "__v": 0
14 }
```

## Exibindo produtos por tag

GET - localhost:3000/products/tags/games

```
Pretty Raw Preview JSON ↗  
1 [  
2 {  
3   "tags": [  
4     "informática",  
5     "mouse",  
6     "games"  
7   ],  
8   "_id": "5d5c65f7cc81b832f8f37dd8",  
9   "title": "Mouse Gamer",  
10  "description": "Mouse Gamaer",  
11  "slug": "mouse-gamer",  
12  "price": 299  
13 },  
14 {  
15   "tags": [  
16     "informática",  
17     "mouse",  
18     "games"  
19   ],  
20   "_id": "5d5c6627cc81b832f8f37dd9",  
21   "title": "Cadeira Gamer",  
22   "description": "Cadeira Gamer",  
23   "slug": "cadeira-gamer",  
24   "price": 1299  
25 }  
26 ]
```

## Atualizando dados de um produto

PUT - localhost:3000/products/5d5c6627cc81b832f8f37dd9

PUT localhost:3000/products/5d5c6627cc81b832f8f37dd9

Params Authorization Headers (1) **Body** Pre-request Script Tests

none form-data x-www-form-urlencoded raw binary **JSON (application/json)**

```
1 {  
2   "title": "Cadeira Gamer 2",  
3   "description": "Cadeira Gamer 2",  
4   "slug": "cadeira-gamer-2",  
5   "price": 2299  
6 }
```

Body Cookies Headers (6) Test Results

Pretty Raw Preview JSON ↗

```
1 [{  
2   "message": "Produto atualizado com sucesso!"  
3 }]
```

Listando os produtos

GET - localhost:3000/products



Pretty Raw Preview JSON ↻

```
1 [ [ 2 { 3   "_id": "5d5c65f7cc81b832f8f37dd8", 4     "title": "Mouse Gamer", 5     "slug": "mouse-gamer", 6     "price": 299 7 }, 8 { 9   "_id": "5d5c6627cc81b832f8f37dd9", 10    "title": "Cadeira Gamer 2", 11    "slug": "cadeira-gamer-2", 12    "price": 2299 13 } 14 ]
```

Excluindo um produto

DELETE - localhost:3000/products/5d5c6627cc81b832f8f37dd9



Body Cookies Headers (6) Test Results

Pretty Raw Preview JSON ↻

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

Listando os produtos

GET - localhost:3000/products



Pretty Raw Preview JSON ↻

```
1 [ [ 2 { 3   "_id": "5d5c6627cc81b832f8f37dd9", 4     "title": "Cadeira Gamer 2", 5     "slug": "cadeira-gamer-2", 6     "price": 2299 7 } 8 ] ]
```

## 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.getByTag = async(req, res, next) => {
  try {
    const data = await repository.getByTag(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 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(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;
```

## Criando um cliente (customer)

POST - localhost:3000/customers

The screenshot shows a Postman request configuration and its response.

**Request Headers:**

- Method: POST
- URL: localhost:3000/customers
- Headers (1): Content-Type (application/json)

**Request Body:**

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

**Response Headers:**

- Content-Type: application/json

**Response Body:**

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

No Studio 3T:

The screenshot shows the MongoDB interface displaying the inserted document.

**Result Table:**

Key	Value	Type
↪ (1) {_id : 5d5c6962cc81b832f8f37dda}	{ 5 fields }	Document
↳ _id	5d5c6962cc81b832f8f37dda	ObjectId
↳ name	Roberto Pinheiro	String
↳ email	betopinheiro1005@yahoo.com.br	String
↳ password	891f095be7ed455ec084e1fc23a3bb21	String
↳ __v	0	Int32

# Aula 29 - Revisitando os Controllers: Order

## Instalação do pacote guid

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

```
C:\laragon\www\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@1.2.9 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"ia32"})
+ guid@0.0.12
added 1 package from 2 contributors and audited 2470 packages in 25.159s
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/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();
}
```

## **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
}));

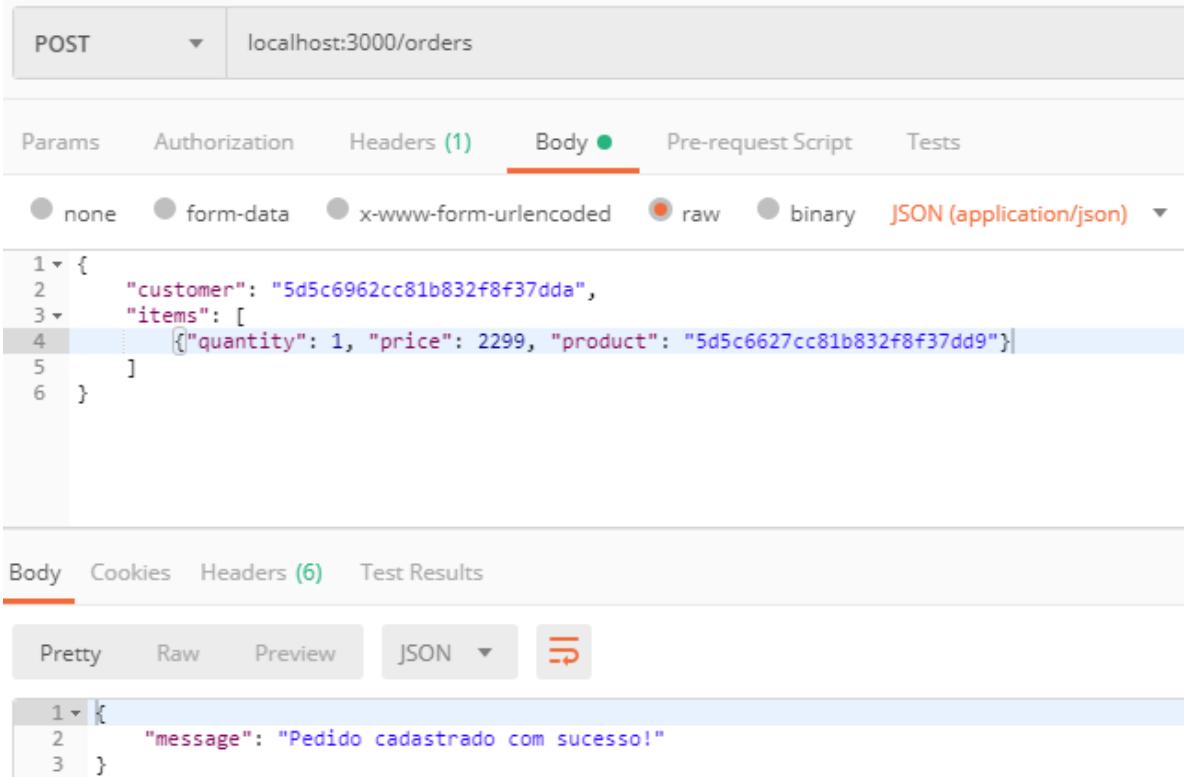
app.use('/', indexRoute);
app.use('/products', productRoute);
app.use('/customers', customerRoute);
app.use('/orders', orderRoute);

module.exports = app;

```

## Testando no Postman

POST - localhost:3000/orders



The screenshot shows the Postman interface with the following details:

- Method:** POST
- URL:** localhost:3000/orders
- Headers:** (1) (highlighted)
- Body:** (highlighted)
- Content Type:** JSON (application/json)
- Request Body (Raw JSON):**

```

1 ▾ {
2   "customer": "5d5c6962cc81b832f8f37dda",
3   "items": [
4     {"quantity": 1, "price": 2299, "product": "5d5c6627cc81b832f8f37dd9"}
5   ]
6 }

```

- Response Body (Pretty JSON):**

```

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

```

No Studio 3T:

Key	Value	Type
» (1) {_id : 5d5c6bb5cc81b832f8f37ddb}	{ 7 fields }	Document
» _id	5d5c6bb5cc81b832f8f37ddb	ObjectId
» status	created	String
» customer	5d5c6962cc81b832f8f37dda	ObjectId
» number	f62f59	String
» items	[ 1 elements ]	Array
» 0	{ 4 fields }	Object
» quantity	1	Int32
» _id	5d5c6bb6cc81b832f8f37ddc	ObjectId
» price	2299	Int32
» product	5d5c6627cc81b832f8f37dd9	ObjectId
» createDate	2019-08-20T21:52:54.036Z	Date
» _v	0	Int32

## Listando os pedidos no Postman

GET - localhost:3000/orders

Pretty    Raw    Preview    JSON ▾

```
1 [ ]
2 {
3   "status": "created",
4   "_id": "5d5c6bb5cc81b832f8f37ddb",
5   "customer": {
6     "_id": "5d5c6962cc81b832f8f37dda",
7     "name": "Roberto Pinheiro"
8   },
9   "items": [
10    {
11      "quantity": 1,
12      "_id": "5d5c6bb6cc81b832f8f37ddc",
13      "price": 2299,
14      "product": {
15        "_id": "5d5c6627cc81b832f8f37dd9",
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:<password>@node-store-cluster-nlcnv.mongodb.net/node-str-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, { 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;
```

# Aula 31 - Encriptando a senha

## Instalação do md5

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

```
C:\laragon\www\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@1.2.9 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"ia32"})
+ md5@2.2.1
added 3 packages from 3 contributors and audited 2474 packages in 37.546s
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'
        });
    }
}
```

```
    }  
};
```

## No Studio 3T

Key	Value	Type
»  (1) {_id : 5d5c6962cc81b832f8f37dda}	{ 5 fields }	Document
_id	5d5c6962cc81b832f8f37dda	ObjectId
name	Roberto Pinheiro	String
email	betopinheiro1005@yahoo.com.br	String
password	891f095be7ed455ec084e1fc23a3bb21	String
__v	0	Int32

# Aula 32 - Enviando email de boas vindas

## SendGrid

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

The screenshot shows the SendGrid API Keys settings page. At the top, a banner states: "Your Email API trial expired. You may still send up to 100 Email API emails per day for testing purposes." Below this, there's a "Create API Key" button. On the left, a sidebar menu includes "Dashboard", "Marketing NEW", "Templates", and "Stats". The main content area features a key icon and the text "Get started creating API Keys". It explains that API keys help protect sensitive account areas like contacts and settings, and allows for multiple keys with different permissions.

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

The screenshot shows the "Create API Key" form. The "API Key Name" field is filled with "teste". Under "API Key Permissions", the "Full Access" option is selected, indicated by a blue circle. The "Full Access" section describes it as allowing access to all endpoints except billing and email validation. There are also "Restricted Access" and "Billing Access" options, each with their own descriptions. At the bottom, there are "Cancel" and "Create & View" buttons.



## API Key Created

Please copy this key and save it somewhere safe.

For security reasons, we cannot show it to you again

SG.zbIpNcObTvCjxa5dmO-acw.hrA3Y-ibkIqKN\_qymY4c0YCzXvpfG0SwD70QBpejOdg

Done

## API Keys

Create API Key

NAME	API KEY	ACTION
teste API Key ID: zbIpNcObTvCjxa5dmO-acw	copy · ..... refresh	gear · v

- 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:<password>@node-store-cluster-nlcnv.mongodb.net/node-str-db?retryWrites=true&w=majority',
  sendgridKey: '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',
  containerConnectionString: 'TBD'
}
```

## Instalação do SendGrid

```
npm install sendgrid@2.0.0 --save
```

```
C:\laragon\www\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 node-str@1.0.0 No description
npm WARN node-str@1.0.0 No repository field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"ia32"})
+ sendgrid@2.0.0
added 46 packages from 68 contributors and audited 2541 packages in 50.504s
found 3 vulnerabilities (1 low, 2 high)
  run `npm audit fix` to fix them, or `npm audit` for details
```

## 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": "^4.1.1",
    "express": "^4.17.1",
    "guid": "0.0.12",
    "http": "0.0.0",
    "md5": "^2.2.1",
    "mongoose": "^5.6.9",
    "sendgrid": "^2.0.0"
  },
  "devDependencies": {
    "nodemon": "^1.19.1"
  }
}
```

## **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));
  } catch (err) {
    res.status(500).send(`Internal Server Error: ${err}`).end();
  }
}
```

```

        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

The screenshot shows the Postman interface. At the top, it says "POST" and "localhost:3000/customers". Below that, under the "Body" tab, there is a JSON payload:

```

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

```

Under the "Body" tab, the response status is "201 Created" with a time of "1393 ms" and a size of "262 B". The response body is:

```

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

```

The screenshot shows the MongoDB Compass interface with the "Result" tab selected. It displays a table with one document:

Key	Value	Type
❶ (1) <code>._id : 5d5c6d8bcc81b832f8f37ddd</code>	{ 5 fields }	Document
❷ <code>._id</code>	<code>5d5c6d8bcc81b832f8f37ddd</code>	ObjectId
❸ <code>name</code>	Roberto Pinheiro	String
❹ <code>email</code>	<code>betopinheiro1005@yahoo.com.br</code>	String
❺ <code>password</code>	<code>891f095be7ed455ec084e1fc23a3bb21</code>	String
❻ <code>__v</code>	0	Int32

The screenshot shows a browser window with a header banner for Land Rover and a message: "A aventura está em nosso DNA. Capacidade e versatilidade para enfrentar qual...". Below the banner, it says "Bem vindo ao Node Store Olá, Roberto Pinheiro, seja bem vindo à Node ...".

• Bem vindo ao Node Store

Yahoo/Spam ★



andrebalteri@balta.io

Para: betopinheiro1005@yahoo.com.br



20 de ago às 03:05



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



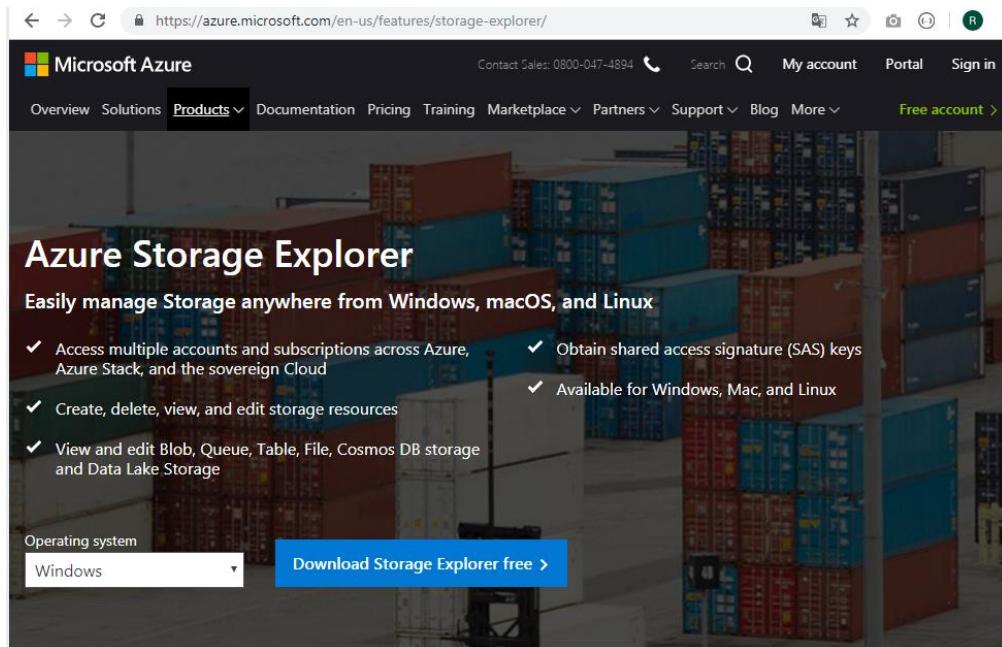
[Responder](#), [Responder a todos](#) ou [Encaminhar](#)



# Aula 33 - Upload da imagem do produto

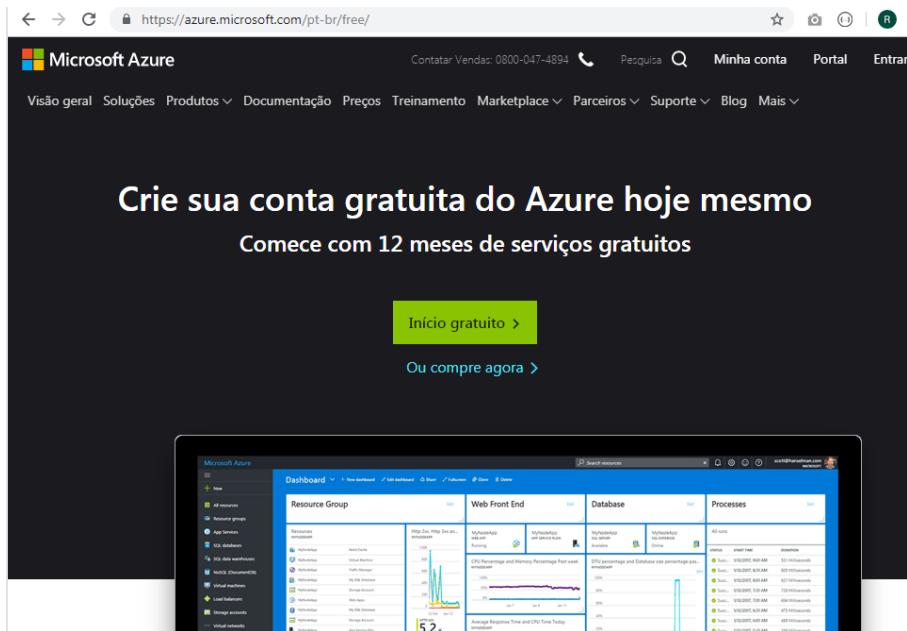
Acesse a URL:

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



- Baixe e instale o programa Storage Explorer free.
- Crie sua conta no Azzure:

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



- Acesse o portal.

The screenshot shows the Microsoft Azure portal interface. On the left, there is a navigation sidebar with various service icons and links: Create a resource, Home, Dashboard, All services (which is selected), FAVORITES (with All resources, Resource groups, App Services, Function App, SQL databases, Azure Cosmos DB, Virtual machines, Load balancers, Storage accounts, Virtual networks, Azure Active Directory, and Monitor), and a general Favorites section. The main content area is titled "Azure services" and shows "See all (100+)" services. It features several service tiles: Virtual machines, App Services, Storage accounts, SQL databases, Azure Database for PostgreSQL, Azure Cosmos DB, Kubernetes services, and Function App. Below these are four promotional cards: Microsoft Learn (Learn Azure with free online training from Microsoft), Azure Monitor (Monitor your apps and infrastructure), Security Center (Secure your apps and infrastructure), and Cost Management (Analyze and optimize your cloud spend for free). At the bottom, there is a "Recent resources" section with links to "See all your recent resources" and "See all your resources".

- Crie um Storage Account.