

# Introdução ao AngularJS

Professor Anderson Henrique

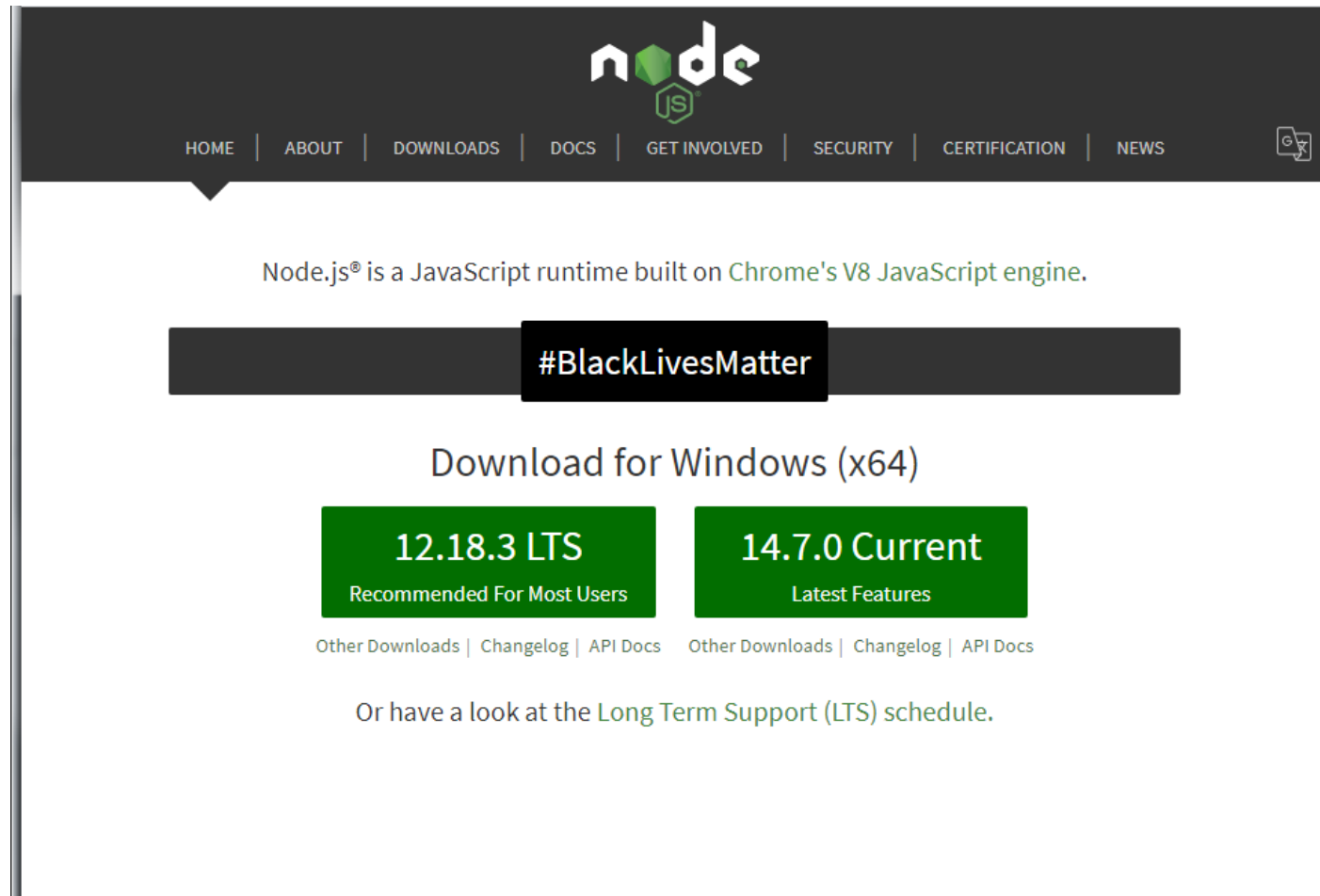


# Utilizando Node.js

Node.js é um interpretador de JavaScript assíncrono com código aberto orientado a eventos, criado por Ryan Dahl em 2009, focado em migrar a programação do Javascript do cliente (frontend) para os servidores, criando aplicações de alta escalabilidade (como um servidor web), manipulando milhares de conexões/eventos simultâneas em tempo real numa única máquina física.

O Node.js (ambiente de execução Javascript no servidor) foi implementado baseado no interpretador V8 JavaScript Engine (interpretador de JavaScript em C++ com código aberto do Google, utilizado no Chrome), com desenvolvimento mantido pela fundação Node.js em parceria com a Linux Foundation.

# Baixando e instalando o Node.js

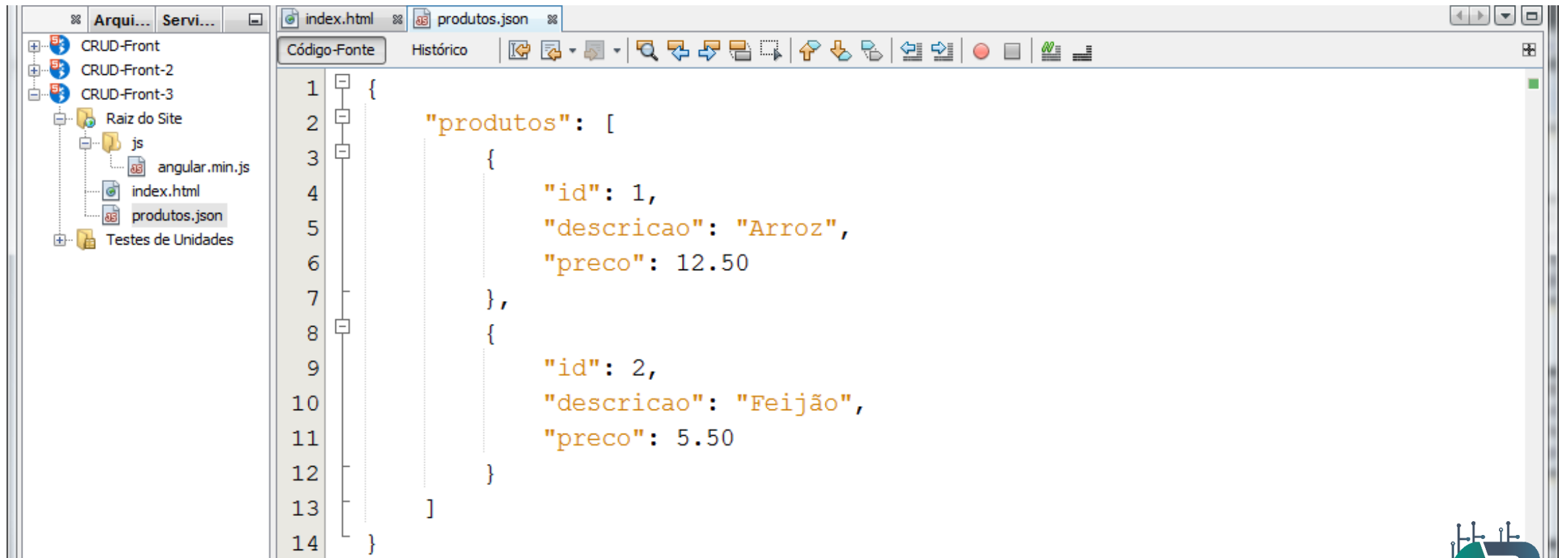


Após a instalação do Node.js, devemos executar esse comando no nosso **CMD** do Windows para a instalação das dependências do nosso servidor json

**npm install -g json-server**

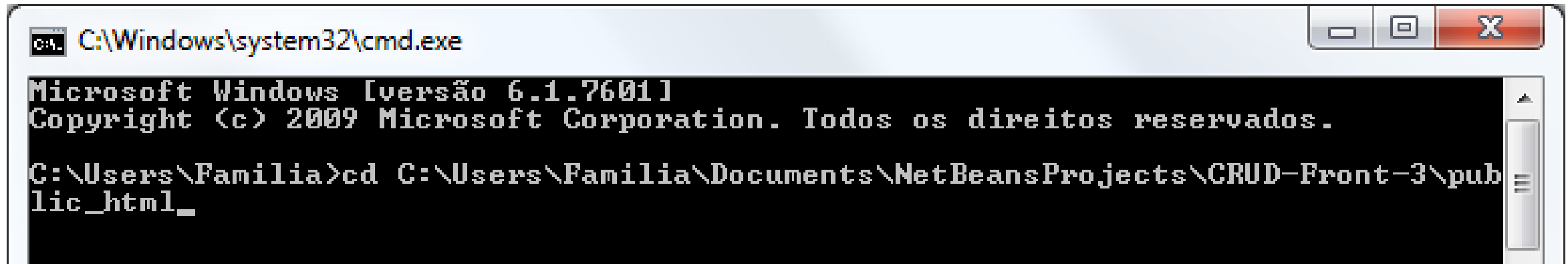
Para testar nosso servidor, vamos criar um arquivo .json de produtos

Para testar nosso servidor, vamos criar um arquivo .json de produtos



```
1 {  
2   "produtos": [  
3     {  
4       "id": 1,  
5       "descricao": "Arroz",  
6       "preco": 12.50  
7     },  
8     {  
9       "id": 2,  
10      "descricao": "Feijão",  
11      "preco": 5.50  
12    }  
13  ]  
14 }
```

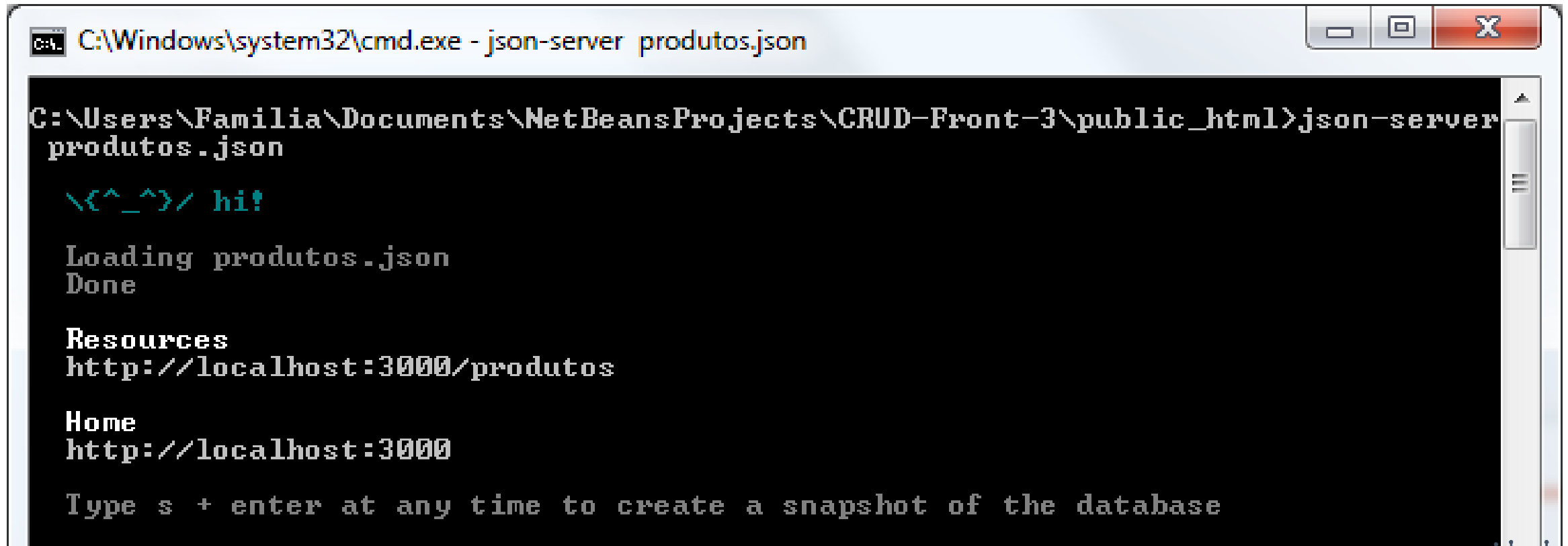
Para rodar nosso arquivo json no servidor, precisamos ir até o diretório no **cmd**



```
C:\Windows\system32\cmd.exe  
Microsoft Windows [versão 6.1.7601]  
Copyright (c) 2009 Microsoft Corporation. Todos os direitos reservados.  
C:\Users\Familia>cd C:\Users\Familia\Documents\NetBeansProjects\CRUD-Front-3\public_html_
```

Execute o comando: **json-server produtos.json**

## Nosso serviço está rodando



```
C:\Windows\system32\cmd.exe - json-server produtos.json

C:\Users\Familia\Documents\NetBeansProjects\CRUD-Front-3\public_html>json-server
produtos.json

\<^_^>/ hi!

Loading produtos.json
Done

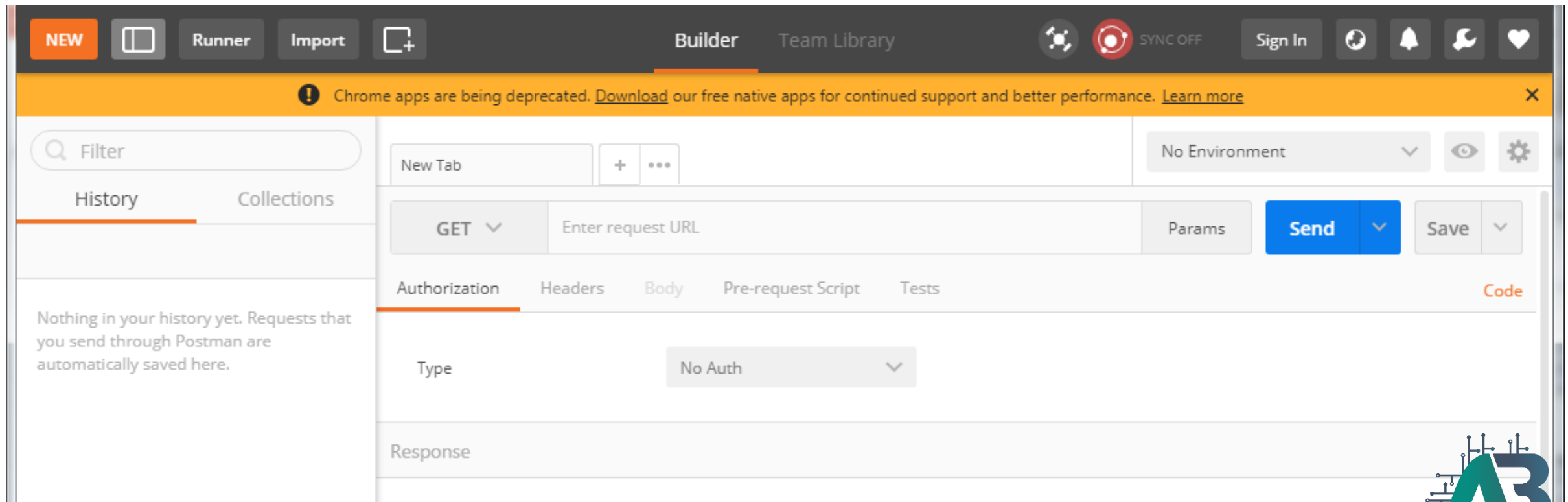
Resources
http://localhost:3000/produtos

Home
http://localhost:3000

Type s + enter at any time to create a snapshot of the database
```

# Utilizando Postman

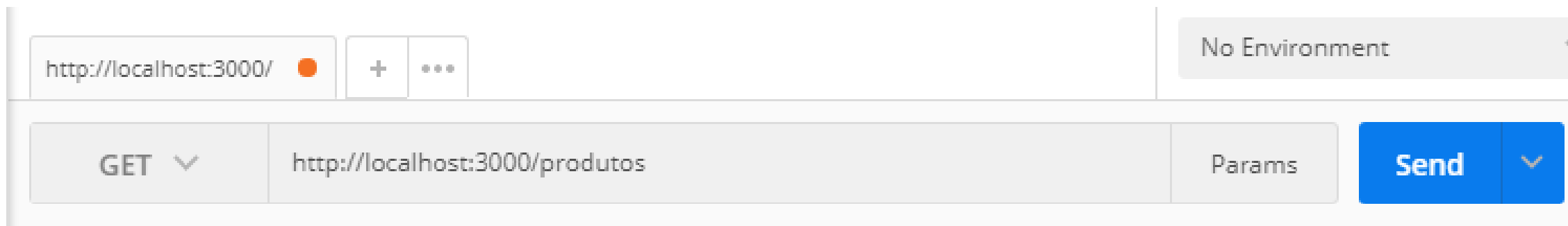
Utilizando API Rest com o nosso servidor json-server





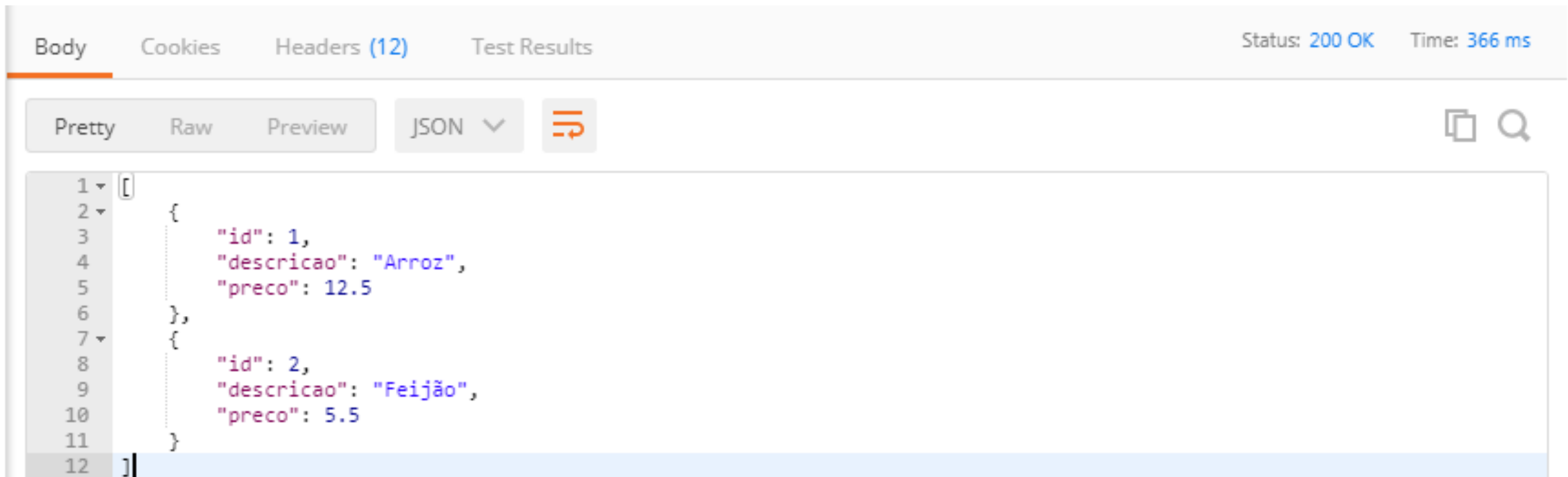
Informe o Resource do servidor json

Para recuperar todos os registros no nosso webservice utilizamos a requisição **GET**



The image shows a REST client interface. At the top, there is a tab labeled 'http://localhost:3000/' with an orange status indicator, a plus sign, and a three-dot menu. To the right of the tab is a dropdown menu showing 'No Environment'. Below the tab, there is a large input field containing the URL 'http://localhost:3000/produtos'. To the left of this field is a dropdown menu showing 'GET' with a downward arrow. To the right of the URL field is a 'Params' label and a blue 'Send' button with a downward arrow.

# Este é o resultado



The screenshot shows a web browser's developer console with the 'Body' tab selected. The status bar at the top right indicates 'Status: 200 OK' and 'Time: 366 ms'. The console content is displayed in 'Pretty' format as a JSON array. The array contains two objects, each representing a food item with an 'id', 'descricao', and 'preco'.

```
1 [
2   {
3     "id": 1,
4     "descricao": "Arroz",
5     "preco": 12.5
6   },
7   {
8     "id": 2,
9     "descricao": "Feijão",
10    "preco": 5.5
11  }
12 ]
```

Para recuperar um registro no nosso webservice utilizamos a requisição **GET** passando o valor do parâmetro id

The screenshot shows a REST client interface with the following details:

- Environment:** No Environment
- Method:** GET
- URL:** http://localhost:3000/produtos/1
- Params:** (empty)
- Send Button:** A blue button labeled "Send".
- Response Status:** 200 OK
- Response Time:** 339 ms
- Response Body:** A JSON object displayed in "Pretty" format:

```
{
  "id": 1,
  "descricao": "Arroz",
  "preco": 12.5
}
```

Para excluir um registro no nosso webservice utilizamos a requisição **DELETE** passando o valor do parâmetro id

The screenshot displays a REST client interface. At the top, the URL bar shows 'http://localhost:3000/' with a status indicator (orange dot), a plus sign, and a menu icon. To the right, it says 'No Environment'. Below this, the method 'DELETE' is selected from a dropdown, and the URL 'http://localhost:3000/produtos/1' is entered. To the right of the URL are 'Params' and a 'Send' button. Below the URL bar, there are tabs for 'Body', 'Cookies', 'Headers (12)', and 'Test Results'. The 'Body' tab is active, showing a JSON response in 'Pretty' format. The response is an array with one object: 

```
[{"id": 2, "descricao": "Feijão", "preco": 5.5}]
```

. The status bar at the top right indicates 'Status: 200 OK' and 'Time: 333 ms'.

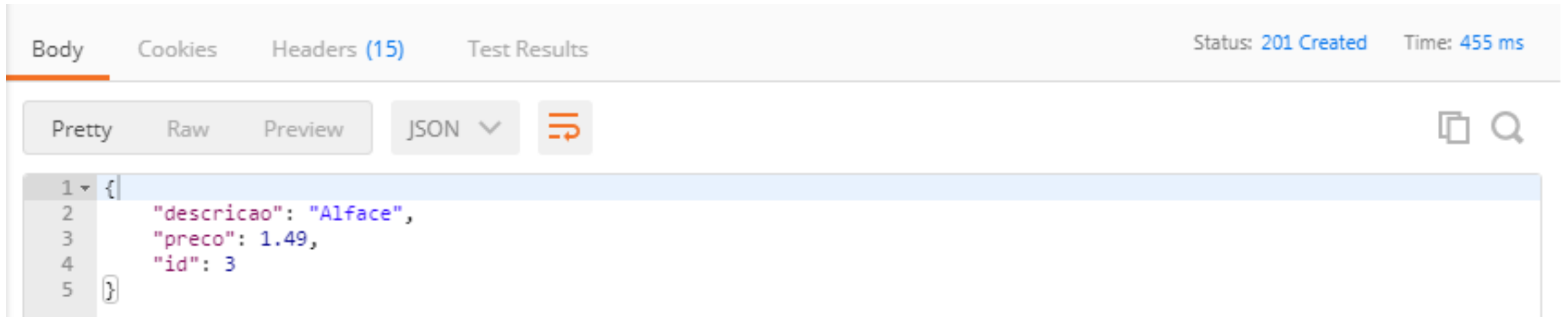
Para adicionar um registro no nosso webservice utilizamos a requisição **POST**

The screenshot shows a REST client interface with the following details:

- URL Bar:** `http://localhost:3000/` with a red status indicator, a plus sign, and a menu icon. A dropdown menu on the right shows `No Environment`.
- Request Configuration:** Method `POST` and URL `http://localhost:3000/produtos`. A `Params` tab and a blue `Send` button are also visible.
- Body Tab:** The `Body` tab is selected, showing radio buttons for `form-data`, `x-www-form-urlencoded`, `raw` (selected), and `binary`. A dropdown menu shows `JSON (application/json)`.
- JSON Body:**

```
1 {  
2   "descricao" : "Alface",  
3   "preco" : 1.49  
4 }  
5 |
```

## Adiciona o produto e cria o id dinamicamente



Se fizemos uma nova requisição **GET**, de todos os produtos este será o resultado

BodyCookiesHeaders (12)Test ResultsStatus: 200 OKTime: 373 ms

PrettyRawPreviewJSON ↕

```
1 [
2   {
3     "id": 2,
4     "descricao": "Feijão",
5     "preco": 5.5
6   },
7   {
8     "descricao": "Alface",
9     "preco": 1.49,
10    "id": 3
11  }
12 ]
```

Para atualizar um registro no nosso webservice utilizamos a requisição **PUT** passando o valor do parâmetro id

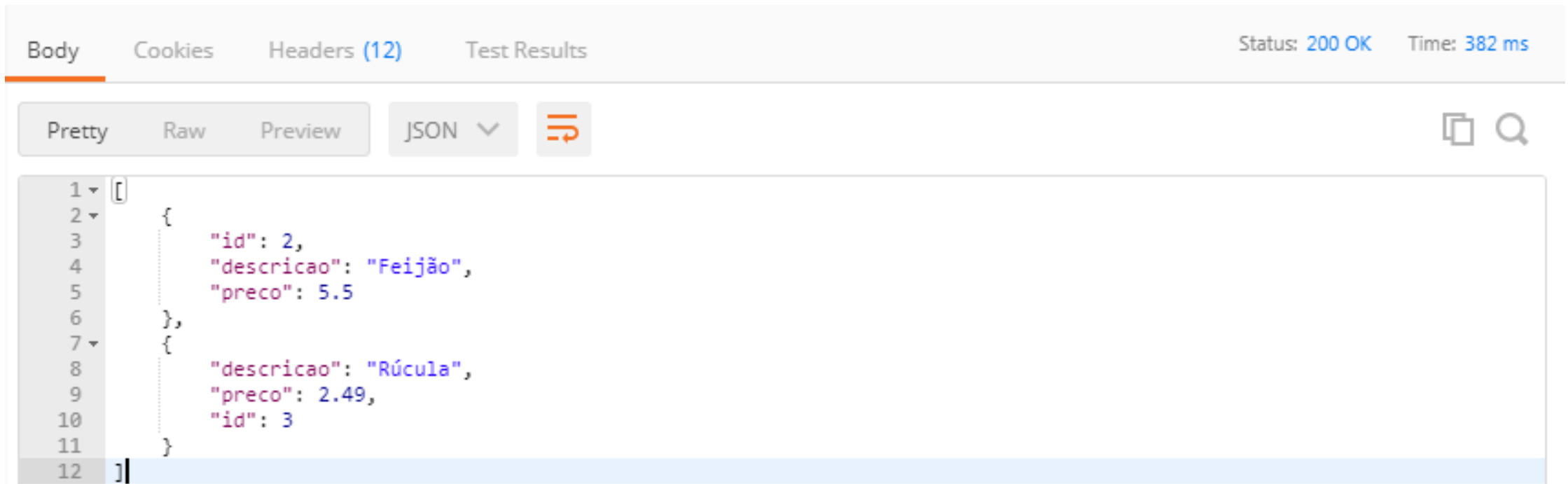
The screenshot shows a REST client interface with the following details:

- Method:** PUT
- URL:** http://localhost:3000/produtos/3
- Params:** (empty)
- Buttons:** Send, Save
- Tabs:** Authorization, Headers (1), Body (selected), Pre-request Script, Tests
- Body Type:** raw (selected), JSON (application/json)
- Body Content:**

```
1 {  
2   "descricao" : "Rúcula",  
3   "preco" : 2.49  
4 }  
5
```



Se fizemos uma nova requisição **GET**, de todos os produtos este será o resultado



The screenshot shows a web browser's developer console with the 'Body' tab selected. The response is a JSON array containing two product objects. The status is 200 OK and the time is 382 ms.

```
[
  {
    "id": 2,
    "descricao": "Feijão",
    "preco": 5.5
  },
  {
    "descricao": "Rúcula",
    "preco": 2.49,
    "id": 3
  }
]
```

Vamos alterar o nosso Service, primeiro removendo o array de produtos, faremos requisições utilizando o service \$http

```
44 angular.module("produtosApp").service("produtosService", function($http){  
45  
46     this.listar = function(){  
47         return $http.get('http://localhost:3000/produtos');  
48     };  
}
```

Ops! Não retornou, porque nas requisições que são assíncronas, temos que utilizar o que chamamos de promessas de devoluções quando estiverem resolvidas

```
19 angular.module("produtosApp", []);
20 angular.module("produtosApp").controller("produtosController",function($scope, produtosService){
21
22     $scope.produto = {};
23
24     produtosService.listar().then(function(resposta){
25         $scope.produtos = resposta.data;
26     });
```

Vamos alterar o nosso Service, alterando o nosso método salvar, se o id existir atualizar registro, senão criar novo registro

```
47 angular.module("produtosApp").service("produtosService", function($http){
48
49     var api = 'http://localhost:3000/produtos';
50
51     this.listar = function(){
52         return $http.get(api);
53     };
54
55     this.salvar = function(produto) {
56         if(produto.id){
57             return $http.put(api + '/' + produto.id, produto);
58         }else{
59             return $http.post(api, produto);
60         }
61     };
62 }
```

Vamos alterar o nosso controller, assim que ele salvar o produto deverá listar os produtos novamente

```
22 $scope.produto = {};  
23 listar();  
24  
25 function listar(){  
26     produtosService.listar().then(function(resposta){  
27         $scope.produtos = resposta.data;  
28     });  
29 }  
30  
31 $scope.salvar = function(produto){  
32     produtosService.salvar(produto).then(listar);  
33     $scope.produto = {};  
34 };
```

Vamos alterar o nosso Service, alterando o nosso método excluir

```
66 this.excluir = function(produto){  
67     return $http.delete(api + '/' + produto.id);  
68 };
```

Assim que excluir o registro, deverá listar os produtos

```
36 $scope.excluir = function(produto){  
37     produtosService.excluir(produto).then(listar);  
38 };
```

**Dúvidas?**

**Professor Anderson Henrique**



## Para a próxima aula

01 – Aplicação Back-End em JAVA

02 – Instalando e Configurando nosso Servidor Apache TOMCAT

03 – Criando nossa Classe, nosso Service e Configurando REST com Java

Professor Anderson Henrique

