

Prefácio

1 – App Base	pág. 2
1.1 – Git	pág. 2
1.2 – Módulos	pág. 2
1.2.1 – Eureka Server	pág. 2
1.2.2 – Gateway (gateway-service)	pág. 2
1.2.3 – App Products (product-service)	pág. 3
1.2.4 – App Business (business-service)	
1.2.5 – App Orders (order-service)	
1.3 – Bancos de Dados	pág. 3
2 – Execução/ Apresentação	pág. 4
2.1 – Controllers	pág. 5
2.1.1 – Products	pág. 6
2.1.2 – Business	pág. 7
2.1.3 – Orders	pág. 9

1 – App Base

A seguir as instruções por tópicos referente a modelagem e construção da aplicação. Segue abaixo o diagrama ilustrativo da aplicação e respectivos módulos.

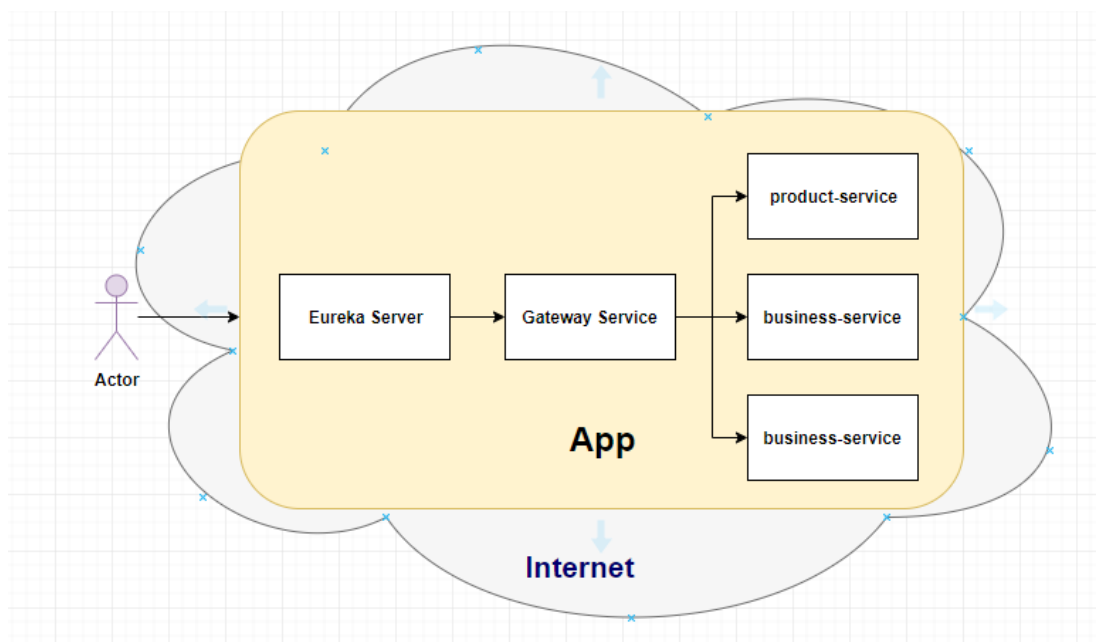


Figura 1 – Diagrama da Aplicação/Módulos

O contexto da aplicação desenvolvida envolve a arquitetura moderna de microserviços. Essa envolveu de acordo com a Figura 1: um servidor de monitoramento de instâncias e módulos em execução, além de zonas disponíveis.

1.1 – Git

Conforme aplicação criada inicialmente, foi modificada de forma modular para atender a arquitetura e aplicações que foram desenvolvidas.

Todo o código encontra-se em: <https://github.com/thiagohernandes/backend-challenge/tree/dev>

1.2 – Módulos

Os módulos criados são independentes e no caso dos Apps: Products, Business e Orders é possível gerar mais do que uma instância independente (bastando apenas gerar um novo deploy alterando a porta) por apps a fim de fornecer novas zonas para os casos de fault-tolerance e disponibilidade.

1.2.1 – Eureka Server

É composto pelo módulo central que registra e monitora as instâncias em execução da arquitetura modular da aplicação e serviços.

1.2.2 – Gateway (gateway-service)

Corresponde ao Zuul Server Gateway da arquitetura. Corresponde ao redirecionamento conforme disponibilidade de instâncias em execução.

1.2.3 – App Products (product-service)

Envolve toda e qualquer estrutura relacionada a produtos. Esse em que pode ser normalizado, implementado e ajustado conforme cases específicos.

Para o cenário em questão, foi criada apenas uma tabela para comunicação com o serviço principal da aplicação, o “order-service”.

1.2.4 – App Business (business-service)

Compreende os serviços relacionados a duas tabelas do que foi proposto. “Store” e “Payments”. Não obstante, por meio desse serviço/módulo/app pode-se implementar novas tabelas e funcionalidades conforme crescimento da aplicação.

1.2.5 – App Orders (order-service)

O módulo independente mais importante da aplicação. Esse está com a responsabilidade de gerar pedidos “Orders” que busca informações dos outros módulos: Business e Products.

1.3 – Bancos de Dados

Simbolicamente, foram criados em localhost para agilizar o desenvolvimento. No entanto, na real necessidade considera-se que cada banco de dados foi criado em servidores/ips distintos.

2 – Execução/ Apresentação

A ferramenta utilizada para o desenvolvimento foi o STS – Spring Tool Service. Segue abaixo demais tecnologias utilizadas:

Banco de dados: PostgreSQL 10

Framework Java: Spring Boot

Java version: 8

O primeiro passo é importar a aplicação e respectivos módulos como um app do tipo Maven.

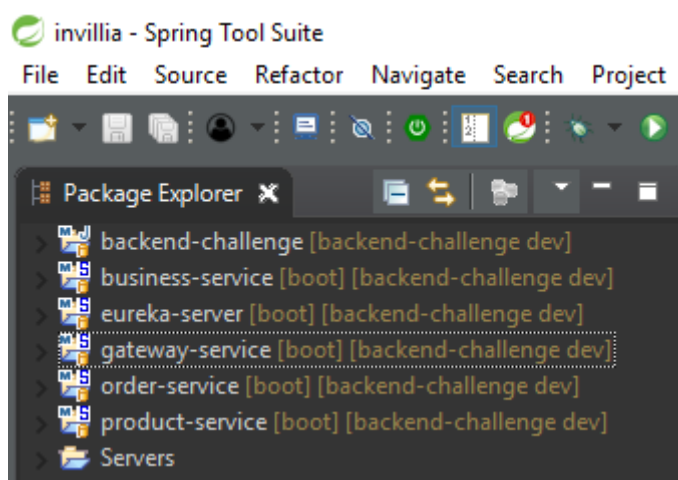


Figura 2 – Aplicação importada na IDE

Em seguida atualizar/baixar as dependências do Maven.

Agora, basta executar (dar um intervalo entre as execuções de 20 segundos ou acompanhar pelo console até que seja executada por completo) como “Spring Boot App” na sequência os projetos:

- 1 – **eureka-server** – classe: AppEurekaServer.java
- 2 – **gateway-service** – classe: AppGateway.java
- 3 – **business-service** – classe: AppBusiness.java
- 4 – **product-service** – classe: AppProduct.java
- 5 – **order-service** – classe: AppOrder.java

Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
BUSINESS-SERVICE	n/a (1)	(1)	UP (1) - DESKTOP-UCD5TPH:business-service:8091
ORDER-SERVICE	n/a (1)	(1)	UP (1) - DESKTOP-UCD5TPH:order-service:8090
PRODUCT-SERVICE	n/a (1)	(1)	UP (1) - DESKTOP-UCD5TPH:product-service:8092
ZUUL-SERVER-GATEWAY	n/a (1)	(1)	UP (1) - DESKTOP-UCD5TPH:zuul-server-gateway:8080

Figura 3 – URL de acesso do Eureka Server: <http://localhost:8761/>

Caso uma nova zona com qualquer um dos serviços (Product-Service, Order-Service ou Business-Service) seja adicionada, será disponibilizada conforme a Figura 4.

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
BUSINESS-SERVICE	n/a (1)	(1)	UP (1) - DESKTOP-UCD5TPH:business-service:8091
ORDER-SERVICE	n/a (1)	(1)	UP (1) - DESKTOP-UCD5TPH:order-service:8090
PRODUCT-SERVICE	n/a (2)	(2)	UP (2) - DESKTOP-UCD5TPH:product-service:8092 - DESKTOP-UCD5TPH:product-service:8093
ZUUL-SERVER-GATEWAY	n/a (1)	(1)	UP (1) - DESKTOP-UCD5TPH:zuul-server-gateway:8080

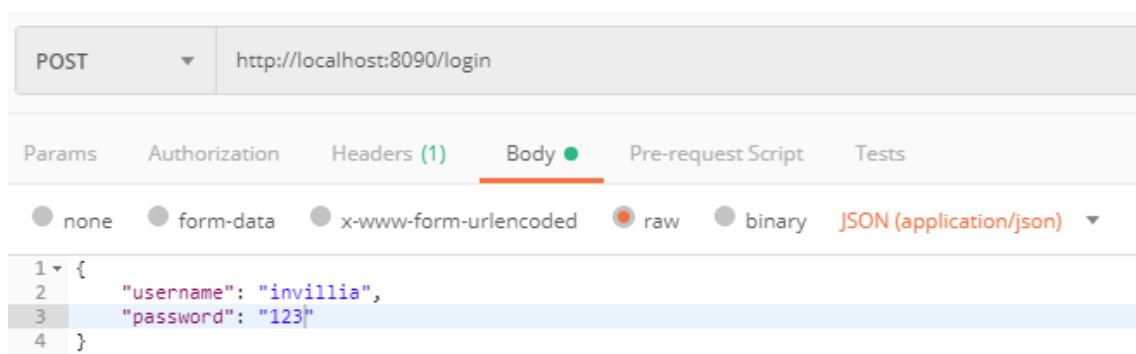
Figura 4 – Nova instância em uma nova zona para o caso de falha ou queda da instância 8092

2.1 – Controllers

Os controllers são as classes que armazenam os “endpoints” dos serviços REST de cada aplicação.

Atenção! Os “endpoints” estão protegidos por JWT Security. Sendo assim, antes é preciso simular o login em todas as aplicações para que os serviços funcionem.

O usuário(“invillia”) e senha(“123”) a ser utilizado é “inMemory”.



POST http://localhost:8090/login

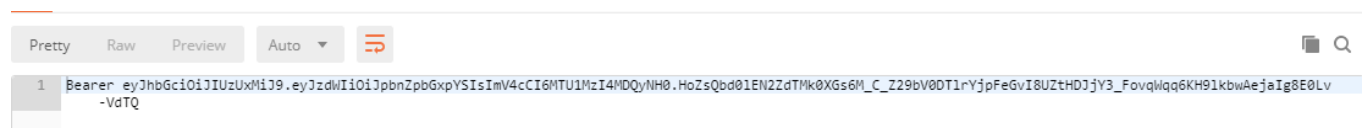
Params Authorization Headers (1) Body Pre-request Script Tests

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

```

1 {
2   "username": "invillia",
3   "password": "123"
4 }
```

Figura 5 – Login pelo Postman



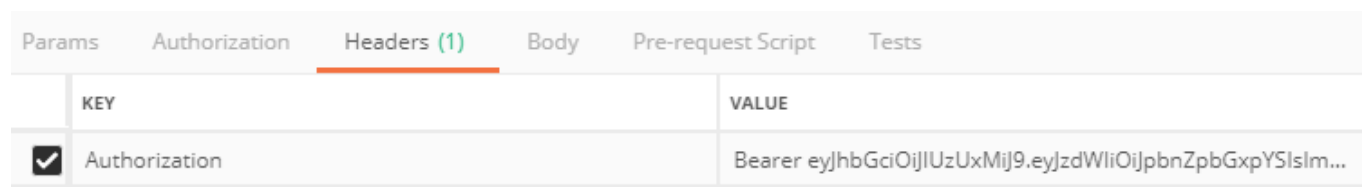
Pretty Raw Preview Auto

```

1 Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJpbmZpbGxpYSIsImV4cCI6MTU1MzI4MDQyNH0.HoZsQbd01EN2ZdTMk0XGs6M_C_Z29bV0DT1rYjpfFeGvI8UZtHDJjY3_FovqIqq6KH91kbwAejaIg8E0Lv-VdTQ
```

Figura 6 – Token gerado

Logo após, copiar o token e adicionar como “Authorization” em todas as chamadas endpoints/REST.



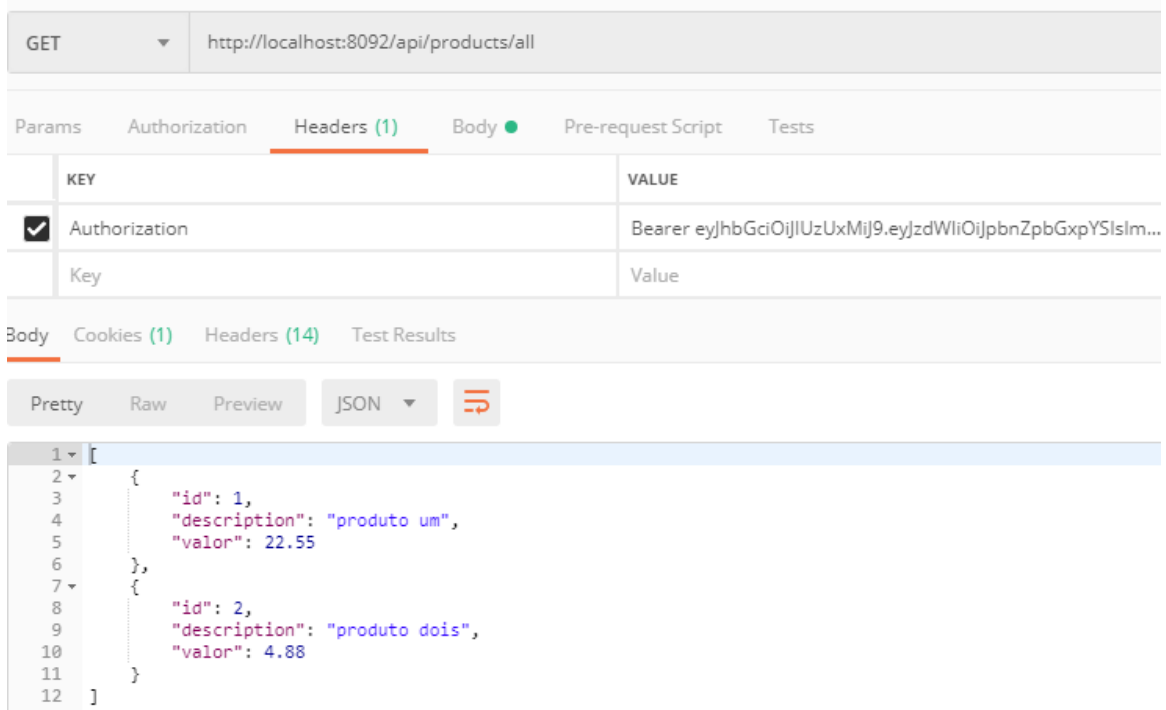
Params Authorization Headers (1) Body Pre-request Script Tests

KEY	VALUE
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJpbmZpbGxpYSIsImV4cCI6MTU1MzI4MDQyNH0.HoZsQbd01EN2ZdTMk0XGs6M_C_Z29bV0DT1rYjpfFeGvI8UZtHDJjY3_FovqIqq6KH91kbwAejaIg8E0Lv-VdTQ

Figura 7 – Authorization Token

Os passos (Figuras: 5 a 7) anteriores devem ser repetidos para as portas das instâncias com as portas: 8090, 8091 e 8092. Caso seja adicionada outra instância em uma outra zona, com a respectiva porta, fazer o mesmo procedimento.

2.1.1 – Products



GET ▼ http://localhost:8092/api/products/all

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

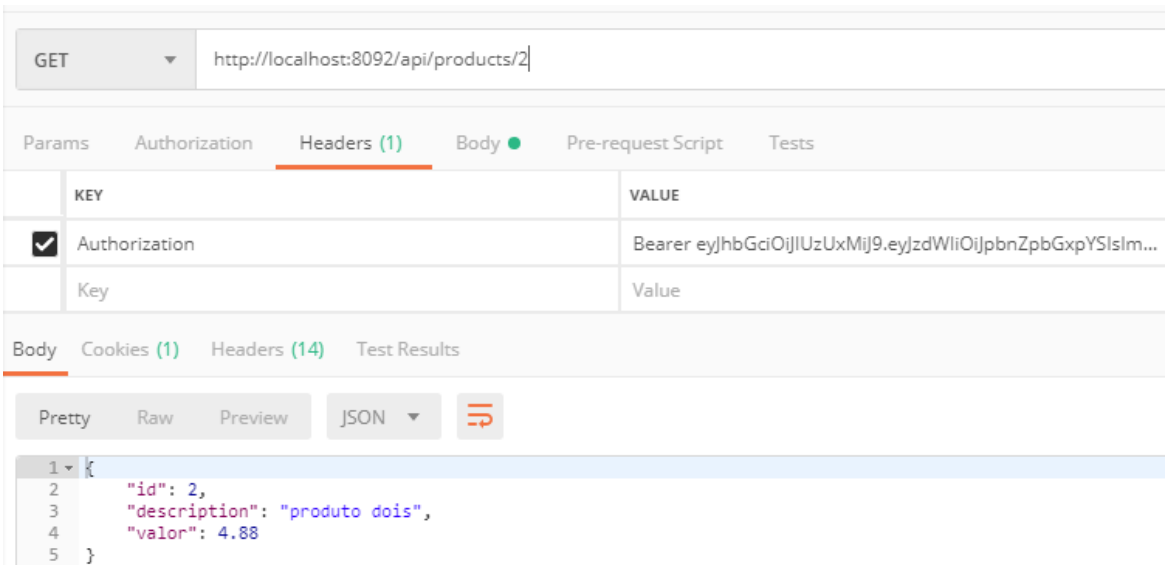
KEY	VALUE
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJpbmZpbGxpYSIsIm...
Key	Value

Body Cookies (1) Headers (14) Test Results

Pretty Raw Preview JSON ↺

```
1 [
2   {
3     "id": 1,
4     "description": "produto um",
5     "valor": 22.55
6   },
7   {
8     "id": 2,
9     "description": "produto dois",
10    "valor": 4.88
11  }
12 ]
```

Figura 8 – Todos os produtos



GET ▼ http://localhost:8092/api/products/2

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

KEY	VALUE
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJpbmZpbGxpYSIsIm...
Key	Value

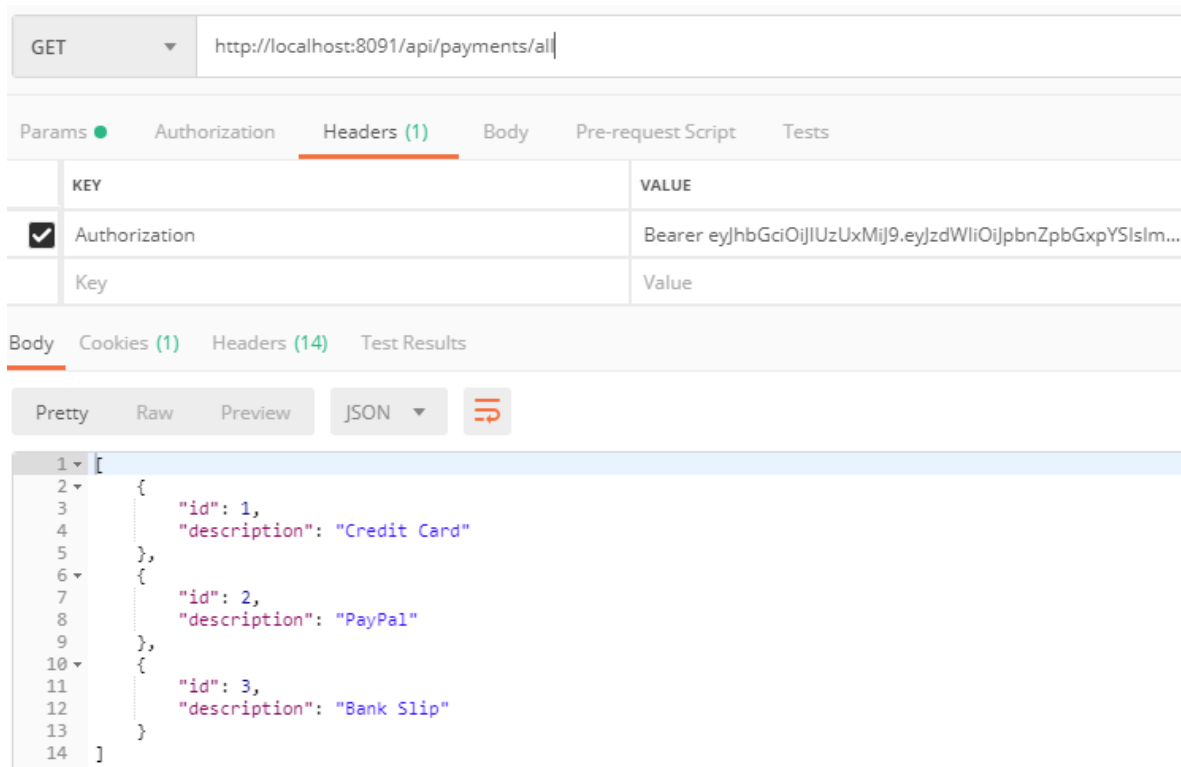
Body Cookies (1) Headers (14) Test Results

Pretty Raw Preview JSON ↺

```
1 {
2   "id": 2,
3   "description": "produto dois",
4   "valor": 4.88
5 }
```

Figura 9 – Produto por id

2.1.2 – Business




GET ▼ http://localhost:8091/api/payments/all

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

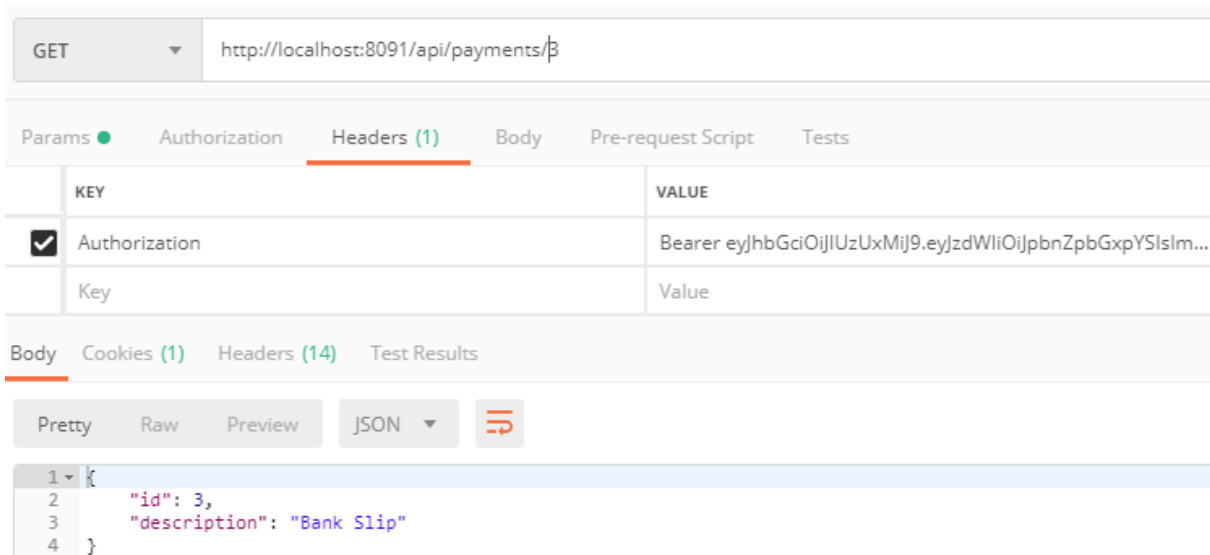
KEY	VALUE
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWliOiJpbmZpbGxpYSIsIm...
Key	Value

Body Cookies (1) Headers (14) Test Results

Pretty Raw Preview JSON ▼ 

```
1 {  
2   {  
3     "id": 1,  
4     "description": "Credit Card"  
5   },  
6   {  
7     "id": 2,  
8     "description": "PayPal"  
9   },  
10  {  
11    "id": 3,  
12    "description": "Bank Slip"  
13  }  
14 }
```

Figura 10 – Todas as formas de pagamento




GET ▼ http://localhost:8091/api/payments/3

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

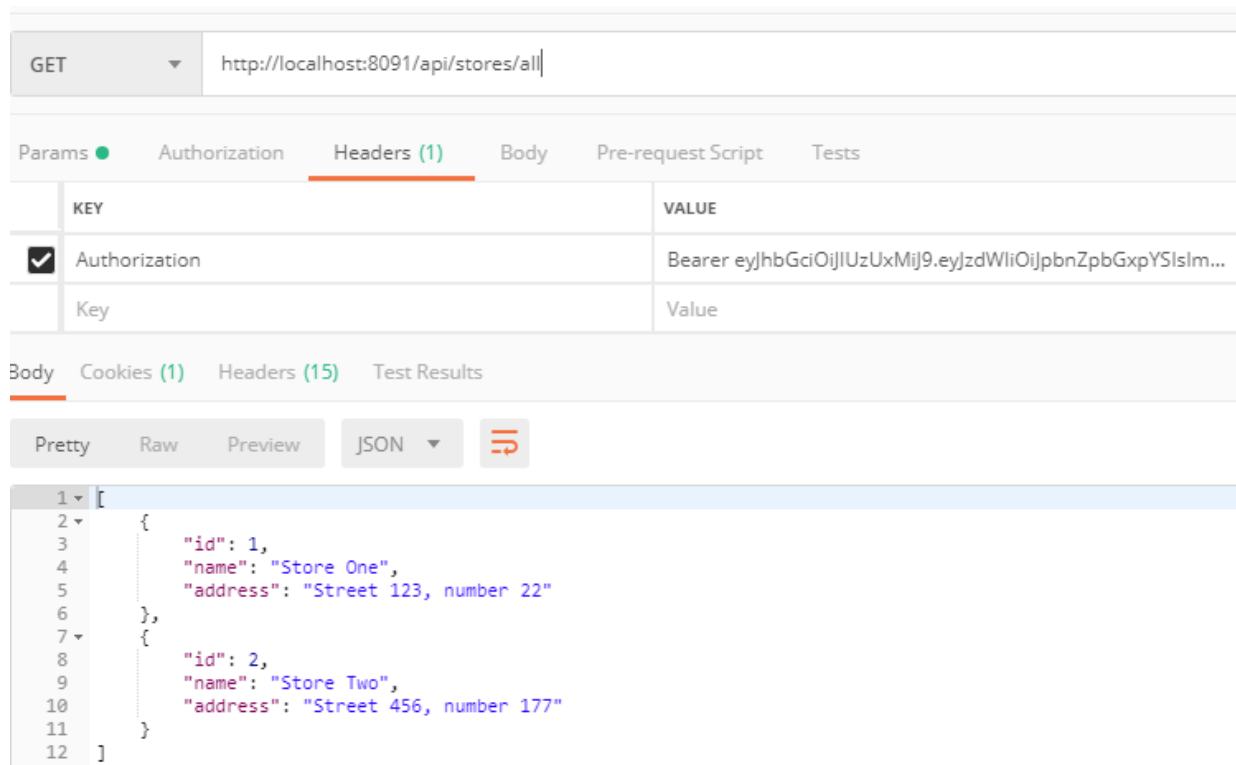
KEY	VALUE
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWliOiJpbmZpbGxpYSIsIm...
Key	Value

Body Cookies (1) Headers (14) Test Results

Pretty Raw Preview JSON ▼ 

```
1 {  
2   "id": 3,  
3   "description": "Bank Slip"  
4 }
```

Figura 11 – Pagamento por id




GET ▼ http://localhost:8091/api/stores/all

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

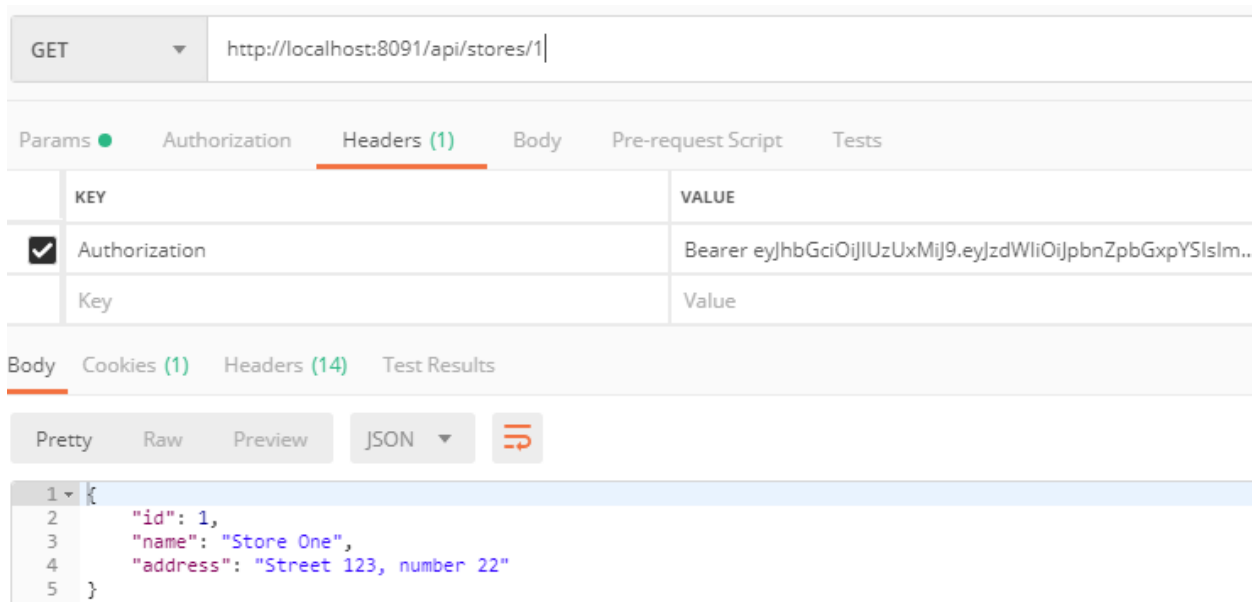
KEY	VALUE
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJpbmZpbGxpYSIsIm...
Key	Value

Body Cookies (1) Headers (15) Test Results

Pretty Raw Preview JSON ▼ 

```
1 [
2   {
3     "id": 1,
4     "name": "Store One",
5     "address": "Street 123, number 22"
6   },
7   {
8     "id": 2,
9     "name": "Store Two",
10    "address": "Street 456, number 177"
11  }
12 ]
```

Figura 12 – Todas as lojas




GET ▼ http://localhost:8091/api/stores/1

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

KEY	VALUE
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJpbmZpbGxpYSIsIm...
Key	Value

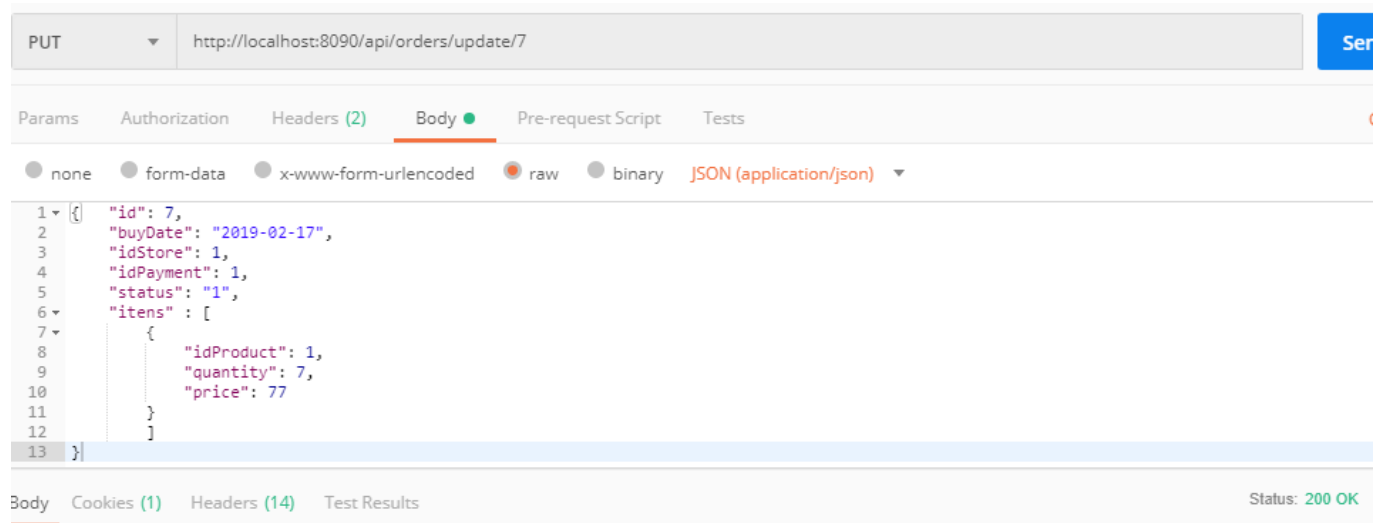
Body Cookies (1) Headers (14) Test Results

Pretty Raw Preview JSON ▼ 

```
1 {
2   "id": 1,
3   "name": "Store One",
4   "address": "Street 123, number 22"
5 }
```

Figura 13 – Loja por id

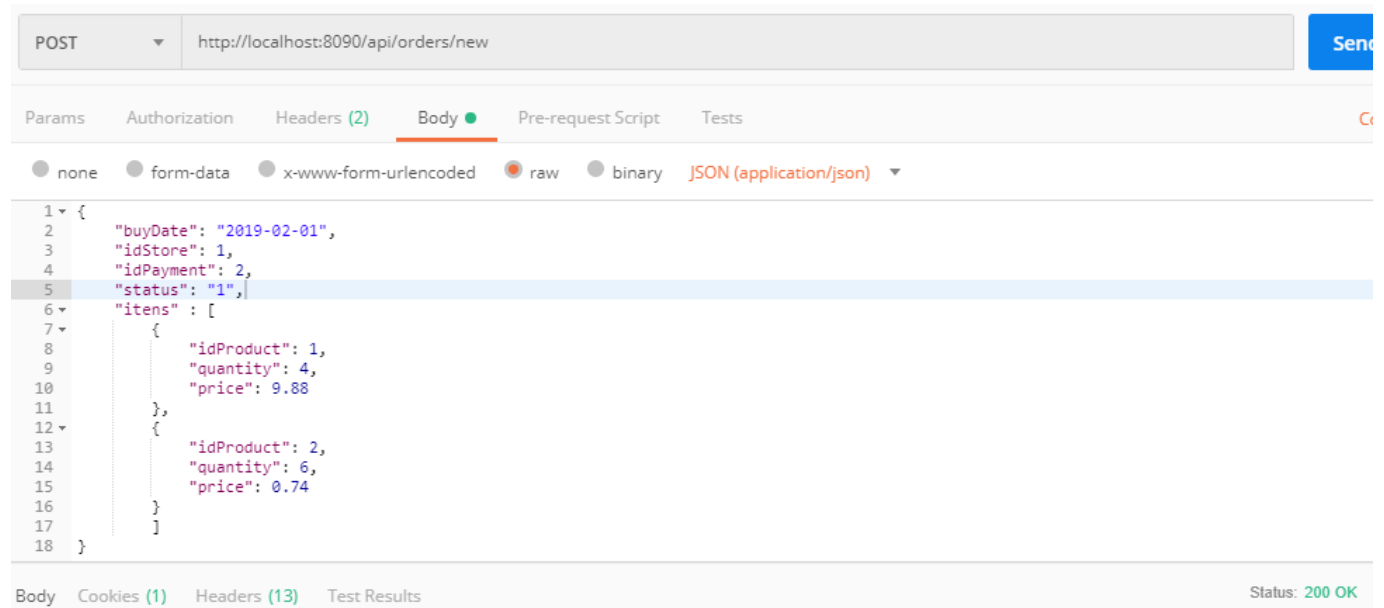
2.1.3 – Orders



The screenshot shows a REST client interface with a PUT request to `http://localhost:8090/api/orders/update/7`. The request body is a JSON object representing an order update. The status bar at the bottom indicates a successful response with `Status: 200 OK`.

```
1 {  
2   "id": 7,  
3   "buyDate": "2019-02-17",  
4   "idStore": 1,  
5   "idPayment": 1,  
6   "status": "1",  
7   "itens" : [  
8     {  
9       "idProduct": 1,  
10      "quantity": 7,  
11      "price": 77  
12    }  
13  ]  
14 }
```

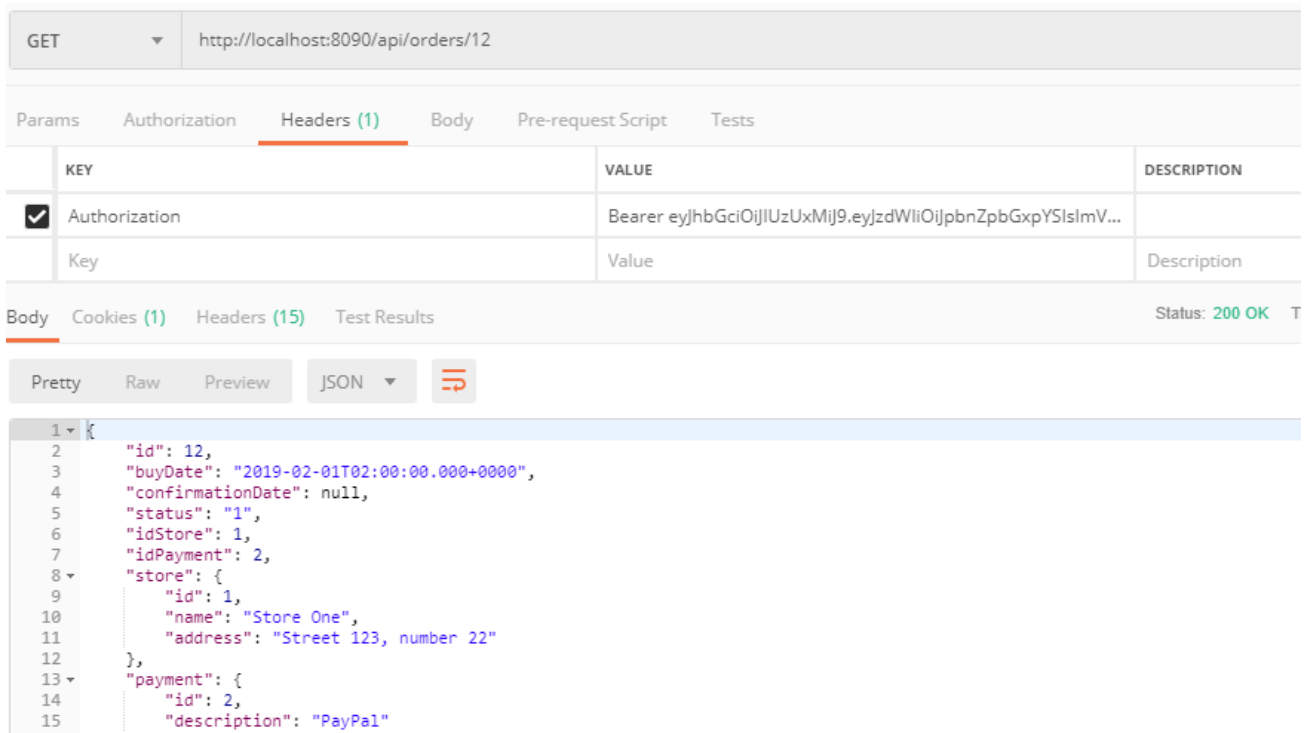
Figura 14 – Alteração de pedido



The screenshot shows a REST client interface with a POST request to `http://localhost:8090/api/orders/new`. The request body is a JSON object representing a new order. The status bar at the bottom indicates a successful response with `Status: 200 OK`.

```
1 {  
2   "buyDate": "2019-02-01",  
3   "idStore": 1,  
4   "idPayment": 2,  
5   "status": "1",  
6   "itens" : [  
7     {  
8       "idProduct": 1,  
9       "quantity": 4,  
10      "price": 9.88  
11    },  
12    {  
13      "idProduct": 2,  
14      "quantity": 6,  
15      "price": 0.74  
16    }  
17  ]  
18 }
```

Figura 15 – Criação de pedido



GET http://localhost:8090/api/orders/12

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

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWliOiJpbmZpbGxpYSIsImV...	
Key	Value	Description

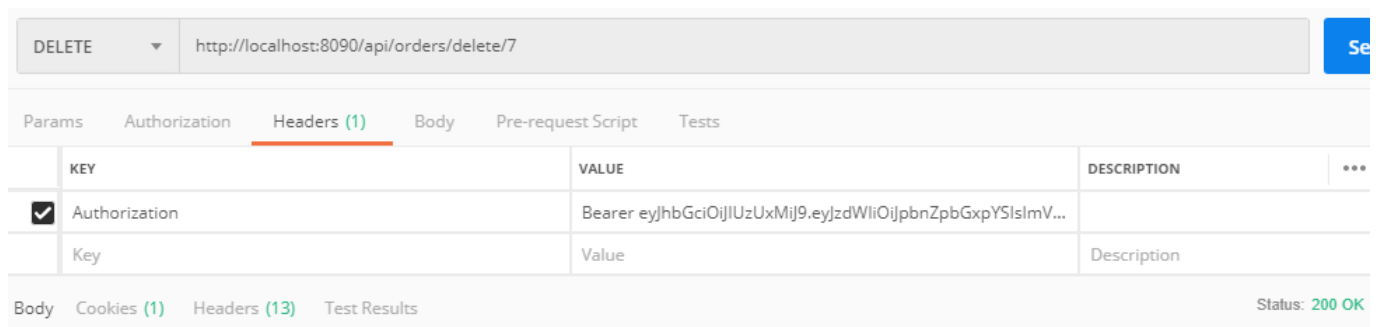
Body Cookies (1) Headers (15) Test Results Status: 200 OK

Pretty Raw Preview JSON

```

1 {
2   "id": 12,
3   "buyDate": "2019-02-01T02:00:00.000+0000",
4   "confirmationDate": null,
5   "status": "1",
6   "idStore": 1,
7   "idPayment": 2,
8   "store": {
9     "id": 1,
10    "name": "Store One",
11    "address": "Street 123, number 22"
12  },
13  "payment": {
14    "id": 2,
15    "description": "PayPal"
16  }
17 }
```

Figura 16 – Pedido por id



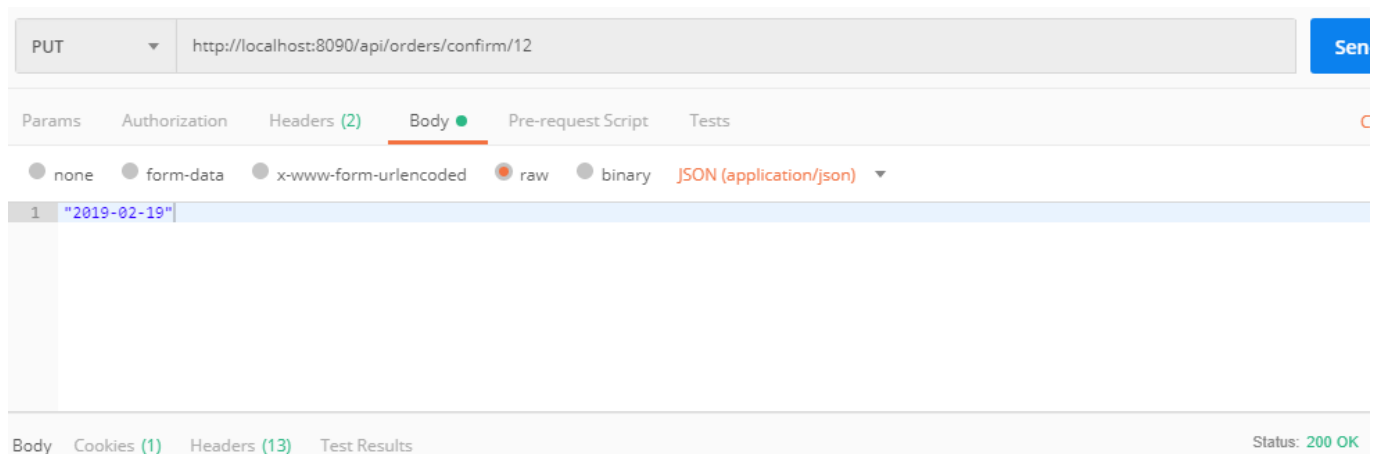
DELETE http://localhost:8090/api/orders/delete/7

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

KEY	VALUE	DESCRIPTION	...
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWliOiJpbmZpbGxpYSIsImV...		
Key	Value	Description	

Body Cookies (1) Headers (13) Test Results Status: 200 OK

Figura 17 – Exclusão de pedido



PUT http://localhost:8090/api/orders/confirm/12

Params Authorization Headers (2) Body Pre-request Script Tests

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

```

1 "2019-02-19"

```

Body Cookies (1) Headers (13) Test Results Status: 200 OK

Figura 18 – Confirmação de pedido