

NOUS

SPRINT #3

Johan Steven Anzola Hernandez

David McEwen Arango

Lina María Cardona Giraldo

Jairo Andrés Arbeláez Calderón

Santiago Rodriguez Serna

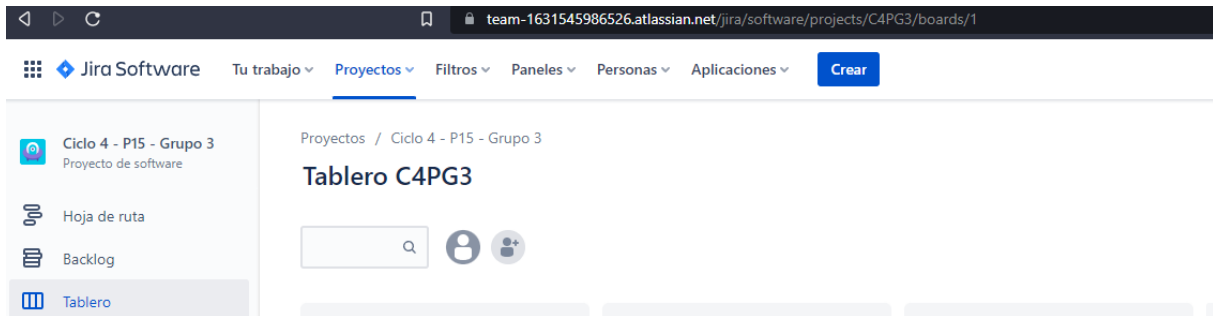
Diciembre 03 de 2021



Ciclo 4 Programación WEB

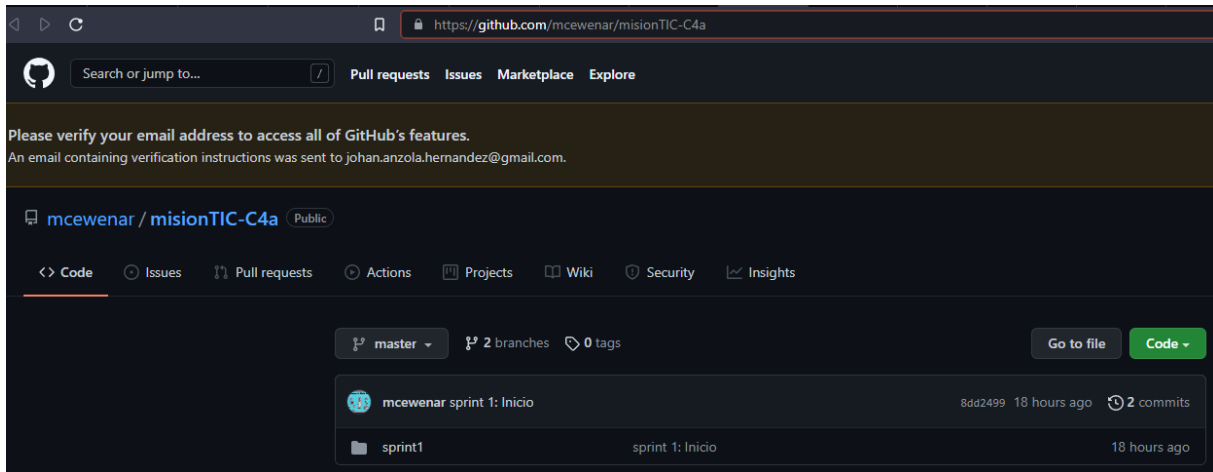
1. Enlace del proyecto en Jira

<https://team-1631545986526.atlassian.net/jira/software/projects/C4PG3/boards/1/backlog?selectedIssue=C4PG3-9>



2. Enlace del proyecto en bitbucket o Git hub

https://github.com/mcewenar/misionTIC-C4a/tree/master/proveedores_ms





3. Lista de tareas realizadas

Para el tercer sprint se desarrolló otro el microservicio de PROVEEDORES con el Framework DJANGO (Python) conectada a una base de datos NoSQL(MongoDB) en Atlas, con una API de tipo REST. Para ello realizaron las siguientes tareas configuradas en el software Jira:

Micro servicio de proveedores

- ☒ Creación de Modelos—Proveedores
- ☒ Creación de Serializadores—Proveedores
- ☒ Creación de Vistas—Proveedores
- ☒ Creación de API—REST
- ☒ Creación y vinculación de Mongo-DB

Micro servicio de Facturas

- ☒ Creación de Modelos
- ☒ Creación de Repositorio
- ☒ Creación de Excepciones
- ☒ Creación de Controladores
- ☐ Despliegue del microservicio y pruebas

4. Explicación general del código realizado

4.1 Micro servicio de proveedores

Se creó el modelo de **proveedores** usando Django, en el cual se va a permitir un CRUD para ingresar, ver, modificar y eliminar mediante un ID autogenerado por el ORM. Para este modelo, se ingresarán por parte del user-admin varios campos tales como:

- Nombre de la compañía (no permite repetir).
- Nombre de contacto.
- Celular
- Dirección
- Ciudad
- Correo electrónico

```
proveedores_ms > proveedoresApp > models > PY proveedor.py > Proveedor
mcewenar, a week ago | 1 author (mcewenar)
1 from django.db import models
2 from django.core.validators import RegexValidator
3 from django.db.models.fields import AutoField
4 from django.db.models.fields.related import ForeignKey
mcewenar, a week ago | 1 author (mcewenar)
5 class Proveedor(models.Model):
6     idRegister = models.AutoField('idRegister',primary_key=True)
7     name_company = models.CharField('Name_company',max_length=50,null=False, unique=True)
8     name_contact = models.CharField('Name_contact',max_length=50,null=False)
9     #validador de celular:
10    phone_regex = phone_regex = RegexValidator(regex=r'^\+?1?\d{9,15}$', message="Phone number must be entered i
11    cel = models.CharField(validators=[phone_regex], max_length=17, blank=False)
12    address = models.CharField('Address',max_length=80,null=False)
13    city = models.CharField('City',max_length=50,null=False)
14    email = models.EmailField('Email', max_length = 100,null=True)
15
```

(https://github.com/mcewenar/misionTIC-C4a/blob/master/proveedores_ms/proveedoresApp/models/proveedor.py)

También, se creó el serializador para proveedores, que permite convertir objetos a JSON y viceversa, en el cual se registran todos los datos que se van a comunicar con la capa del cliente.

```
Dockerfile M PY proveedorSerializer.py 1 X
proveedores_ms > proveedoresApp > serializers > PY proveedorSerializer.py > ProveedorSerializer > Meta
mcewenar, a week ago | 1 author (mcewenar)
1 from rest_framework import serializers
2 from proveedoresApp.models.proveedor import Proveedor
mcewenar, a week ago | 1 author (mcewenar)
3 class ProveedorSerializer(serializers.ModelSerializer):
mcewenar, a week ago | 1 author (mcewenar)
4     class Meta:
5         model = Proveedor
6         fields = ['idRegister','name_company','name_contact','cel','address','city','email']
mcewenar, a we
```

https://github.com/mcewenar/misionTIC-C4a/blob/master/proveedores_ms/proveedoresApp/serializers/proveedorSerializer.py



Para generar la vista `Proveedor` que va a mostrar un endpoint que comunicará con el api-gateway que está basado en clases que heredan comportamientos ya definidos por Django: la clase **`ProveedorListCreateView`**, permite listar todos los proveedores y crear un proveedor basado en un único ID. Luego está la clase **`ProveedorRetrieveUpdateDestroy`**, como su nombre lo indica, permite leer, actualizar y eliminar un proveedor basado en un ID específico.

```
proveedores_ms > proveedoresApp > views > PY proveedorView.py > ProveedorListCreateView
mcewenar, a week ago | 1 author (mcewenar)
1  from rest_framework import generics
2  from django.core.exceptions import ObjectDoesNotExist
3  from proveedoresApp.models.proveedor import Proveedor
4  from proveedoresApp.serializers.proveedorSerializer import ProveedorSerializer
5  #from rest_framework import views, status
6  #from rest_framework.response import Response
7
8  #List, Create
mcewenar, a week ago | 1 author (mcewenar)
9  class ProveedorListCreateView(generics.ListCreateAPIView): mcewenar, a week ago * app proveedores, realizad
10     queryset = Proveedor.objects.all()
11     serializer_class = ProveedorSerializer
12
13  #Read, Update, Delete
mcewenar, a week ago | 1 author (mcewenar)
14  class ProveedorRetrieveUpdateDestroy(generics.RetrieveUpdateDestroyAPIView):
15     queryset = Proveedor.objects.all()
16     serializer_class = ProveedorSerializer
17
```

https://github.com/mcewenar/misionTIC-C4a/blob/master/proveedores_ms/proveedoresApp/views/proveedorView.py

Creamos 2 endpoints para que, próximamente, serán consumidas por el API-GATEWAY mediante una conexión con api rest:

```
proveedores_ms > proveedoresProject > PY urls.py > ...
mcewenar, a week ago | 1 author (mcewenar)
1 from proveedoresApp.views.proveedorView import ProveedorListCreateView, ProveedorRetrieveUpdateDestroy
2 #from proveedoresApp.views.proveedorView import proveedorApi
3 from django.contrib import admin
4 from django.urls import path
5 from proveedoresApp import views mcewenar, a week ago * app proveedores, realizado. Falta desplegar la ap...
6 #from django.conf.urls import url,include
7
8
9 urlpatterns = [
10     path('admin/', admin.site.urls),
11     path('proveedores/', views.ProveedorListCreateView.as_view()), #Ver todos los proveedores, crear un proveedor
12     path('proveedor/<int:pk>', views.ProveedorRetrieveUpdateDestroy.as_view()), #Borrar, editar y consultar un
13 ]
14
```

Para conectar Django con la base de datos MongoDB en Atlas, realizamos un par de pasos adicionales:

1. Instalamos 2 dependencias específicas:

- djongo == 1.3.6
- dnspython == 2.1.0

```
proveedores_ms > requirements.txt
mcewenar, a week ago | 1 author (mcewenar)
1 django==3.2.7
2 djangoRESTframework==3.12.4
3 djangoRESTframework-simplejwt==4.8.0
4 dnspython==2.1.0
5 gunicorn==20.1.0
6 djongo==1.3.6
7 django-heroku==0.3.1 mcewenar, a week ago * app proveedores, realizado. Falta desplegar la ap...
```

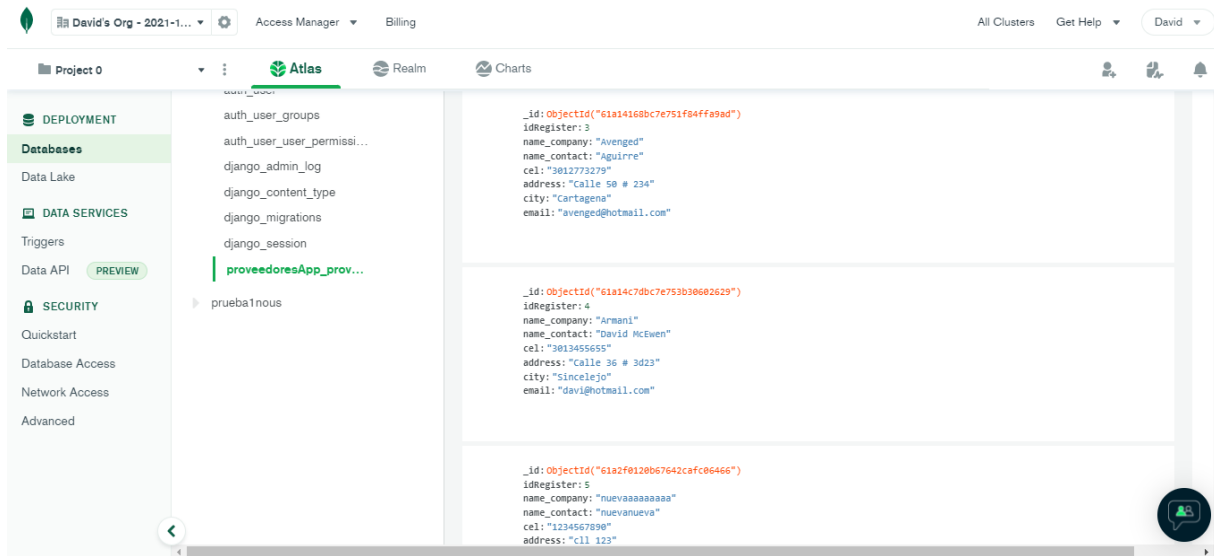
Luego, en el settings.py del proyecto de Django, creamos la conexión con la BD desplegada en la nube con Atlas:



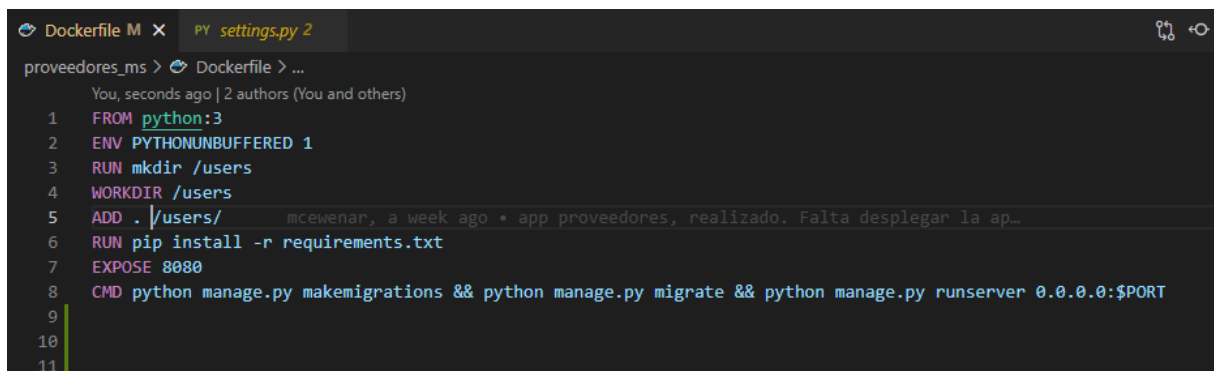
```
proveedores_ms > proveedoresProject > PY settings.py > ...
90
91
92 WSGI_APPLICATION = 'proveedoresProject.wsgi.application'
93
94
95 # Database
96 # https://docs.djangoproject.com/en/3.2/ref/settings/#databases
97
98 DATABASES = {
99     'default': {
100         'ENGINE': 'django',
101         'NAME': 'Prueba1NOUS',
102         'ENFORCE_SCHEMA': False,
103         'CLIENT': {
104             "name": "NOUSProveedor",
105             "host": "mongodb+srv://mcewenar:1234@prueba1nous.xg57g.mongodb.net/NOUSProveedor?retryWrites=true",
106             "username": "mcewenar",
107             "password": "1234",
108             "authMechanism": "SCRAM-SHA-1",
109         }
110     }
111 }
112
113
114 # Password validation
115 # https://docs.djangoproject.com/en/3.2/ref/settings/#auth-password-validators
116
```

(https://github.com/mcewenar/misionTIC-C4a/blob/master/proveedores_ms/proveedoresProject/settings.py)

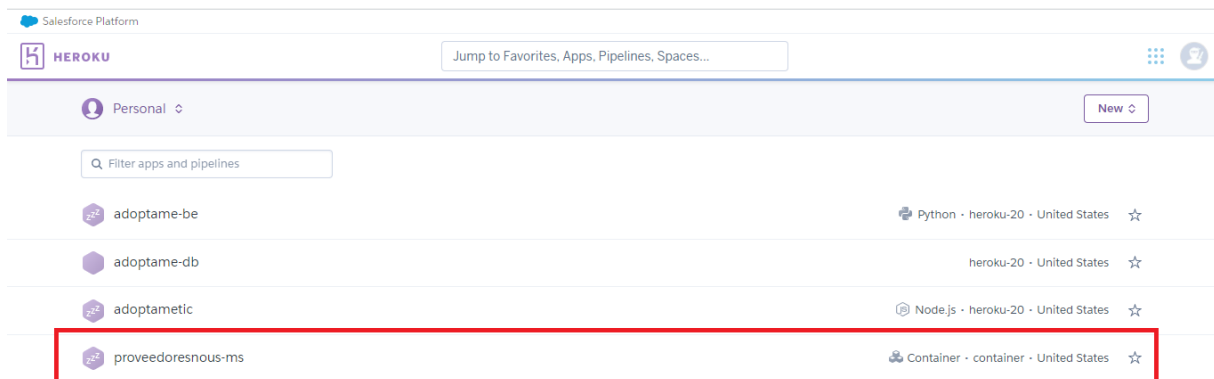
La BD en Atlas, se ve así:



Con los pasos anteriores, creamos un CRUD para proveedores, pero nos queda faltando el despliegue del contenedor en Heroku. Para esto usamos el archivo de aprovisionamiento (Dockerfile):

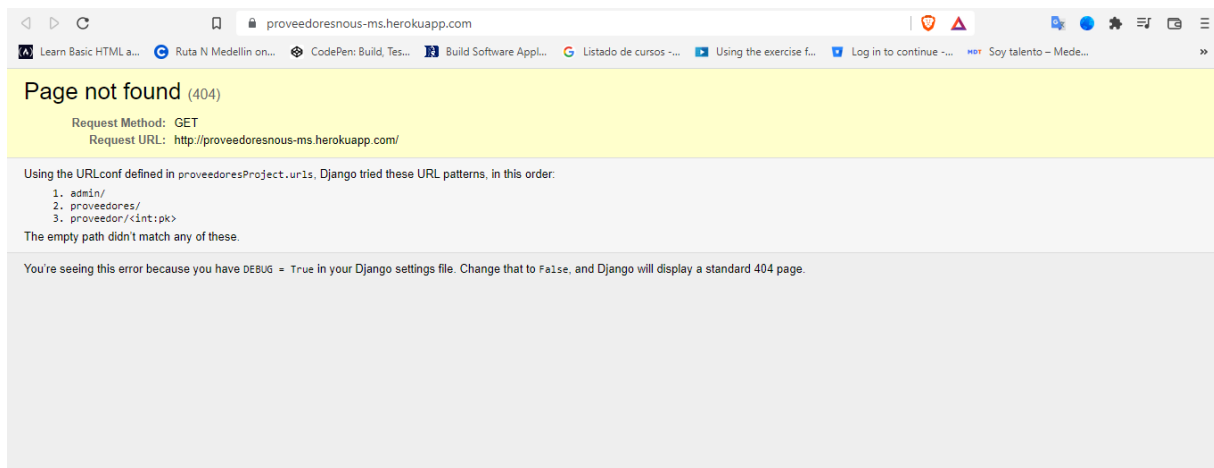


Luego creamos una aplicación en Heroku donde vamos a alojar nuestro microservicio **proveedoresnous-ms**:



Se despliega mediante un proceso y, cuando se aloje en la web, veremos esto. A partir de este momento, se puede consumir este microservicio desde cualquier parte.

URL: <https://proveedoresnous-ms.herokuapp.com/>



Hacemos una petición de tipo get en Postman que nos arrojará todos los registros de proveedores:



```

1  {
2    {
3      "idRegister": 3,
4      "name_company": "Avenged",
5      "name_contact": "Aguirre",
6      "cel": "3012773279",
7      "address": "Calle 50 # 234",
8      "city": "Cartagena",
9      "email": "avenged@hotmail.com"
10   },
11   {
12     "idRegister": 4,
13     "name_company": "Armani",
14     "name_contact": "David McEwen",
15     "cel": "3013455655",
16     "address": "Calle 36 # 3d23",
17     "city": "Sincelejo",
18     "email": "davi@hotmail.com"
19   }
20 }

```

4.2 Micro servicio de Facturas

Para este micro servicio se desea Guardar y consultar las facturas emitidas por el carrito realizado en el frontend que hará uso del localStorage para generar las listas, esta información será recibida por el api-gateway que gestionará las peticiones al presente microservicio y así llevar el registro de facturas. Para cumplir esto se hará uso de **MongoDB** como parte del componente de base de datos no-relacional y iniciaremos el espacio de trabajo con **Spring Initializr** desde su página principal <https://start.spring.io/> La configuración será la siguiente:

- Project: Maven Project
- Language: Java Version 11
- Spring Boot: 2.5.7
- Packaging: Jar
- Dependencias:
 - Spring Web
 - Spring Data MongoDB

El código se desarrollará con la ayuda de **IntelliJ IDEA**, se realiza la conexión con la base de datos esto se realiza desde el archivo "application.properties", se crea la estructura del proyecto la cual será la siguiente:

Creación de models:

En la carpeta models se crea el modelo del producto este recibe la información necesaria del producto para poderla guardar en el modelo de Factura que cuenta con un arreglo de los productos que se

agregue al carrito de compras, los dos archivos se pueden ver de la siguiente manera con sus respectivos setter y getter que por practicidad no se pegará ese fragmento de código:

```
package com.misiontic.facturacion_ms.models;
import org.springframework.data.annotation.Id;
public class Product {
    @Id
    private String idproduct;
    private String name;
    private Integer amount;
    private Integer value;

    public Product(String idproduct, String name, Integer amount, Integer value){
        this.idproduct = idproduct;
        this.name = name;
        this.amount = amount;
        this.value = value;
    }
}
```

```
1 package com.misiontic.facturacion_ms.models;
2 import org.springframework.data.annotation.Id;
3 import java.util.Date;
4
5 import java.util.List;
6
7 public class Factura {
8     @Id
9     private String idfactura;
10    private String username;
11    private String address;
12    private List<Product> products;
13    private Integer totalValue;
14    private Date date;
15
16    public Factura(String idfactura, String username, String address, List<Product> products, Integer totalValue, Date date) {
17        this.idfactura = idfactura;
18        this.username = username;
19        this.address = address;
20        this.products = products;
21        this.totalValue = totalValue;
22        this.date = date;
23    }
24 }
```

Creación de repositories:

En la carpeta de repositorio se establece la gestión de la base de datos, pero esta vez solo del modelo Factura ya que este incluye los Productos como objetos.

```

1 package com.misiontic.facturacion_ms.repositories;
2 import com.misiontic.facturacion_ms.models.Factura;
3 import org.springframework.data.mongodb.repository.MongoRepository;
4 import java.util.List;
5 public interface FacturaRepository extends MongoRepository<Factura, String> {
6     List<Factura> findByUsername (String username);
7 }
8
9

```

Creación de exceptions:

Para las excepciones, durante las pruebas se identificó necesario crear excepciones para consultar facturas por nombres distintos al usuario

Creación de Controllers:

En esta carpeta se crea el controlador de Factura, debido a que se necesita tener el historial de facturas éstas no se podrán borrar ni actualizar, pero si, podremos crearlas y consultarlas, por medio de decoradores de **Spring Boot** como **@PostMapping** **@GetMapping**, Adicionalmente con el decorador **@RestController** se identifica a la clase como un controlador:

```

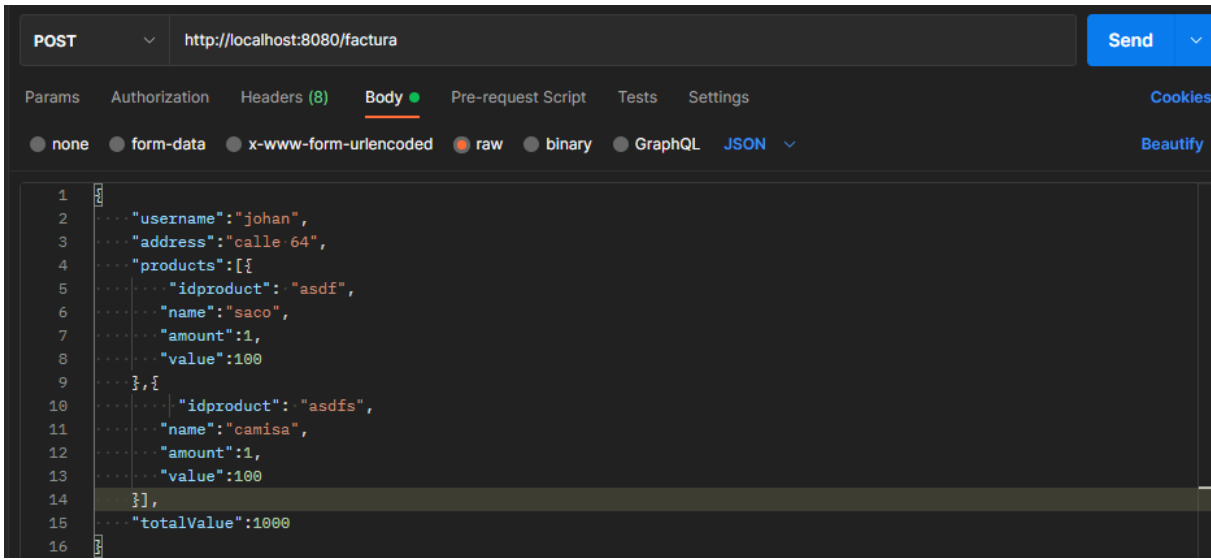
1 package com.misiontic.facturacion_ms.controllers;
2 import com.misiontic.facturacion_ms.models.Factura;
3 import com.misiontic.facturacion_ms.repositories.FacturaRepository;
4 import org.springframework.web.bind.annotation.*;
5 import java.util.Date;
6 import java.util.List;
7
8 @RestController
9 public class FacturaController {
10     private final FacturaRepository facturaRepository;
11     public FacturaController(FacturaRepository facturaRepository){
12         this.facturaRepository = facturaRepository;
13     }
14     @PostMapping("/factura")
15     @ResponseBody
16     Factura newFactura(@RequestBody Factura factura){
17         factura.setDate(new Date());
18         return facturaRepository.save(factura);
19     }
20     @GetMapping("/factura/{username}")
21     List<Factura> userFactura(@PathVariable String username){
22         // Factura userFactura = facturaRepository.findById(username).orElse(null);
23         List<Factura> facturas = facturaRepository.findByUsername(username);
24         return facturas;
25     }
26 }

```

Despliegue del microservicio y pruebas:

Dado que se deben realizar pequeñas modificaciones identificadas en las pruebas se aplazó el despliegue, pero a continuación se puede observar que cumple con los principales objetivos del microservicio:

Prueba Post

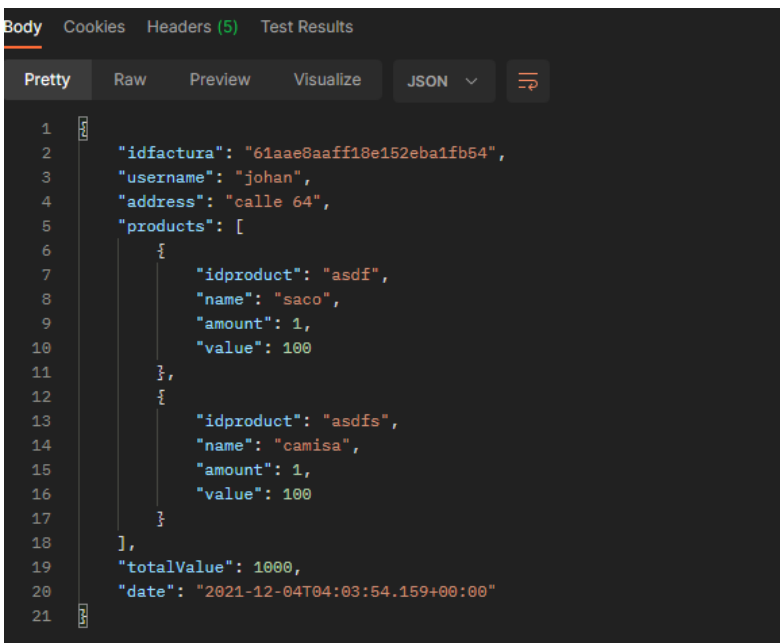


```

1  {
2    "username": "johan",
3    "address": "calle 64",
4    "products": [
5      {
6        "idproduct": "asdf",
7        "name": "saco",
8        "amount": 1,
9        "value": 100
10     },
11     {
12       "idproduct": "asdfs",
13       "name": "camisa",
14       "amount": 1,
15       "value": 100
16     }
17   ],
18   "totalValue": 1000
19 }

```

Respuesta:



```

1  {
2    "idfactura": "61aae8aaff18e152eba1fb54",
3    "username": "johan",
4    "address": "calle 64",
5    "products": [
6      {
7        "idproduct": "asdf",
8        "name": "saco",
9        "amount": 1,
10       "value": 100
11     },
12     {
13       "idproduct": "asdfs",
14       "name": "camisa",
15       "amount": 1,
16       "value": 100
17     }
18   ],
19   "totalValue": 1000,
20   "date": "2021-12-04T04:03:54.159+00:00"
21 }

```



Comprobamos en la base de datos:

+ Create Database

Q NAMESPACES

Facturacion_ms

factura

myFirstDatabase

Facturacion_ms.factura

COLLECTION SIZE: 576B TOTAL DOCUMENTS: 2 INDEXES TOTAL SIZE:

Find Indexes Schema Anti-Patterns 0 Aggregations

FILTER { field: 'value' }

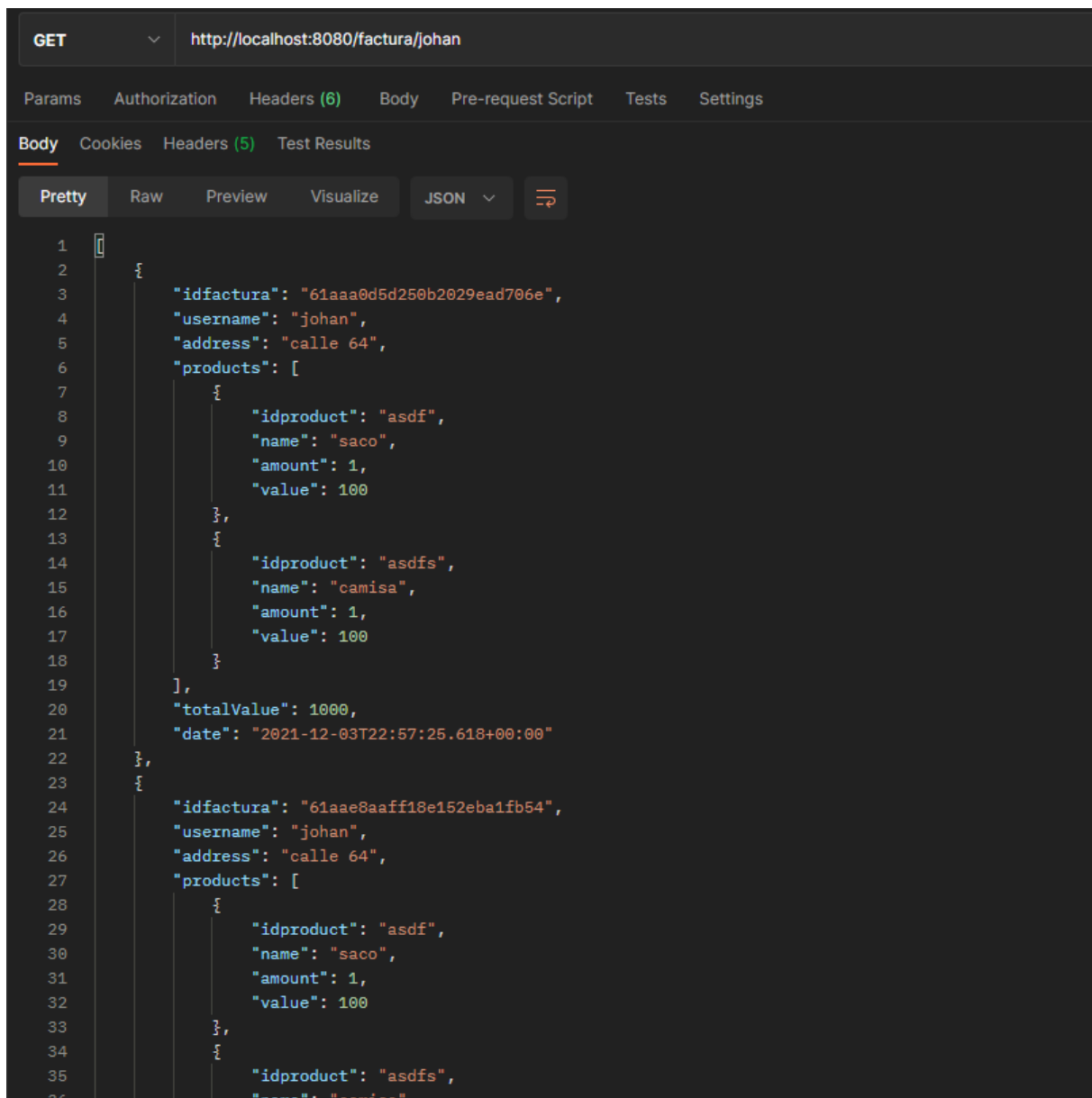
QUERY RESULTS 1-2 OF 2

```

_id: ObjectId("61aaa0d5d250b2029ead706e")
username: "johan"
address: "calle 64"
products: Array
  0: Object
    _id: "asdf"
    name: "saco"
    amount: 1
    value: 100
  1: Object
    _id: "asdfs"
    name: "camisa"
    amount: 1
    value: 100
totalValue: 1000
date: 2021-12-03T22:57:25.618+00:00
_class: "com.misiontic.facturacion_ms.models.Factura"

```

Prueba Get



```

1  [
2    {
3      "idfactura": "61aaa0d5d250b2029ead706e",
4      "username": "johan",
5      "address": "calle 64",
6      "products": [
7        {
8          "idproduct": "asdf",
9          "name": "saco",
10         "amount": 1,
11         "value": 100
12       },
13       {
14         "idproduct": "asdfs",
15         "name": "camisa",
16         "amount": 1,
17         "value": 100
18       }
19     ],
20     "totalValue": 1000,
21     "date": "2021-12-03T22:57:25.618+00:00"
22   },
23   {
24     "idfactura": "61aae8aaff18e152eba1fb54",
25     "username": "johan",
26     "address": "calle 64",
27     "products": [
28       {
29         "idproduct": "asdf",
30         "name": "saco",
31         "amount": 1,
32         "value": 100
33       },
34       {
35         "idproduct": "asdfs",
36         "name": "camisa",
  
```

Como se puede apreciar, cumple con los objetivos del Ms_Facturas queda pendiente la tarea del despliegue y unión con el repositorio principal para el siguiente Spring.