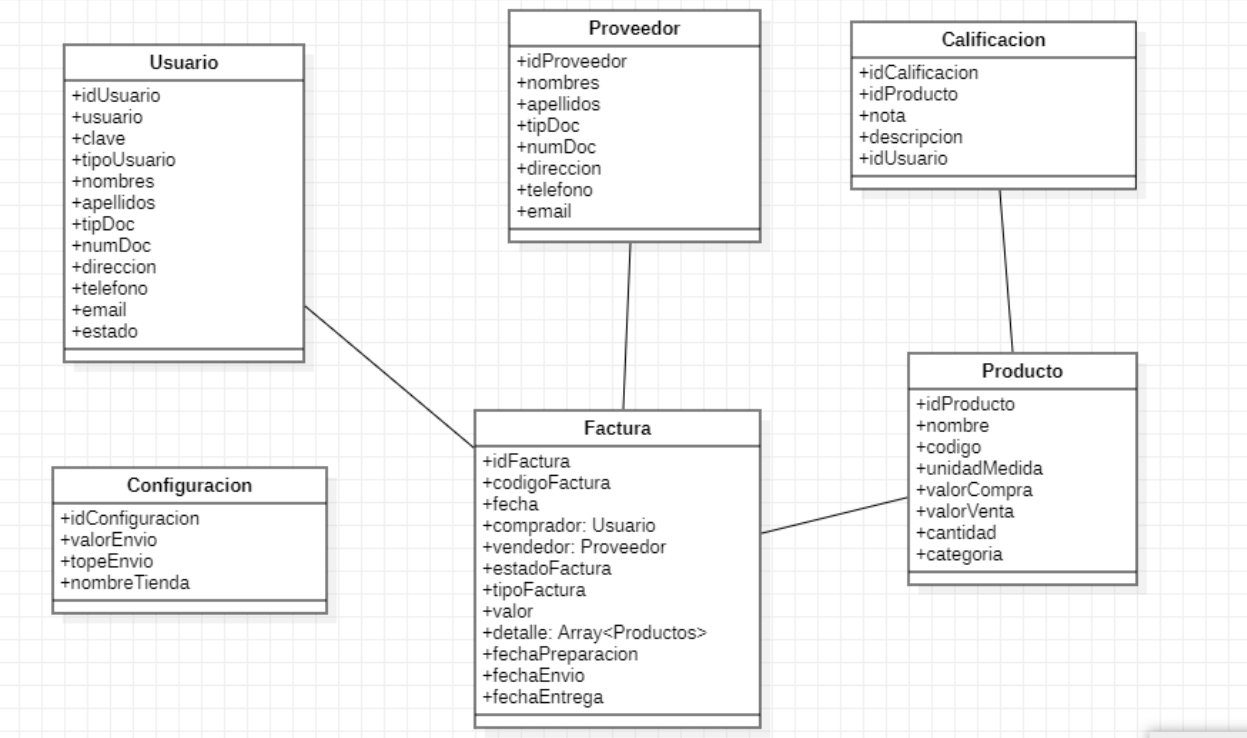
Ciclo 4A

SPRINT 2 – DESARROLLANDO EL BACKEND

EQUIPO: U29_MiDulceOnline_10

DIAGRAMA DE COLECCIONES DE LA BASE DE DATOS



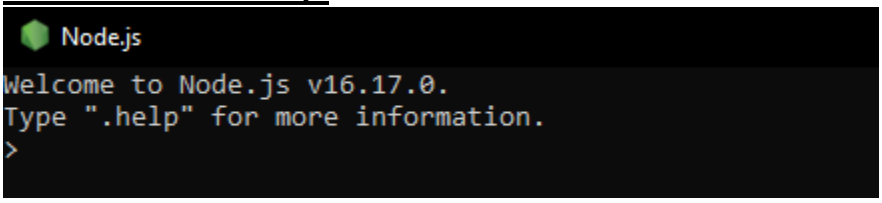
COMANDOS BASICOS DE LA ESTRUCTURA BACKEND Y DEPENDENCIAS REQUERIDAS

```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "start": "nodemon app.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "mongoose": "^6.6.7"
  },
  "devDependencies": {
    "nodemon": "^2.0.20"
  }
}
```

DOCUMENTACIÓN DEL PROCESO REALIZADO:

Descripción detallada del proceso realizado para el montaje del backend, incluyendo las dificultades presentadas.

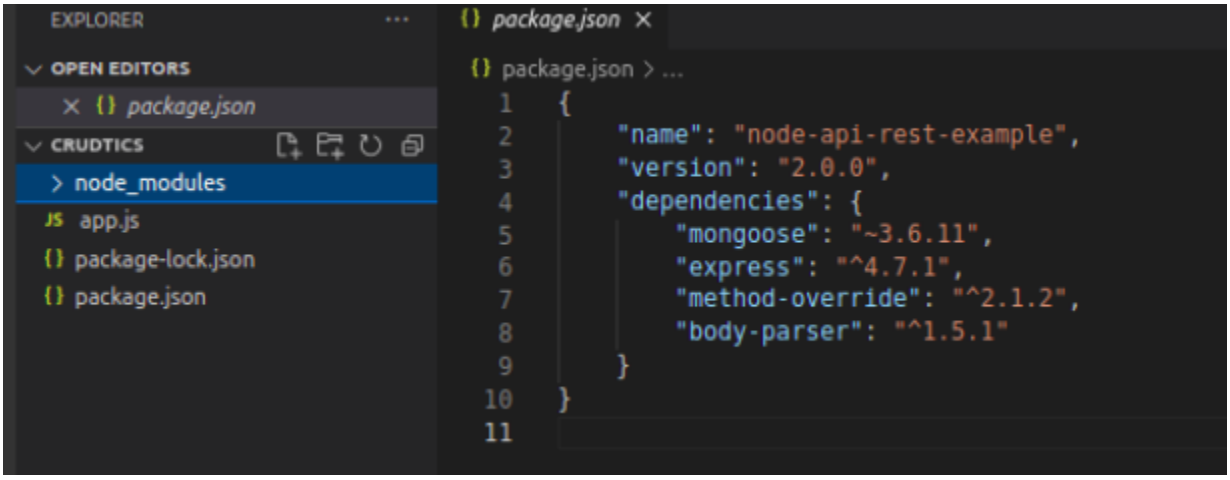
a) Instalación de nodejs



b) Configuración de Dependencias en VScode

Paso 1. Se Crea un directorio en el repositorio donde se llevan el proyecto, con su nombre en minúscula, separado por guiones.

Paso 2. Se abre la carpeta creada con el editor de código Vscode (Visual Studio Code)



Desde la terminal, se inicializa este paquete desde la terminal con el comando:

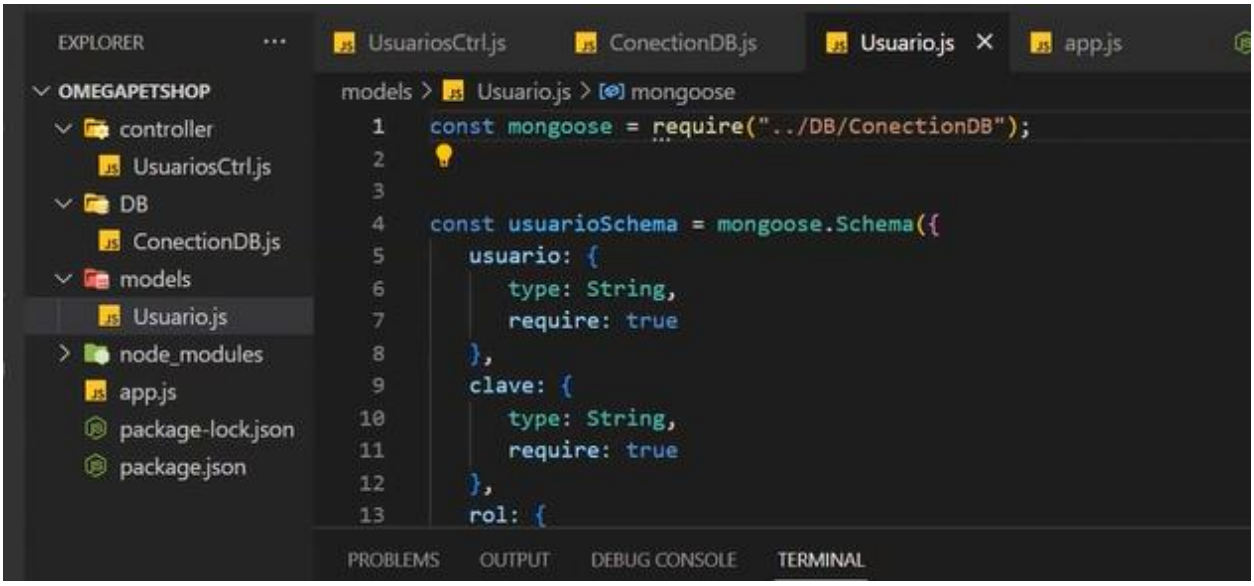
npm init

se crea automáticamente los paquetes json y se inicia la instalación de los paquetes necesarios para desarrollar el backend con los siguientes comandos:

- npm install express –save, se generará la carpeta node_modules, en la cual podrás almacenar los módulos necesarios para tu proyecto
- npm install nodemon -D --save
- npm install cors --save
- npm install mongoose –sabe
- modificar script package.json con el siguiente script para inicializar archivo nodemoon:

```
"scripts": {
   "start": "nodemon app.js"

 }
```

c) **<u>creación de la estructura base del backend, con las siguientes carpetas:</u>**



● routers: En esta se especificarán cada uno de los manejadores de rutas montables y modulares.
● models: Especificarán los modelos donde se trabaja con los datos, por tanto, contendrá mecanismos para acceder a la información y también para actualizar su estado.
● controllers: funciona para obtener los datos solicitados de los modelos y crear la información necesaria para devolvérsela a los clientes en formato json.

d) **Creamos un archivo index.js** , app.js

creamos los archivos de index.js, el cual será nuestro archivo de puno de entrada inicial, en el cual tiene el siguiente código:

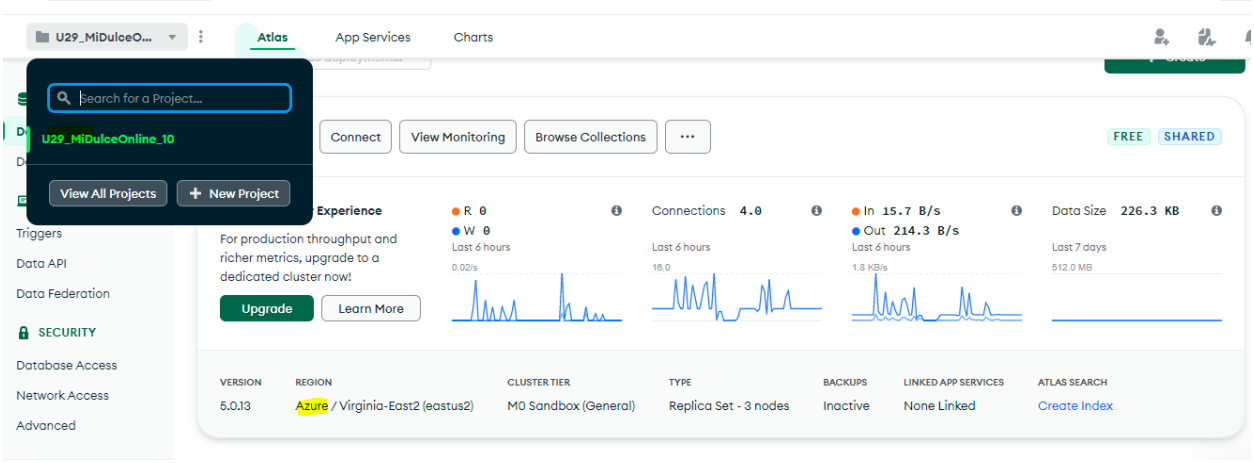
```
//Este archivo es el punto de entrada del proyecto
//Llamado  a la app la cual se encuentra en el archivo app.js
var app = require("./app");
var mongoose = require("./conexBD/db");
var port = process.env.PORT || 4000;//Llamado al puerto por variables de
entorno y asignación de puerto en caso de no haberla

app.listen(port, () => {//Ejecución de la función que pone en ejecución la
aplicación en el puerto indicado
    console.log("servidor corriendo ok");
});
```

El archivo app.js es creado como archivo en donde se definen las importaciones

```
var express = require("express");
var cors = require("cors");
var app = express();

app.use(cors());
app.use(express.json());

app.use(require('./routers/routers'));

app.use(express.urlencoded({
    extended: true
}));


//Comando para indicar indicar condificacion de externo
app.use((req, res, next) => {
    res.header("Access-Control-Allow-Origin", "*");
    res.header(
        "Access-Control-Allow-Headers",
        "Authorization",
        "X-API-KEY",
        "Origin",
        "X-Requested-With",
        "Content-Type, Accept",
        "Access-Control-Allow-Request-Method"
    );
    res.header("Access-Control-Allow-Methods", "GET, POST, OPTIONS, PUT,
DELETE");
    res.header("Allow", "GET, POST, OPTIONS, PUT, DELETE");
    next();
});

module.exports = app;
```

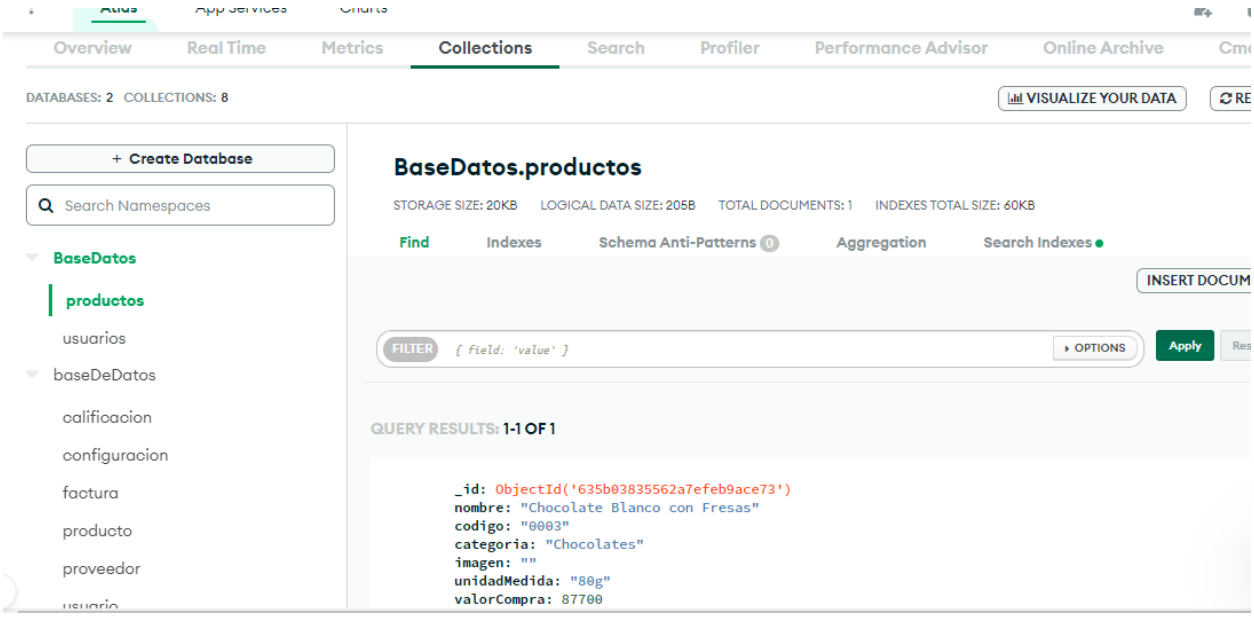
e) <u>Creación de colecciones y documentos desde la plataforma online de MongoDB:</u>

Creamos un usuario y agregamos el clúster en donde se implementará la base de datos del proyecto, para este caso se utilizó la tecnología de AZURE, se modificó el nombre del proyecto:



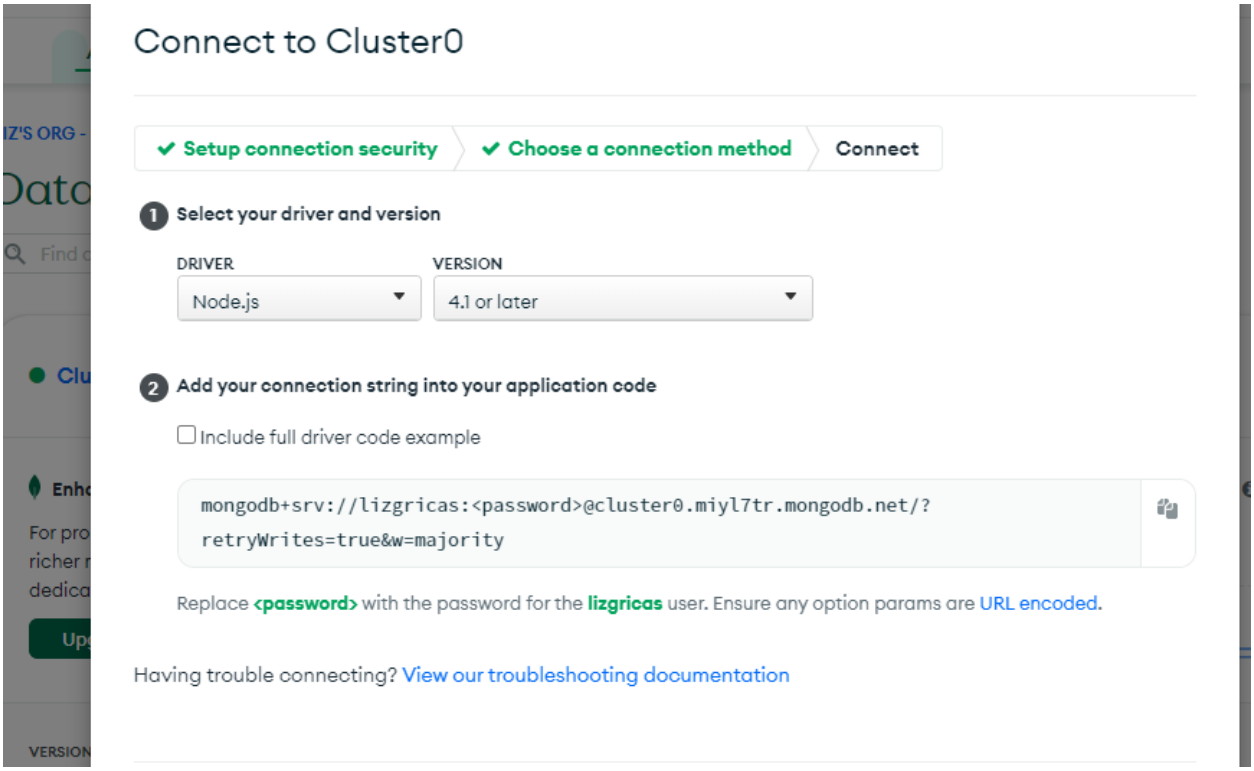
f) **<u>Creación de base de datos de prueba</u>**:

Creamos una base de datos de prueba directamente en la plataforma ingresando las colecciones de acuerdo al diagrama de clases creado en el sprint 1:

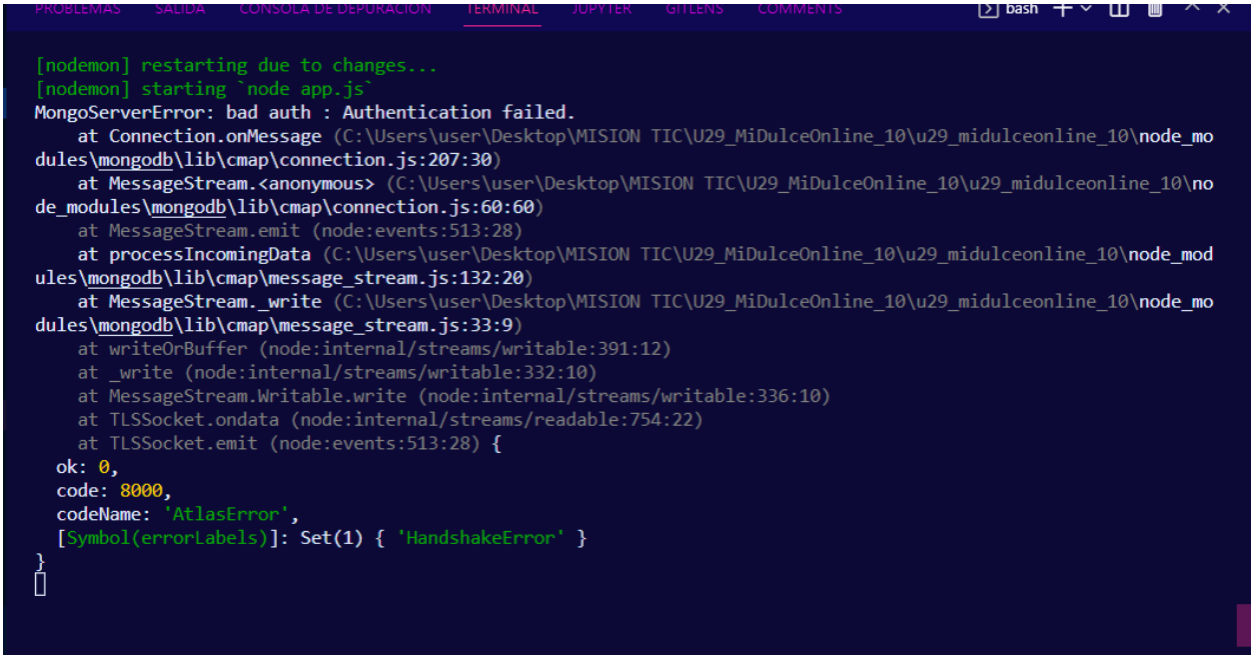


g) **<u>Conexión de Servidor y Base de datos:</u>**

Desde el archivo ConectionDB.js, creado en VScode, se ejecutó la conexión con la cuenta creada en MongoDB, añadiendo el url arrojado por la plataforma en el ítem connect – connect your application; se modificó igualmente el password según los datos ingresados inicialmente en la creación del clúster:

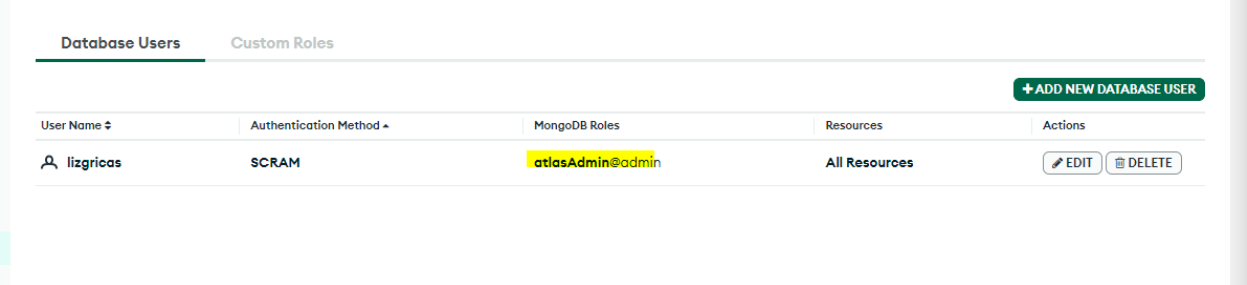


Cuando probamos la conexión al clúster de MongoDB con el comando npm start desde la terminal, ésta arrojo un error de autentificación fallida:

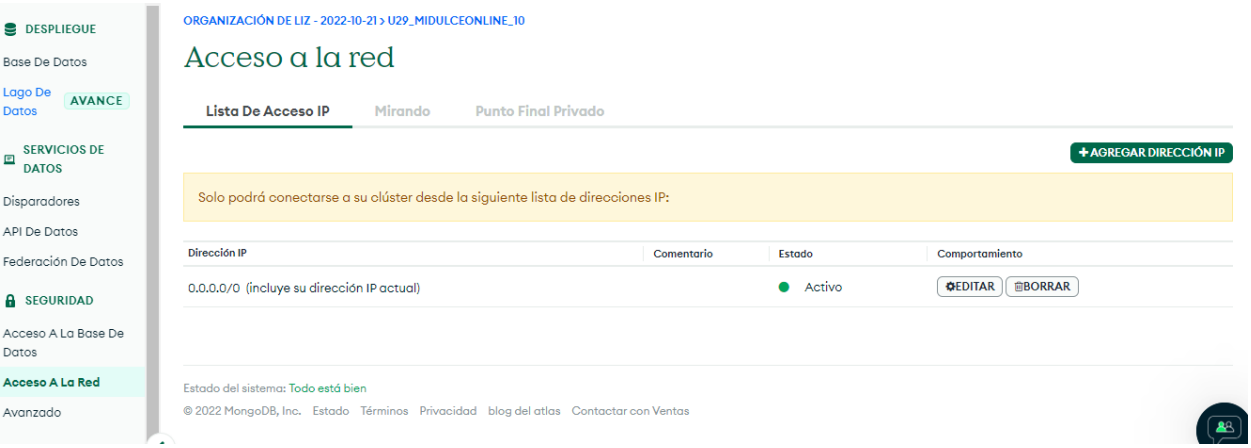


Para este inconveniente se debió realizar las siguientes modificaciones en la configuración de acceso del clúster en MongoDB:

Se modificó la clave (y se actualizó este dato en el archivo ConectionDB.js) y el rol de usuario por administrador para tener todos los permisos:



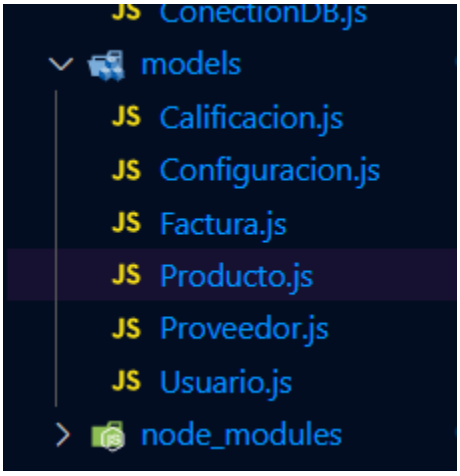
Se habilitó el acceso abierto a las IP que deseen conectarse al clúster, sin discriminar la lista de direcciones IP:



Se ejecutó nuevamente desde la terminal npm start y se estableció conexión con el clúster satisfactoriamente.

h) **<u>Creación de modelos de las colecciones requeridas en el proyecto</u>**:

se diseñó la representación lógica de las entidades (Tablas) en la carpeta models, agregando un archivo con extensión js por cada entidad requerida, es decir por producto se creó el archivo Producto.js, con la siguiente estructura, y así sucesivamente hasta completar todas las clases:




```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const productoSchema = new Schema({
    nombre: {
        type: String,
        required: true,
        unique: true
    },
    codigo: {
        type: String,
        required: true,
        unique: true
    },
    categoria: {
        type: String,
        required: true
    },
    imagen: {
        type: String,
    },
    unidadMedida: {
        type: String,
        required: true
    },
    descripcion: {
        type: String,
        required: true
    },
    valorCompra: {
```

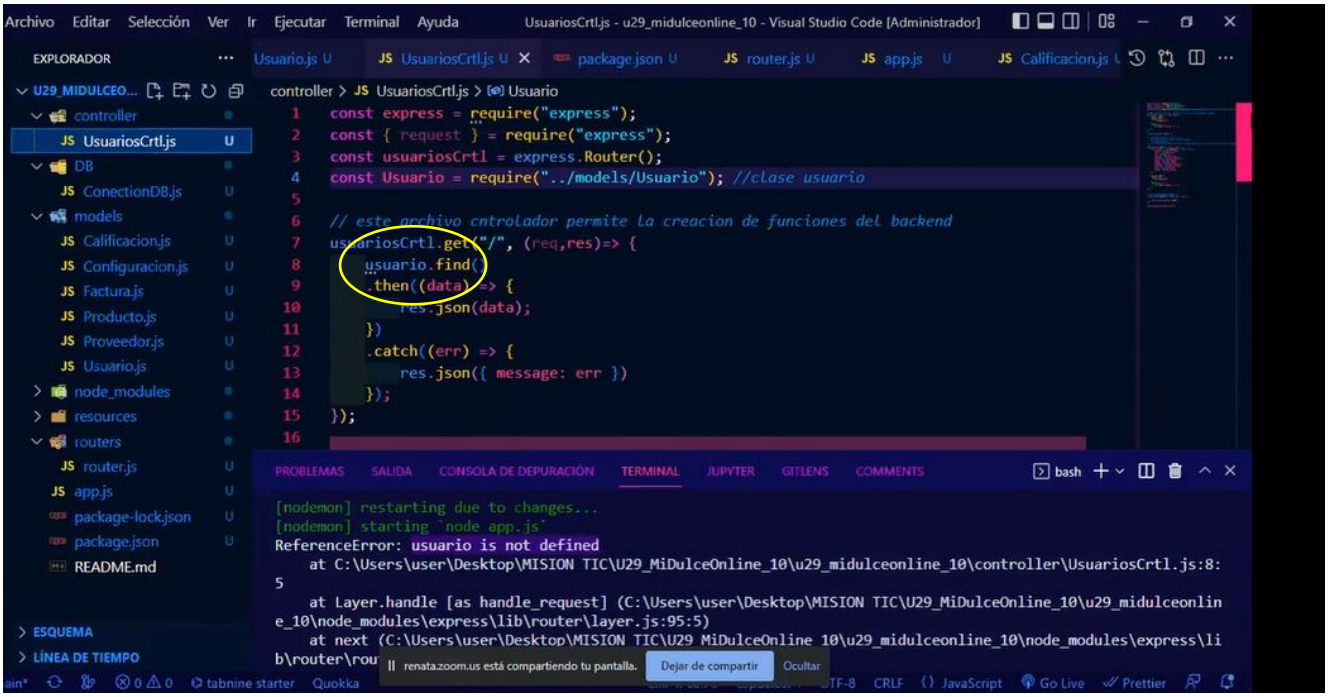
```
type: Number,
        required: true
    },
    valorVenta: {
        type: Number,
        required: true
    },
    sinRebaja: {
        type: Number,
        required: true
    },
    cantidad: {
        type: Number,
        required: true
    },
    calificacion: {
        type: Number
    },
    proveedor: {
        type: String,
        required: true
    }
}
    , {
        collection: 'productos',
        versionKey: false
    }
);



var Producto = mongoose.model('productos', productoSchema);
module.exports = Producto
```

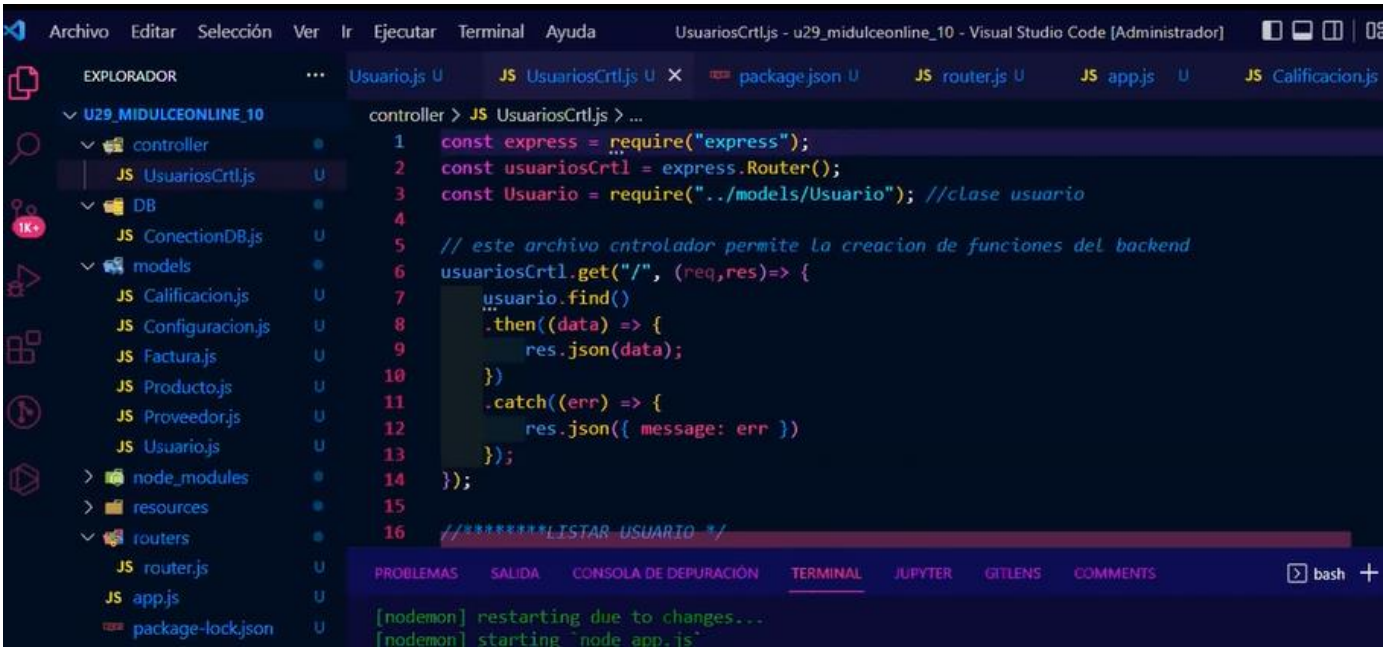
i) **Creación de controladores:**

se crea la carpeta controller y se ingresa por cada clase un archivo de extensión js con la siguiente estructura, definiendo los métodos get, post, delete, update, patch que serán las rutas para las peticiones al backend según lo requiera cada entidad:

```
const Producto = require("../models/Producto");

function all(req, res) {
    Producto.find()
        .then((data) => {
            res.json(data);
        })
        .catch((err) => {
            res.json({ message: err })
        });
}

function search(req, res) {

    let hopa = req.params;
    Producto.findById(hopa.id)
        .then((data) => {
            res.json(data);
        })
        .catch((err) => {
            res.json({ message: err })
        });
}

function save(req, res) {

    const producto = new Producto(req.body);

    console.log(producto)
    producto.save()
        .then((data) => {
            res.json(data);
        })
        .catch((err) => {
            res.json({ message: err })
        });

}

function destroy(req, res) {
    Producto.deleteOne({ _id: req.params.id })
        .then((data) => {
            res.json(data);
        })
        .catch((err) => {
```

- Se ejecuta desde el explorador con el link http://localhost:4000/prueba la funcionalidad del backend pero no se ejecuta la API
- Al ejecutar nuevamente desde la terminal el paquete de mongoose para establecer conexión con el clúster, la consola arrojó un error a la hora de guardar las colecciones creadas en el servidor, en la nube de la base de datos; el evento arrojó un Error de usuario no definido; para este inconveniente se debió modificar la constante usuario, con la letra inicial en mayúscula ya que representa la clase Usuario y no una nueva variable.





- se ejecuta nuevamente luego de realizar las modificaciones y se conecta a la base de datos satisfactoriamente, se procede a realizar las pruebas en el postman, de las colecciones creadas.

LISTADO DE LOS ENDPOINTS DEL PROYECTO:

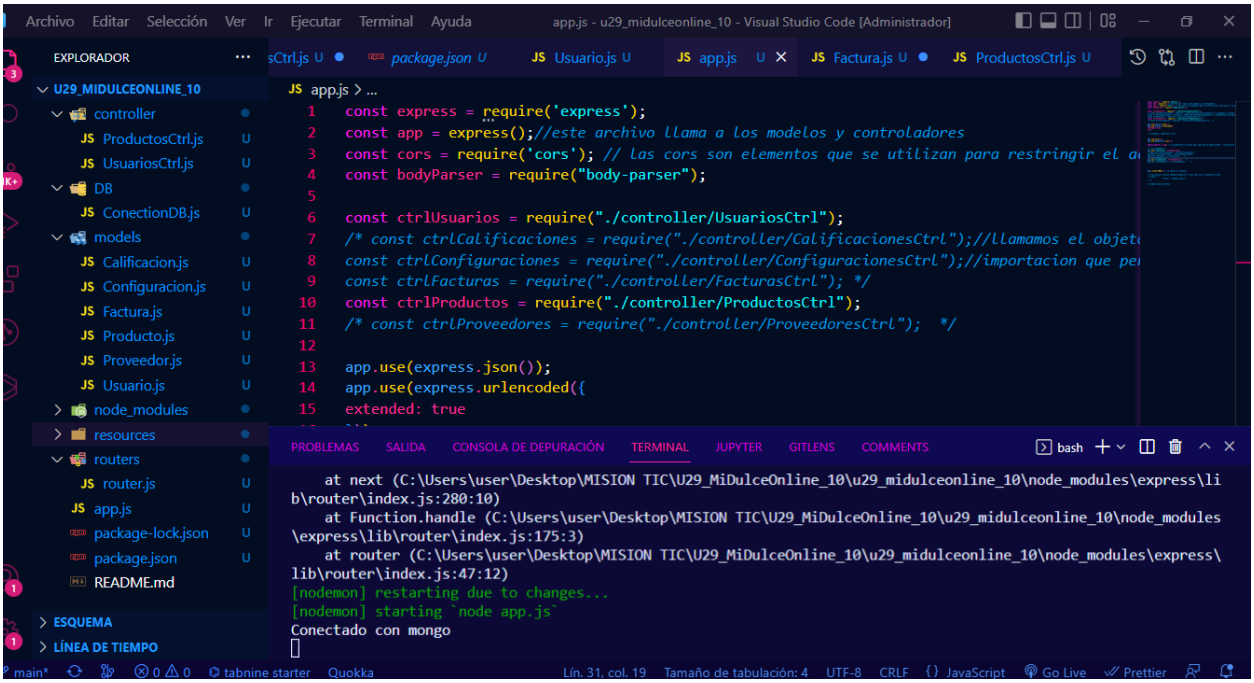
El listado de los endpoints del backend, junto con algunas capturas de pantalla del funcionamiento de estos en Postman, se presentan a continuación:

MODELOS Y CONTROLADORES EN BACKEND DEL PROYECTO

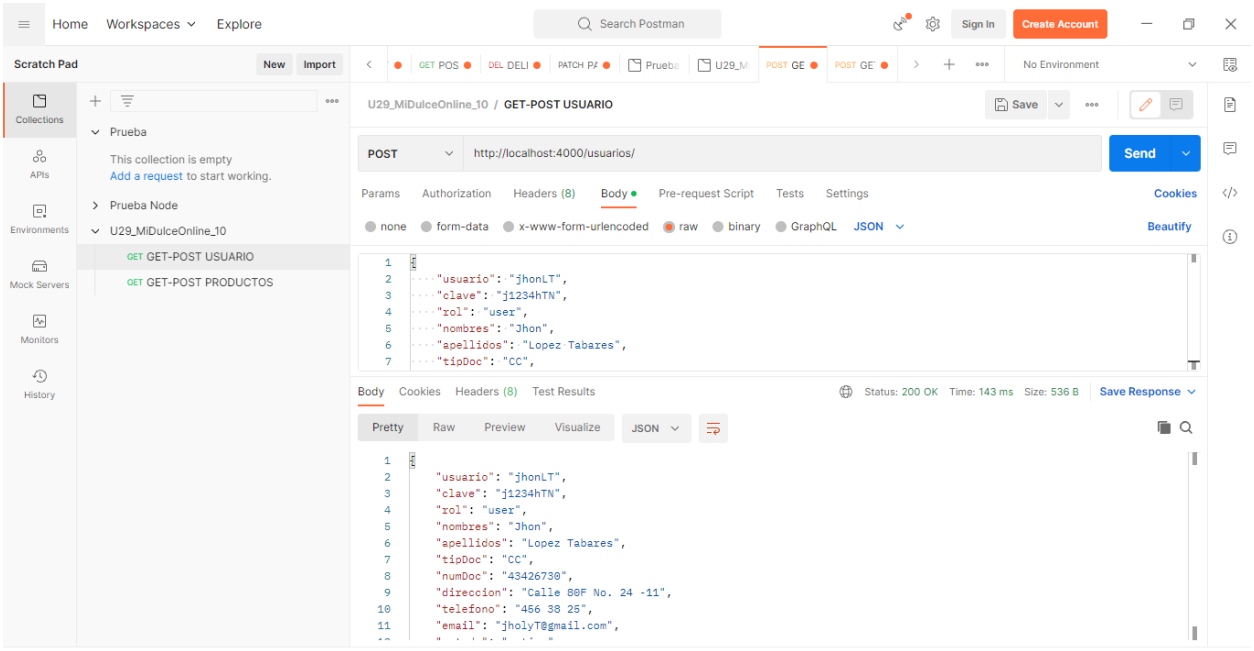
Universidad Industrial de Santander

Misión TIC 2022

Ciclo 4A



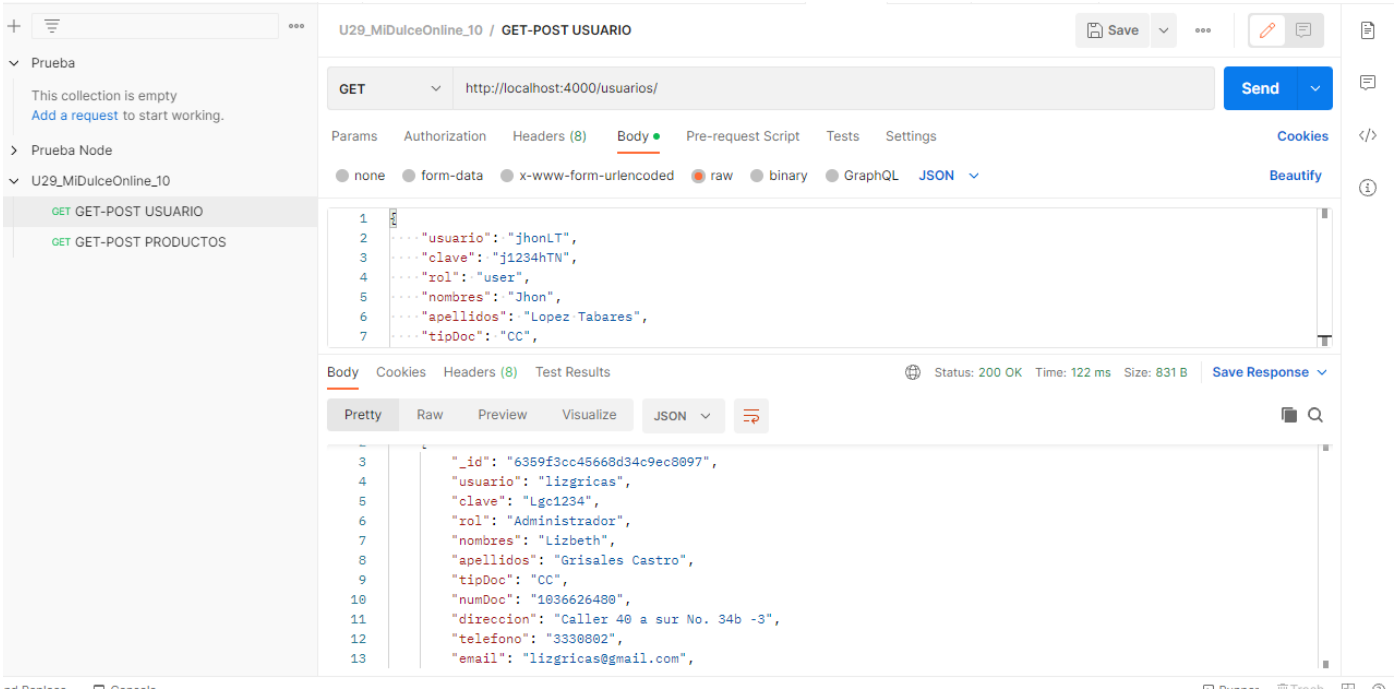
GET – POST COLECCIÓN USUARIOS

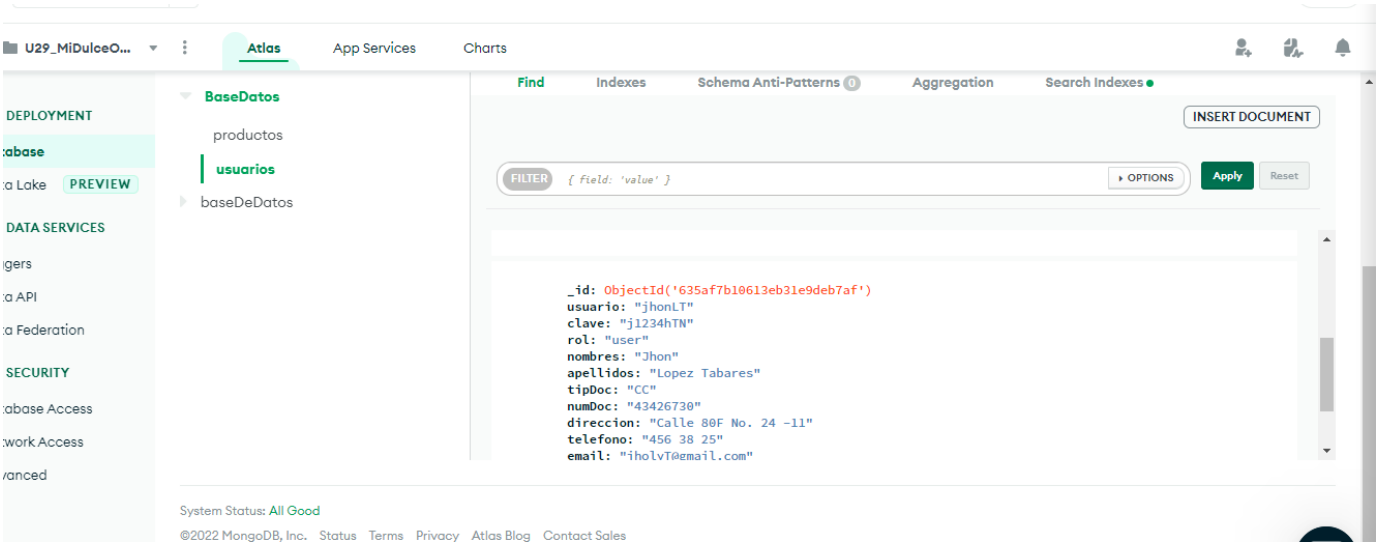


Universidad Industrial de Santander

Mision TIC 2022

Ciclo 4A



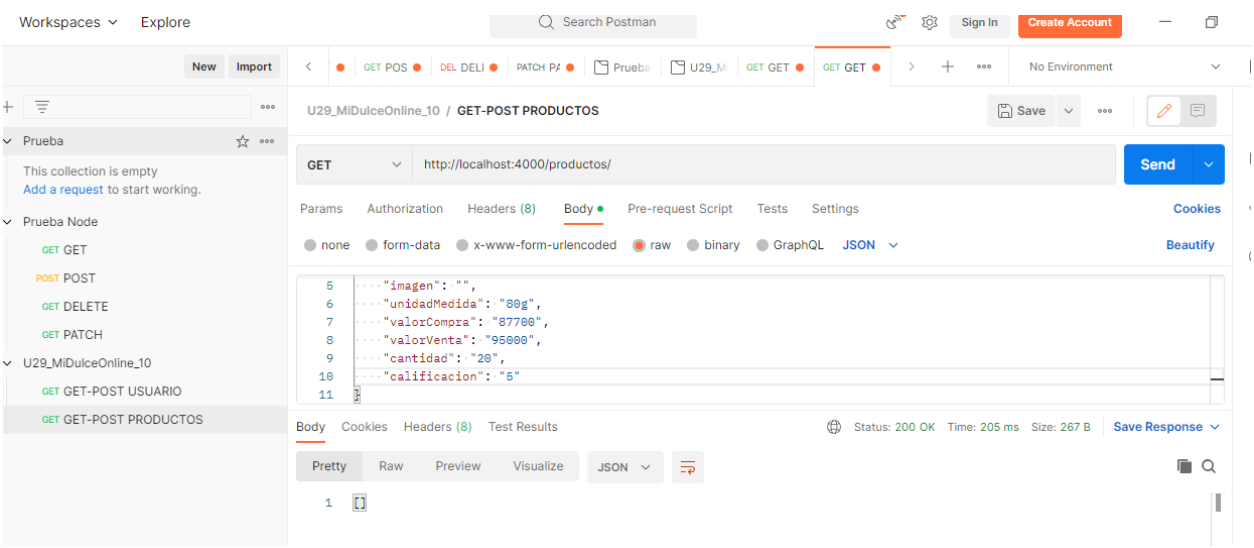


GET – POST COLECCIÓN PRODUCTOS

Universidad Industrial de Santander

Mision TIC 2022

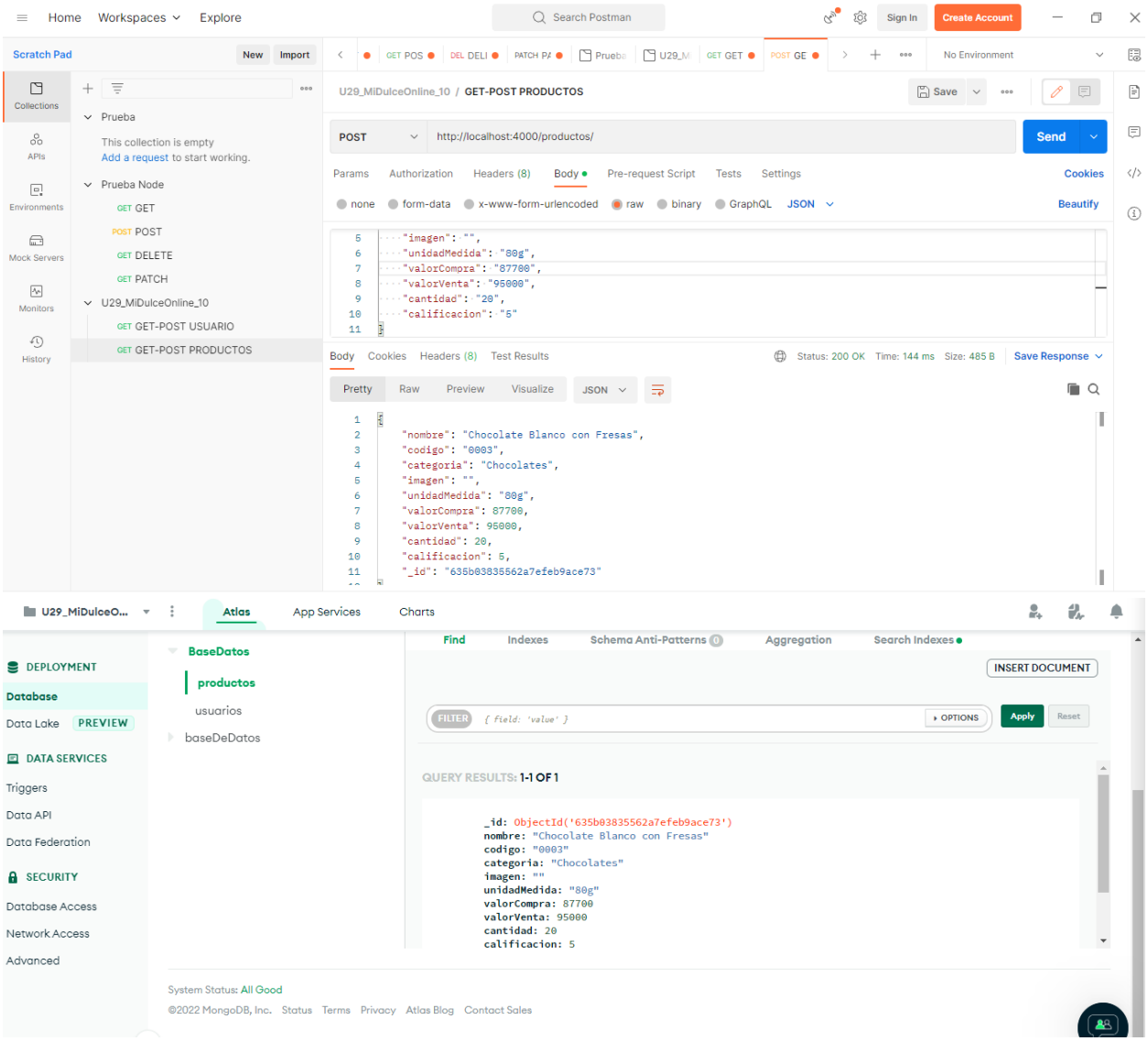
Ciclo 4A

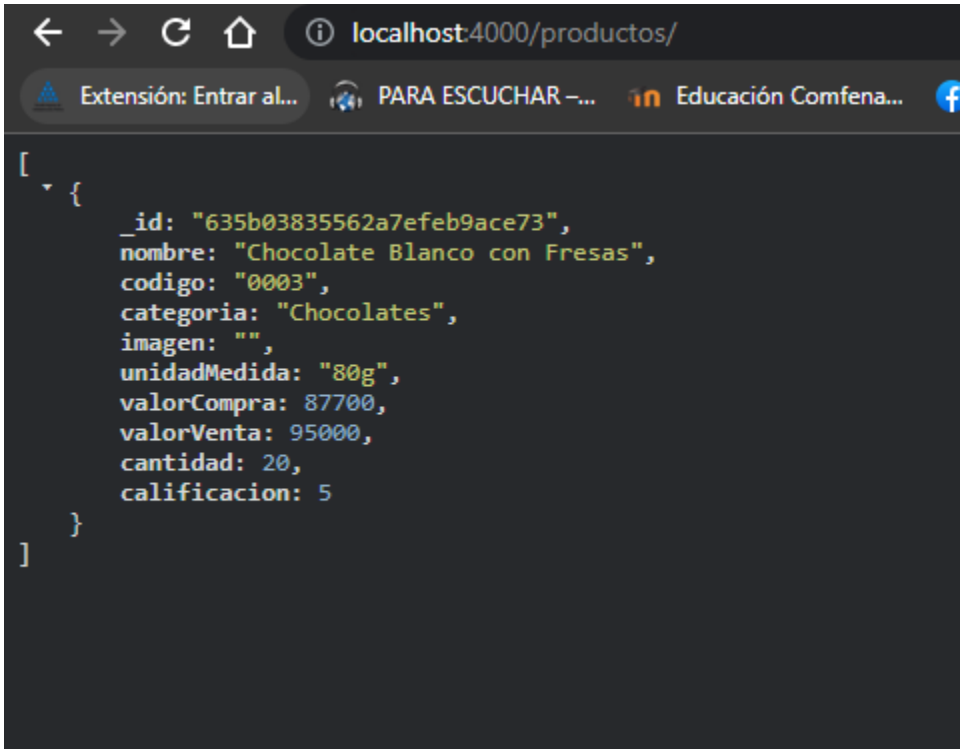


Universidad Industrial de Santander

Mision TIC 2022

Ciclo 4A





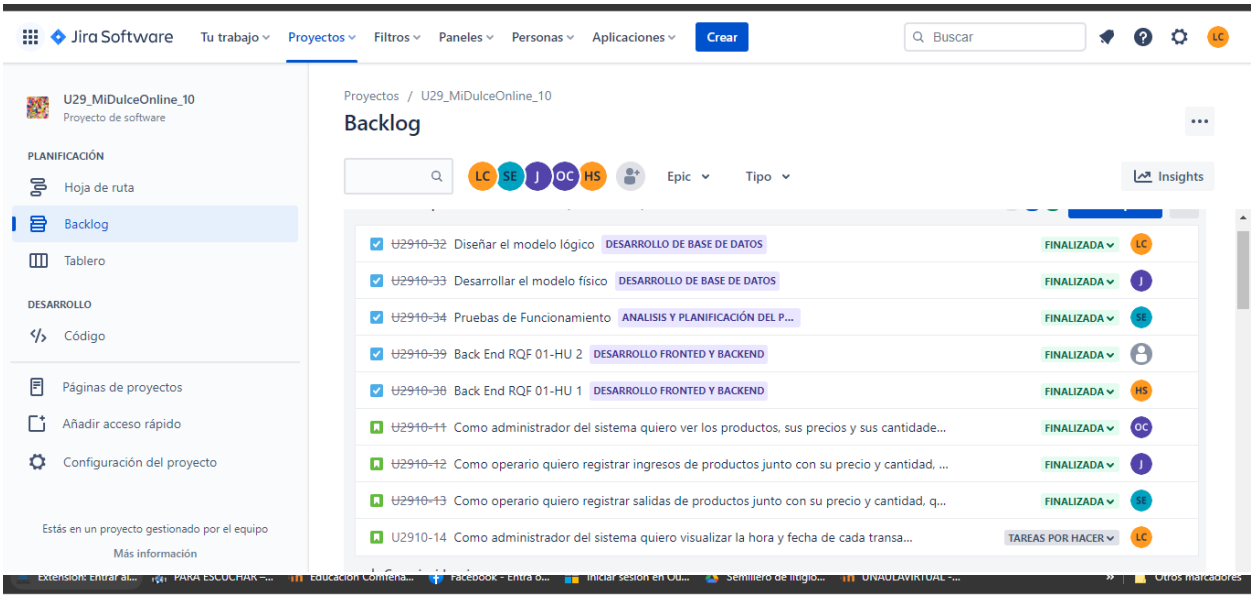
EVIDENCIAS DE USO DE JIRA:

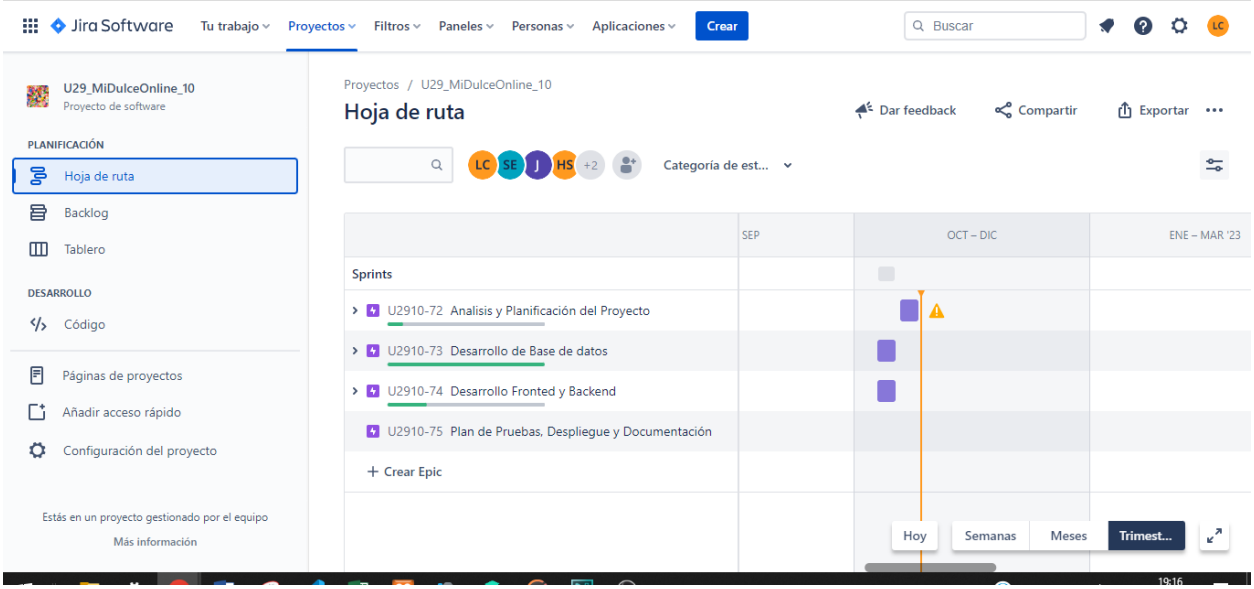
Capturas de pantalla del uso de Jira y de historias de usuario

Universidad Industrial de Santander

Misión TIC 2022

Ciclo 4A





Universidad Industrial de Santander

Mision TIC 2022

Ciclo 4A