I. Preparación del entorno

1. **Comprobar que tengo Node.js instalado**. Utilizo cmd todo el proyecto. Ejecuto node -v => **v24.11.1**

Node.js es un entorno que permite ejecutar JavaScript fuera del navegador. Antes de Node, JavaScript solo funcionaba dentro de Google Chrome / Firefox / Safari. Ahora con Node.js puedo usar JavaScript para crear aplicaciones reales, no solo páginas web.

2. **Comprobar qué versión tengo de npm** (*Node Package Manager*, gestor de paquetes que viene con Node.js y que me permitirá instalar librerías como express, mongoose, dotenv, etc. Para ello ejecuto npm -v => **11.6.2**
3. **Crear carpeta del proyecto** mediante mkdir backend-products-users
4. **Situarme en la carpeta del proyecto** mediante cd backend-products-users para trabajar siempre dentro de ella.
5. **Crear** index.js como archivo inicial desde el que arrancará todo el proyecto de backend. El index.js se crea en vacío mediante type NUL > index.js
6. Posteriormente le iremos añadiendo todas las funciones que debe tener:
   a) Cargar las variables de entorno => require("dotenv").config();
   b) Arrancar el servidor Express

```
const express = require("express");
const app = express();
```

   c) Definir los middlewares => app.use(express.json());
   d) Conectar a la BBDD de MongoDB => mongoose.connect(process.env.MONGO_URI)
   e) Incluir las rutas, como por ejemplo app.use("/api/users", userRoutes); o app.use("/api/products", productRoutes);
   f) Levantar el servidor

```
app.listen(PORT, () => {
 console.log(`Servidor escuchando en puerto ${PORT}`);
});
```

7. **Inicializar npm**

Para ello ejecuto npm init -y → crea package.json automáticamente ...\backend-products-users\package.json

```
{
 "name": "backend-products-users",
 "version": "1.0.0",
 "description": "",
 "main": "index.js",
 "scripts": {
  "test": "echo \"Error: no test specified\" && exit 1"
 },
 "keywords": [],
 "author": "",
 "license": "ISC",
 "type": "commonjs"
}
```

npm necesita un archivo de configuración y este archivo es el package.json,que se crea automáticamente cuando inicializo npm. Con el comando npm init -y le digo que cree un proyecto npm y que genere un package.json con todos los valores por defecto (-y)
Este archivo guarda los datos del proyecto (versión, autor, descripción, su punto de entrada "main": "index.js", etc) guarda las dependencias al instalar express, mongoose, dotenv...también guarda los scripts que voy creando

8. Y para ello se ejecutará node index.js o npm run dev (con nodemon)

9. Crear estructura base de carpetas. Mi proyecto debe tener las siguientes carpetas.

```
backend-products-users/
│
├── models/
│   ├── User.model.js
│   └── Product.model.js
│
├── routes/
│   ├── user.routes.js
│   └── product.routes.js
│
├── controllers/
│   ├── user.controller.js
│   └── product.controller.js
│
├── middlewares/
│   ├── auth.middleware.js
│   ├── role.middleware.js
│   └── upload.middleware.js
│
├── utils/
│   └── cloudinary.js
│
├── seeds/
│   └── product.seed.js
│
├── index.js
├── package.json
├── .env
└── .gitignore
```

El uso de cada carpeta es el siguiente:
.- routes para las rutas de registro y login
.- controllers para la lógica del registro, hash y creación del usuario en la BBDD
.- middlewares necesarios para la autenticación con JWT, para los roles y para la subida de imágenes a cloudinary
.- utils para la configuración de cloudinary (servicio de almacenamiento de imágene**s** en la nube)
.- seeds para el script necesario para crear productos. **seed** (o semilla) es un *script especial* que sirve para insertar muchos datos de forma automática en una BBDD.Según el enunciado "Se deberá realizar una semilla que nos permita subir un array de datos a una de las colecciones". Y por tanto usaremos una semilla para crear los productos.
.- models para los modelos de datos. Según el enunciado para productos y para usuarios (Necesitas mínimo 2 modelos en MongoDB. Y con un dato relacionado entre ellos (El modelo User tendrá un **dato relacionado** (array de favoritos) que guardará IDs de Product))

Para ello desde /backend-products-users/ ejecuto los siguientes comandos:

```
mkdir models
mkdir  routes
mkdir controllers
mkdir middlewares
mkdir utils
mkdir seeds
```

10. Instalar las dependencias base del servidor

Hay que instalar Express, que crea el servidor. Dotenv para leer las variables de entorno del .env y Nodemon para que se reinicie de forma automática el servidor durante el desarrollo. Para ello ejecutamos

```
npm install express  =>
added 68 packages, and audited 69 packages in 2s
16 packages are looking for funding
 run `npm fund` for details
```

found 0 vulnerabilities

npm install dotenv =>
added 1 package, and audited 70 packages in 764ms
17 packages are looking for funding
run `npm fund` for details
found 0 vulnerabilities

npm install nodemon --save-dev =>
added 27 packages, and audited 97 packages in 2s
21 packages are looking for funding
run `npm fund` for details
found 0 vulnerabilities

Al instalar las dependencias se crea la carpeta node_modules, que guardará todas las dependencias. Es una carpeta enorme que no debe subirse al GitHub y que debe estar en el .gitignore. También se crea el package-lock.json que guarda las versiones exactas y que sí se debe subir al GitHub.

11. Crear archivo .env

El archivo .env guarda valores privados (puertos, claves, URLs, tokens), y dotenv los carga automáticamente dentro de process.env
Creo el fichero mediante el comando type NUL > .env
Lo abrimos y añadimos el puerto PORT=4000
Más adelante añadiremos:

- URL de MongoDB
- Cloudinary
- JWT_SECRET

Ahora tenemos que modificar el index.js para usar dotenv. dotenv es una librería de Node.js que permite cargar las variables de entorno desde mi archivo .env a mi proyecto. Así en el .env guardo datos sensibles como el MONGO_URI y luego en el código lo uso a través de process.env.MONGO_URI

Para ello abrimos index.js y en la primera línea ponemos require("dotenv").config();
Además modificamos la parte del puerto, si existe PORT en el .env usará ese valor y si no existe usará el 4000

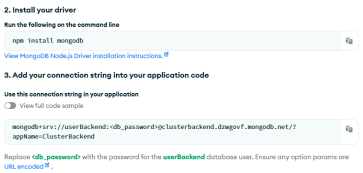
```
const PORT = process.env.PORT || 4000;
app.listen(PORT, () => {
  console.log(`Servidor escuchando en http://localhost:${PORT}`);
});
```

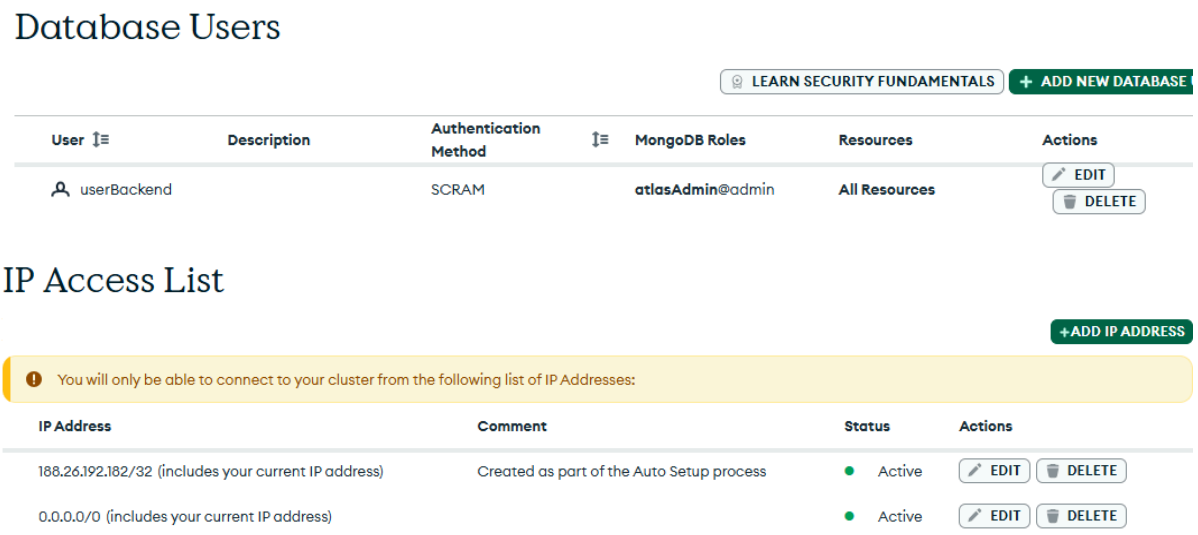
12. Configurar conexión a MongoDB

Abro sesión con email en https://www.mongodb.com/cloud/atlas/register.
Creo proyecto con nombre backend-products-users
Creo un cluster, de nombre ClusterBackend
A continuación creo mi usuario y mi passw userBackend **FcYrfgpaLGzbw8Qt**

En Built-in Role Select one built-in role for this user selecciono Atlas admin
Añado mi IP Address 188.26.192.182/32 y configuro como Allow access from anywhere (0.0.0.0/0)
Y obtengo mi URL de conexión, mi MONGO_URI que debo añadir al index.js debajo de PORT

MONGO_URI=mongodb+srv://userBackend:**FcYrfgpaLGzbw8Qt**@clusterbackend.dzwgovf.mongodb.net/?appName=ClusterBackend





Instalo mongoose, que es una librería de Node.js que sirve para conectar mi aplicación con MongoDB npm install mongoose
Debo añadir mongoose y la conexión a MongoDB a mi index.js quedando así

```
require("dotenv").config();
const express = require("express");
const mongoose = require("mongoose");

const app = express();
app.use(express.json());

// Rutas

mongoose
 .connect(process.env.MONGO_URI)
 .then(() => console.log("Conectado a MongoDB ✔️"))
 .catch((err) => console.log(err));

const PORT = process.env.PORT || 4000;
app.listen(PORT, () => {
 console.log(`Servidor escuchando en http://localhost:${PORT}`);
});
```

II. Los Modelos de datos

Tengo que a crear las estructuras de datos de mi backend: User y Product
Dentro de la carpeta /models creo

```
type NUL > models\User.model.js
type NUL > models\Product.model.js
```

Crear el User.model.js
Para crear el modelo User debo tener en cuenta que el usuario tendrá: nombre, email, password, rol = “user” por defecto, image (que vendrá de Cloudinary), un array relacionado (por ejemplo favoritos) y un control de duplicados en ese array.
Mi User.model.js contiene:

```
const mongoose = require("mongoose");
const userSchema = new mongoose.Schema(
 {
  username: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },

  // Rol: por defecto siempre 'user'
  role: { type: String, enum: ["user", "admin"], default: "user" },
```

```
    // Imagen subida a Cloudinary
    image: { type: String },

    // Array relacionado con Products (IDs)
    favorites: [
     {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Product",
     },
    ],
   },
   {
    timestamps: true,
    versionKey: false,
   }
);

const User = mongoose.model("User", userSchema);
module.exports = User;
```

Crear Product.model.js
He elegido crear un modelo con nombre, descripción y precio.
Mi Product.model.js contiene:

```
const mongoose = require("mongoose");
const productSchema = new mongoose.Schema(
 {
  name: { type: String, required: true },
  description: { type: String },
  price: { type: Number, required: true },
 },
 {
  timestamps: true,
  versionKey: false,
 }
);

const Product = mongoose.model("Product", productSchema);
module.exports = Product;
```

Añadir al usuario un array de IDs relacionados
Para ello al usuario le añado productos favoritos con products: [{ type: mongoose.Schema.Types.ObjectId, ref: "Product" }]
Es decir un array en el que cada elemento del array es un ID de Mongo que pertenece a la colección Product
Y esto lo añado a mi User.model.js quedando como sigue:

```
const mongoose = require("mongoose");
const userSchema = new mongoose.Schema(
 {
  username: { type: String, required: true, unique: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  image: { type: String, default: "" },
  role: { type: String, enum: ["user", "admin"], default: "user" },
  products: [
    { type: mongoose.Schema.Types.ObjectId, ref: "Product" }
   ]
 },
 {
```

```
    timestamps: true,
    versionKey: false,
  }
);

module.exports = mongoose.model("User", userSchema);
```

III. La autenticación

Crear carpetas controllers, routes, middlewares, utils

Primero tenemos que crear 4 carpetas:
/controllers=> donde reside la lógica del backend, es decir, crear usuarios, borrar usuarios, conectarse a la BBDD, etc
/routes => para registrar las rutas, es decir, las URLs a las que un cliente (un frontend, un navegador…) puede acceder para pedirle algo al backend.
/middlewares => para funciones intermedias entre el cliente y el controlador como la autenticación con token, subir imagen a cloudinary, etc
/utils => para funciones auxiliares

Instalar bcrypt

Necesito instalar el módulo bcrypt necesario para el registro y login. Para ello ejecuto npm install bcrypt

Instalar jsonwebtoken

Para hacer el login con **JWT** primero lo instalo mediante npm install jsonwebtoken
Después creo una variable de entorno. Para ello en el .env añadimos el JWT_SECRET con el valor que queramos, por ejemplo, poierfgnmw0eiouvn40289vbun3e08br4eb (es la clave para firmar tokens JWT)

Un JWT (JSON Web Token) es una cadena firmada por el servidor que contiene datos del usuario (ID, email, rol…).
Para hacer login con JWT se necesitan 3 piezas obligatorias:
generateToken (crear tokens con jwt.sign)
loginUser (buscar usuario → validar contraseña → emitir token)
Ruta /login registrada en Express

Crear generateToken

JWT requiere jwt.sign. Primero creamos /utils/jwt.js:

```
const jwt = require("jsonwebtoken");

const generateToken = (user) => {
 return jwt.sign(
  { id: user._id, email: user.email, role: user.role },
  process.env.JWT_SECRET,
  { expiresIn: "2h" }
 );
};

module.exports = { generateToken };
```

Para probarlo necesitamos crear un controlador. Creamos el controllers/user.controller.js

```
// controllers/user.controller.js
const { generateToken } = require("../utils/jwt");

// Prueba de creación de token
const testToken = (req, res) => {
 // Simulamos un usuario como si viniera de la base de datos
 const fakeUser = {
  _id: "123456789",
  email: "test@example.com",
  role: "user",
 };
```

```
  const token = generateToken(fakeUser);

  res.json({
   message: "Token generado correctamente",
   token,
  });
 };

 module.exports = {
  testToken,
 };
```

Añadimos la ruta y la registramos en index.js
Y lo probamos con http://localhost:4000/api/users/test-token
Y nos responde {"message":"Token generado correctamente","token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjEyMzQ1Njc4OSIsImVtYWlsIjoidGVzdEBleGFtcGxlLmNvbSIsInJvbGUiOiJ1c2VyIiwiaWF0IjoxNzYzMzkyODg4LCJleHAiOjE3NjM0MDAwODh9.irjAQzbykMDtCG_nsjchUcbz3GbL1-zSffBlBASlWJo"}

Crear registro de usuarios (con hashing + creación + imagen opcional)
El registro incluye obligatoriamente:

- Hash de contraseña con bcrypt
- El rol siempre “user”
- Subir la imagen a Cloudinary
- Guardar el usuario en MongoDB
- Devolver el usuario sin password
- Evitar duplicados por email

Instalo Cloudinary con npm install cloudinary multer multer-storage-cloudinary

Tengo que tener una cuenta en https://cloudinary.com/
****Cloudinary recomienda ahora NO usar multer-storage-cloudinary. Y subir los archivos a Cloudinary directamente desde tu controlador. Multer solo maneja el archivo en local (en memoria) y Cloudinary recibe el archivo y lo almacena en la nube
Instalo multer normal: npm install multer

Mi cloud name es **dzeiqqxhz**
Mi cloud ID es **8dfecac657168f985bab94732cf52d**
Mi API KEY es 976592752763889
Mi API SECRET es ZMRvJezctTPQI2_1yFXnhXzMP5E

Añadir la configuración en utils\cloudinary.js

```
const cloudinary = require("cloudinary").v2;
cloudinary.config({
 cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
 api_key: process.env.CLOUDINARY_API_KEY,
 api_secret: process.env.CLOUDINARY_API_SECRET,
});
module.exports = cloudinary;
```

Añadir las variables de entorno a .env

```
CLOUDINARY_CLOUD_NAME=dzeiqqxhz
CLOUDINARY_API_KEY=976592752763889
CLOUDINARY_API_SECRET=ZMRvJezctTPQI2_1yFXnhXzMP5E
```

Creo el middlewares\upload.middleware.js para la subida de la imagen

```
const multer = require("multer");
const storage = multer.memoryStorage();
```

```
const upload = multer({ storage });
module.exports = upload;
```

Actualizo el usercontroller.js

```
// controllers/user.controller.js
const User = require("../models/User.model");
const bcrypt = require("bcrypt");
const { generateToken } = require("../utils/jwt");
const cloudinary = require("../utils/cloudinary");

// Función auxiliar para subir buffer a Cloudinary
const uploadBufferToCloudinary = (buffer) => {
 return new Promise((resolve, reject) => {
  const stream = cloudinary.uploader.upload_stream(
   { folder: "users" },
   (error, result) => {
    if (error) reject(error);
    else resolve(result);
   }
  );
  stream.end(buffer);
 });
};

const registerUser = async (req, res) => {
 try {
  const { username, email, password } = req.body;

  if (!username || !email || !password)
   return res.status(400).json({ message: "Todos los campos son obligatorios" });

  const existingUser = await User.findOne({ email });
  if (existingUser)
   return res.status(400).json({ message: "Ese email ya existe" });

  const hashedPassword = await bcrypt.hash(password, 10);

  let imageUrl = "";

  if (req.file) {
   const uploaded = await uploadBufferToCloudinary(req.file.buffer);
   imageUrl = uploaded.secure_url;
  }

  const newUser = await User.create({
   username,
   email,
   password: hashedPassword,
   image: imageUrl,
   role: "user",
  });

  const userResponse = {
   _id: newUser._id,
   username: newUser.username,
   email: newUser.email,
   image: newUser.image,
   role: newUser.role,
```

```
    };

    return res.status(201).json({
     message: "Usuario registrado correctamente",
     user: userResponse,
    });

   } catch (error) {
    return res.status(500).json({ message: "Error al registrar usuario", error });
   }
  };

  module.exports = { registerUser };
```

Añado la ruta a routes/user.routes.js

```
  const upload = require("../middlewares/upload.middleware");
  router.post("/register", upload.single("image"), registerUser);
```

Para hacer las pruebas voy a utilizar Insomnia. Para ello me lo descargo desde https://insomnia.rest/
Con Insomnia puedo probar el backend sin necesidad de crear un frontal.
Creo un POST con username, email, password e image y el resultado es:


```
{
    "message": "Usuario registrado correctamente",
    "user": {
        "_id": "691ded42131ece17ba4a150d",
        "username": "Laura",
        "email": "tara65@gmail.com",
        "image":
"https://res.cloudinary.com/dzeiqqxhz/image/upload/v1763568961/users/vqiwhbois9tkgduha2up.jpg",
        "role": "user"
    }
}
```


Crear login y subida de imagen a Cloudinary cuando se crea un usuario

Para hacer login de usuario con JWT necesito buscar el usuario por email, comparar la contraseña con bcrypt, generar el token JWT y después devolver el token y los datos del usuario
Probar qué ocurre si el usuario no existe o si la contraseña es incorrecta

Primero pruebo POST http://localhost:4000/api/users/login con body=JSON

```
{
 "email": "tara65@gmail.com",
 "password": "test112233"
}
```

Y me devuelve


```
{
    "message": "Login correcto",
    "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MWRlZDQyMTMxZWNlMTdiYTRhMTUwZCI
sImVtYWlsIjoidGFyYTY1QGdtYWlsLmNvbSIsInJvbGUiOiJ1c2VyIiwiaWF0IjoxNzYzNTcwMjMyLCJleH
AiOjE3NjM1Nzc0MzJ9.5LV1DhplJtwzbQzVYka7FMhvNTswDbjUCZ7jz0g7QYs",
    "user": {
        "_id": "691ded42131ece17ba4a150d",
        "username": "Laura",
        "email": "tara65@gmail.com",
        "image":
"https://res.cloudinary.com/dzeiqqxhz/image/upload/v1763568961/users/vqiwhbois9tkgduha2up.jpg",
        "role": "user"
    }
}
```

Crear middleware auth, validación del token JWT
Tengo que crear un middleware de autenticación. Un "*middleware es una función que se ejecuta entre la petición(request) y la respuesta (response) de tu servidor, por lo tanto, puede ejecutarse antes de llegar al controlador*"
Esta función decide si me deja continuar en función del token.
Creo el fichero auth.middleware.js dentro de la carpeta de middlewares.
Este middleware lee el token, lo valida y llama a la función next si es válido.
Voy a probarlo con Insomnia mediante GET http://localhost:4000/api/users/profile

Para ello primero hago login en Insomnia mediante POST http://localhost:4000/api/users/login con body=JSON

```
{
  "email": "tara65@gmail.com",
  "password": "test112233"
}
```

Y me devuelve el token JWT

```
{
        "message": "Login correcto",
        "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MWRlZDQyMTMxZWNlMTdiYTRhMTUwZCIsImVtYWlsIjoidGFyYTY
1QGdtYWlsLmNvbSIsInJvbGUiOiJ1c2VyIiwiaWF0IjoxNzYzNTczMjcxLCJleHAiOjE3NjM1ODA0NzF9.n5uqcxmyIL6SkdHy3yMt
kYYM4A-xAZXUcYCmISFDm5Y",
        "user": {
                "_id": "691ded42131ece17ba4a150d",
                "username": "Laura",
                "email": "tara65@gmail.com",
                "image": "https://res.cloudinary.com/dzeiqqxhz/image/upload/v1763568961/users/vqiwhbois9tkgduha2up.jpg",
                "role": "user"
        }
}
```

ya que sin token no puedo probar rutas protegidas como /profile.

Y me devuelve

```
{
        "message": "Ruta protegida OK",
        "userData": {
                "id": "691ded42131ece17ba4a150d",
                "email": "tara65@gmail.com",
                "role": "user",
                "iat": 1763572554,
                "exp": 1763579754
        }
}
```

Por tanto:
.- El login genera un JWT válido Post → /api/users/login → token OK
.- Insomnia utiliza el token Auth → Bearer Token funcionando
.- El middleware protege la ruta ya que sólo responde si el token es válido
.- Y devuelve los datos del usuario (id, email, role, iat, exp)

Crear el middleware de roles de admin y de user
Tengo que crear el archivo role.middleware.js en la carpeta de /middlewares
En función del rol se tienen unos permisos u otros. Se trata de comprobar si tienes permiso para realizar una acción.
.- Un usuario con rol user puede borrar su propia cuenta.
.- Un usuario con rol user no puede borrar otros usuarios. Un admin sí.
.- Un usuario con rol user no puede cambiar roles. Un admin sí.

```
// middlewares/role.middleware.js

const checkRole = (...allowedRoles) => {
 return (req, res, next) => {
  try {
   const userRole = req.user?.role;

   if (!userRole) {
    return res.status(401).json({
     message: "No se ha podido obtener el rol del usuario",
    });
   }

   if (!allowedRoles.includes(userRole)) {
    return res.status(403).json({
     message: "No tienes permisos para esta acción",
    });
   }

   next();
  } catch (error) {
   return res.status(500).json({
    message: "Error en el middleware de roles",
    error,
   });
  }
 };
};

module.exports = { checkRole };
```

Pruebas realizadas

1 — Un usuario normal (role: "user") intenta acceder a una ruta protegida con admin.

Antes de nada y como es posible que me haya caducado el token hago login con

POST http://localhost:4000/api/users/login

con body JSON igual a

```
{
 "email": "tara65@gmail.com",
 "password": "test112233"
}
```

Para que me devuelva un nuevo token


```
{
        "message": "Login correcto",
        "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MWRlZDQyMTMxZWNlMTdiYTRhMTUwZCIsImVtYWlsIjoi
dGFyYTY1QGdtYWlsLmNvbSIsInJvbGUiOiJ1c2VyIiwiaWF0IjoxNzYzNjQ4MjU1LCJleHAiOjE3NjM2NTU0NTV9.6Ri
pZAO3QrLx7wjpNnLp56E0SNkf_6r9xLZOe-sSabg",
        "user": {
                "_id": "691ded42131ece17ba4a150d",
                "username": "Laura",
                "email": "tara65@gmail.com",
                "image":
"https://res.cloudinary.com/dzeiqqxhz/image/upload/v1763568961/users/vqiwhbois9tkgduha2up.jpg",
                "role": "user"
        }
```

}

Al hacer login el usuario introduce email y password y el Backend comprueba si existe y si la contraseña es correcta
Si todo es correcto el Backend devuelve el token jwt. Este token representa que este usuario está autenticado.

Y ahora hago esta primera prueba con este token

GET http://localhost:4000/api/users/test-admin
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MWRlZDQyMTMxZWNlMTdiYTRhMTUwZCIsImVtYWlsIjoidGFyYTY1QGdtYWlsLmNvbSIsInJvbGUiOiJ1c2VyIiwiaWF0IjoxNzYzNjQ4MjU1LCJleHAiOjE3NjM2NTU0NTV9.6RipZAO3QrLx7wjpNnLp56E0SNkf_6r9xLZOe-sSabg
Resultado esperado: 403 - No tienes permisos para esta acción

```
{
    "message": "No tienes permisos para esta acción"
}
```

2 — Un usuario admin accede a esa ruta y devuelve ok
El primer admin, según el enunciado, se crea directamente en MongoDB Atlas:
Para ello accedo a MongoDB Atlas y en mi colección test.users de mi ClusterBackend hago "Insert Document" para crear un nuevo documento. La passw la he hasheado antes.

Introduzco los siguientes datos:

{"_id":{"$oid":"69203024b5c020afdb6f44a2"},"username":"admin","email":"admin@example.com","password":"$2b$10$RZzfjoZjOZuRf5bwP4KJFeo6u5Rz7ss7f8y0E8fkFuWNxhDB6./uq","role":"admin","image":""}

Ahora hago login con el admin POST http://localhost:4000/api/users/login

Body (JSON):

```
{
  "email": "admin@example.com",
  "password": "admin123"
}
```

Y me devuelve

```
{
    "message": "Login correcto",
    "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjAzMDI0YjVjMDIwYWZkYjZmNDRhMiIsImVtYWlsIjoiYWRtaW5AZXhhbXBsZS5jb20iLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE3NjM3Mjg4NjksImV4cCI6MTc2MzczNjA2OX0.eBbTp_ZlyxuW8bUs8SgtCJQ6y72-rl5Bqv9Z0WoAPzw",
    "user": {
        "_id": "69203024b5c020afdb6f44a2",
        "username": "admin",
        "email": "admin@example.com",
        "image": "",
        "role": "admin"
    }
}
```

Ahora prueba la ruta protegida con el admin

Probar ruta protegida SOLO para admin mediante GET http://localhost:4000/api/users/test-admin
con Headers:
Key: Authorization
Value:
Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjAzMDI0YjVjMDIwYWZkYjZmNDRhMiIsImVtYWlsIjoiYWRtaW5A

ZXhhbXBsZS5jb20iLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE3NjM3MTcyMDQsImV4cCI6MTc2MzcyNDQwNH0.ZhDQyvLasLvihvVTOs15zvfUbLjrsqNb1CXr1LIw3Hw

Y me devuelve

```
{
  "message": "Acceso permitido SOLO a admin"
}
```

3 — Un usuario normal (role: "user") intenta borrar otra cuenta, pero como no es admin da error 403
Para esta prueba debo tener 2 usuarios normales (role: "user") y uno intenta borrar a otro.
Mi usuario A (testeva123)


```
{
    "message": "Usuario registrado correctamente",
    "user": {
        "_id": "6920ae1185376e2038b3fd5c",
        "username": "Eva",
        "email": "evaperez@gmail.com",
        "image": "https://res.cloudinary.com/dzeiqqxhz/image/upload/v1763749391/users/pgrcpmnigovqap3cr9nc.jpg",
        "role": "user"
    }
}
```


Mi usuario B (elisa123)


```
{
    "message": "Usuario registrado correctamente",
    "user": {
        "_id": "6920ae4d85376e2038b3fd5f",
        "username": "Elisa",
        "email": "elisaperez@gmail.com",
        "image": "https://res.cloudinary.com/dzeiqqxhz/image/upload/v1763749451/users/x6t6u5y6dbnajvlisqy3.jpg",
        "role": "user"
    }
}
```


El usuario A intenta borrar al usuario B
Para ello hago login con el usuario A

POST http://localhost:4000/api/users/login

Body JSON:

```
{
  "email": "evaperez@gmail.com",
  "password": "testeva123"
}
```

Y me devuelve


```
{
    "message": "Login correcto",
    "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjBhZTExODUzNzZlMjAzOGIzZmQ1YyIsImVtYWlsIjoiZXZhcGVyZXpAZ21haWwuY29tIiwicm9sZSI6InVzZXIiLCJpYXQiOjE3NjM3NDk1MzcsImV4cCI6MTc2Mzc1NjczN30.xCME2iwpH21gKjBZNSbXRA6mtP5bOOBPGx-1SgPulLw",
    "user": {
        "_id": "6920ae1185376e2038b3fd5c",
        "username": "Eva",
        "email": "evaperez@gmail.com",
```

```
                "image":
"https://res.cloudinary.com/dzeiqqxhz/image/upload/v1763749391/users/pgrcpmnigovqap3cr9nc.jpg",
                "role": "user"
            }
        }
```


Hago delete del usuario B con el token del usuario A mediante

DELETE http://localhost:4000/api/users/**6920ae4d85376e2038b3fd5f**

con Auth Bearer

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjBhZTExODUzNzZlMjAzOGIzZmQ1YyIsImVtYWlsIjoiZXZhcGVyZXpAZ21haWwuY29tIiwicm9sZSI6InVzZXIiLCJpYXQiOjE3NjM3NDk1MzcsImV4cCI6MTc2Mzc1NjczN30.xCME2iwpH21gKjBZNSbXRA6mtP5bOOBPGx-1SgPuILw

Y devuelve

```
{
    "message": "No tienes permisos para esta acción"
}
```

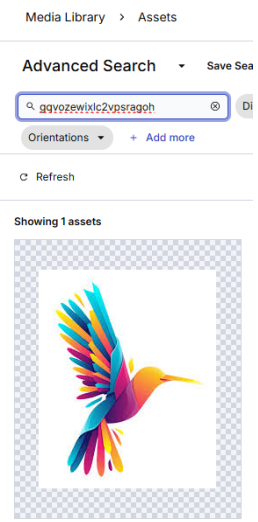
4.- El Usuario C se borra a sí mismo, ok y además su imagen se elimina en Cloudinary.

Creo el usuario con POST http://localhost:4000/api/users/register con Form Data introduciendo username=Elvira, email=elvi23@gmail.com, password=elvi23123 e image.

Y me devuelve


```
{
    "message": "Usuario registrado correctamente",
    "user": {
            "_id": "69219b1873b350734a89cb2b",
            "username": "Elvira",
            "email": "elvi23@gmail.com",
            "image":
"https://res.cloudinary.com/dzeiqqxhz/image/upload/v1763810071/users/gqvozewixlc2vpsragoh.jpg",
            "role": "user"
    }
}
```


Y la imagen ha quedado registrada en Cloudinary



Hago delete del usuario A mediante su id y su token

DELETE http://localhost:4000/api/users/**69219b1873b350734a89cb2b**

con Auth Bearer Token y el token de C

Como es posible que me haya expirado el token del usuario C lo vuelvo a solicitar

```
POST http://localhost:4000/api/users/login
Body (JSON):
{
 "email": "elvi23@gmail.com",
 "password": "elvi23123"
}
```

Y me devuelve


```
{
    "message": "Login correcto",
    "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjE5YjE4NzNiMzUwNzM0YTg5Y2IyYiIsImVtYWlsIjoiZWx2aTIzQGdtYWlsLmNvbSIsInJvbGUiOiJ1c2VyIiwiaWF0IjoxNzYzODEwMzAzLCJleHAiOjE3NjM4MTc1MDN9.4_g9IYOoA7-rq_TL4H6YsMefGrWEGhjLnT3Ayn7IKUk",
    "user": {
            "_id": "69219b1873b350734a89cb2b",
            "username": "Elvira",
```

```
            "email": "elvi23@gmail.com",
            "image":
"https://res.cloudinary.com/dzeiqqxhz/image/upload/v1763810071/users/gqvozewixlc2vpsragoh.jpg",
            "role": "user"
        }
    }
```


Ahora ya puedo hacer el delete con

DELETE http://localhost:4000/api/users/**69219b1873b350734a89cb2b**

con Auth Bearer Token y el token de C que es

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjE5YjE4NzNiMzUwNzM0YTg5Y2IyYiIsImVtYWlsIjoiZWx2aTIzQGdtYWlsLmNvbSIsInJvbGUiOiJ1c2VyIiwiaWF0IjoxNzYzODEwMzAzLCJleHAiOjE3NjM4MTc1MDN9.4_g9IYOoA7-rq_TL4H6YsMefGrWEGhjLnT3Ayn7IKUk

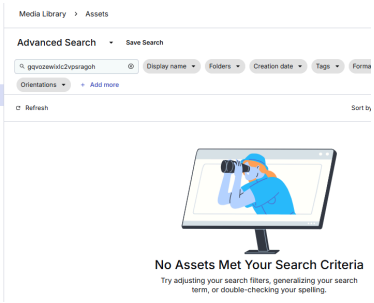
Y me devuelve


```
{
    "message": "Usuario eliminado correctamente"
}
```


Ahora debo comprobar en Cloudinary que la imagen con public_id del user A ya no existe. Y que en MongoAtlas que el documento fue eliminado.

Entro en Cloudinary, https://cloudinary.com/console, en Media Library, folders users y busco el public_id **gqvozewixlc2vpsragoh**

Y devuelve



5.- El Usuario admin borra al usuario B

Mi usuario B (elisa123) - "_id": "6920ae4d85376e2038b3fd5f",

DELETE http://localhost:4000/api/users/6920ae4d85376e2038b3fd5f

con Auth Bearer Token y el token de admin que es

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjAzMDI0YjVjMDIwYWZkYjZmNDRhMiIsImVtYWlsIjoiYWRtaW5AZXhhbXBsZS5jb20iLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE3NjM4MjU2MjMsImV4cCI6MTc2MzgzMjgyM30.1z9hHMrJZkyafeUiqPG78LBudZkTaZCISW1tM65ookM

Y devuelve


```
{
    "message": "Usuario eliminado correctamente"
}
```


6.- Sólo admin puede cambiar roles. Por ejemplo poner como admin a Adela.

Hago login con admin y recupero su token

POST http://localhost:4000/api/users/login con body=JSON


```
{
  "email": "admin@example.com",
  "password": "admin123"
}
```


Y me devuelve


```
{
    "message": "Login correcto",
```

```
        "token":
    "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjAzMDI0YjVjMDIwYWZkYjZmNDRhMiIsImVtYWlsIjoiYWRtaW5AZXhhbXBsZS5jb20iLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE3NjM4OTQxOTgsImV4cCI6MTc2MzkwMTM5OH0.4jRHr5T3gQBxYyfVdX1tuOSACjPkhA0Hl2cT7gm488w",
        "user": {
            "_id": "69203024b5c020afdb6f44a2",
            "username": "admin",
            "email": "admin@example.com",
            "role": "admin"
        }
    }
```


PUT http://localhost:4000/api/users/691c84e1131ece17ba4a150a/role

con Auth Bearer Token y el token de admin que es

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjAzMDI0YjVjMDIwYWZkYjZmNDRhMiIsImVtYWlsIjoiYWRtaW5AZXhhbXBsZS5jb20iLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE3NjM4OTQxOTgsImV4cCI6MTc2MzkwMTM5OH0.4jRHr5T3gQBxYyfVdX1tuOSACjPkhA0Hl2cT7gm488w

y body JSON igual a


```
{
  "role": "admin"
}
```


Y devuelve


```
{
    "message": "Rol actualizado correctamente",
    "user": {
        "image": "",
        "_id": "691c84e1131ece17ba4a150a",
        "username": "AdelaActual",
        "email": "newbell78@gmail.com",
        "password": "\"password\":
  \"$2b$10$kXXq/ax6ydyibVuZ2UmtUuWbMbnG2lfsxCbOjZg3Kpod2wI2/Q6QC\"\n",
        "role": "admin",
        "favorites": [],
        "createdAt": "2025-11-18T14:38:25.994Z",
        "updatedAt": "2025-11-23T10:38:35.448Z"
    }
}
```


Ahora vuelvo a dejarla con rol user para la siguiente prueba

PUT http://localhost:4000/api/users/691c84e1131ece17ba4a150a/role

con Auth Bearer Token y el token de admin que es

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjAzMDI0YjVjMDIwYWZkYjZmNDRhMiIsImVtYWlsIjoiYWRtaW5AZXhhbXBsZS5jb20iLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE3NjM4OTQxOTgsImV4cCI6MTc2MzkwMTM5OH0.4jRHr5T3gQBxYyfVdX1tuOSACjPkhA0Hl2cT7gm488w

y body JSON igual a


```
{
  "role": "user"
}
```


Y devuelve


```
{
    "message": "Rol actualizado correctamente",
    "user": {
        "image": "",
        "_id": "691c84e1131ece17ba4a150a",
        "username": "AdelaActual",
        "email": "newbell78@gmail.com",
```

```
            "password": "\"password\":
        \"$2b$10$kXXq/ax6ydyibVuZ2UmtUuWbMbnG2lfsxCbOjZg3Kpod2wI2/Q6QC\"\n",
            "role": "user",
            "favorites": [],
            "createdAt": "2025-11-18T14:38:25.994Z",
            "updatedAt": "2025-11-23T10:40:59.047Z"
        }
    }
```


7.- Probar que user normal no puede cambiar roles
Hago login con Ana para obtener su token y luego intento que cambie el rol de Adela a admin
POST http://localhost:4000/api/users/login con body=JSON


```
{
 "email": "anasanz@gmail.com",
 "password": "ana112233"
}
```


Y me devuelve su token


```
{
    "message": "Login correcto",
    "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjFkZjBmMDVhODc3OGJkYzdkZmM0OSIsIm
VtYWlsIjoiYW5hc2FuekBnbWFpbC5jb20iLCJyb2xlIjoidXNlciIsImlhdCI6MTc2Mzg5NDY2NywiZXhwIjo
xNzYzOTAxODY3fQ.k3_bx9TzRQWOq07AE1Fppoj5yc1wHkLQE4OJzMnK2m8",
    "user": {
        "_id": "6921df0f05a8778bdc7dfc49",
        "username": "AnaActual",
        "email": "anasanz@gmail.com",
        "image":
"https://res.cloudinary.com/dzeiqqxhz/image/upload/v1763827470/users/vy9yyt580adtl8gi56xn.jpg",
        "role": "user"
    }
}
```


Y ahora intento que Ana cambie el rol de Adela a admin
PUT http://localhost:4000/api/users/691c84e1131ece17ba4a150a/role
con Auth Bearer Token y el token de Ana que es
**eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjFkZjBmMDVhODc3OGJkYzdkZmM0OSIsImVt
YWlsIjoiYW5hc2FuekBnbWFpbC5jb20iLCJyb2xlIjoidXNlciIsImlhdCI6MTc2Mzg5NDY2NywiZXhwIjoxNz
YzOTAxODY3fQ.k3_bx9TzRQWOq07AE1Fppoj5yc1wHkLQE4OJzMnK2m8**
y body JSON igual a


```
{
 "role": "admin"
}
```


Y devuelve


```
{
    "message": "No tienes permisos para esta acción"
}
```


IV. CRUD de colecciones

CRUD usuarios

El CRUD se refiere a Create, Read, Update y Delete.
El Create ya lo hago con el register, y ya he usado el Delete. Me falta leer usuarios, con GET y actualizar usuarios con Update.
Tengo que crear las rutas y los controladores. Y luego hacer las pruebas con Insomnia y comprobar resultados en el Mongo..
1-Seleccionar todos los elementos de una colección. Sólo puede admin según el enunciado (un usuario normal no puede cambiar… ni acceder a todo".
Debo añadir la ruta router.get("/", auth, checkRole("admin"), getAllUsers); a user.routes.js
Debo añadir a **user.controller.js** lo siguiente:

```
const getAllUsers = async (req, res, next) => {
 try {
  const users = await User.find().select("-password");
  return res.status(200).json(users);
 } catch (error) {
  return next(error);
 }
};
```

Para probarlo en Insomnia

Hago login con admin. Copio el token que me devuelve y hago un GET http://localhost:4000/api/users
con Headers: Auth: Bearer token


```
{
	"message": "Login correcto",
	"token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjAzMDI0YjVjMDIwYWZkYjZmNDRhMiIsImVtYWlsIjoiYWRtaW5AZXhhbXBsZS5jb20iLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE3NjM4MjU2MjMsImV4cCI6MTc2MzgzMjgyM30.1z9hHMrJZkyafeUiqPG78LBudZkTaZCISW1tM65ookM",
	"user": {
		"_id": "69203024b5c020afdb6f44a2",
		"username": "admin",
		"email": "admin@example.com",
		"role": "admin"
	}
}
```


GET http://localhost:4000/api/users
con Headers: Auth: Bearer token

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjAzMDI0YjVjMDIwYWZkYjZmNDRhMiIsImVtYWlsIjoiYWRtaW5AZXhhbXBsZS5jb20iLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE3NjM4MjU2MjMsImV4cCI6MTc2MzgzMjgyM30.1z9hHMrJZkyafeUiqPG78LBudZkTaZCISW1tM65ookM

Y me devuelve todos los usuarios


```
[
	{
		"image": "",
		"_id": "691c84e1131ece17ba4a150a",
		"username": "Adela",
		"email": "newbell78@gmail.com",
		"role": "user",
		"favorites": [],
		"createdAt": "2025-11-18T14:38:25.994Z",
		"updatedAt": "2025-11-18T14:38:25.994Z"
	},
	{
		"image":
"https://res.cloudinary.com/dzeiqqxhz/image/upload/v1763568961/users/vqiwhbois9tkgduha2up.jpg",
		"_id": "691ded42131ece17ba4a150d",
		"username": "Laura",
		"email": "tara65@gmail.com",
		"role": "user",
		"favorites": [],
		"createdAt": "2025-11-19T16:16:02.307Z",
		"updatedAt": "2025-11-19T16:16:02.307Z"
	},
	{
		"image": "",
		"favorites": [],
```

```
                "_id": "69203024b5c020afdb6f44a2",
                "username": "admin",
                "email": "admin@example.com",
                "role": "admin"
        },
        {
                "image":
"https://res.cloudinary.com/dzeiqqxhz/image/upload/v1763749451/users/x6t6u5y6dbnajvlisqy3.jpg",
                "_id": "6920ae4d85376e2038b3fd5f",
                "username": "Elisa",
                "email": "elisaperez@gmail.com",
                "role": "user",
                "favorites": [],
                "createdAt": "2025-11-21T18:24:13.331Z",
                "updatedAt": "2025-11-21T18:24:13.331Z"
        }
]
```


2-Un user sólo puede ver su información pero un admin puede ver la información de cualquier usuario

Como admin acceder a la información del user (por ejemplo del user Adela)

GET http://localhost:4000/api/users/691c84e1131ece17ba4a150a

con Headers: Auth: Bearer token

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjAzMDI0YjVjMDIwYWZkYjZmNDRhMiIsImVtYWlsIjoiYWRtaW5AZXhhbXBsZS5jb20iLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE3NjM4MjU2MjMsImV4cCI6MTc2MzgzMjgyM30.1z9hHMrJZkyafeUiqPG78LBudZkTaZCISW1tM65ookM

Y devuelve


```
{
        "image": "",
        "_id": "691c84e1131ece17ba4a150a",
        "username": "Adela",
        "email": "newbell78@gmail.com",
        "role": "user",
        "favorites": [],
        "createdAt": "2025-11-18T14:38:25.994Z",
        "updatedAt": "2025-11-18T14:38:25.994Z"
}
```



```
{
 "message": "Usuario encontrado",
 "user": {
  "_id": "...",
  "username": "...",
  "email": "...",
  "role": "user",
  "image": { ... }
 }
}
```


Y como otro user normal intentar acceder a la información del user Adela

Creo el usuario con POST http://localhost:4000/api/users/register con Form Data introduciendo username=Ana, email=anasanz@gmail.com, password=ana112233 e image.


```
{
        "message": "Usuario registrado correctamente",
        "user": {
                "_id": "6921df0f05a8778bdc7dfc49",
```

```
                "username": "Ana",
                "email": "anasanz@gmail.com",
                "image":
"https://res.cloudinary.com/dzeiqqxhz/image/upload/v1763827470/users/vy9yyt580adtl8gi56xn.jpg",
                "role": "user"
        }
}
```


Hago login como Ana

POST http://localhost:4000/api/users/login con body=JSON


```
{
 "email": "anasanz@gmail.com",
 "password": "ana112233"
}
```


Devuelve


```
{
        "message": "Login correcto",
        "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjFkZjBmMDVhODc3OGJkYzdkZmM0OSIsImVtYWlsIjoiYW5hc2FuekBnbWFpbC5jb20iLCJyb2xlIjoidXNlciIsImlhdCI6MTc2MzgyNzYyNSwiZXhwIjoxNzYzODM0ODI1fQ.9ZP10M8BCeTMkkTNKryXOyVQjsfHkM0Z4zJ5u-K09tM",
        "user": {
                "_id": "6921df0f05a8778bdc7dfc49",
                "username": "Ana",
                "email": "anasanz@gmail.com",
                "image":
"https://res.cloudinary.com/dzeiqqxhz/image/upload/v1763827470/users/vy9yyt580adtl8gi56xn.jpg",
                "role": "user"
        }
}
```


Y ahora solicito la información de Adela con el token de Ana que es un user normal

GET http://localhost:4000/api/users/691c84e1131ece17ba4a150a

con Headers: Auth: Bearer token

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjFkZjBmMDVhODc3OGJkYzdkZmM0OSIsImVtYWlsIjoiYW5hc2FuekBnbWFpbC5jb20iLCJyb2xlIjoidXNlciIsImlhdCI6MTc2MzgyNzYyNSwiZXhwIjoxNzYzODM0ODI1fQ.9ZP10M8BCeTMkkTNKryXOyVQjsfHkM0Z4zJ5u-K09tM

Y devuelve


```
{
"message": "No tienes permisos para ver este usuario"
}
```


3- Un user normal accede a su información, por ejemplo, el user Ana

GET http://localhost:4000/api/users/6921df0f05a8778bdc7dfc49

con Headers: Auth: Bearer token

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjFkZjBmMDVhODc3OGJkYzdkZmM0OSIsImVtYWlsIjoiYW5hc2FuekBnbWFpbC5jb20iLCJyb2xlIjoidXNlciIsImlhdCI6MTc2MzgyNzYyNSwiZXhwIjoxNzYzODM0ODI1fQ.9ZP10M8BCeTMkkTNKryXOyVQjsfHkM0Z4zJ5u-K09tM

Y devuelve


```
{
"image": {
        "url":
"https://res.cloudinary.com/dzeiqqxhz/image/upload/v1763827470/users/vy9yyt580adtl8gi56xn.jpg",
        "public_id": "users/vy9yyt580adtl8gi56xn"
},
"_id": "6921df0f05a8778bdc7dfc49",
"username": "Ana",
```

"email": "anasanz@gmail.com",
"role": "user",
"favorites": [],
"createdAt": "2025-11-22T16:04:31.647Z",
"updatedAt": "2025-11-22T16:04:31.647Z"
}

Para probar la parte del UPDATE debo realizar 2 pruebas:
.- Un user normal solo puede actualizar su propia información:
Por ejemplo el user normal Ana actualiza su username
Primero hago login para obtener su token mediante POST http://localhost:4000/api/users/login
con body JSON igual a

{
"email": "anasanz@gmail.com",
"password": "ana112233"
}
Y devuelve
{
"message": "Login correcto",
"token":
"**eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjFkZjBmMDVhODc3OGJkYzdkZmM0OSIsImVtYWlsIjoiYW5hc2FuekBnbWFpbC5jb20iLCJyb2xlIjoidXNlciIsImlhdCI6MTc2MzgyODc0MCwiZXhwIjoxNzYzODM1OTQwfQ.UEqPPuJ2NWI4AoIfmDLTPLxO04bYxIEPvSNQl0U--xk**",
"user": {
"_id": "6921df0f05a8778bdc7dfc49",
"username": "Ana",
"email": "anasanz@gmail.com",
"image":
"https://res.cloudinary.com/dzeiqqxhz/image/upload/v1763827470/users/vy9yyt580adtl8gi56xn.jpg",
"role": "user"
}
}
Y ahora actualizo su username mediante PUT http://localhost:4000/api/users/6921df0f05a8778bdc7dfc49
con Auth: Bearer Token
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjFkZjBmMDVhODc3OGJkYzdkZmM0OSIsImVtYWlsIjoiYW5hc2FuekBnbWFpbC5jb20iLCJyb2xlIjoidXNlciIsImlhdCI6MTc2MzgyODc0MCwiZXhwIjoxNzYzODM1OTQwfQ.UEqPPuJ2NWI4AoIfmDLTPLxO04bYxIEPvSNQl0U--xk
y con body Form Data para
username=AnaActual

Y devuelve el dato actualizado

{
"message": "Usuario actualizado correctamente",
"user": {
"_id": "6921df0f05a8778bdc7dfc49",
"username": "AnaActual",
"email": "anasanz@gmail.com",
"image": {
"url":
"https://res.cloudinary.com/dzeiqqxhz/image/upload/v1763827470/users/vy9yyt580adtl8gi56xn.jpg",
"public_id": "users/vy9yyt580adtl8gi56xn"
},
"role": "user"
}
}

4- Un user normal no puedo modificar la información de otro user normal
También pruebo que Ana no puede modificar el username de Adela mediante

PUT http://localhost:4000/api/users/691c84e1131ece17ba4a150a
con Auth: Bearer Token
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjFkZjBmMDVhODc3OGJkYzdkZmM0OSIsImVtYWlsIjoiYW5hc2FuekBnbWFpbC5jb20iLCJyb2xlIjoidXNlciIsImlhdCI6MTc2MzgyODc0MCwiZXhwIjoxNzYzODM1OTQwfQ.UEqPPuJ2NWI4AoIfmDLTPLxO04bYxIEPvSNQl0U--xk
y con body Form Data para
username=AdelaActual

Y lógicamente devuelve

```
{
    "message": "No tienes permisos para editar este usuario"
}
```

5- Un user admin puede actualizar la información de cualquier usuario:
Por ejemplo admin cambia el username de Adela

PUT http://localhost:4000/api/users/691c84e1131ece17ba4a150a
con Auth: Bearer Token
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjAzMDI0YjVjMDIwYWZkYjZmNDRhMiIsImVtYWlsIjoiYWRtaW5AZXhhbXBsZS5jb20iLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE3NjM4MjU2MjMsImV4cCI6MTc2MzgzMjgyM30.1z9hHMrJZkyafeUiqPG78LBudZkTaZCISW1tM65ookM
y con body Form Data para
username=AdelaActual

Y funciona correctamente

```
{
    "message": "Usuario actualizado correctamente",
    "user": {
        "_id": "691c84e1131ece17ba4a150a",
        "username": "AdelaActual",
        "email": "newbell78@gmail.com",
        "image": "",
        "role": "user"
    }
}
```

6- Añadir/eliminar favoritos evitando duplicados
Los favoritos **siempre** se guardan dentro del usuario, en su array favorites dentro del modelo.
Debo añadir la lógica sin duplicados al fichero user.controller.js. y crear las rutas correspondientes.
Mediante la función toggleFavorite que sirve para añadir o eliminar un producto de la lista de favoritos del usuario. Es como un interruptor: si no existe lo añade y si existe lo elimina. Y la función getFavorites devuelve la lista completa de productos favoritos de un usuario.

a) Para insertar un favorito
POST http://localhost:4000/api/users/ID_DEL_USUARIO/favorites/ID_DEL_PRODUCTO
con Auth: **Bearer token del usuario**
y body vacío
por ejemplo id usuario = 691c84e1131ece17ba4a150a e id producto = en próximos pasos

Y devolvería
{ "message": "Producto añadido a favoritos", "favorites": ["..."] } o

{ "message": "Producto eliminado de favoritos", "favorites": [] }

b) Para ver los favoritos (por ejemplo de Ana)
GET **http://localhost:4000/api/users/**6921df0f05a8778bdc7dfc49**/favorites**
con Auth Bearer token
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjFkZjBmMDVhODc3OGJkYzdkZmM0OSIsImVt

YWlsIjoiYW5hc2FuekBnbWFpbC5jb20iLCJyb2xlIjoidXNlciIsImlhdCI6MTc2MzgyNzYyNSwiZXhwIjoxNzY
zODM0ODI1fQ.9ZP10M8BCeTMkkTNKryXOyVQjsfHkM0Z4zJ5u-K09tM

CRUD productos

Ahora tengo que crear todo lo relativo a los productos. Creo Product con nombre, descripción y precio. Los products no tienen que tener imagen en cloudinary. Solo crea/edita/elimina Products un admin. La seed debe borrar la colección y luego insertar los nuevos productos. Y la seed no incluye imágenes.

Creo los siguientes archivos:

- Product.model.js
- product.controller.js (CRUD completo)
- product.routes.js
- la seed products.seed.js

Pruebas CRUD productos realizadas

1- Obtener todos los productos mediante GET http://localhost:4000/products

devuelve un array vacío [] porque aún no he creado productos pero significa que la ruta funciona

2- Creo productos como admin, porque sólo admin puede crearlos para ello hago login con el admin

POST http://localhost:4000/api/users/login

Body (JSON):

```
{
  "email": "admin@example.com",
  "password": "admin123"
}
```

Y me devuelve

```
{
        "message": "Login correcto",
        "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MjAzMDI0YjVjMDIwYWZkYjZmNDRhMiIsI
mVtYWlsIjoiYWRtaW5AZXhhbXBsZS5jb20iLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE3NjM4ODYxODE
sImV4cCI6MTc2Mzg5MzM4MX0.kqS7Qd6wtfR9YJA_FVxm_SRS1h8Lz7KXs3w5uv43INo",
        "user":{
                "_id": "69203024b5c020afdb6f44a2",
                "username": "admin",
                "email": "admin@example.com",
                "role": "admin"
        }
}
```

Y ahora para crear el producto POST http://localhost:4000/products

con Headers Auth Bearer token (el token de admin)

y con body JSON igual a:

```
{
  "name": "camiseta blanca básica",
  "description": "Una camiseta muy suave y cómoda",
  "price": 19.99,
  "image": "https://via.placeholder.com/200"
}
```

y me devuelve

```
{
  "name": "camiseta blanca básica",
  "description": "Una camiseta muy suave y cómoda",
  "price": 19.99,
  "_id": "6922c563a4edbe58603fef67",
  "createdAt": "2025-11-23T08:27:15.393Z",
  "updatedAt": "2025-11-23T08:27:15.393Z"
}
```

3- Actualizar productos como admin, sólo admin puede hacerlo.

Mediante PUT http://localhost:4000/products/ID_PROD

siendo ID_PROD = 6922c563a4edbe58603fef67

con Headers Auth Bearer token (el token de admin)
y con body JSON igual a:

```
{
 "name": "Camiseta blanca premium",
  "description": "Una camiseta muy suave y cómoda",
  "price": 23.99
}
```

Y devuelve

```
{
  "_id": "6922c563a4edbe58603fef67",
  "name": "Camiseta blanca premium",
  "description": "Una camiseta muy suave y cómoda",
  "price": 23.99,
  "createdAt": "2025-11-23T08:27:15.393Z",
  "updatedAt": "2025-11-23T08:44:48.070Z"
}
```

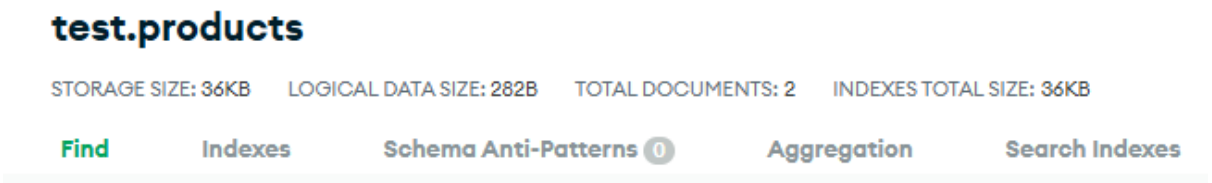
4- Borrar un producto como admin, sólo admin puede hacerlo.
Mediante DELETE http://localhost:4000/products/ID_PROD
siendo ID_PROD = 6922c563a4edbe58603fef67
con Headers Auth Bearer token (el token de admin)

Y devuelve "Product deleted"

5- Probar seed mediante node seeds/products.seed.js y devuelve

```
[dotenv@17.2.3] injecting env (6) from .env -- tip: 📡 add observability to secrets:
https://dotenvx.com/ops
Cargando .env desde: ...\backend-products-users\.env
MONGO_URI:
mongodb+srv://userBackend:FcYrfgpaLGzbw8Qt@clusterbackend.dzwgovf.mongodb.net/?appName=Cl
usterBackend
Conectado para seed
Colección de productos borrada
Productos insertados
Desconectado
```

6- Y en MongoDB, colección de productos



{"_id":{"$oid":"6922d12dc38a6327e7773955"},"name":"Producto 1","description":"Descripción del producto 1","price":{"$numberDouble":"10.99"},"createdAt":{"$date":{"$numberLong":"1763889453698"}},"updatedAt":{"$date":{"$numberLong":"1763889453698"}}}

Añadir un producto a favoritos
Lo hace un user normal con su token, por ejemplo Laura

POST http://localhost:4000/api/users/login
con body JSON igual a

```
{
 "email": "tara65@gmail.com",
 "password": "test112233"
}
```

Para que me devuelva un nuevo token

```
{
```

```
        "message": "Login correcto",
        "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MWRlZDQyMTMxZWNlMTdiYTRhMTUwZCIsImVtYWlsIjoidGFyYTY1QGdtYWlsLmNvbSIsInJvbGUiOiJ1c2VyIiwiaWF0IjoxNzYzODkwNDEwLCJleHAiOjE3NjM4OTc2MTB9.B5fSA_EPwn21Zig_f3yb3ikFlxQUGrwjuL52Rb2nPMc",
        "user": {
                "_id": "691ded42131ece17ba4a150d",
                "username": "Laura",
                "email": "tara65@gmail.com",
                "role": "user"
        }
}
```


Y ahora añado el favorito con su token

POST http://localhost:4000/api/users/691ded42131ece17ba4a150d/favorites/6922d12dc38a6327e7773955
con Auth Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MWRlZDQyMTMxZWNlMTdiYTRhMTUwZCIsImVtYWlsIjoidGFyYTY1QGdtYWlsLmNvbSIsInJvbGUiOiJ1c2VyIiwiaWF0IjoxNzYzODkwNDEwLCJleHAiOjE3NjM4OTc2MTB9.B5fSA_EPwn21Zig_f3yb3ikFlxQUGrwjuL52Rb2nPMc

Y devuelve


```
{
        "message": "Producto añadido a favoritos",
        "favorites": [
                "6922d12dc38a6327e7773955"
        ]
}
```


7- Eliminar un favorito.
Usaría el mismo POST ya que tengo la función toggle
POST http://localhost:4000/api/users/691ded42131ece17ba4a150d/favorites/6922d12dc38a6327e7773955
con Auth Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MWRlZDQyMTMxZWNlMTdiYTRhMTUwZCIsImVtYWlsIjoidGFyYTY1QGdtYWlsLmNvbSIsInJvbGUiOiJ1c2VyIiwiaWF0IjoxNzYzODkwNDEwLCJleHAiOjE3NjM4OTc2MTB9.B5fSA_EPwn21Zig_f3yb3ikFlxQUGrwjuL52Rb2nPMc
Y devuelve


```
{
        "message": "Producto eliminado de favoritos",
        "favorites": []
}
```