

**Proyecto nuevo en 03\_Server**

**Tienda Online**

**Terminal**

**express --view=pug --git TiendaOnline**

**Terminal**

**cd TiendaOnline**

**Terminal**

**npm install**

**Editar el package.json y editar el script con nodemon**

  "scripts": {

    "start": "node ./bin/www",

    "dev": "nodemon ./bin/www"

  },

**Terminal**

**npm run dev**

**tiendaonline@0.0.0 dev**

**nodemon ./bin/www**

**[nodemon] 2.0.20**

**[nodemon] to restart at any time, enter `rs`**

**[nodemon] watching path(s): \*.\***

**[nodemon] watching extensions: js,mjs,json**

**[nodemon] starting `node ./bin/www`**

Listo!!!! Ya arranca el servidor con nodemon

OBJETIVO PRINCIPAL

##OBJETIVO

**Creación de una API que gestione los productos de una tienda Online.**

**### GET /api/products**

**1ª ETAPA 🡪 REQUERIMIENTOS**

* **Recuperar todos los productos de la tabla products.**
  + **La url GET /api/products debe funcionar** 
    - **Respuesta: Responda con status 200**
  + **Debe devolver datos en formato JSON**
    - **Respuesta: Responda con la cabecera Content-Type igual a application /json**
  + **Debe devolver**

**NOTA :Para realizar estas pruebas debemos como siempre utilizar librerías.**

**Terminal**

**npm install jest supertest --save-dev**

**NOTA : Instalar estás librerías pero solo para el tiempo de desarrollo, es decir cuando yo suba este proyecto la librería no viene para instalar.**

**package.json 🡪 nos aparecerá ahora**

**“devDependencies”**

***Librerías instaladas***

***jest :*** Ejecutor de las pruebas

***supertest*** : Permitir peticiones web supertest

* **Creamos una carpeta llamada tests sobre el proyecto.**
* **Dentro de la carpeta test creamos la carpeta api**
* **Dentro de la carpeta api creamos el fichero products.spec.js**
* **Instalamos nueva librería, instala todas las herramientas de la librería jest.**

**Terminal**

**npm install @types/jest --save-dev**

**2ª ETAPA 🡪 DESARROLLO DE LAS PRUEBAS**

* **Dentro de products.spec.js**

**describe** 🡪  **sirve para poder organizar nuestras pruebas.**

**it ( ) 🡪 sirve para comenzar a crear una prueba, la “prueba de la prueba”.**

**expect ( ) la ejecución de lo que queremos probar.**

**.tobe ( ) el resultado que debe dar.**

*describe*('Api de productos', () => {

*describe*('GET api/api/products', () => {

*it*('debería resolver correctamente la suma', () => {

*expect*(3 + 2).*toBe*(5);

        })

    })

})

**3ª ETAPA 🡪 EJECUCIÓN DE LAS PRUEBAS**

* **Dentro de package.json**

**Creamos el scripts de “test”**

  "scripts": {

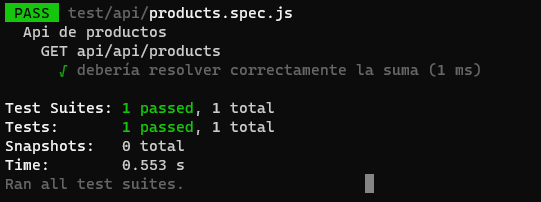
    "start": "node ./bin/www",

    "dev": "nodemon ./bin/www",

    "test": "jest"

**Terminal**

**Lanzamos npm run test**

**Resultado**

RESULTADO OK!!!

**2ª ETAPA 🡪 DESARROLLO DE LAS PRUEBAS**

* **Dentro de products.spec.js**

**Requerimos la librería supertest.**

const request = *require*('supertest');

**Requerimos la app**

const app = *require*('../../app')

**Sobre esta aplicación lanzamos el método GET de está url y me lo envías.**

*request*(app).*get*('/api/products').*send*();

**Como nos devuelve una promesa pues async-await**

*it*('debería responder con status 200', *async* () => {

         const responde = *await* *request*(app).*get*('/api/products').*send*();

        })

**Para esa respuesta espero que me devuelvas un status 200**

*expect*(response.statusCode).*toBe*(200)

**Código**

const request = *require*('supertest');

const app = *require*('../../app')

*describe*('Api de productos', () => {

*describe*('GET api/api/products', () => {

*it*('debería responder con status 200', *async* () => {

            const responde = *await* *request*(app).*get*('/api/products').*send*();

*expect*(response.statusCode).*toBe*(200);

        });

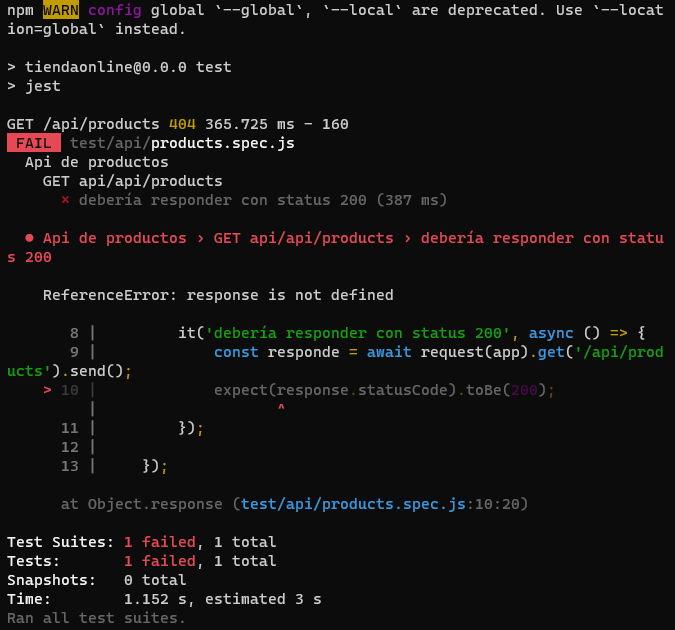
    });

});

**3ª ETAPA 🡪 EJECUCIÓN DE LAS PRUEBAS**

**Terminal**

**npm run test**

**Resultado**

* **Como nos da FAILED lógicamente porque tenemos que desarrollar el código, es decir pasar la prueba status 200.**

**INTEGRACION DE MongoDB**

**Terminal**

**npm install mongoose dotenv**

* **Dentro de proyecto a la altura del proyecto creamos el fichero .env**
* **Dentro de .env** **para hace la ruta hacia MongoDB**

MONGO\_URL="mongodb://127.0.0.1:27017/tienda\_Online"

* **Dentro de bin/www**

**Obtenemos variables de entorno**

*require*('dotenv').*config*();

**CONFIGURACIÓN DE LA BASE DE DATOS**

* **Creamos a la altura del proyecto la carpeta /config**
* **Dentro de /config creo el archivo db.js**

**Intentamos conectar con la BD.**

const mongoose = *require*('mongoose');

mongoose.*connect*(process.env.MONGO\_URL);

* **Dentro de bin/www**

**Conecto a la BD de MongoDB**

*require*('../config/db');

**Terminal**

**Levantamos con npm run dev**

* **Creamos una carpeta /models a la altura del proyecto**
* **Creamos dentro de /models el fichero product.model.js**

**Requerimos la librería mongoose**

const mongoose = *require*('mongoose');

**Requerimos el objeto para crear la estructura, que posteriormente vamos a utilizar.**

const Schema = mongoose.Schema;

**Creamos una new aplicación de la librería Schema, y hay que crear la estructura y/o campos que va a “*tener un producto*”. Hay que poner el campo de tipo específico de mongoDB.**

const productSchema = new *Schema*({

    name: String,

    description: String,

    department: String,

    price: Number,

    available: Boolean,

    stock: Number

});

**Exportamos este Schema llamado al método mongoose.model()**

**Asociamos product del esquema creado para cada producto.**

module.exports = mongoose.*model*('product', productSchema);

* **Ahora vamos a utilizar la estructura creada, vamos a products.js**

**Requerimos la estructura creada “Product”**

const Product = *require*('../../models/product.model');

**Para recuperar los campos de esa estructura.**

**Método find()** 🡪 **recordemos que devuelve una promesa, podemos resolver tanto como con tipo async – await o con .then .catch**

const router = *require*('express').*Router*();

const Product = *require*('../../models/product.model');

router.*get*('/', (req, res) => {

    Product.*find*()

        .*then*(products => res.*json*(products))

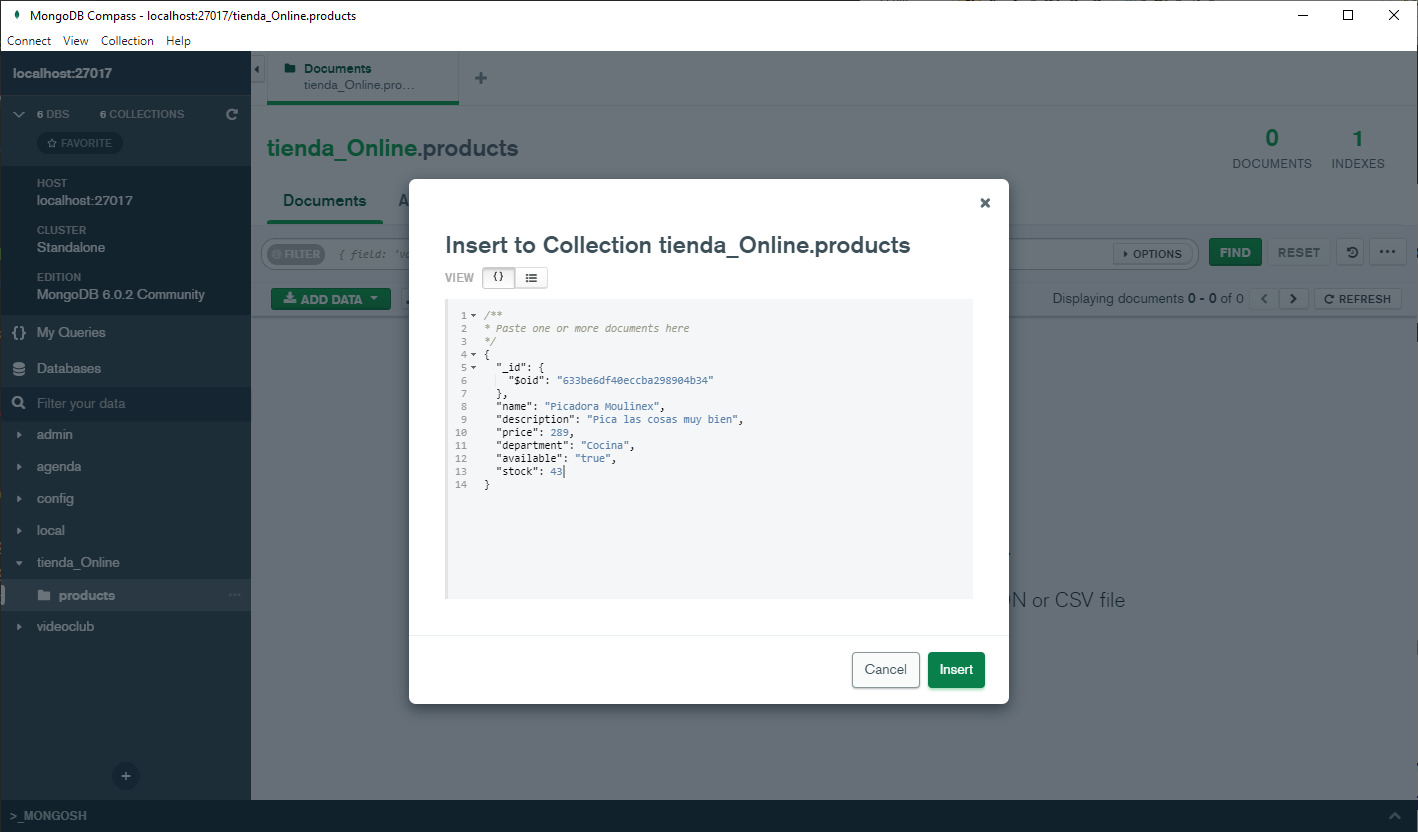
        .*catch*(err => res.*json*(err));

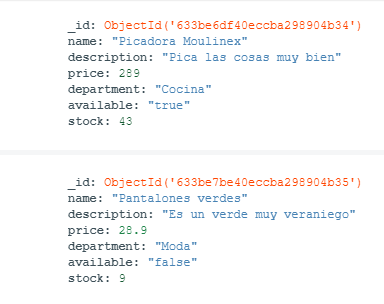
});

module.exports = router;

* **Creamos a la altura del proyecto el archivo de peticiones.rest**

MONGO\_URL="mongodb://127.0.0.1:27017/tienda\_Online"

* **Ahora vamos a MongoDB Compass y comprobamos que en la base de datos se ha creado tienda\_Online**
* **Creamos directamente dentro de MongoDB Compass un producto, para hacer pruebas.**
* **Creamos varios productos e insertamos.**



* **Si ahora vamos a peticiones.rest lanzamos y nos debería sacar los productos que acabamos de crear.**



**EJERCICIO PROFESOR**

**### GET /api /products / PRODUCTID**

**-Recuperar el único producto a partir de su ID.**

**-**Extraemos el id del producto de **req.params**

**-**Para recuperar el producto podemos usar **findById**

* **Vamos a products.js**

**Cuando creamos rutas alternativas, por ejemplo productoId va con dos ( : )**

const router = *require*('express').*Router*();

const Product = *require*('../../models/product.model');

router.*get*('/:productId', *async* (req, res) => {

    const { productId } = req.params;

    const product = *await* Product.*findById*(productId);

    res.*json*(product)

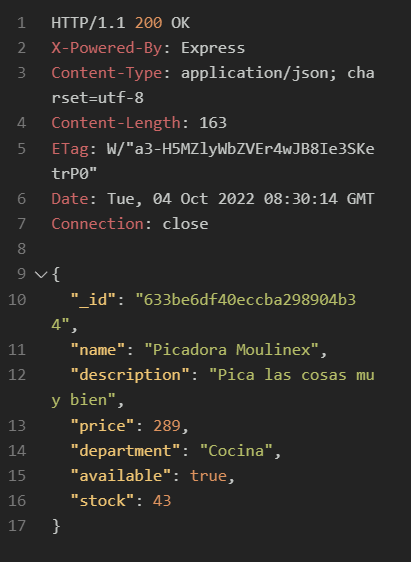
});

module.exports = router;

* **Vamos a peticiones.rest creamos la petición y lanzamos**

*###*

GET http://localhost:3000/api/products/633be6df40eccba298904b34



* **Vamos a especificar más la url a la hora de poner mal la id que nos salte un error, dentro del products.js**

const router = *require*('express').*Router*();

const Product = *require*('../../models/product.model');

router.*get*('/:productId', *async* (req, res) => {

    const { productId } = req.params;

    try {

        const product = *await* Product.*findById*(productId);

*if* (!product) {

*return* res.*json*({ error: 'No existe el producto' })

        }

        res.*json*(product);

    } catch {

        res.*json*({ error: 'No existe el producto' })

    }

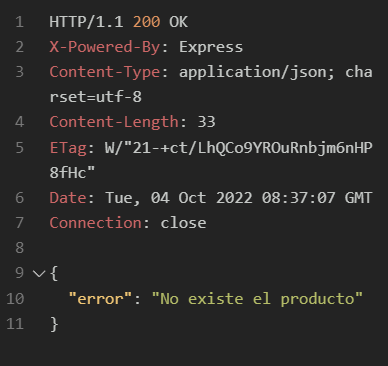
});

module.exports = router;

* **Vamos a probar en peticiones.rest poniendo mal la id**

*###*

GET http://localhost:3000/api/products/633be6df40eccba298904b34dasd



**TEST**

* **Vamos a api/products.spec.js para hacer la prueba de la petición de /api /products / PRODUCTID**

const request = *require*('supertest');

const mongoose = *require*('mongoose');

const app = *require*('../../app')

*describe*('Api de productos', () => {

*beforeAll*(*async* () => {

*await* mongoose.*connect*('mongodb://127.0.0.1:27017/tienda\_Online')

    });

*afterAll*(*async* () => {

*await* mongoose.*disconnect*();

    });

*describe*('GET /api/products/PRODUCTID', () => {

*it*('debería devolver un JSON correcto', *async* () => {

            const response = *await* *request*(app).*get*('/api/products/633be6df40eccba298904b34').*send*();

*expect*(response.statusCode).*toBe*(200);

*expect*(response.headers['content-type']).*toContain*('application/json');

        });

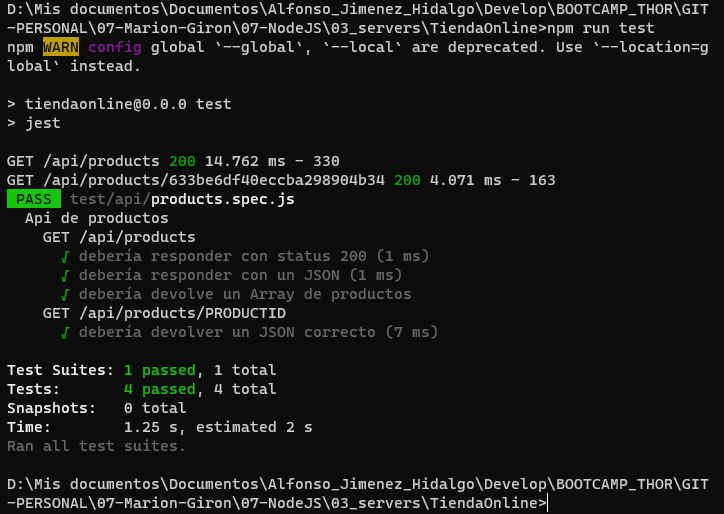
    })

});

* **Lanzamos en la terminal**

**Terminal**

**npm run test**



**CREACION DE USUARIOS**

* **Creamos un fichero nuevo dentro de models, llamado user.models.js**

const mongoose = *require*('mongoose');

const Schema = mongoose.Schema;

const userShema = new *Schema*({

    username: String,

    email: String,

    password: String,

    active: Boolean,

    role: { type: String, default: 'regular' }

});

module.exports = mongoose.*model*('user', userShema);

* **Generamos nueva RUTA.**

**## POST /api / users / register**

**-Recibe todos los datos del usuario a través del Body.**

**-Inserta dichos datos dentro de la BD (create)**

* **Dentro de api.js creamos los enganches hacia /api/users.js**

const router = *require*('express').*Router*();

const apiProductsRouter = *require*('./api/products');

const apiUsersRouter = *require*('./api/users');

router.*use*('/products', apiProductsRouter);

router.*use*('/users', apiUsersRouter);

module.exports = router;

* **Dentro de /api/users.js**

const router = *require*('express').*Router*();

const User = *require*('../../models/user.model');

router.*post*('/register', *async* (req, res) => {

    const user = *await* User.*create*(req.body);

    res.*json*(user);

});

module.exports = router;

* **Creamos otro fichero de peticiones como autentication.rest**

POST http://localhost:3000/api/users/register

Content-Type: application/json

{

    "username": "ajhidalgo94",

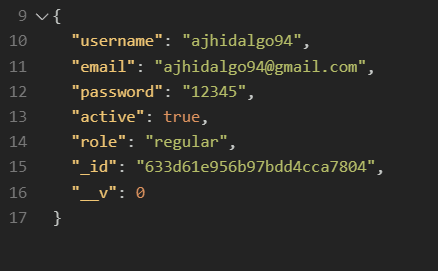
    "email": "ajhidalgo94@gmail.com",

    "password": "12345",

    "active": true

}

**Resultado de la petición**



* **Instalamos librería bcryptjs para poder hace encriptaciones de password.**

**Terminal**

**npm install bcryptjs**

* **Dentro de users.js**

**Importamos librería y encriptados la password**

const bcrypt = *require*("bcryptjs")

const router = *require*('express').*Router*();

const bcrypt = *require*("bcryptjs")

const User = *require*('../../models/user.model');

router.*post*('/register', *async* (req, res) => {

    req.body.password = bcrypt.*hashSync*(req.body.password, 12);

    try {

        const user = *await* User.*create*(req.body);

        res.*json*(user);

    } catch {

        res.*json*({error: err.message })

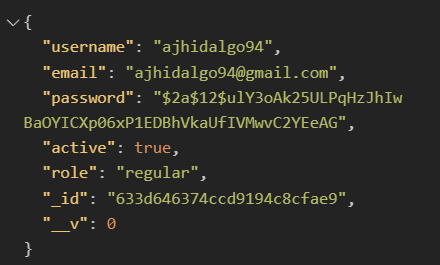
    }

});

module.exports = router;

* **Dentro de autentication.rest**

**Lanzamos de nuevo y ya veremos la password encriptado del usuario registrado.**



LOGIN

* **Dentro de user.js**

**Miramos si el email existe en la BD.**

**Creamos nueva ruta.**

const router = *require*('express').*Router*();

const bcrypt = *require*("bcryptjs")

const User = *require*('../../models/user.model');

router.*post*('/register', *async* (req, res) => {

    req.body.password = bcrypt.*hashSync*(req.body.password, 12);

    try {

        const user = *await* User.*create*(req.body);

        res.*json*(user);

    } catch {

        res.*json*({ error: err.message })

    }

});

router.*post*('/login', *async* (req, res) => {

    const { email, password } = req.body;

    const user = *await* User.*findOne*({ email });

    res.*json*(user);

});

module.exports = router;

* **Dentro de autentication.rest lanzamos el login**

*###*

POST http://localhost:3000/api/users/login

Content-Type: application/json

{

    "email": "ajhidalgo94@gmail.com",

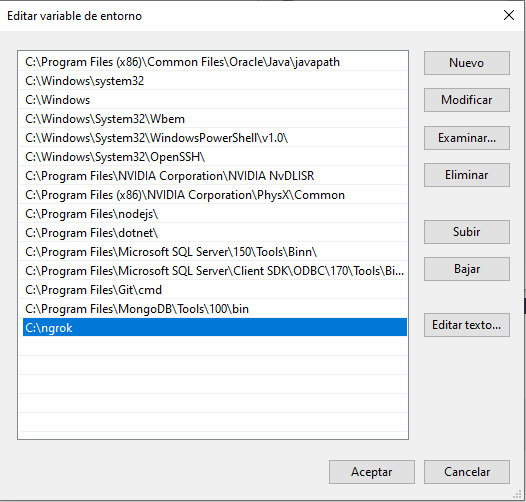
    "password": "12345"

}

**Resultado**



SUBIR LA API ONLINE

* **Descargamos el archivo ngrok**
* **Descomprimimos en disco local C:\ngrok**
* **Añadimos a las variables de entorno**

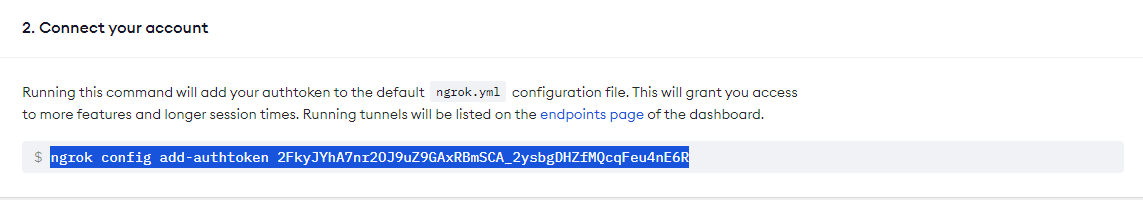
NOTA: SI EJECUTAMOS EL COMANDO NGROK QUE EL WINDOWS SEPA DONDE ESTÁ ESE COMANDO PARA QUE LO RECONOSCA

* **Abrimos el Terminal y introducimos el comando**

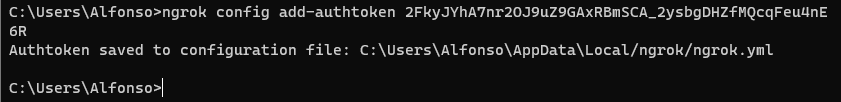
**nkgrok**

**Nos aparecerá una ayuda de todo lo que podemos hacer.**

* **El siguiente paso es obtener un token en la página de nggrok, para ello tenemos que estar registrados.**



* **Copiamos y lo pegamos en el terminal**

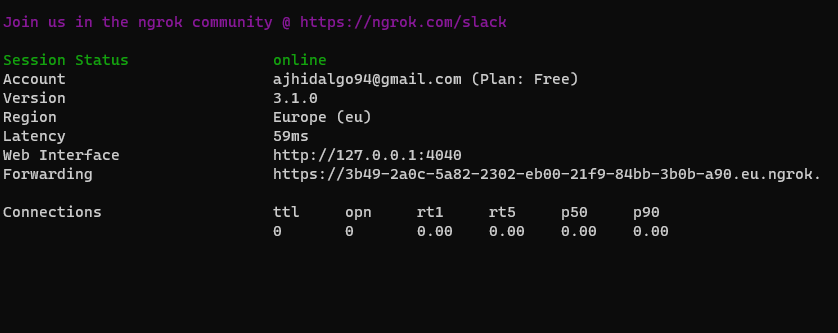


**NOTA: TENEMOS QUE TENER LA API LEVANTADA**

* **Ahora vamos a levantar una url online que apunte al puerto 3000 de nuestro ordenador.**

**Terminal**

**ngrok http 3000**

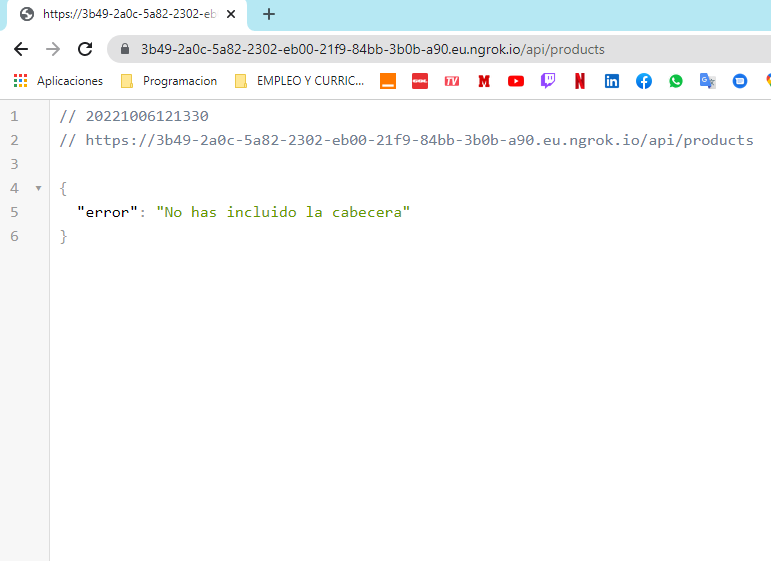


* **La url que nos sale en Forwarding es la url que cualquier usuario puede acceder desde cualquier sitio.**



* **Pues en NAVEGADOR podemos probar**

**https://3b49-2a0c-5a82-2302-eb00-21f9-84bb-3b0b-a90.eu.ngrok.io/api/products**



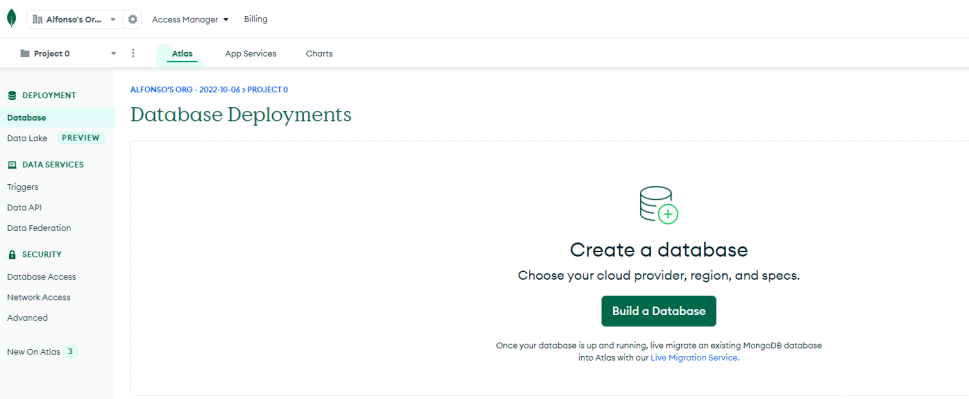
SUBIR LA API A UN SERVIDOR

* **Crear bases de datos online, a través de Mongo DB**

<https://www.mongodb.com/atlas/database>

**Tenemos que registrarnos.**

* **Creamos una nueva base de datos online.**



* **Importamos nueva base de datos en MongoDB Compass**

**Copiamos la ruta y editamos la contraseña**

**mongodb+srv://root1804017@thordb.emarxdc.mongodb.net/test**

**Connect**