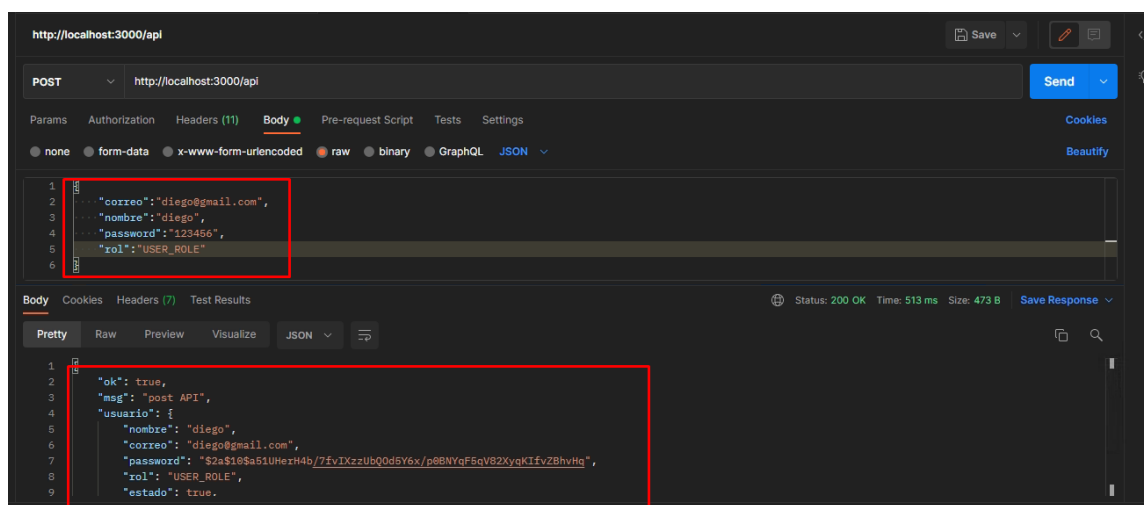


## 11.B4. CRUD DE PRODUCTOS EN NODEJS CON LOGIN, GOOGLE SIGN IN Y SUBIDA DE ARCHIVOS

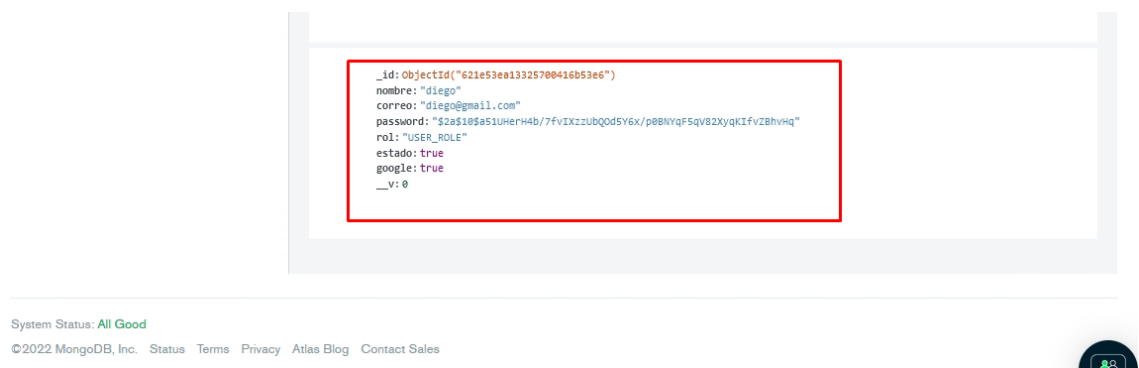
Completa la aplicación web con el CRUD de productos de la tabla que te ha tocado añadiendo:

1. Un login a la tabla de usuarios y una validación por token

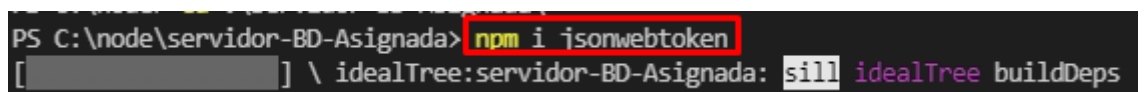
Registramos un usuario con el posmant.



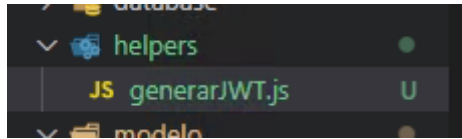
Miramos que se ha subido correctamente a la base de datos.



Para poder usar la validación por token hay que instalar npm i jsonwebtoken



Crear la carpeta Helpers y dentro un archivo javascript con el código para generar el token

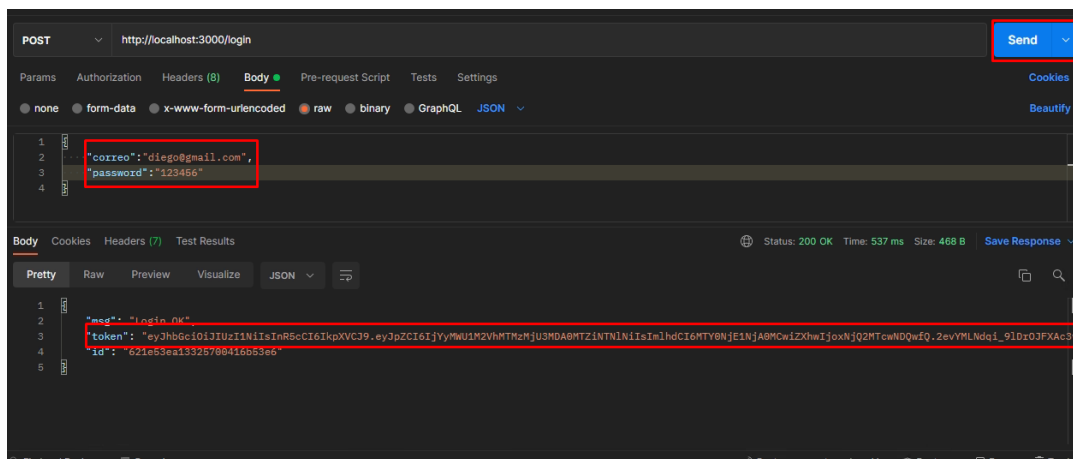


```
1 const jwt = require('jsonwebtoken');
2 const generarJWT = ( id = '' ) => {
3   return new Promise( (resolve, reject) => {
4
5     const payload = { id };
6
7     jwt.sign( payload, process.env.SECRETORPRIVATEKEY, {
8       expiresIn: '4h'
9     }, ( err, token ) => {
10
11       if ( err ) {
12         console.log(err);
13         reject( 'No se pudo generar el token' )
14       } else {
15         resolve( token );
16       }
17     })
18   })
19 }
20
21 module.exports = {
22   generarJWT
23 }
```

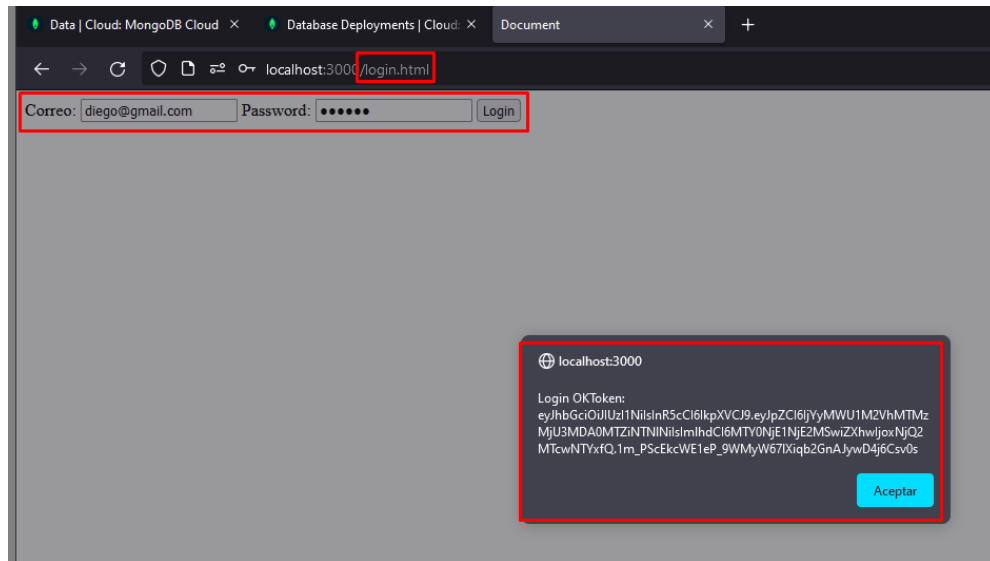
Se importa en el servidor.

```
const { generarJWT } = require("../helpers/generarJWT");
```

Antes de hacer el login probamos que funcione con el postman

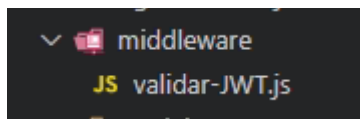


Desde login.html nos conectamos a base de datos y funciona correctamente y genera el token.



## 2. Una protección de todas las rutas validando el token

Para validar el token en las rutas, con la siguiente estructura de carpetas.



### Código validar-JWT.js

```
1 const {response,request} = require('express');
2 const jwt = require('jsonwebtoken');
3 const validarJWT = (req=request,res=response,next)=>
4 {
5   const token = req.header('x-token');
6   if (!token) {
7     return res.status(401).json({
8       msg:'No hay token en la petición'
9     })
10  }
11  try {
12    const payload = jwt.verify(token,process.env.SECRETORPRIVATEKEY);
13    console.log('PAYLOAD:',payload);
14    next();
15  } catch (error) {
16    console.log(error);
17    res.status(401).json({
18      msg:'El token no es válido'
19    })
20  }
21 }
22 module.exports = {validarJWT }
```

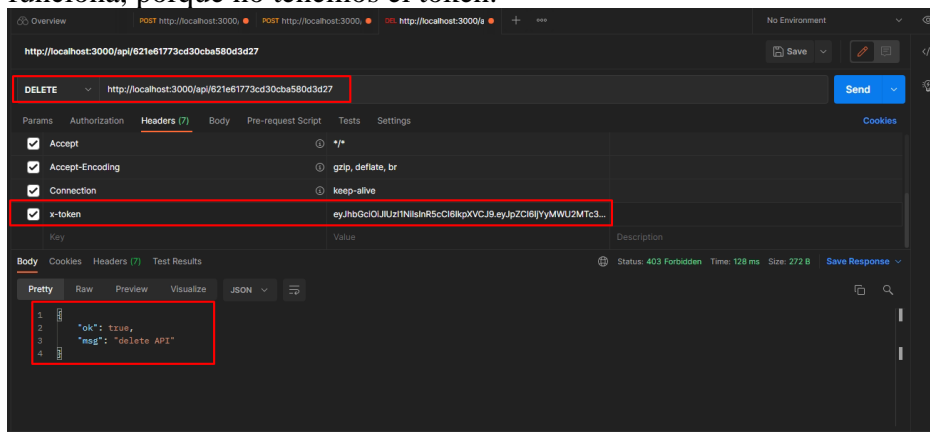
Hay que importarlo

```
const { generarJWT } = require("../helpers/generarJWT");  
const { validarJWT } = require("../middleware/validar-JWT");
```

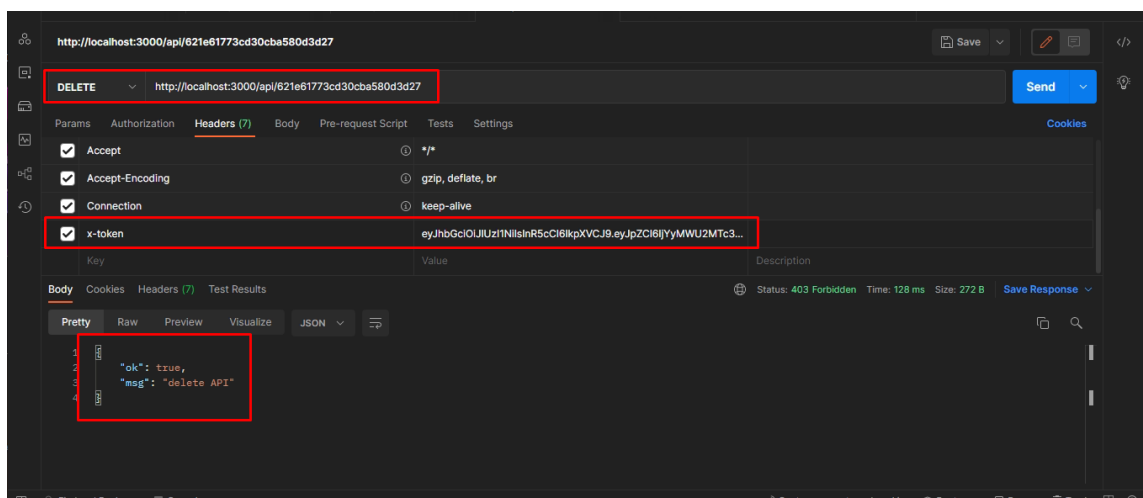
Para cuando se borra un usuario compruebe si el token es correcto.

```
    })  
    this.app.delete('/api/:id', validarJWT, async function (req, res) {  
      const id = req.params.id;  
      await Usuario.findByIdAndDelete(id);  
      res.status(403).json({  
        ok: true,  
        msg: 'delete API'  
      })  
    })  
    this.app.get('/saludo', function (req, res) {
```

Si probamos a borrar un usuario como lo hacíamos antes desde postman no funciona, porque no tenemos el token.



Si añades el token correspondiente a ese usuario puedes eliminarlo si pone otro token que te dice token no es válido.



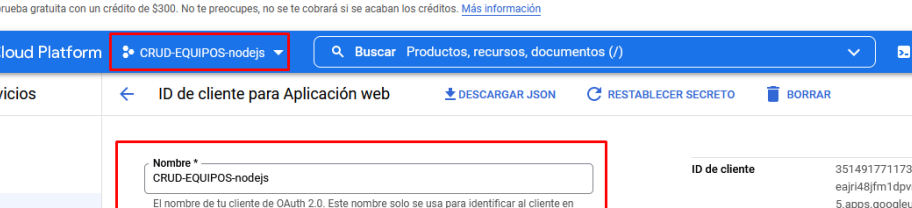
### 3. Un login con Google

Instalamos la librería de Google.

```
PS C:\node\servidor-BD-Asignada> npm install google-auth-library
[redacted] \ idealTree:servidor-BD-Asignada: sill idealTree buildDeps
```

```
PS C:\node\servidor-BD-Asignada> npm install google-auth-library --save
```

Hay que crear un proyecto en Google cloud para conseguir la id del clientes y la secreta.



Comienza tu prueba gratuita con un crédito de \$300. No te preocupes, no se te cobrará si se acaban los créditos. [Más información](#)

Google Cloud Platform **CRUD-EQUIPOS-nodejs**  Productos, recursos, documentos (/)

API y servicios **ID de cliente para Aplicación web** [DESCARGAR JSON](#) [RESTABLECER SECRETO](#) [BORRAR](#)

**Nombre \***  
CRUD-EQUIPOS-nodejs

El nombre de tu cliente de OAuth 2.0. Este nombre solo se usa para identificar al cliente en la consola y no se mostrará a los usuarios finales.

**Los dominios de los URI que agregues a continuación se incorporarán automáticamente a tu pantalla de consentimiento de OAuth como dominios autorizados.**

**Orígenes autorizados de JavaScript**

Para usar con solicitudes de un navegador

**URI 1 \***

**URI 2 \***

**ID de cliente** 351491771173-eajri48jfm1dpvmpogovt21ar2abvh5.apps.googleusercontent.com

**Secreto del cliente** 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

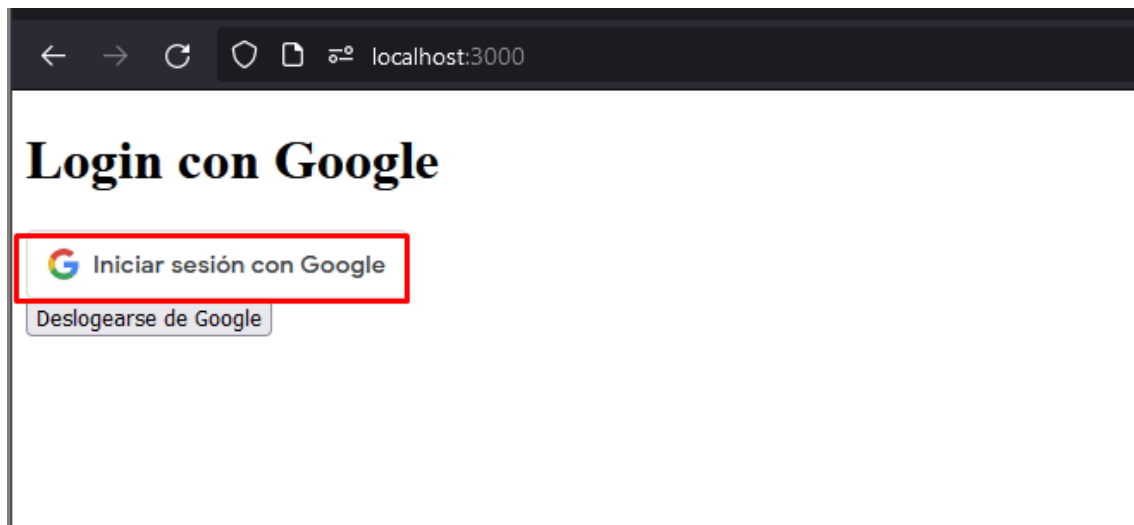
**Fecha de creación** 15 de febrero de 2022, 20:21:06 GMT+1

[+ AGREGAR URI](#)

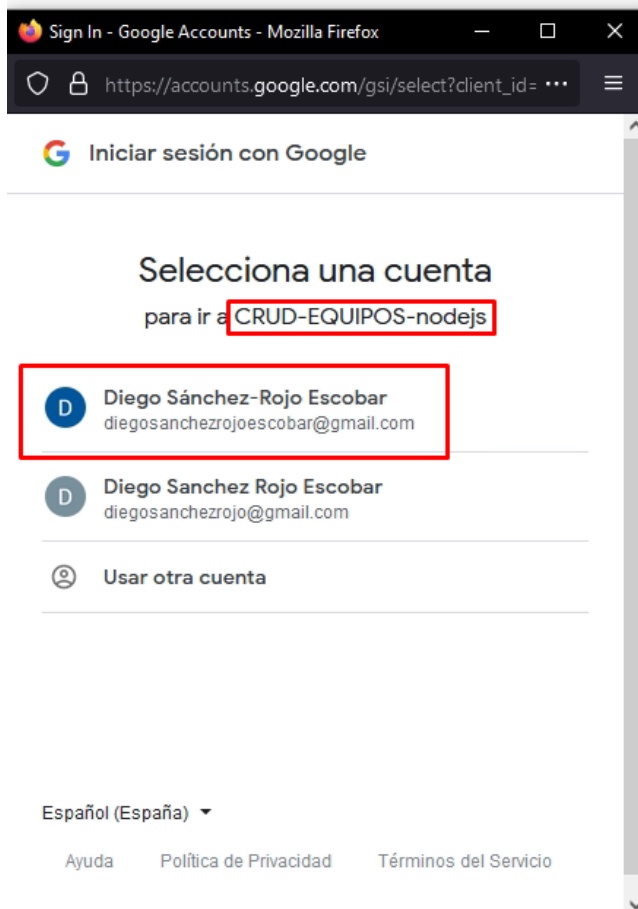
Probamos primero el login con Google desde postman antes que en modo grafico

[illegible]

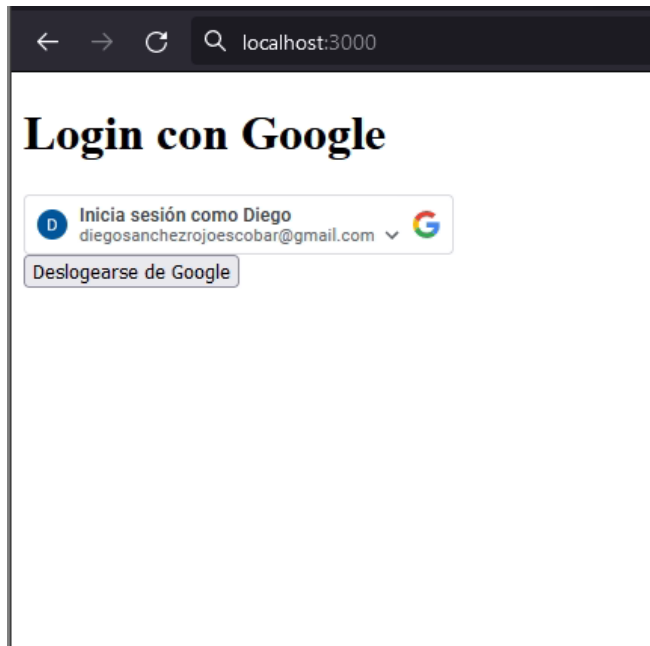
Desde el navegador le damos en iniciar sesión con Google.



Iniciamos sesión con el usuario que queremos.



Inicia sesión con Google



#### 4. Subida de archivos

Instalamos lo express-fileupload

```
found 0 vulnerabilities
PS C:\node\servidor-BD-Asignada> npm i express-fileupload
[ ] \ idealTree:servidor-BD-Asignada: sill idealTree buildDeps
```

Instalamos uuid para cambiar el nombre y que no coincidan.

```
found 0 vulnerabilities
PS C:\node\servidor-BD-Asignada> npm install uuid
[ ] \ idealTree:servidor-BD-Asignada: sill idealTree buildDeps
```

Lo importamos en el servidor.

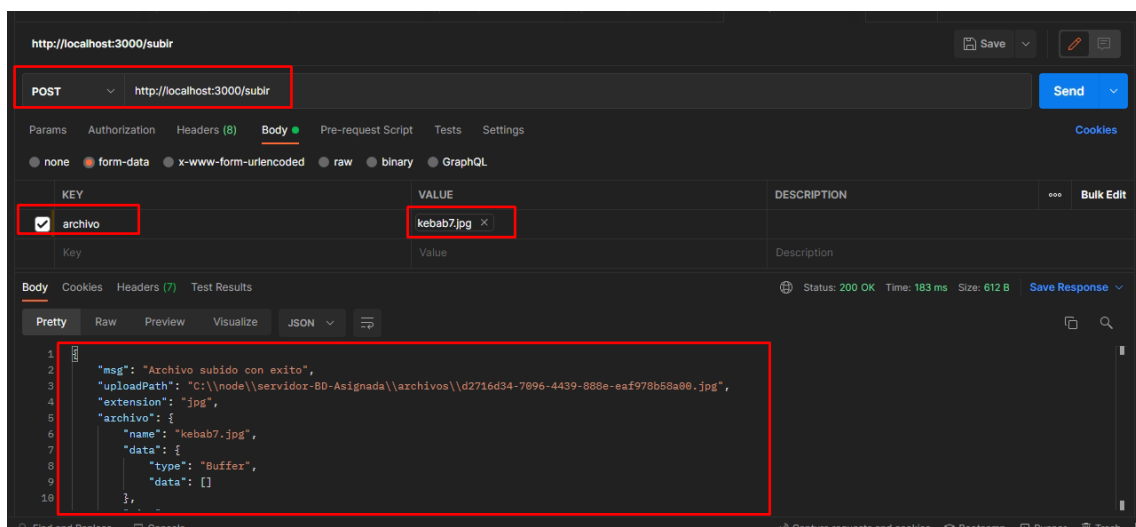
```
1 const { validarJWT } = require("../middleware/validar-
2 const { OAuth2Client } = require('google-auth-library');
3 const fileUpload = require('express-fileupload');

const fileUpload = require('express-fileup
const { v4: uuidv4 } = require('uuid');
class Server {
```

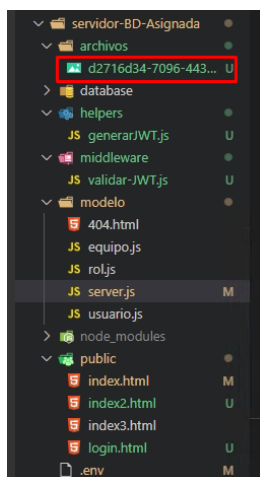
Añadimos lo siguiente en el middleware para poder crear archivos temporales.

```
middlewares() {  
  this.app.use(express.json()); //Middleware para leer json;  
  this.app.use(express.static("public"));  
  //Middleware para servir la carpeta public  
  this.app.use(fileUpload({  
    useTempFiles: true,  
    tempFileDir: '/tmp/'  
  }));  
}
```

Se pueden subir archivos a nuestro servidor



Nos metemos en la carpeta archivos que es donde guardamos las imágenes





Sube la foto y le cambia el nombre para que no coincidan los nombres. (No tenía otra foto que poner).

