

**Tecnologías de desarrollo en el servidor**

*Definición del proyecto Integrador*

**PRESENTA**

Lic. Ingeniería en Nanotecnología: Leonardo Daniel Hernández García

Lic. Ingeniería en Desarrollo de Software: Sergio Isaac Santana Jiménez

Lic. Ingeniería en Desarrollo de Software: José Eduardo Pérez Valenzuela

Profesor:

Francisco Javier Sevilla Medina

Tlaquepaque, Jalisco, 6 de febrero del 2025

### My-ecommerce Iteso

### Descripción:

Como estudiantes del ITESO, hemos experimentado el proceso de compra y venta a través del grupo de WhatsApp *"Ventas ITESO"*, donde los usuarios pueden ofertar y adquirir productos mediante mensajes. Sin embargo, este método presenta diversas limitaciones. Los compradores deben desplazarse constantemente por el chat para encontrar publicaciones anteriores sin garantía de que los productos sigan disponibles. Por su parte, los vendedores deben republicar con frecuencia para evitar que sus ofertas se pierdan entre tantas publicaciones. Una aplicación más estructurada optimizaría este proceso, brindando una experiencia más eficiente y accesible para todos.

### Integrantes:

* Leonardo Daniel Hernández García
* Sergio Isaac Santana Jiménez
* José Eduardo Pérez Valenzuela

### Stack de tecnologías a utilizar:

1. **Backend y Frameworks:**

* Express: Framework para la creación de servidores.
* Handlebars: plantillas para vistas dinámicas.
* Passport y Passport-google-oauth20: Autenticación de usuarios.
* Mongoose: Modelado de datos con MongoDB.
* Multer: Manejo de archivos.
* NodeMailer: Envío de correos electrónicos.
* Socket.io: Comunicación en tiempo real.
* Jsonwebtoken: Gestión de sesiones con token JWT.

1. **Servicios y APIs:**

* AWS SDK (s3): Almacenamiento de archivos en la nube.
* Stripe: Procesamiento de pagos.

1. **Documentación:**

* Swagger-jsdoc.
* Swagger-ui-express

1. **Desarrollo y Testing:**

* Typescript
* Jest: pruebas Automatizadas.
* Nodemon: Recarga automática del servidor en desarrollo.
* Doteenv: Gestión de variables de entorno.

### Roles y acceso a la aplicación:

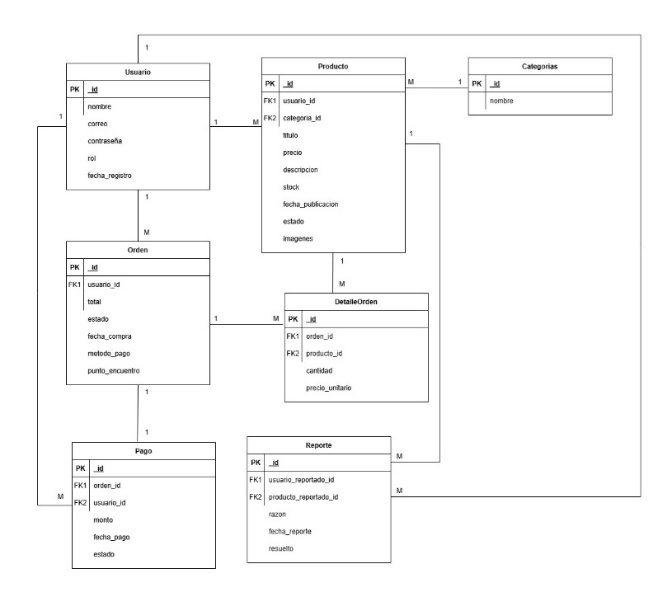
#### Usuarios que pueden registrarse e iniciar sesión

* **Usuarios comunes:** A través de un correo electrónico.
* **Administradores:** Con privilegios adicionales para moderar la plataforma.

#### Funciones por rol

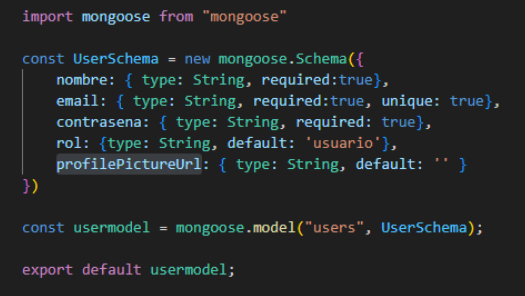
* **Usuario:**  
  Puede iniciar sesión, publicar productos para la venta, comprar artículos publicados por otros usuarios y reportar productos que no cumplan con las normas.
* **Administrador:**  
  Tiene permisos para eliminar productos inapropiados, revisar reportes de los usuarios, gestionar el cumplimiento de las políticas de la plataforma y dar de baja usuarios que infrinjan las reglas del sistema.

### Diagrama entidad-relación de la base de datos:

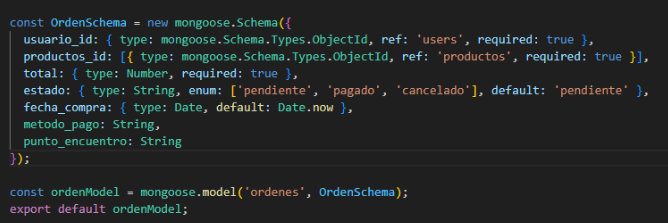


A continuación, se mostrarán unos ejemplos de los esquemas utilizados para la base de datos basada en el diagrama entidad-relación.

Usuarios:

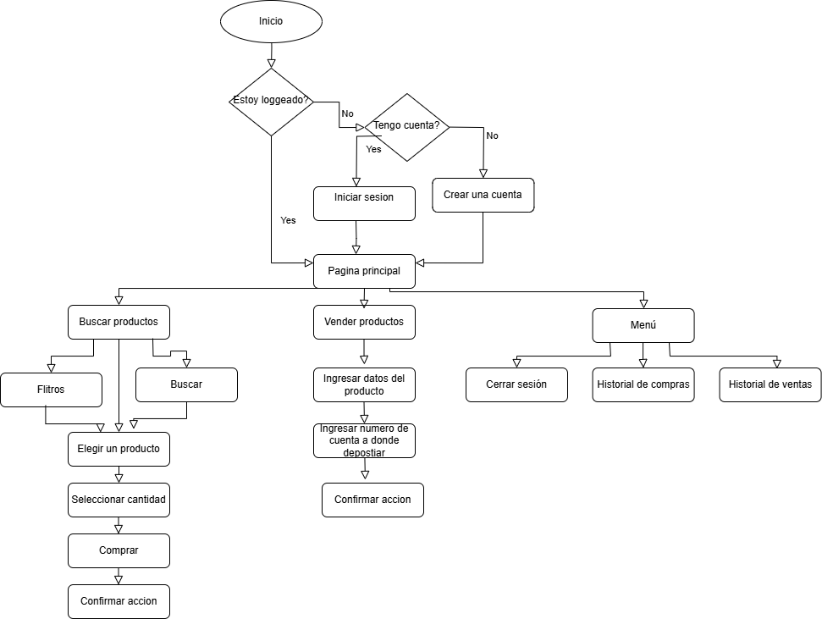


Orden:



Las demás entidades también cuentan con sus respectivos esquemas, pero por redundancia no se pondrán las fotos de estos modelos.

### Diagrama de flujo:

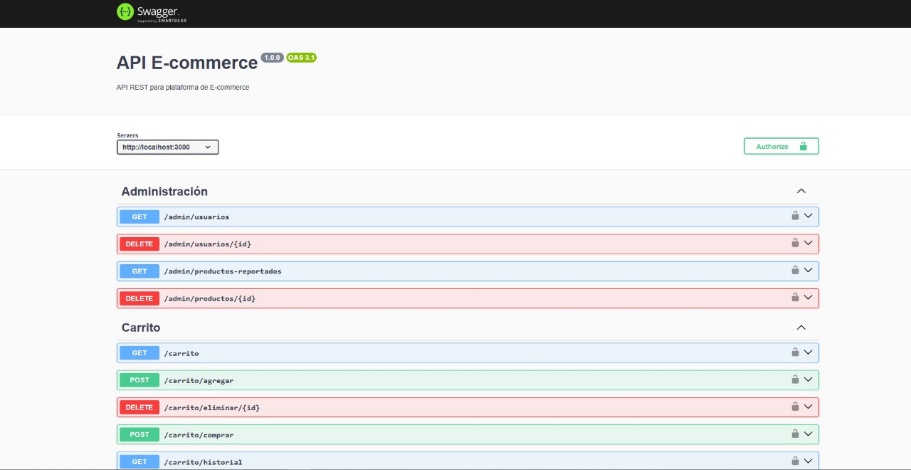


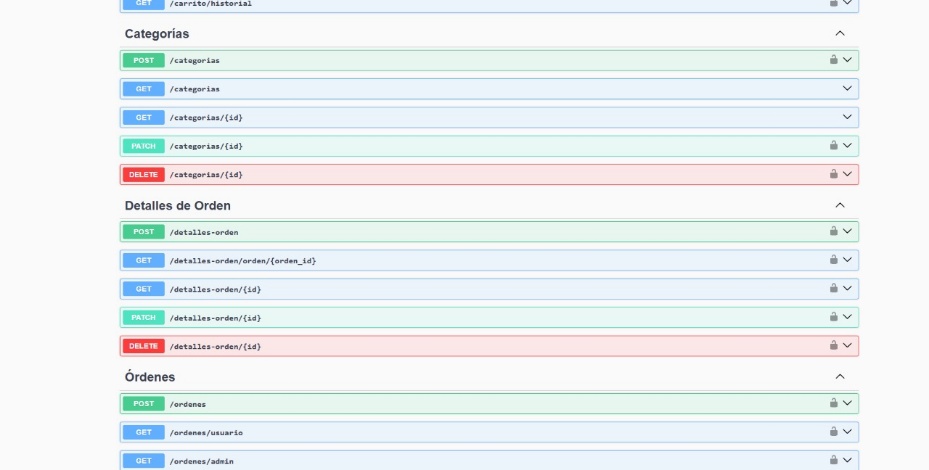
En el caso del flujo se espera que siempre se inicie en la autenticación del usuario, por eso, es necesario que se inicie sesión desde el inicio para poder acceder a los demás endpoints, posteriormente a iniciar sesión se puede buscar productos y comprarlos, por otro lado, también esta la ruta que se puedan dar de altas productos y venderlos, finalmente se puede cerrar la sesión.

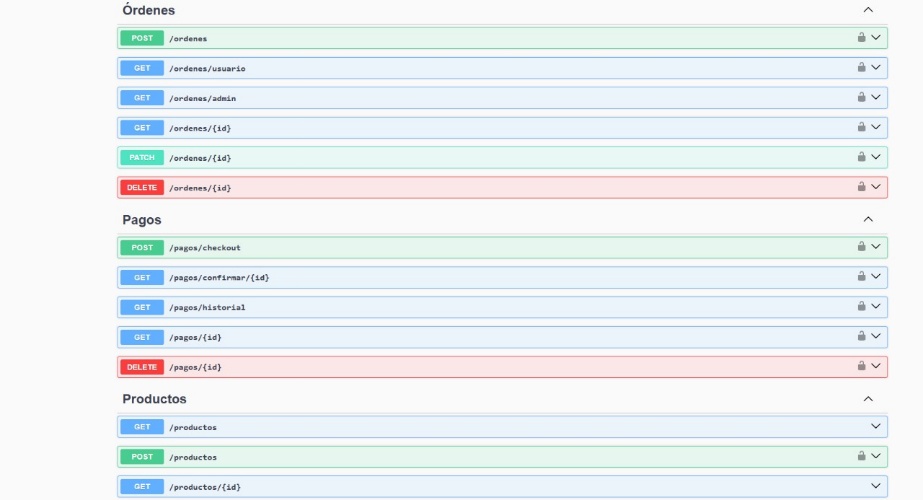
### Documentación:

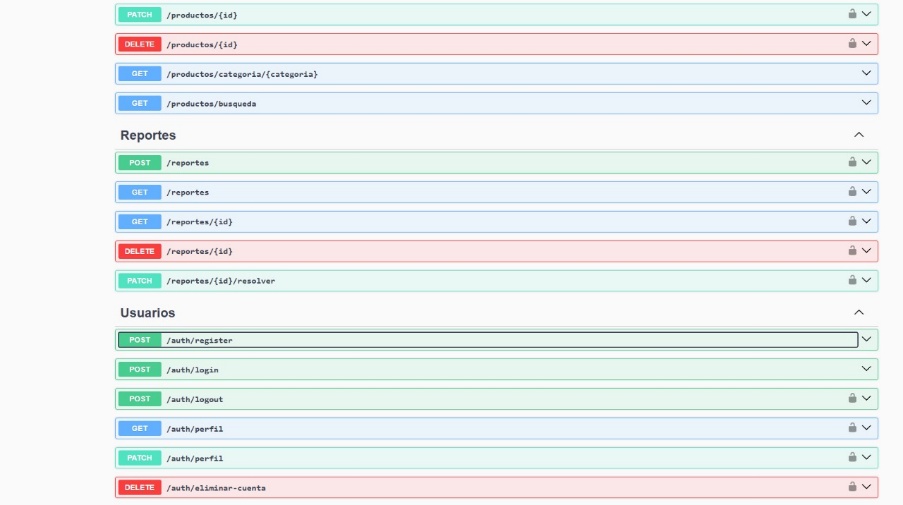
A continuación, se mostrará la configuración Swagger y su visualización, el endpoint creado para entrar a la documentación es localhost:3000/swagger (la primera imagen es la configuración de swagger):











Todos los endpoints fueron probados por medio de Postman para su verificación y su funcionamiento.

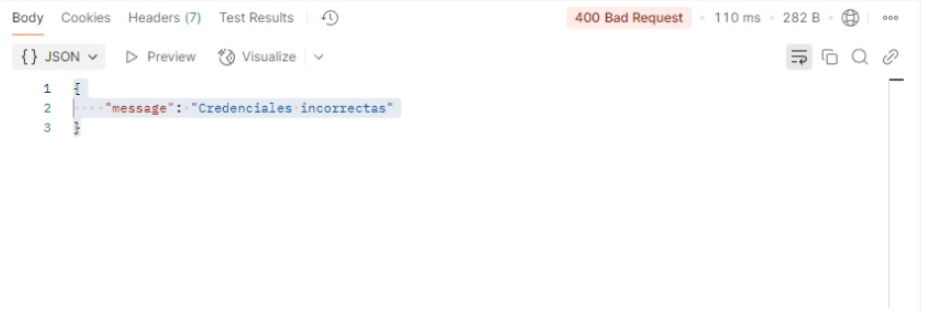
### Autenticación y permisos:

Para los usuarios, se implementó un middleware que verifica la autenticación mediante un token. Este token se genera al iniciar sesión utilizando JWT (JSON Web Tokens). En el caso del administrador, se aplican dos middlewares: el primero valida que el usuario esté autenticado, y el segundo verifica su rol. Este último requiere que el usuario tenga un atributo admin para poder acceder a ciertos endpoints restringidos.

Los usuarios al tener token pueden tanto comprar productos como dar de alta para su venta, mientras que el administrador puede eliminar productos como usuarios.

**Usuarios:**

* En el caso de usuario una vez intenta iniciar sesión por medio de su correo, se valida en la base de datos, que este exista, posteriormente se valida la contraseña, si la contraseña es correcta y el usuario también entonces se genera un token por medio de jwt, este mismo token es necesario para el endpoint del perfil y los demás endpoint.
* En caso de no contar con el token:



**Administrador:**

* En el caso del administrador es muy parecido al del usuario, pero además del token, se valida que tenga un rol de administrador para poder entrar a los demás endpoints.
* Usuario sin rol de admin:

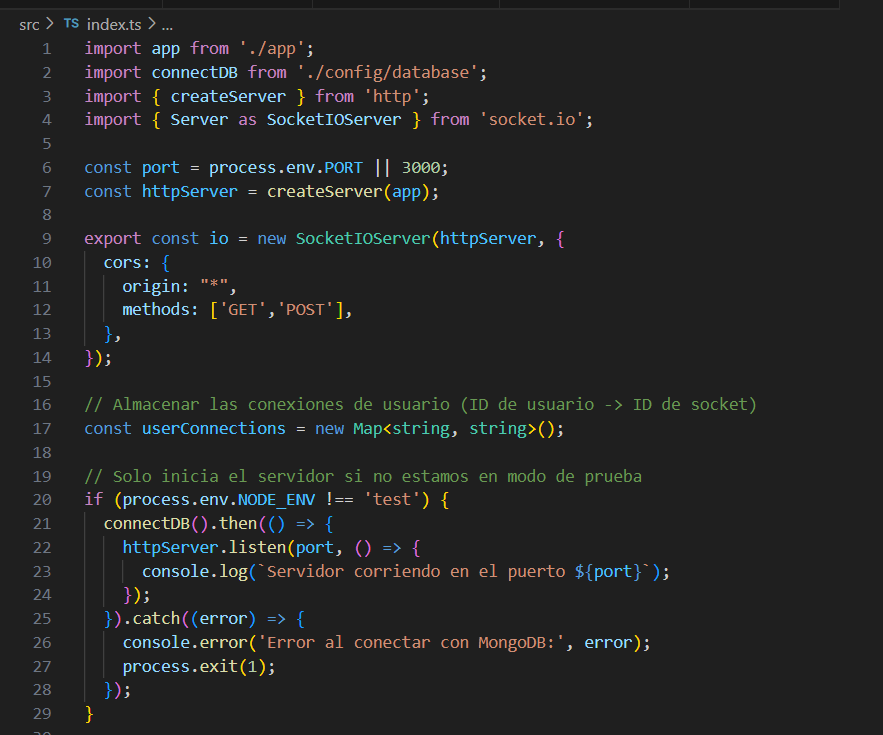


### Socket.io y su configuración:

La comunicación en tiempo real se implementó utilizando Socket.io para proporcionar notificaciones instantáneas entre vendedores y compradores.

**Configuración inicial:**

En el archivo index.ts se configura el servidor Socket.io junto con el servidor HTTP:

Esta configuración incluye:

* La creación del servidor HTTP utilizando Express
* La inicialización del servidor Socket.io sobre el servidor HTTP
* La configuración de opciones como CORS para permitir conexiones desde distintos orígenes

**Gestión de conexiones de usuarios:**

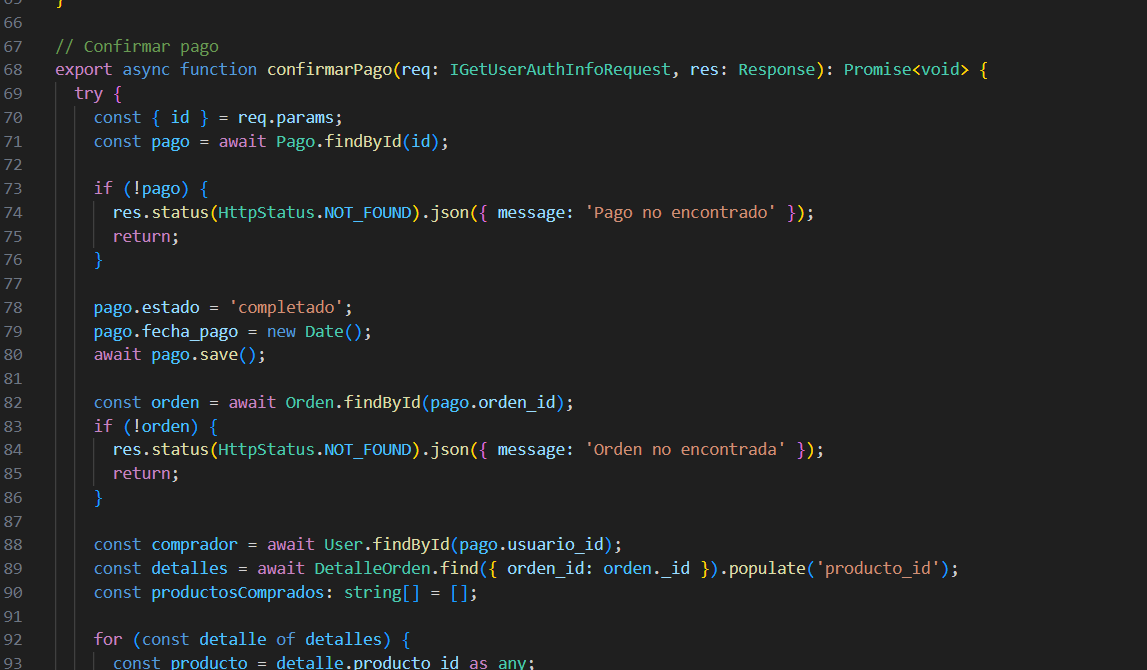
El sistema mantiene un registro de las conexiones activas mediante un mapa que relaciona los IDs de usuario con sus respectivos IDs de socket:

En este código se puede observar:

* Un mapa userConnections que almacena la relación entre usuario y socket
* Un evento 'register-user' que registra nuevas conexiones de usuarios
* El manejo de la desconexión de usuarios para mantener actualizado el mapa

**Notificaciones de compra:**

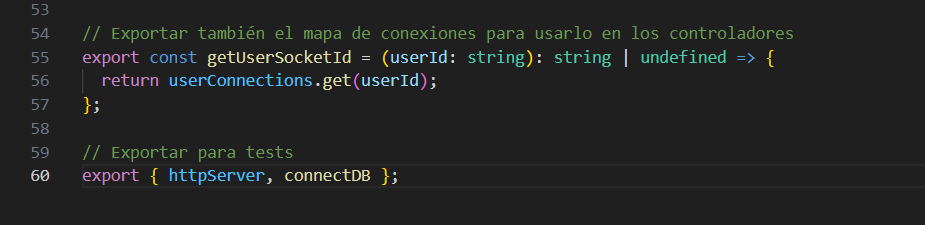
En el archivo Pagos.ts, dentro de la función confirmarPago, se implementa el envío de notificaciones a vendedores cuando se completa una compra:

Este código:

* Obtiene el ID de socket del vendedor usando la función c
* Envía una notificación directamente al socket del vendedor con información de la venta
* Implementa un fallback para casos donde el vendedor no está conectado

**Función auxiliar para notificaciones:**

Para facilitar la obtención del ID de socket de un usuario específico, se implementó la función getUserSocketId:

Esta función simple pero útil permite localizar rápidamente el socket de un usuario para enviarle notificaciones directas, evitando broadcasts innecesarios.  
  
**Flujo completo de notificaciones en tiempo real:**

1. Usuario se conecta al servidor Socket.io desde el frontend.
2. El cliente emite un evento 'register-user' con su ID para registrar la conexión.
3. El servidor almacena el mapeo entre ID de usuario y ID de socket.
4. Un comprador realiza una compra y confirma su pago.
5. El sistema identifica al vendedor de cada producto comprado.
6. Se busca el socket del vendedor usando su ID de usuario.
7. Se envía notificación directa al socket del vendedor con detalles de la venta.
8. Si el vendedor no está conectado, se emite el evento a un canal específico para recepción posterior.
9. Cuando el usuario se desconecta, se elimina su registro del mapa de conexiones.

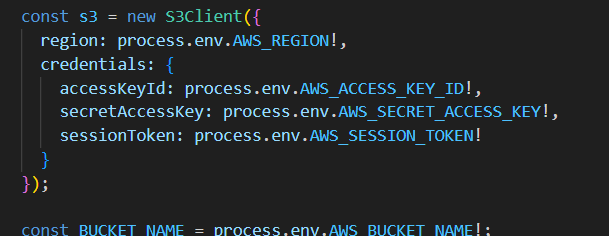
**Variables importantes:**

* io: Instancia principal de Socket.io exportada desde index.ts.
* httpServer: Servidor HTTP sobre el que se monta Socket.io.
* userConnections: Mapa que relaciona IDs de usuario con IDs de socket.
* socket.id: Identificador único de cada conexión Socket.io.
* vendedorSocketId: ID de socket del vendedor para enviar notificaciones.
* vendedorId: ID del usuario vendedor del producto.
* canal específico: Formato 'nueva-compra-${vendedorId}' para notificaciones dirigidas.
* getUserSocketId: Función auxiliar para obtener el ID de socket de un usuario.

### Flujo de carga y gestión de archivos con AWS S3:

**Configuración del cliente S3:**

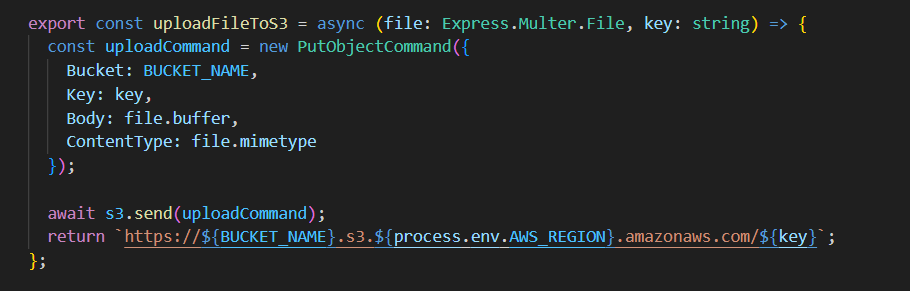
AWS SDK v3 requiere un cliente configurado, previamente antes de hacer peticiones. Las credenciales se cargan desde variables de entorno porque hard-codearlas en el código sería un riesgo de seguridad. El cliente debe saber la región porque cada bucket S3 está asociado a una región específica de AWS.



Esta configuración requiere la definición de las siguientes variables de entorno: AWS\_REGION, AWS\_ACCESS\_KEY\_ID, AWS\_SECRET\_ACCESS\_KEY, AWS\_SESSION\_TOKEN y AWS\_BUCKET\_NAME, las cuales son necesarias para habilitar el acceso y funcionamiento de los servicios de almacenamiento en AWS S3, todas están definidas en el archivo env.

**Subida de archivos genérica:**  
  
función uploadFileToS3:

S3 necesita tres elementos para subir un archivo: el destino (bucket), la ubicación dentro del bucket (key), y el contenido. El buffer es necesario porque S3 API trabaja con datos en memoria, no con archivos en disco. La URL se construye manualmente porque AWS no provee automáticamente la URL pública en la respuesta.

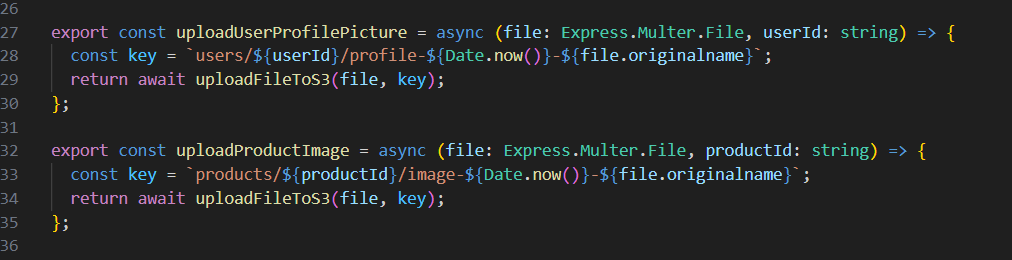


**Subida de imágenes de perfil y de producto:**  
  
Función uploadUserProfilePicture:

La estructura de carpetas "users/{userId}/" mantiene organizados los archivos por usuario, facilitando gestión posterior. El timestamp en la clave previene colisiones cuando un usuario sube múltiples fotos seguidas. Se llama a uploadFileToS3 en lugar de implementar todo de nuevo porque respeta el principio DRY.

Función uploadProductImage:

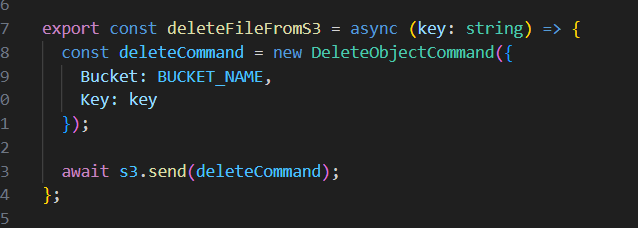
Similar al anterior, pero con "products/" en lugar de "users/", porque la estructura de directorios en S3, debe reflejar el tipo de contenido almacenado, facilitando backups, permisos y limpieza posterior.



**Eliminación de archivos:**

Función deleteFileFromS3:

S3 requiere la clave exacta porque no hay "búsqueda" de archivos en el API, solo eliminación directa por ruta. Es una operación irreversible, por eso los controladores verifican existencia antes de llamar esta función.



**Flujo completo de gestión de archivos:**

1. Cliente envía archivo multipart al endpoint.
2. Multer procesa el archivo en memoria.
3. Se verifica el usuario/producto.
4. Se elimina archivo anterior si existe.
5. Se sube nuevo archivo a S3.
6. Se actualiza URL en la base de datos.
7. Se retorna confirmación con URL pública.

Variables importantes:

* file.buffer: Contenido del archivo en memoria
* key: Ruta/nombre único del archivo en S3
* imageUrl/profilePictureUrl: URLs almacenadas en los modelos

### Pruebas por Postman:

A continuación, se verán imágenes de las pruebas realizadas por Postman para subir una foto de perfil por medio de AWS S3.

Subir foto de perfil:

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.**

**Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.**

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.**

Subir foto de producto:

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.**

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.**

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.**

Eliminar foto:

**Interfaz de usuario gráfica, Texto

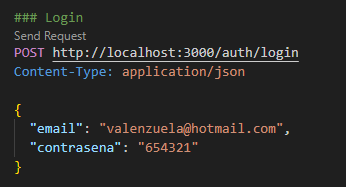
El contenido generado por IA puede ser incorrecto.**

**Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.**

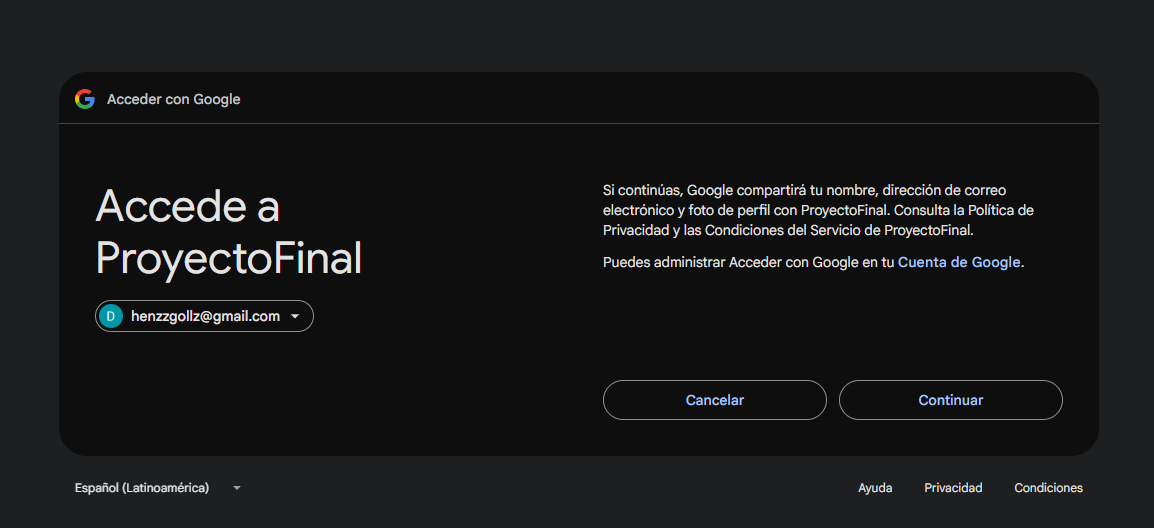
### Autenticación por Google y credenciales:

A continuación, se mostrará la autenticación por medio de credenciales para iniciar sesión, para esto es necesario haberte registrado en la base de datos, y acceder por medio de tu correo de usuario y tu contraseña.

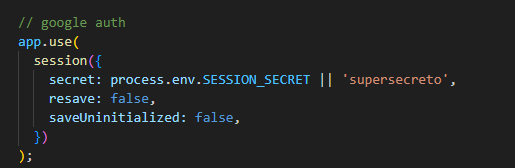
****

****

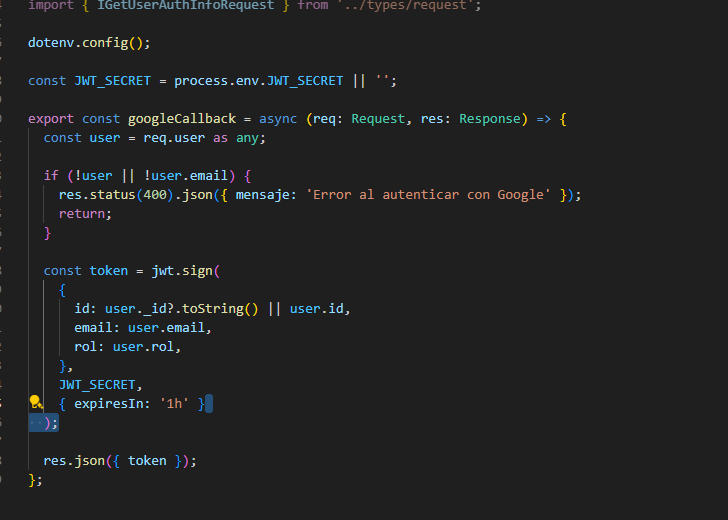
Finalmente, para iniciar sesión y autenticarte por medio de Google se necesita entrar al endpoint (auth/google), en donde se te pedirá permiso para obtener tus datos.



A continuación, se mostrará la configuración para la autenticación por Google, en este caso se ve el archivo en donde se inicia la sesión y la variable de entorno que se utiliza.



Controlador para el endpoint de auth/Google en donde se verifica que el usuario exista y que cuente con correo, posteriormente se hace un token por medio con el email y el nombre del usuario para devolverlo en formato json.

****

### Envío de correos:

La funcionalidad de envío de correos electrónicos está implementada para notificar tanto a compradores como a vendedores sobre las acciones relacionadas con una compra. Esta característica mejora la experiencia del usuario y garantiza la trazabilidad del proceso de venta.

Configuración técnica:

Se utilizó el módulo nodemailer para gestionar el envío de correos electrónicos desde la plataforma. La autenticación del correo se gestiona a través de variables de entorno (EMAIL\_USER, EMAIL\_PASS) y se configura un transporter con el servicio de Gmail.

Texto

El contenido generado por IA puede ser incorrecto.

La función enviarCorreo recibe el destinatario, el asunto y el cuerpo del correo en formato HTML, permitiendo así correos estructurados y personalizados.

Correos a vendedores:

Cuando un comprador confirma su pago, el backend recupera los productos incluidos en la orden y por cada uno:

1. Se identifica al vendedor del producto.
2. Se genera un correo de notificación de venta, que incluye:

* El nombre del producto.
* La cantidad comprada.
* El total pagado.
* El correo del comprador.

1. Se envía el correo al vendedor y se emite una notificación vía Socket.io.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Correos a compradores}.

Una vez procesado el pago, también se notifica al comprador. Este correo contiene:

* Un listado con todos los productos adquiridos.
* El total pagado.
* La fecha y hora de la transacción.
* El punto de encuentro para la entrega (si aplica).

Esto permite al comprador contar con un comprobante inmediato de su compra.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Seguridad y buenas prácticas:

* Las credenciales del correo se almacenan únicamente en variables de entorno (.env) y no se hardcodean.
* El contenido HTML está estructurado para facilitar la lectura del correo en distintos clientes de email.
* Los correos se envían solo si el usuario tiene una dirección válida en la base de datos.

Variables de entorno utilizadas

* EMAIL\_USER: Correo del remitente (cuenta Gmail usada para enviar).
* EMAIL\_PASS: Contraseña de aplicación generada desde Google (no la contraseña de acceso normal).

Flujo completo del envío

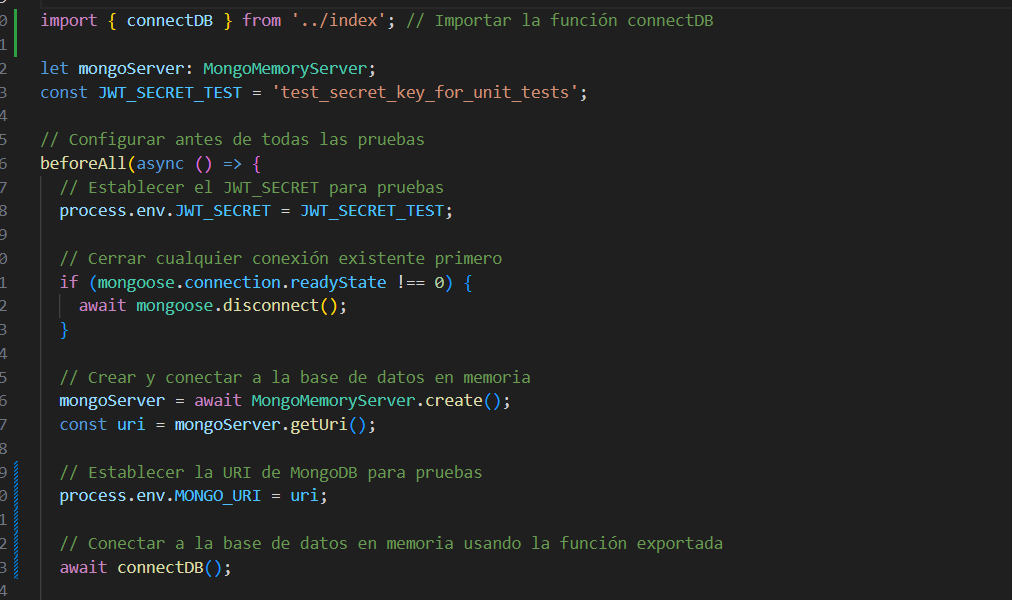
1. El usuario realiza una compra y se genera una orden.
2. Se confirma el pago desde Stripe.
3. Se recuperan los detalles de la orden.
4. Se notifican todos los vendedores involucrados.
5. Se notifica al comprador con los detalles completos.
6. Se registra la compra en la base de datos y se actualiza el stock de los productos.

### Pruebas unitarias:

Las pruebas unitarias son una parte fundamental del desarrollo de software para garantizar la calidad y funcionalidad del código. En este proyecto, se implementaron pruebas automatizadas utilizando Jest, un framework de pruebas para JavaScript que permite validar el correcto funcionamiento de los diferentes componentes del sistema.

**Configuración del entorno de pruebas:**

Se configuró un entorno aislado utilizando MongoDB Memory Server, lo que permite ejecutar pruebas sin afectar a la base de datos de producción. Cada prueba se ejecuta con una instancia limpia de la base de datos, garantizando que los resultados sean consistentes y repetibles.

  
La configuración incluye:

* Creación de una base de datos MongoDB en memoria para las pruebas
* Configuración de variables de entorno específicas para pruebas
* Registro de todos los modelos necesarios
* Funciones de limpieza entre pruebas

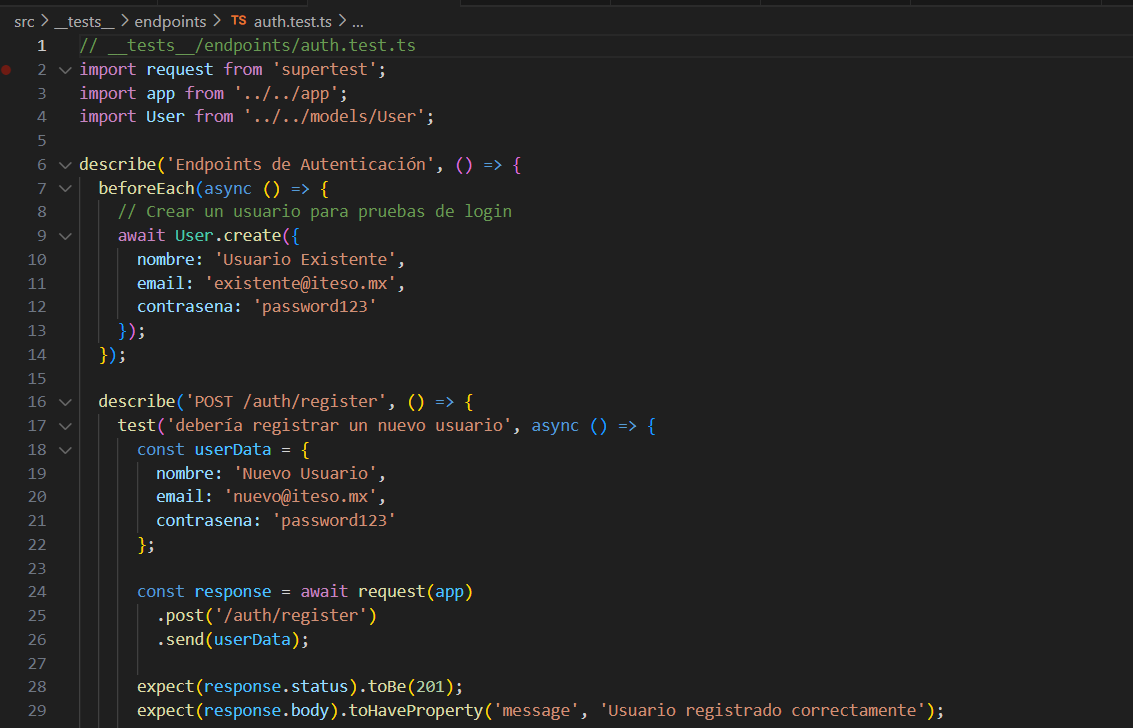
**Estructura de las pruebas:**

Las pruebas siguen una estructura organizada por módulos, correspondiendo cada archivo de prueba a un controlador o ruta específica de la aplicación. Esto permite identificar rápidamente la fuente de cualquier fallo.

**Pruebas de autenticación:**

Se implementaron pruebas exhaustivas para el sistema de autenticación, validando tanto el registro como el inicio de sesión de usuarios. Estas pruebas verifican:

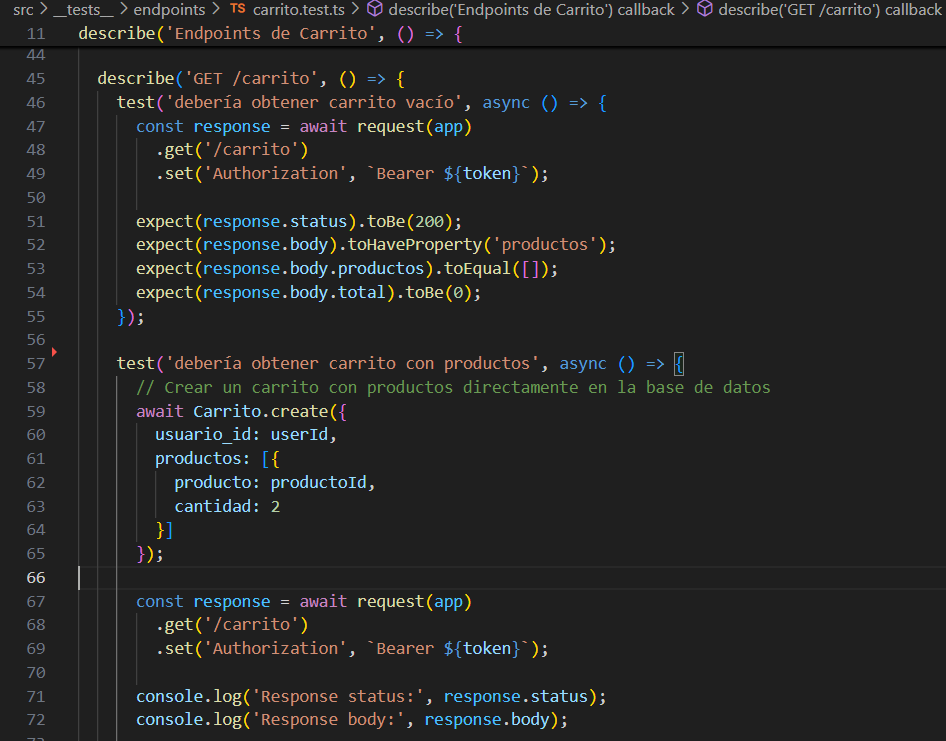
* Registro exitoso de nuevos usuarios
* Rechazo de correos electrónicos duplicados
* Generación correcta de tokens JWT
* Validación de credenciales incorrectas



**Pruebas del carrito de compras:**

Las pruebas del carrito de compras verifican la funcionalidad completa del ciclo de compra:

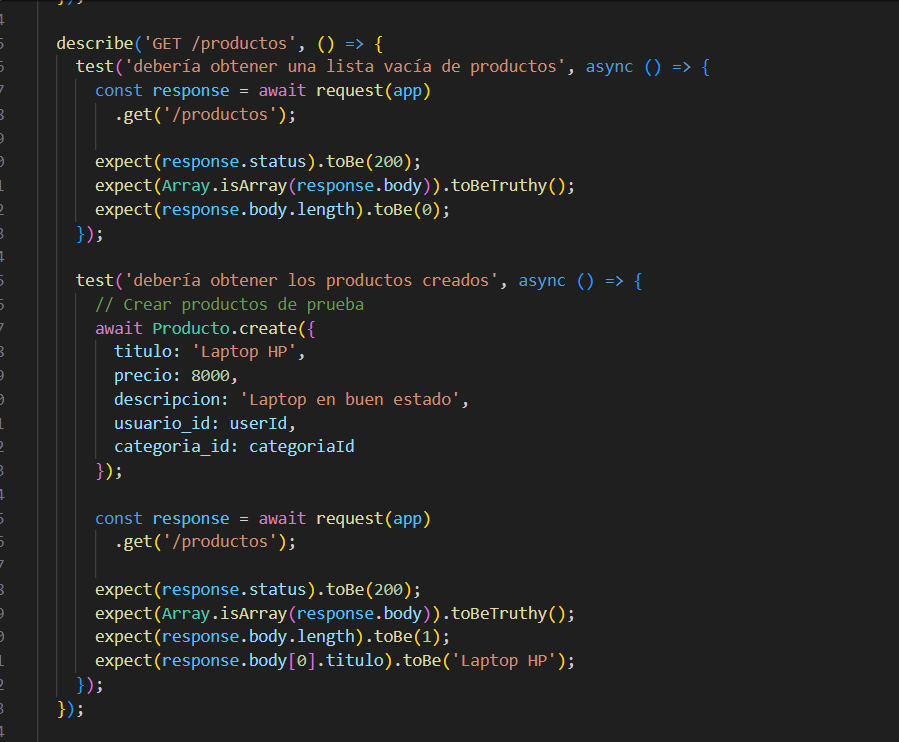
* Obtención del carrito vacío
* Agregar productos al carrito
* Eliminar productos del carrito
* Procesamiento de la compra



**Pruebas de gestión de productos:**

Se implementaron pruebas para validar las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) de productos:

* Obtención de lista de productos
* Creación de nuevos productos
* Validación de permisos de usuario
* Rechazo de operaciones no autorizadas



**Ejecución de pruebas:**

Las pruebas se ejecutan mediante el comando npm test, que utiliza la configuración de Jest definida en jest.config.js. Los resultados muestran el número de pruebas aprobadas, fallidas y el tiempo de ejecución.

Para ejecutar pruebas específicas, se puede utilizar:

npm test -- -t "nombre de la prueba"

**Flujo completo de ejecución de pruebas:**

1. Inicialización de la base de datos en memoria al comenzar las pruebas.
2. Registro de todos los modelos de Mongoose necesarios.
3. Creación de datos de prueba para cada caso (usuarios, productos, categorías).
4. Ejecución de la acción a probar (autenticación, operaciones del carrito, etc.).
5. Verificación de los resultados esperados mediante assertions.
6. Limpieza de la base de datos después de cada prueba.
7. Cierre de conexiones al finalizar todas las pruebas.

**Variables importantes:**

* mongoServer: Instancia de MongoMemoryServer para pruebas aisladas.
* JWT\_SECRET\_TEST: Clave secreta específica para generar tokens en pruebas.
* userId: ID de usuario creado para las pruebas.
* token: Token JWT generado para simular autenticación.
* productoId: ID de producto utilizado en pruebas de carrito y compras.
* categoriaId: ID de categoría usado en pruebas de productos.
* response: Objeto que contiene la respuesta de las peticiones HTTP.
* expect(): Función de Jest para realizar comprobaciones.

### Variables de entorno (.env):

* Mongo\_URI=mongodb+srv://nt729409:Indiano501@cluster0.mjyd5.mongodb.net/publicaciones?retryWrites=true&w=majority&appName=Cluster0
* PORT=3000
* JWT\_SECRET=mi\_clave\_secreta
* AWS\_REGION=us-east-1
* AWS\_ACCESS\_KEY\_ID= ASIAXSTVNQPTST742EUF
* AWS\_SECRET\_ACCESS\_KEY=OLbAzPg10DOd3GKiIn0tmBqHuZZLZgjIuMAqojVL
* AWS\_SESSION\_TOKEN=
* AWS\_BUCKET-NAME=archivos-servidor
* EMAIL\_USER=jopev0224@gmail.com
* EMAIL\_PASS=szlw pwdf cvcc bdwy
* STRIPE\_SECRET\_KEY=sk\_test\_51RNonZC0GDpVbMLZrZ6Uvay4nKVr94ppIfl9xWpFu41Gdmrjy0aZoGuNwRDT7vZzn25B8ZU2a5NYfe1KMvAJb58F00EnYWy5p8
* FRONTEND\_URL=https://httpbin.org
* GOOGLE\_CLIENT\_ID=675661748109tsdgpcn22fo9vtrl9uvl1pp1i8b4pu4.apps.googleusercontent.com
* GOOGLE\_CLIENT\_SECRET=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
* GOOGLE\_CALLBACK\_URL=http://localhost:3000/auth/google/callback
* SESSION\_SECRET=un\_secreto\_seguro