Nombre:	**Matrícula:**
1. Mauricio Hernández Quintanar 2. Juan Manuel Diaz Rivas 3. Daniella Hernández Hernández	1. 3001084 2. 7090780 3. 2989955
Desarrollo Fullstack	**Docente:** Guillermo Ambriz Carreon
Fecha de entrega: Sábado 24/1/2026 a las 11:59pm. A través de CANVAS	

Referencias

[1] Google. (2026). *Gemini* (versión del 30 de enero) [Modelo de lenguaje amplio]. Recuperado en enero 30, 2026 dehttps://gemini.google.com/app

[2] OpenAI. (2026). *ChatGPT* (versión del 30 de enero) [Modelo de lenguaje amplio]. Recuperado en enero 30, 2026 de https://www.openai.com/chatgpt

[3] npm, Inc. (s.f.). *About npm* Recuperado en enero 30, 2026 de https://www.npmjs.com/about

[4] MDN Web Docs. (2024). *Introducción a Express y Node.* Recuperado de https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/Introduction

[5] mongoosejs. (s.f.). *Mongoose* (Version 9.1.5) Recuperado en enero 31, 2026 de https://mongoosejs.com/

[6] Limpalair, C. (2022). *Hash Tables, Rainbow Table Attacks, and Salts*. Cybr. https://cybr.com/certifications-archives/hash-tables-rainbow-table-attacks-and-salts/

[7] M. Jones., J. Bradley., N. Sakimura. (2015) *JSON Web Token (JWT)* (RFC 7519). IETF. https://datatracker.ietf.org/doc/html/rfc7519

Bibliografía y otros recursos utilizados

[1] Amazon Web Services. (2021). *¿Qué es una interfaz de programación de aplicaciones (API)?* Amazon Web Services, Inc. Recuperado en enero 30, 2026 de https://aws.amazon.com/es/what-is/api/

[2] Express (s.f.). *Installing Express*. Recuperado en enero 30, 2026 de https://expressjs.com/en/starter/installing.html

[3] npm, Inc. (2025). *jsonwebtoken - npm* Recuperado en enero 30, 2026 de https://www.npmjs.com/package/jsonwebtoken

[4] npm, Inc. (2025). *dotenv - npm* Recuperado en enero 30, 2026 de https://www.npmjs.com/package/dotenv

[5] npm, Inc. (2026). *cors - npm* Recuperado en enero 30, 2026 de https://www.npmjs.com/package/cors#usage

[6] npm, Inc. (2026). *mongoose - npm* Recuperado en enero 30, 2026 de

https://www.npmjs.com/package/mongoose

[7] npm, Inc. (2025). *bcryptjs - npm* Recuperado en enero 30, 2026 de https://www.npmjs.com/package/bcryptjs

[8] MDN Web Docs. (2025). *Cross-Origin Resource Sharing (CORS) - HTTP* Recuperado en enero 31, 2026 de https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CORS

[9] w3schools (s.f.). *Node.js Built-in Modules* Recuperado en enero 31, 2026 de https://www.w3schools.com/nodejs/ref_modules.asp

[10] Universidad Tecmilenio. (2026) *Desarrollo Full Stack - Tema 7. Desarrollo backend con Node.js* Recuperado en enero 31 del curso en Canvas https://utm-cdn-labcontenidos-htfaarehf2gcfycs.a01.azurefd.net/contenido/profesional/lsti2313/explicas/tema-07.html?version=1

[11] Universidad Tecmilenio. (2026) *Desarrollo Full Stack - Tema 8. Express.js* Recuperado en enero 31 del contenido del curso en Canvas https://utm-cdn-labcontenidos-htfaarehf2gcfycs.a01.azurefd.net/contenido/profesional/lsti2313/explicas/tema-08.html?version=1

[12] Universidad Tecmilenio. (2026) *Desarrollo Full Stack - Tema 9. Bases de datos (operaciones CRUD)* Recuperado en enero 31 del curso en Canvas https://utm-cdn-labcontenidos-htfaarehf2gcfycs.a01.azurefd.net/contenido/profesional/lsti2313/explicas/tema-09.html?version=1

[13] npm, Inc. (2025). *nodemon - npm* Recuperado en enero 31, 2026 de https://www.npmjs.com/package/nodemon

[14] https://www.mongodb.com/docs/drivers/node/current/integrations/mongoose/mongoose-get-started/

[15] https://mongoosejs.com/docs/validation.html

[16] Mongoose v9.1.5: Validation https://mongoosejs.com/docs/validation.html#built-in-validators

[17]

Aplicación Fullstack

Avance de Proyecto

Introducción:

Para la entrega de este avance del proyecto decidimos mantener el mismo enfoque que en entregas anteriores, es decir, un sitio web para una tienda de ropa. Durante el desarrollo de esta actividad, creamos y realizamos las configuraciones iniciales de nuestra aplicación.

Crear y configurar una base de datos en backend, instalación de módulos y diseño del frontend fueron algunos de los principales trabajos.

Objetivos:

- Creación de un frontend usando HTML, CSS y JavaScript
- Creación de base de datos
- Creación y configuración de un servidor Express
- Autenticación de inicios de sesión
- Operaciones CRUD en base de datos

A) Configuración del proyecto

1. Inicia un nuevo proyecto con npm init en Node.js.

Node.js es un framework que se caracteriza por su forma asíncrona de trabajar y el uso de promesas, como en javascript ES6 (MDN Web Docs, 2024). Esto significa que es ideal para proyectos con solicitudes constantes, como una página web. Todo el sistema (frontend y backend) puede escribirse con javascript gracias a Node.js.

Cuando se instala Node su gestor de paquetes se incluye en la descarga. Node Package Manager facilita la importación de dependencias (librerías de Node.js) y su administración (npm, Inc., 2026). Al usar el comando `npm init` se crean dos archivos en la carpeta del proyecto (package.json y package-lock.json). El primero contiene toda la información relevante de la aplicación, nombre, autor, licencia, dependencias, repositorio de github, etc.

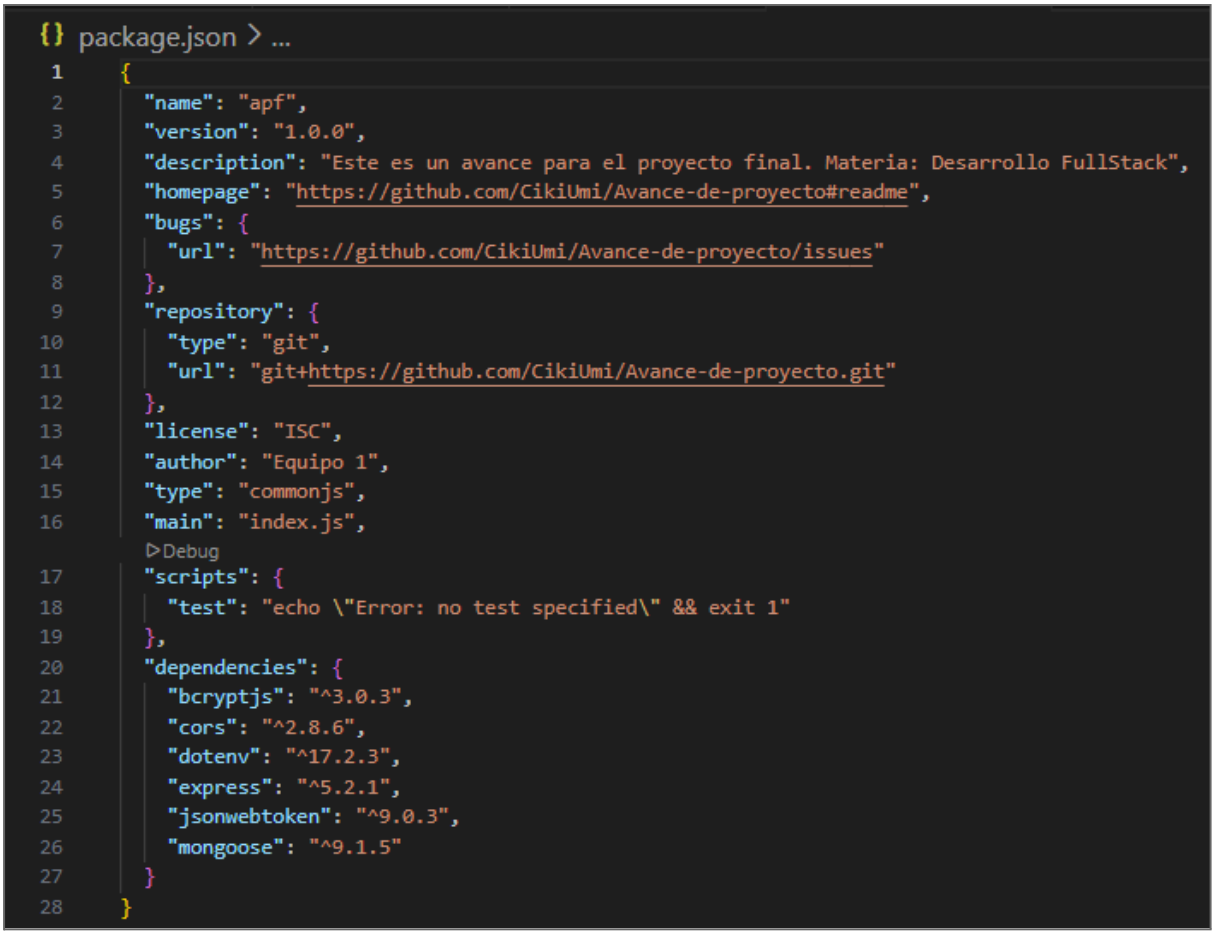


Imagen 1. Captura de pantalla con información de la página web. 'package.json'.

El segundo contiene las librerías de Node.js con su versión e información. Para descargar las dependencias del proyecto y poder utilizarlas se usa el comando `npm install`. (No hace falta subir el directorio node_modules al repositorio en github.)

2. Elige la base de datos que se utilizará (MongoDB, MySQL o PostgreSQL)

En una tienda en línea, un catálogo de nuestros productos con imágenes es algo esencial. MongoDB trabaja con bases de datos no relacionales, es bastante flexible y la mejor opción para almacenar datos no estructurados como los archivos.

Por esto y su entorno en la nube, elegimos trabajar con una base en MongoAtlas.

3. Instala las dependencias necesarias: express, jsonwebtoken, bcryptjs, cors. Dependiendo de la base de datos seleccionada, instala también mongoose (para MongoDB) o pg (para PostgreSQL).

Las dependencias se descargaron con `npm install`:

- **express**
  Un framework popular para Node.js bastante flexible y no dogmático, sin restricciones de estructura o herramientas para el proyecto. Sirve para definir rutas, su origen, y administrar las solicitudes y respuestas como `GET` o `POST` de acuerdo con su protocolo HTTP (Mozilla, 2024). Puede trabajar con APIs.

- **jsonwebtoken**
  Como sugiere su nombre, crea tokens web de verificación y los almacena en un JSON. Cuando un usuario inicia sesión se genera una cadena de texto que representa un *claim* (información suya guardada como clave y valores), se utiliza para verificar su identidad en cada petición que realiza. (Jones et al, 2015)

- **bcryptjs**
  De acuerdo con los casos de uso que npm, Inc. (2025) propone, esta librería se utiliza para la encriptación de contraseñas. Toma el texto y lo transforma en un hash utilizando sales (‘salt’ en inglés, caracteres aleatorios) para evitar que con el uso de tablas hash se puedan obtener las contraseñas (Limpalair, 2022).

- **cors**
  “Un middleware de Node.js para Express/Connect que aplica encabezados de respuesta CORS” (npm, Inc., 2026). Es decir, el CORS (Cross-Origin Resource Sharing) que indica el origen de las peticiones de otros servidores para dejarlas pasar. Un ejemplo es la comunicación entre front y backend (Mozilla, 2025).

- **mongoose**
  Es un ODM (Object Data Modelling) que facilita las operaciones relacionadas a la base de datos como la creación de objetos. Traduce los comandos y funciones para trabajar con los documentos de Mongo en javascript como si fueran objetos, permite realizar queries y validaciones (mongoosejs, 2025).

4. **Configura el proyecto con variables de entorno usando dotenv.**

Imagen 2. Captura de pantalla con ejemplo de archivo .env y las variables de entorno.

Las variables de entorno almacenan valores que permanecerán estáticos por toda la sesión. Un ejemplo es el puerto que recibe las solicitudes: no cambia en todo el uso de la aplicación. A diferencia de las variables globales, estas contienen información sensible como claves. Dotenv se encarga de ocultar sus valores.

Por ejemplo, en `process.env.MONGO_URI` se extrae el valor del archivo .env; no hay necesidad de colocar el enlace de conexión a la base en otros documentos.

5. **Consulta un ejemplo de configuración de variables de entorno utilizando dotenv para almacenar credenciales y claves sensibles, mediante una herramienta de inteligencia artificial (IA) Adapta el ejemplo proporcionado a las necesidades específicas del proyecto.**

De acuerdo con Gemini (2026), una buena práctica para almacenar variables de entorno es ocultar el archivo y omitirlo del repositorio en Github. utilizando un archivo .gitignore podemos señalar qué archivos no subir. En este caso se ocultó el

directorio con los módulos de Node.js y el archivo .env.

Además de pedir apoyo con la explicación de ciertos términos y módulos, solicitamos a ChatGPT y Gemini algunos ejemplos de estructuras para el directorio principal del proyecto. Ambos coincidieron en las siguientes carpetas:

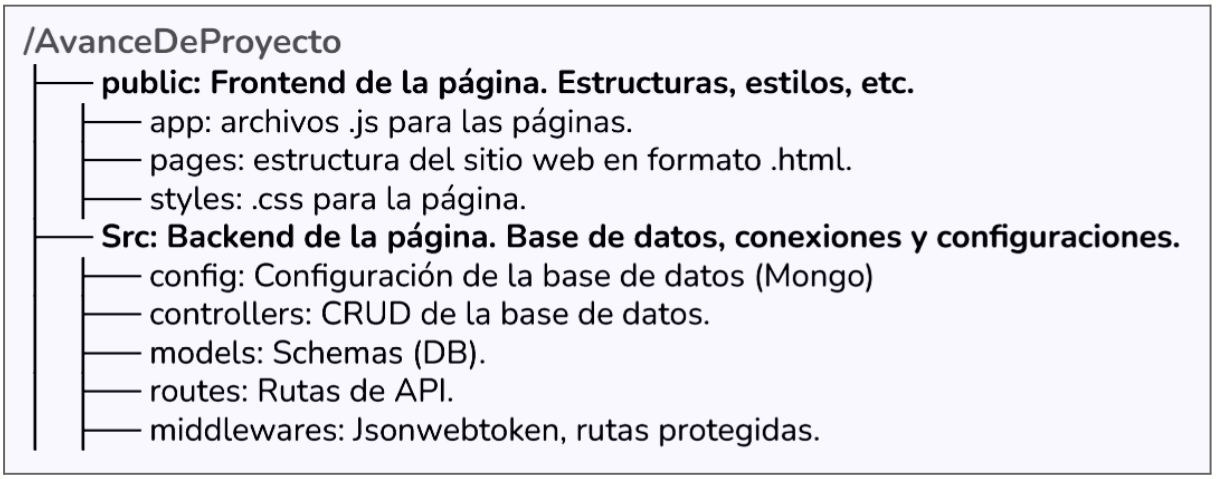
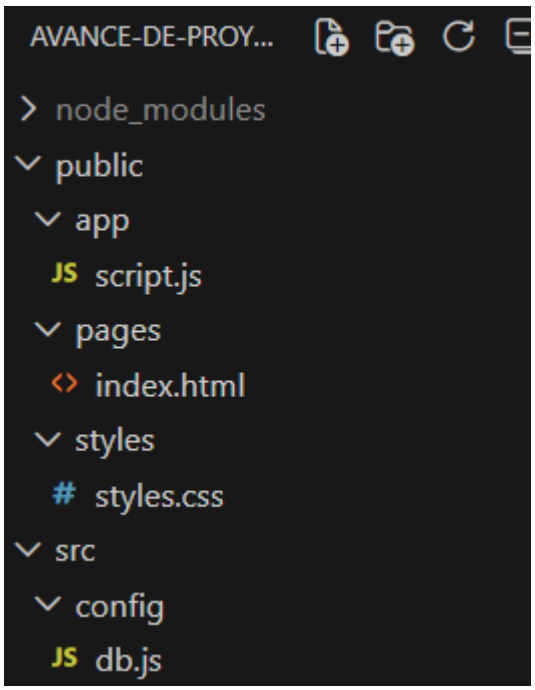
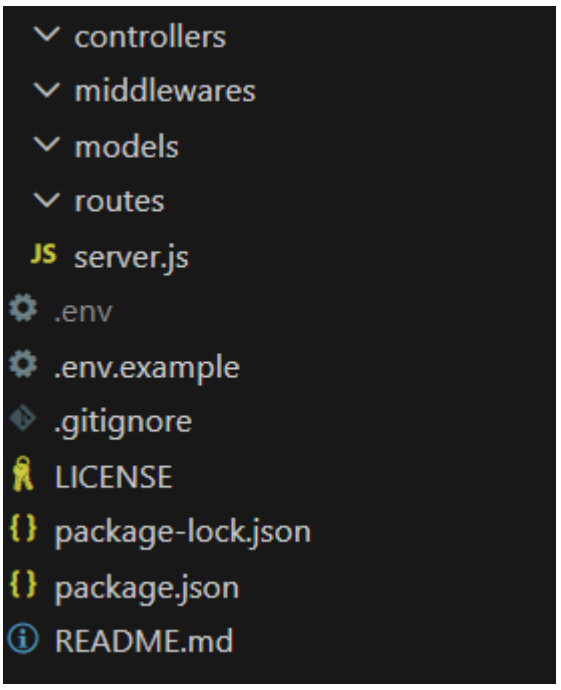


Diagrama 1. Organización del directorio para el proyecto.

Decidimos seguir la recomendación y aplicamos el mismo orden. Todo el proyecto está almacenado de esa forma para dividir y separar los archivos, permitiendo una mejor organización e implementación de modularidad en la página.

También creamos un archivo .env de ejemplo con variables vacías para subir a Github y que sirva de referencia a quien descargue o utilice el proyecto.





Imágenes 3 y 4. Captura de pantalla con la estructura del proyecto, carpetas, archivos, etc..

B) Backend (API RESTful con Express.js)

1. Crea un servidor básico en Express.

Con Express instalado podemos crear un servidor que reciba solicitudes y pueda enviar respuestas.

Utilizando el archivo server.js creado durante la estructuración del proyecto, podemos hacer la conexión desde el cliente. Primero seleccionamos el puerto desde .env para que reciba solicitudes, importamos las dependencias necesarias, y con una aplicación de express asignamos una respuesta.

src > JS server.js > ...

```
1  // por el package.json y el tipo (commonjs), se usan require y module.export en vez de import from y export
2
3  const dotenv = require('dotenv'); /* Variables */
4  const express = require('express'); /* Framework */
5
6  dotenv.config(); /* agarra variables de .env */
7  const app = express();
8  const PORT = process.env.PORT;
9
10 // Servidor :D
11 app.get('/', (req, res) => {
12   res.send('Servidor sencillo con express y node :DD!');
13 });
14
15 // Arrancar servidor
16 app.listen(PORT, () => {
17   console.log(`Servidor escuchando en http://localhost:${PORT}`);
18 });
```

Imagen 5. Captura de pantalla con el código para crear un servidor simple en express.

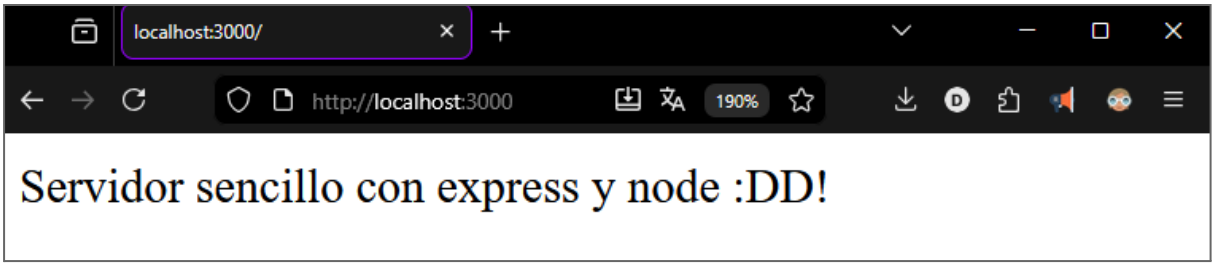


Imagen 6. Captura de pantalla con localhost:3000/ respuesta del servidor en express.

Ahora mismo sólo regresa una cadena de texto sencilla, pero esta misma función se utilizó en pasos posteriores para mostrar la página. Respondemos a la solicitud con un archivo, con el documento html de nuestra tienda online.

```
app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, '..', 'public', 'pages', 'index.html'));
});
```

Imagen 7. Captura de pantalla con la función que devuelve el archivo index.html.

2. Configura las rutas básicas para realizar operaciones CRUD (Create, Read, Update, Delete) utilizando una base de datos local, ya sea MongoDB, MySQL o PostgreSQL.

Primero se configuró la conexión a nuestra base de datos en MongoAtlas:

```
Node.js v24.11.0
PS C:\Users\fluff\OneDrive\Escritorio\_tareas\Fullstack\apf\Avance-de-proyecto> node src/server.js
[dotenv@17.2.3] injecting env (3) from .env -- tip: 🔐 prevent building .env in docker: https://dotenvx.com/prebuild
Servidor escuchando en http://localhost:3000
¡Conexión exitosa a MongoDB!
```

Imagen 8. Captura de pantalla con ejemplo de archivo .env y las variables de entorno.

En el archivo de server.js, debajo de la respuesta del servidor, se importan las dependencias cors y express.json para acceder a la base de datos. También se llama una función creada en el archivo config/db.js que toma el enlace con las credenciales de acceso a Mongo desde el archivo .env y realiza la conexión con mongoose.

.env | db.js

src > config > db.js > ...

```
1  /* Fragmento de código extraído de MongoAtlas */
2  const mongoose = require('mongoose');
3
4  const connectDB = async () => { /* nombre de función */
5    try {
6      // Create a Mongoose client with a MongoClientOptions object to set the Stable API version
7      /* Conexión a MongoDB usando variable de .env */
8
9      await mongoose.connect(process.env.MONGO_URI); /* obtenido de MongoAtlas */
10     console.log("¡Conexión exitosa a MongoDB!");
11   } catch (error) {
12     // Por si algo ocurre
13     console.error('Error al conectar a MongoDB:', error.message);
14     process.exit(1); /* Cierra si hay algo mal */
15   }
16 };
17
18 module.exports = { connectDB }; /* Para poder importarla en otros lugare */
```

Imagen 9. Captura de pantalla con variante de la función que MongoAtlas ofrece para conectarse.

Antes de la configuración de las operaciones CRUD definimos nuestras colecciones con los atributos generales para tener una base al crear los objetos. Nos decidimos por 4 distintas: dos para manejar la compra y envíos de los productos, un catálogo-inventario y una para almacenar las cuentas de los usuarios.

- **CATÁLOGO**
  - modelo
  - tipo
  - talla
  - precio
  - existencias
  - costoUnitario
  - imagen
  - descripcion
  - color

- **CARRITO DE COMPRAS**
  - productos
    [
    - ID_Producto
    - cantidad
    - costoUnitario
    - subtotal

    ]
  - usuario
  - total

- **PEDIDO**	- **CUENTA DE USUARIO**
- user - productos [- ID_Producto - cantidad - Costo Unitario - subtotal] - precioTotal - descuento - fecha - metodoPago (débito, crédito, transferencia, depósito, robado) - estado (en proceso, entregado, cancelado)	- nombre - correo - password - rol - direccion [- Calle - Estado - Municipio - Asentamiento - Núm Interno - Núm Externo] - lastLogin - creationDate

Tabla 1. Atributos de las colecciones en MongoDB.

Usando la lista anterior como base, organizamos los archivos para la creación de nuestra API. Se definieron los modelos de los esquemas (estructura y validación de datos para su creación) y sus colecciones para poder utilizarlos en la programación de los controladores (las operaciones CRUD) y se crearon las rutas para su uso en la página (`POST` , `GET` , `PUT` , `DELETE`).

```
1   const express = require('express');
2   const router = express.Router();
3
4   const {
5     createPedido,
6     updatePedido,
7     getPedido,
8     getAllPedidos,
9     deletePedido
10  } = require('../controllers/pedidoController');
11
12  router.post('/', createPedido);    /* Crear pedido */
13  router.put('/:id', updatePedido);      /* para editar el pedido */
14  router.get('/:id', getPedido);       /* Leer uno */
15  router.get('/', getAllPedidos);       /* Leer todos */
16  router.delete('/:id', deletePedido);       /* Eliminar uno */
17
18  module.exports = router;
```

Imagen 10. Captura de pantalla con la configuración de rutas para los pedidos.

A continuación, los endpoints (rutas) de nuestra API para llamar a las funciones:

cuentaUsuario		
Método	**Ruta**	**Acción que realiza**
POST	**/api/cuentaUsuario**	Crear una cuenta para usuario
GET	**/api/cuentaUsuario/:id**	Leer una cuenta de usuario
PUT	**/api/cuentaUsuario/:id**	Actualizar un usuario
DELETE	**/api/cuentaUsuario/:id**	Eliminar un usuario

Tabla 2. Endpoints de cuentaUsuario para uso de operaciones CRUD.

carrito		
Método	**Ruta**	**Acción que realiza**
POST	**/api/carrito**	Crear un carrito para un usuario
GET	**/api/carrito/:userID**	Obtener el carrito de usuario
PUT	**/api/carrito/:userID**	Añadir un producto al carrito del usuario
PUT	**/api/carrito/:userID/add**	Actualizar carrito del usuario
DELETE	**/api/carrito/:userID**	Eliminar carrito del usuario

Tabla 3. Endpoints de carrito para uso de operaciones CRUD.

catalogo		
Método	**Ruta**	**Acción que realiza**
POST	**/api/catalogo**	Crear un producto en el catálogo
GET	**/api/catalogo**	Obtener todos los productos
GET	**/api/catalogo/:id**	Obtener un producto por ID
PUT	**/api/catalogo/:id**	Actualizar un producto por ID
DELETE	**/api/catalogo/:id**	Eliminar un producto por ID

Tabla 4. Endpoints de catalogo para uso de operaciones CRUD.

pedido		
Método	**Ruta**	**Acción que realiza**
POST	**/api/pedido**	Crear un nuevo pedido
GET	**/api/pedido**	Obtener todos los pedidos
GET	**/api/pedido/:id**	Actualiza un usuario
PUT	**/api/pedido/:id**	Elimina un usuario
DELETE	**/api/pedido/:id**	Actualiza un usuario

Tabla 5. Endpoints de pedido para uso de operaciones CRUD.

Una vez configurado todo, se definieron los roles y permisos de las operaciones para trabajar más adelante con ellos:

Admin	**Crear**	**Leer**	**Actualizar**	**Eliminar**
Catalogo				
Carrito				
Pedido				
Cuentas				

Tabla 7. Permisos CRUD para el rol de Administrador (Admin).

Empleado	**Crear**	**Leer**	**Actualizar**	**Eliminar**
Catalogo				
Carrito				
Pedido				
Cuentas				

Tabla 8. Permisos CRUD para el rol de Empleado.

Cliente	**Crear**	**Leer**	**Actualizar**	**Eliminar**
Catalogo				
Carrito				

Pedido				
Cuentas				

Tabla 9. Permisos CRUD para el rol de Cliente.

3. **Implementa un sistema de autenticación utilizando JWT. Los usuarios deberán iniciar sesión para realizar ciertas operaciones. Dependiendo de tus reglas de negocio, las operaciones de lectura pueden ser abiertas, mientras que las operaciones de creación, modificación y eliminación pueden quedar protegidas.**

```
// POST /api/auth/login
router.post('/login', async (req, res) => {
  try {
    const { email, password } = req.body;
    if (!email || !password) return res.status(400).json({ message: 'Se necesita correo y contraseña' });

    const user = await User.findOne({ email });
    if (!user) return res.status(401).json({ message: 'Credenciales inválidas' });

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(401).json({ message: 'Credenciales inválidas' });

    const token = generateToken(user);
    res.json({
      message: 'Autenticado',
      user: { id: user._id, name: user.name, email: user.email },
      token,
    });
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Error del servidor' });
  }
});
```

Imagen 11. Captura de pantalla con ruta de autentificación, función para login.

Utilizando JWT para generar tokens, bcrypt para encriptar contraseñas y dotenv para indicar el tiempo de validez de una sesión con `TOKEN_EXPIRES_IN`, creamos tres funciones que administran registro de usuarios, el login y sus peticiones. Dos de ellas son parte de la API y cuentan con su ruta:

auth		
Método	**Ruta**	**Acción que realiza**
POST	**/api/auth/register**	Registrar un nuevo usuario
POST	**/api/auth/login**	Iniciar sesión y obtener token JWT

Tabla 5. Endpoints de pedido para uso de operaciones CRUD.

El registro solicita un correo, un usuario y una contraseña. En caso de cumplir todas las especificaciones, se encripta la contraseña y se almacena la cuenta nueva en la base de datos. Al iniciar sesión se comparan las claves. En caso de coincidir se le

otorga un token (con duración de una hora) al usuario.

Este Token se envía junto a las peticiones, decide si se pueden cumplir o no según su estado, los datos del usuario, etc.

4. **Utiliza middleware personalizado para manejar errores y validaciones.**

Con las funciones de JWT, podemos crear un middleware reutilizable. Este nos permite crear tokens y evaluarlos. Verificando si fueron firmados con nuestra contraseña creada en /dotenv.

src > middleware > JS auth.js > ...

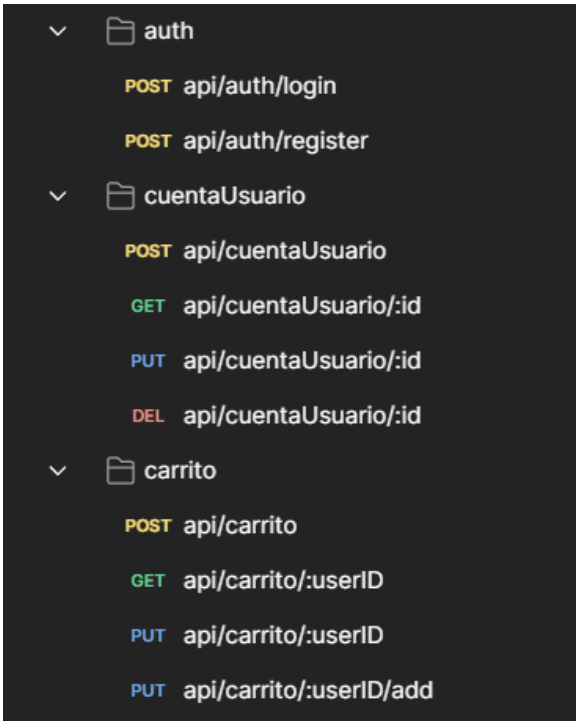
```
1  const jwt = require('jsonwebtoken');
2
3  const auth = (req, res, next) => {
4    const authHeader = req.headers.authorization || '';
5    const token = authHeader.startsWith('Bearer ') ? authHeader.slice(7) : null;
6    if (!token) return res.status(401).json({ message: 'Sin Token' });
7
8    try {
9      const payload = jwt.verify(token, process.env.JWT_SECRET);
10     // payload puede contener { id: ..., iat: ..., exp: ... }
11     req.user = { id: payload.id };
12     next();
13   } catch (err) {
14     return res.status(401).json({ message: 'El Token es inválido o ya expiró.' });
15   }
16 };
17
18 module.exports = auth;
```

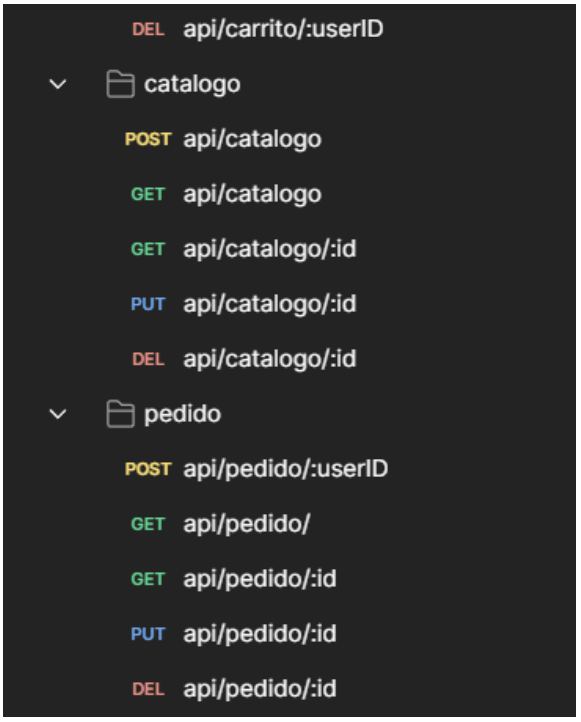
Imagen 12. Captura de pantalla con middleware de autenticación.

5. **Consulta una herramienta de inteligencia artificial (IA) para generar un middleware básico de manejo de errores en Express.js. Adapta el middleware a los requerimientos específicos de la aplicación y realiza pruebas para verificar su funcionalidad. Esto te permitirá visualizar ejemplos prácticos de manejo de errores en Express.js, aprender a adaptarlos a las necesidades del proyecto y mejorar la robustez del backend.**

Utilizamos la integración de Copilot en Github. Nos dio los ejemplos de rutas y controles para el inicio de sesión. Explicó paso a paso el procedimiento de verificación y encriptación de datos. Después de analizar y comprender las soluciones que nos dio, decidimos adaptarlas ligeramente para implementarlas a nuestro proyecto. Los nombres de variables, documentos placeholder del middleware, ruta de la autenticación (auth) y el archivo server.js se modificaron e integraron. Con esto finalizamos el backend de la página web.

Es buena idea probar tus APIs, antes de hacer el front end para saber si la lógica detrás de ella funciona. Postman es una buena herramienta para poder hacerlo desde solicitudes directas. (Gemini, 2025). Siguiendo este consejo, con la aplicación de escritorio hicimos un par de pruebas: creación de usuarios, carrito de compras, etc. Primero metimos todas las rutas a Postman en una colección nueva:





Imágenes 13 y 14. Captura de pantalla con las rutas almacenadas en Postman para pruebas

Durante la primera prueba, los usuarios se crearon exitosamente, pero notamos que se almacenaban en una base de datos nueva llamada 'test'. Se actualizó la variable MONOG_URI para incluir la base de datos y se renombraron las colecciones en plural y todo en minúsculas como mongoose lo hace. Esto arregló los problemas:

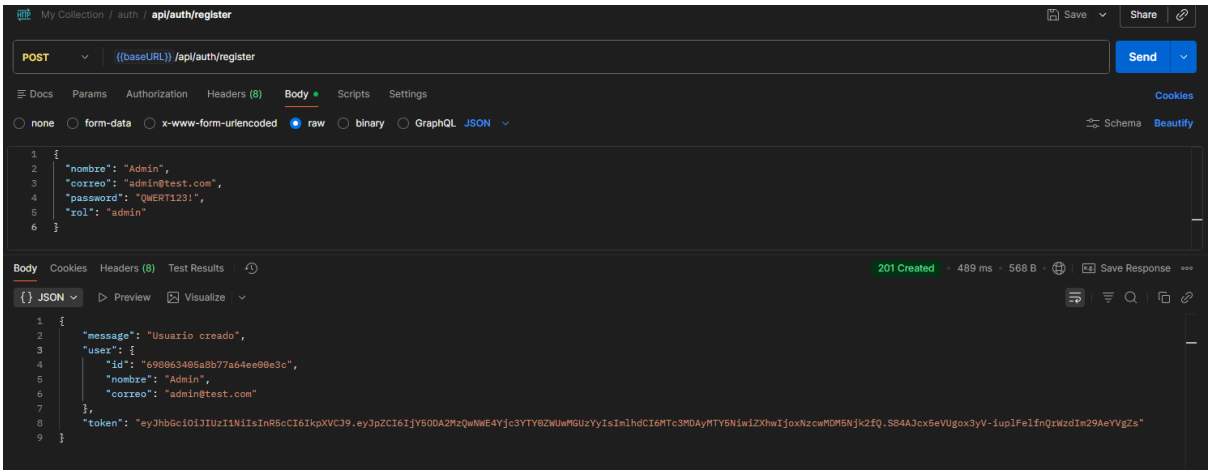


Imagen 15. Captura de pantalla del usuario creado en Postman.

Imagen 16. Captura de pantalla del usuario nuevo en la base de datos.

Durante las pruebas, nos dimos cuenta de que el diseño inicial de atributos era algo

ineficiente. Definimos un orden para los atributos e hicimos más dinámicos a los objetos. Por ejemplo, los productos ahora tienen diferentes modelos y cada uno tiene tallas y existencias. Al trabajar con JSONs, es posible omitir algunos atributos: si vendieramos collares no hay necesidad de colocar una talla.

Imagen 17. Captura de pantalla del formato nuevo para los objetos del catálogo en MongoDB.

C) Frontend (HTML, CSS, JavaScript)

1. **Desarrolla un frontend básico que incluya una página de inicio de sesión y una página principal donde se muestren los datos gestionados desde el backend (como lista de tareas, usuarios, productos, etc.).**

2. **Utiliza *CSS* para el diseño visual (aplicar los principios básicos de diseño UI/UX) y *JavaScript* básico para manipular el DOM, enviar peticiones al servidor y recibir respuestas.**

   - Consulta una herramienta de inteligencia artificial (IA) para obtener sugerencias de diseño en CSS para la página de inicio de sesión. Evalúa las propuestas y adapta las más adecuadas para optimizar la experiencia del usuario.

     Esto fomentará la creatividad en el diseño y permitirá explorar ideas innovadoras generadas por IA, mejorando la apariencia y usabilidad de la interfaz mientras se alinean con los principios fundamentales de diseño UI/UX.

3. **Implementa interacciones dinámicas utilizando JavaScript, como agregar y eliminar elementos de una lista.**

D) Despliegue preliminar

1. **Realiza el despliegue de la aplicación en un entorno local.**

2. **Si es posible, configura un despliegue inicial en un servicio en la nube como *Heroku*, *Render* o *AWS* para el backend.**

E) Entrega avance del reto

1. **Incluye los siguientes elementos en un documento en formato Word o PDF:**
   - Código fuente completo del backend y frontend, subido a un repositorio en GitHub.
   - Documentación técnica donde expliques la estructura del proyecto y los endpoints de la API.
   - Graba un video demostrativo que muestre el funcionamiento del frontend y de las operaciones CRUD en el backend.
   - URL del despliegue preliminar en la nube.

https://github.com/CikiUmi/Avance-de-proyecto.git

Enlace al repositorio en Github con el proyecto,

Conclusiones

Mauricio - 3001084

Esta es, sin lugar a dudas, uno de los proyectos más difíciles que hemos hecho hasta ahora. La configuración de una página es completamente diferente a lo que había pensado. Existen gran cantidad de eventos que deben mantenerse y manejarse para el funcionamiento de la web como la conocemos.

Me gustaría seguir con este proyecto para poder pulirlo y tener la experiencia completa, por ejemplo, crear el servidor y publicar la página.

Juan - 7090780

En conclusión este proyecto permite el desarrollo de un aapp web completa , integrando cosas ya vistas en clase como el frontend y backend, ocupando un abase de datos de mongoDb para poder guardar la información, además se creó un frontend funcional con html, CSS y javascript, aplicando principios de UI/Ux y diseño repositorio para asegurar una correcta visualización en diversas plataformas

Daniella - 2989955

El salto de dificultad entre actividades fue bastante grande. Creo que la parte más lenta fue el inicio, entender todas las dependencias que instalamos, para qué estábamos configurando ciertas cosas y el proceso de poco a poco construir la página. Como vimos en clase, el desarrollo Full Stack no está pensado para que una sola persona haga todo el trabajo. Definir todos los elementos antes de comenzar fue mi parte favorita, crear la estructura de la base de datos, definir los espacios para cada elemento, etc.