MÓDULO DE PROYECTO

Técnico Superior en Desarrollo de Aplicaciones Web

"Sistema de Gestión Web para Farmacéuticas"

Autor: "Francisco Javier Belda Alovera"

Profesor tutor: "Mari Luz Elola"

Alcalá de Henares, "Junio" de "2025"

Índice

---

1. Introducción y justificación

Este documento describe el desarrollo de un sistema de gestión para una farmacéutica, cuya finalidad es facilitar el control de inventario de medicamentos, el seguimiento de pedidos a proveedores, la gestión de empleados y la planificación de sus turnos de trabajo, así como la administración de usuarios con distintos roles (Gerente y Empleado). El proyecto se ha utilizado Angular en el frontend, y en el backend ha sido desarrollado con Node.js utilizando Express y MongoDB como sistema de gestión de base de datos.

Motivación

La verdad que hay varias razones por las que me decanté por este proyecto y algunas de ellas son que varios de mis familiares trabajan/han trabajado en el sector sanitario y han tratado con el sector farmacéutico concretamente y me han ayudado bastante al darme consejos para realizar la parte visual del proyecto, además de que quería crear una aplicación que pudiera abordar el sector profesional sanitario.

Objetivos

- Desarrollar una aplicación web que permita:
    - Autenticación de usuarios con roles diferenciados (Gerente/Empleado).
    - Visualizar, crear y editar registros de medicamentos.
    - Gestionar pedidos a proveedores, registrar nuevos pedidos, actualizar su estado y cancelarlos.
    - Controlar el inventario mediante actualización automática de stock al realizar pedidos o registrar ventas.
    - Administrar proveedores y asociar pedidos a cada uno.
    - Gestionar información de empleados y planificar sus turnos de trabajo en un calendario mensual.
- Implementar el backend con una API RESTful segura, robusta y escalable.
- Utilizar MongoDB para el almacenamiento de datos, aprovechando su flexibilidad en colecciones y documentos.
- Desplegar la aplicación en un entorno de producción sencillo, con instrucciones claras de instalación y configuración.

Medios hardware y software a utilizar

- **Hardware:**
    - Ordenador de desarrollo con al menos 8 GB de RAM y procesador de cuatro núcleos.
    - Servidor (vía cloud o VPS) con al menos 2 GB de RAM para despliegue.
- **Software:**
    - Visual Studio Code (IDE).
    - Node.js (versión 18.x o superior).
    - Angular CLI (versión 19.x).
    - MongoDB (versión 8.x) como gestor de base de datos.
    - Git para control de versiones.
    - Navegador web moderno (Chrome, Firefox).

---

2. Estudio de viabilidad del proyecto

2.1. Viabilidad económica

Estimación de costos:

- **Hosting y servidor:**
    - VPS en proveedor cloud (DigitalOcean, Hetzner o similar) con 2 GB RAM: ≈ €6-8/mes.
    - Dominio web ("misistemafarmaceutica.com", ejemplo): ≈ €10/año.
    - Certificado SSL (Let's Encrypt): gratuito.
- **Licencias y herramientas:**
    - Angular, Node.js, Express, MongoDB Community Edition: todos de código abierto, sin coste de licencia.
    - IDE Visual Studio Code: gratuito.
    - Git (control de versiones): gratuito.
- **Costes adicionales:**
    - (Opcional) Base de datos en servicio gestionado (MongoDB Atlas Free Tier o equivalente): gratuito para entornos de desarrollo. Planes de pago a partir de €9/mes en producción.

Coste aproximado total anual:

- Servidor VPS: €6 × 12 = €72
- Dominio: €10
- **Total aproximado: €82/año**

Retorno de la inversión (ROI):

La aplicación está orientada a pymes farmacéuticas o farmacias de tamaño medio que requieran un sistema básico de gestión de inventario y pedidos. Si se cobrara una suscripción mensual de €20, con 5 clientes activos sería posible recuperar la inversión en menos de un año (€20 × 5 × 12 = €1 200). Además, el coste de mantenimiento anual (VPS + dominio) es de aproximadamente €82, por lo que la aplicación podría generar beneficios a partir del segundo año si se mantiene esta cartera de clientes.

Hay que tener en cuenta en un futuro en caso de tener un gran abanico de clientes seria necesario aumentar el tamaño del servidor para evitar largos tiempos de carga y/o caídas de este.

2.2. Viabilidad legal

Cumplimiento normativo:

- **Protección de datos (GDPR):**
    - La aplicación gestiona datos personales de empleados (nombre, apellidos, correo, teléfono, calendario de turnos) y proveedores (nombre, dirección, correo). Es necesario garantizar el cumplimiento del Reglamento General de Protección de Datos (GDPR) de la UE.
    - Política de privacidad: se debe incluir un aviso legal y de privacidad en la interfaz.
    - Gestión de cookies: Angular CLI genera cookies para la sesión de la aplicación; se debe informar al usuario y darle la opción de consentimiento.
- **Comercio electrónico / venta de medicamentos:**
    - No se manejan datos de pacientes, solo compras a proveedores y ventas a clientes.

Licencias de software:

- **Frameworks y librerías usadas:**
    - Angular: MIT.
    - Node.js y Express: MIT.
    - Mongoose (ODM para MongoDB): MIT.
    - Axios: MIT.
    - QuickChart (carga dinámica): MIT.
    - Otras dependencias estándar (dotenv, cors) bajo licencia MIT.

Al usar únicamente tecnologías y librerías de código abierto (mayoritariamente bajo licencia MIT), no existen costes de licenciamiento y se permite la redistribución comercial.

Propiedad intelectual:

- El proyecto es totalmente original, desarrollado íntegramente por el autor.
- No se reutiliza ningún código propietario de terceros.
- Se respetan las licencias MIT de las dependencias, que permiten uso comercial y modificación sin obligación de abrir el código.
- La documentación y contenidos textuales (comentarios, textos de interfaz) han sido creados por el autor.
- Se propone licenciar la propia aplicación bajo licencia MIT, permitiendo uso, copia, modificación y distribución sin restricciones, siempre que se mantenga la atribución.

2.3. Viabilidad de tiempo o cronograma

Mi planificación iba a ser de unas 8 o 9 semanas, no puedo decir que tiempo le dediqué en total a cada sección debido a que iba trabajando con todo sobre la marcha pero aquí cometo mas o menos como me distribuí:

- **El análisis de requisitos y diseño:**
  - No tardé mucho debido a que ya tenia claro como quería que se viera la aplicación y que funcionalidades iba a tener, con lo que le dediqué un par de días.
- **Configuración del entorno:**
  - Al igual que con el análisis no le dedique mucho tiempo ya que sabia lo que quería/necesitaba, pero según iba avanzando en el desarrollo del proyecto veía que necesitaba nuevas colecciones en la base de datos e importaciones tanto en el frontend como en el backend.
- **Desarrollo del frontend y backend:**
  - Los iba desarrollando juntos ya que buscaba que al realizar algo en el frontend funcionase perfectamente, se conectase con el backend y que persistiera en la base de datos y así tener un seguimiento de objetivos cumplidos y ver los restantes. Sin duda esto fue lo que me llevó más tiempo dedicándole probablemente unas 7 u 8 semanas.
- **Pruebas e integración:**
  - Las pruebas y la integración al git iban de la mano con lo mencionado anteriormente ya que en cuanto conseguía que partes importantes del proyecto funcionasen lo integraba al github. El tiempo que le dedique este integrado en el desarrollo del frontend y backend.
- **Despliegue y validación final:**
  - Este apartado duró poco ya que al hacer integraciones poco a poco de objetivos cumplidos ya sabia que la aplicación estaba finalizada.
- **Elaboración de la memoria:**
  - En cuanto a la memoria le he dedicado menos de una semana ya que me aseguré de comprender las especificaciones y así en unos pocos días la tendría completada. Probablemente le dedicase unos 5 días mas o menos.

Cabe destacar que no utilicé ninguna metodología ágil. Simplemente elaboré un documento de texto con las tareas por realizar y las fui tachando conforme las completaba.

Evaluación de desviaciones:

- En el desarrollo del frontend y del backend he de mencionar que me tomó mas tiempo del que esperaba debido a la creación de un nuevo componente de calendario para la planificación de turnos de empleados y me atasqué un tiempo.
- En las pruebas aunque el tiempo utilizado iba de la mano del desarrollo del frontend y backend me constó mas tiempo debido a problemas en la comunicación entre estos por parámetros mal gestionados entre otras razones.

La realidad fue que le dediqué prácticamente más de dos meses debido a contratiempos como los mencionados anteriormente pero cabe destacar que todo lo que tenia pensado meter en la aplicación esta integrado y es funcional.

---

3. Análisis y diseño del proyecto

3.1. Descripción de la arquitectura web

El sistema está basado en una arquitectura cliente-servidor con los siguientes componentes:

- **Frontend (SPA Angular):** El cliente se implementa como una aplicación Single Page Application (SPA) usando Angular 19. La comunicación con el backend se realiza mediante llamadas HTTP a una API RESTful. La aplicación se despliega estática mediante un servidor web (por ejemplo, Nginx) o directamente desde el mismo servidor Node.js.
- **Backend (API REST Node.js):** El servidor está construido con Node.js y Express. Sigue un patrón de organización MVC simplificado:
    - **Modelos (Mongoose):** Definen esquemas de MongoDB para entidades como Usuario, Empleado, Medicamento, Proveedor y Pedido.
    - **Controladores:** Contienen la lógica de negocio para cada entidad (operaciones CRUD, autenticación, actualización de turnos, registros de pedidos).
    - **Rutas:** Definen los endpoints HTTP (GET, POST, PUT, DELETE) y conectan a los controladores correspondientes.
- **Base de datos (MongoDB):** Se utiliza MongoDB como sistema de base de datos NoSQL. Las colecciones principales son:
    - **usuarios:** Información de acceso y rol de cada usuario.
    - **empleados:** Detalles personales y calendario de turnos para cada empleado.
    - **medicamentos:** Datos de cada medicamento, stock, pedidos y ventas asociadas.
    - **proveedores:** Información de proveedores y pedidos asociados.
    - **pedidos:** Registro independiente de cada pedido a proveedor, usado para control de stock.

- **Comunicación y flujo de datos:**
    - El usuario inicia sesión desde el frontend, enviando credenciales a `/auth/login` (backend).
    - El backend valida las credenciales (busca en colección `usuarios`) y devuelve datos de usuario sin la contraseña.
    - El frontend almacena la información en localStorage (AuthStorageService) y redirige a la sección principal según el rol.
    - Para cada módulo (Medicamentos, Proveedores, Pedidos, Empleados), Angular realiza peticiones HTTP (usando Axios) al backend a través de sus servicios.
    - El backend responde con datos en formato JSON que el cliente procesa y muestra dinámicamente.

Este esquema permite una separación clara entre la lógica de presentación (frontend) y la lógica de negocio (backend), facilitando el mantenimiento, la escalabilidad y la posibilidad de desplegar partes del sistema por separado.

3.2. Tecnologías y herramientas utilizadas

Frontend:

- **Angular 19:** Framework principal para construir aplicaciones SPA. Se configuran rutas, guardias de autenticación (AuthGuard) y componentes modularizados.
- **TypeScript:** Tipado estático para mayor robustez en el código.
- **Vite:** Herramienta de bundling utilizada por Angular CLI para desarrollo y producción.
- **HTML5 y CSS3:** Maquetación y estilos de los componentes.
- **Bootstrap:** Para estilos rápidos y responsivos.
- **Axios:** Cliente HTTP para realizar peticiones al backend.
- **QuickChart (Api):** Se utiliza en el componente de detallesmedicamento para visualizar gráficos de los pedidos y ventas de un medicamento.

Backend:

- **Node.js (v18.x) y Express:** Entorno de ejecución y framework web para construir la API.
- **TypeScript:** Para tipado y mejor mantenimiento del código.
- **Mongoose:** ODM (Object Data Modeling) para conectarse a MongoDB y definir esquemas de datos.
- **MongoDB (v8.x):** Base de datos NoSQL documental.
- **dotenv:** Para gestionar variables de entorno (URI de MongoDB, puerto, claves, etc.).
- **cors:** Middleware para habilitar CORS y permitir peticiones desde el dominio del frontend.
- **Nodemon:** Herramienta para reinicio automático del servidor en desarrollo.

Integración y pruebas:

- **Postman:** Herramienta para probar endpoints del backend.
- **Herramientas de depuración de Angular (Augury, Developer Tools):** Para inspeccionar componentes y estado.

Seguridad:

- **AuthGuard:** Guardias de ruta en Angular que comprueban si el usuario está autenticado y tiene permisos de rol.
- **Variables de entorno:** Uso de archivo .env para no exponer credenciales en el repositorio.

Despliegue y hosting:

- **Backend (Nodejs + Express):** Desde la carpeta del proyecto nodejs → *npm install* y tras eso *npm run dev*.
- **Frontend (Angular 19):** Desde la carpeta del proyecto angular → *npm install* seguido de *ng serve -o.*
- **MongoDB local:** Abre una terminal y pon mongod, asegurate de que en el .env está bien configurado.

3.3. Análisis de usuarios (perfiles de usuario)

El sistema contempla dos perfiles de usuario principales:

1. **Gerente:**
    - **Funcionalidades permitidas:**
        - Crear, editar y borrar medicamentos.
        - Crear, modificar y eliminar pedidos a proveedores.
        - Visualizar y modificar los turnos de todos los empleados.
        - Acceder a todas las rutas de la aplicación.
    - **Restricciones:** Ninguna.
2. **Empleado/Invitado:**
    - **Funcionalidades permitidas:**
        - Visualizar listado de medicamentos y exportar el listado en pdf.
        - Consultar detalles de un medicamento.
        - Realizar ventas y recibir los pedidos.
        - Listar los pedidos.
        - Consultar sus propios datos de perfil y modificar información personal (nombre, apellidos, correo, contraseña).
    - **Restricciones:**
        - No puede crear medicamentos.
        - No puede gestionar pedidos.

El modelo de los usuarios dispone de un atributo rol que almacena “Gerente” o “Empleado”. El guard valida en cada ruta si el usuario autenticado cumple con los roles indicados en la configuración de rutas.

3.4. Definición de requisitos funcionales y no funcionales

Requisitos funcionales:

1. **Autenticación de usuarios:**
    - El sistema debe permitir el inicio de sesión mediante correo y contraseña.
    - El sistema debe distinguir entre roles de usuario (Gerente/Empleado) y permitir o denegar accesos.
2. **Gestión de medicamentos:**
    - Visualizar lista completa de medicamentos.
    - Ver detalles de un único medicamento (nombre, ubicación, cantidad, tipo, descripción, ingredientes, pedidos y ventas).
    - Crear nuevo medicamento: especificar nombre, ubicación, tipo, descripción e ingredientes.
    - Editar cantidad de medicamento al realizar un pedido (introduciendo el numero del pedido) o una venta (introduciendo el numero de la receta y la cantidad solicitada).
    - Borrar un medicamento.
3. **Gestión de pedidos:**
    - Visualizar lista de pedidos registrados.
    - Crear nuevo pedido: indicar nombre de medicamento, cantidad pedida y el proveedor.
    - Editar cantidad pedida de un pedido existente (si no ha sido recibido y aplicado).
    - Eliminar pedido (si no ha sido recibido y aplicado).
4. **Gestión de perfil personal:**
    - Visualizar y editar datos de usuario propio (nombre, apellidos, correo y contraseña).
5. **Visualización de calendario de turnos (solo Gerente):**
    - Mostrar calendario mensual para cada empleado, con diferente color para cada turno.
    - Permitir navegar entre meses.

Requisitos no funcionales:

1. **Usabilidad:**
    - Interfaz intuitiva y clara.
    - Validación de formularios (campos obligatorios, formatos mínimos).
2. **Seguridad:**
    - Control de acceso basado en roles.
    - Protección contra inyecciones XSS y CSRF (Angular y Express proporcionan mecanismos básicos).
3. **Rendimiento:**
    - Respuestas del backend en menos de 200 ms en peticiones habituales.
    - Carga inicial de la SPA en menos de 2 segundos en conexión estándar.
4. **Escalabilidad:**
    - Arquitectura desacoplada que permita replicar el backend o usar un balanceador de carga.
    - Uso de MongoDB, que facilita escalado horizontal.
5. **Mantenibilidad:**
    - Código organizado en módulos y servicios, con nomenclatura clara.
    - Comentarios breves en puntos críticos de la lógica de negocio.
    - Uso de TypeScript para tipado y mayor robustez.
6. **Disponibilidad:**
    - Despliegue continuo mediante nodemon en entornos de desarrollo.
7. **Compatibilidad y Responsividad:**
    - La interfaz se adapta a distintos tamaños de pantalla (desktop, tablet, móvil) gracias al uso de Bootstrap y diseño responsivo.

3.5. Estructura de navegación

A continuación se detalla el mapa de rutas internas de la aplicación Angular:

Rutas públicas

```
 /                → Muestra la página de bienvenida.

 /login           → Punto de acceso para autenticación.
```

Rutas privadas (requieren autenticación)

```
 /principal        → Panel principal con la navegación a los
                     distintos módulos.

 /medicamentos     → Muestra el listado de todos los medicamentos.

 /medicamentos/:id → Muestra la información detallada de un
                     medicamento seleccionado.

 /nuevo-medicamento (solo Gerente)
                   → Permite al gerente crear un nuevo medicamento.

 /pedidos          → Muestra el listado de pedidos existentes.

 /nuevo-pedido    (solo Gerente)
                   → Permite al gerente realizar un nuevo pedido.

 /perfil           → Permite ver y editar los datos del perfil del
                     usuario.

 /horario          → Presenta el calendario con los turnos de los
                     empleados.
```

Guardias de ruta y permisos:

- El servicio `AuthGuard` comprueba si el usuario está autenticado y si su rol coincide con el rol requerido por la ruta.
- Rutas accesibles a cualquier usuario autenticado: `/medicamentos`, `/medicamentos/:id`, `/perfil`, `/pedidos`
- Rutas restringidas solo a "Gerente": `/nuevo-medicamento`, `/nuevo-pedido`

El flujo de navegación es el siguiente:

1. El usuario accede a la página de inicio y procede a identificarse.
2. Tras autenticarse correctamente, se redirige a `/principal`.
3. En `/principal`, el menú de cabecera muestra opciones según el rol:
    - **Gerente:** Medicamentos | Pedidos | Horario | Perfil | Cerrar Sesión
    - **Empleado:** Medicamentos | Pedidos | Perfil | Cerrar Sesión
4. Cada enlace lleva al componente correspondiente, y `AuthGuard` valida el acceso en segundo plano.

3.6. Organización de la lógica de negocio

Backend (Node.js + Express)

- **Modelos (Mongoose):**
    - **`Usuario`**: esquema con campos `_id`, nombre, apellidos, correo, password y rol.
    - **`Empleado`**: esquema con `_id`, nombre, apellidos, correo, teléfono y un array `calendario` de 12 elementos (uno por mes); cada mes almacena un mapa `days` que mapea el día del mes al turno (“M”, “T” o “L”).
    - **`Medicamento`**: esquema con `_id`, nombre, cantidad (stock), ubicación, tipo, descripción, ingredientes, `stock_minimo`, y dos arrays de subdocumentos `pedidos` y `ventas`.
    - **`Proveedor`**: esquema con `_id`, código (único), nombre, dirección, teléfono, email y un array `pedidos` (subdocumentos con número de pedido, nombre de medicamento y cantidad).
    - **`Pedido`**: esquema independiente con `_id`, número de pedido (único), nombre de medicamento, cantidad pedida y código de proveedor.
- **Controladores principales:**
    - **`auth.controller.ts`:**
        - `login`: busca usuario por correo, compara contraseña (en texto plano en este proyecto de ejemplo), devuelve datos sin el password.
        - `updateProfile`: permite al usuario autenticado actualizar nombre, apellidos, correo y contraseña.
    - **`medicamento.controller.ts`:**
        - `getMedicamentos`, `getMedicamentoById`, `addMedicamento`, `updateMedicamento`, `deleteMedicamento`.
        - Al actualizar stock tras un pedido o venta, ajusta el campo `cantidad` y añade o elimina subdocumentos en `pedidos` o `ventas`.

- **`proveedor.controller.ts`:**
    - `getProveedores`, `addPedidoToProveedor`, `removePedidoFromProveedor`, `updatePedidoInProveedor`.
    - Al añadir un pedido al proveedor, llama internamente al controlador de pedidos para crear el documento global y luego empuja el subdocumento en el array de ese proveedor. Hace lo mismo al borrar y al actualizar.
- **`pedido.controller.ts`:**
    - `getPedidos`, `getPedidoByNumero`, `addPedido`, `updatePedido`, `deletePedido`.
    - Al crear un pedido añade un subdocumento en `proveedores.pedidos`.
    - Al actualizar o eliminar un pedido, ajusta correspondientemente el stock y los subdocumentos relacionados.
    - No permite actualizar ni borrar un pedido si ya ha sido recibido.
- **`empleado.controller.ts`:**
    - `getEmpleados`, `getEmpleadoById`, `updateEmpleadoCalendario`.
    - Al actualizar permite cambiar el turno del empleado elegido.

- **APIs externas:**
    - **QuickChart.io:** Es una API externa que sirve para mostrar datos en forma de gráficos. Se conecta a ella a través de una url, dándole los datos para realizar el gráfico desde su servidor y los devuelve como imagen.

3.7. Modelo de datos simplificado

1. Usuario (`usuarios`)

```
{
  _id: ObjectId,
  nombre: String,           // Obligatorio
  apellidos: String,        // Obligatorio
  correo: String,           // Obligatorio, Único
  password: String,         // Obligatorio
  rol: String               // "Gerente" o "Empleado"
}
```

Relaciones: Ninguna explícita; solo controla autenticación y permisos.

---

2. Empleado (`empleados`)

```
{
  _id: ObjectId,
  nombre: String,           // Obligatorio
  apellidos: String,        // Obligatorio
  telefono: String,         // Obligatorio
  correo: String,           // Obligatorio, Único
  calendario: [
    { days: Map<String, String> }  // Array de 12 objetos, uno por
                                      mes. days mapea "1"→"M"/"T"/"L"
  ]
}
```

Relaciones: Ninguna.

---

3. **Medicamento** (`medicamentos`)

```
{
  _id: ObjectId,
  nombre: String,           // Obligatorio
  cantidad: Number,         // Obligatorio (stock actual)
  ubicacion: String,        // Opcional (ej. "Estantería A1")
  tipo: String,             // Obligatorio (ej. "Analgésico")
  descripcion: String,      // Opcional
  ingredientes: [String],   // Opcional
  stock_minimo: Number,     // (por defecto 15)
  pedidos: [
    { numero_pedido: String, cantidad_pedida: Number }
  ],
  ventas: [
    { numero_receta: String, cantidad_vendida: Number }
  ]
}
```

Relaciones:

- El array `pedidos` contiene subdocumentos que referencian el número de pedido y la cantidad relacionada.

---

4. **Proveedor** (`proveedores`)

```
{
  _id: ObjectId,
  codigo: String,           // Obligatorio, Único
  nombre: String,           // Obligatorio
  direccion: String,        // Opcional
  telefono: String,         // Opcional
  email: String,            // Opcional
  pedidos: [
    { numero_pedido: String, nombre_medicamento: String,
      cantidad_pedida: Number }
  ]
}
```

Relaciones:

- Cada subdocumento de `pedidos` comparte el mismo `numero_pedido` que el documento global en la colección `pedidos`.

---

5. Pedido (`pedidos`)

```
{
  _id: ObjectId,
  numero_pedido: String,         // Obligatorio, Único
  nombre_medicamento: String,    // Obligatorio
  cantidad_pedida: Number,       // Obligatorio
  codigo_proveedor: String       // Obligatorio (coincide con
                                     Proveedor.codigo)
}
```

Relaciones:

- `codigo_proveedor` apunta al `Proveedor.codigo`.

---

4. Conclusiones

Resultados obtenidos:

- Se ha desarrollado con éxito una aplicación web completa que satisface los objetivos planteados:
    - Autenticación de usuarios con control de acceso basado en roles.
    - Gestión integral de medicamentos: registro, actualización de stock, detalle y eliminación.
    - Control de pedidos a proveedores: creación, edición y eliminación.
    - Planificación de turnos para empleados mediante calendario interactivo.
    - Edición de perfil de usuario.
    - Interfaz responsiva y navegación fluida.
- La separación de responsabilidades entre frontend y backend ha permitido un desarrollo modular y escalable.
- La implementación de MongoDB como base de datos NoSQL ha facilitado el modelado de subdocumentos (pedidos dentro de medicamentos y proveedores, calendario dentro de empleados).

Retos encontrados y soluciones implementadas:

1. **Gestión de subdocumentos en MongoDB:**
    - Dificultad inicial al actualizar mapas anidados para el calendario de empleados.
    - Solución: comprobar si `days` era instancia de `Map`; en caso contrario, convertirlo de objeto JSON a `Map` antes de modificar.
2. **Validaciones y manejo de errores:**
    - Implementación de unicidad en campos (`numero_pedido`, `correo`).
    - Captura de errores 500 (servidor) y 400/401 (peticiones inválidas, credenciales incorrectas).
3. **Configuración de CORS entre frontend y backend:**
    - Habilitación de middleware `cors` en Express con origen permitido (dominio de Angular).
4. **Implementación de AuthGuard en Angular:**
    - Configuración de rutas protegidas por rol (`Gerente`) para evitar accesos no autorizados.
5. **Diseño de interfaz responsiva:**
    - Utilización de Bootstrap para asegurar adaptabilidad en distintos dispositivos.
6. Implementación de la API externa QuickChart:
    - Al tratar de implementar la API, trataba de usarla de forma local como una librería parecida llamada chart.js y al pasarle los datos no hacia nada.
    - La solución fue con la inteligencia artificial y la información que proporcionan ver como llamar al servidor y mandarle los datos para mostrarlos.

Aprendizajes y mejoras futuras:

- **Aprendizajes:**
    - Integración completa de stack MEAN (Angular + Node.js + Express + MongoDB).
    - Modelado de datos en MongoDB con subdocumentos y mapas.
    - Uso de TypeScript en backend y frontend para mayor seguridad de tipos.
    - Patrón MVC en el contexto de API RESTful.
    - Protección de rutas y autorización basada en roles en Angular.
    - Llamadas a APIs externas.
- **Mejoras futuras:**
    - **Cifrado de contraseñas:** Implementar bcrypt.
    - **Tokens JWT:** Generar tokens en login y usar autenticación Bearer.
    - **Generar PDFs de pedidos:** Para poder hacer un seguimiento y dejar constancia.
    - **Panel estadístico:** Usar QuickChart de nuevo en los pedidos para hacer un seguimiento de que tipo de medicamentos hay mas ventas.
    - **Aplicación móvil:** Crear una aplicación móvil más básica para hacer un seguimiento de los pedidos para los gerentes y que los empleados puedan comprobar de forma más sencilla sus horarios.
    - **Mostrar/Editar/Eliminar proveedores y empleados:** De esta forma se podría dar más independencia y libertad de gestionar por completo la aplicación a la empresa que contrata esta.

---

5. Bibliografía y fuentes de información

- **Documentación oficial de Angular**. (https://angular.io/)
- **Documentación oficial de Node.js**. (https://nodejs.org/)
- **Express.js Guide**. (https://expressjs.com/)
- **Mongoose Documentation**. (https://mongoosejs.com/)
- **MongoDB Manual**. (https://docs.mongodb.com/manual/)
- **TypeScript Handbook**. (https://www.typescriptlang.org/docs/)
- **Bootstrap Documentation**. (https://getbootstrap.com.com/docs)
- **Axios GitHub Repository**. (https://github.com/axios/axios)
- **QuickChart Documentation**. (https://quickchart.io/documentation/)
- **Reglamento General de Protección de Datos (GDPR)**.
- **Licencia MIT**. (https://opensource.org/licenses/MIT)

---

6. Anexos

6.1. Manual de usuario

1. Acceso al sistema

1. Abrir navegador y acceder a la url facilitada.
2. En la pantalla de inicio, ir a “Iniciar Sesión”.
3. Introducir correo y contraseña.
4. Si las credenciales son correctas, se redirige a `/principal`.

---

2. Menú principal

- **Gerente:** Medicamentos | Pedidos | Horario | Perfil | Cerrar Sesión.
- **Empleado:** Medicamentos | Pedidos | Perfil | Cerrar Sesión.

---

3. Gestión de medicamentos

- **Listar Medicamentos:**
  En el menú, hacer clic en “Medicamentos” y se mostrarán todos los medicamentos paginados con los datos más relevantes.
- **Ver Detalles:**
  Hacer clic en el icono “Detalles” de un medicamento. Se muestra la información completa del medicamento: nombre, ubicación, cantidad, tipo, descripción, ingredientes, gráfico de pedidos y gráfico de ventas.
- **Generar PDF:**

  Al pulsar el botón generará un PDF con todos lo medicamentos que se muestren en ese momento a pesar de la paginación.
- **Crear Medicamento (Gerente):**
  Hacer clic en “Nuevo Medicamento”. Rellenar formulario con:
  - Nombre (obligatorio)
  - Cantidad inicial (obligatorio)
  - Tipo (obligatorio)
  - Descripción (opcional)
  - Ubicación (opcional)
  - Ingredientes (campo de texto o tags; opcional)
- **Editar Stock:**
  Al darle al botón “Actualizar” saltará una pantalla donde introducir un número, ha de ser de 4 dígitos (receta) o de 6 (numero de pedido).

  En caso de ser 4 dígitos saldrá un campo para introducir la cantidad a restar indicando una venta, por otro lado si es de 6 solo has de poner el numero del pedido y se añadirá la cantidad al stock.
- **Eliminar Medicamento (Gerente):**
  En la tabla, hacer clic en icono “Borrar”. Confirmar el mensaje saliente en caso de estar seguro.

---

4. Gestión de pedidos

- **Listar Pedidos:**
  En el menú, hacer clic en “Pedidos”. Aparece una tabla con los pedidos.
- **Crear Pedido (Gerente):**
  Hacer clic en “Nuevo Pedido”. Rellenar:
  - Número de pedido (único, obligatorio)
  - Nombre de medicamento (debe existir)
  - Cantidad pedida (obligatorio)
  - Código de proveedor (debe existir)
    Hacer clic en “Realizar Pedido”.
  - Disminuye el stock en `medicamentos`.
  - Crea documento en `pedidos`.
  - Añade subdocumento en `proveedores.pedidos`.
- **Editar Pedido (Gerente):**
  En la tabla de pedidos, hacer clic en “Editar”. Modificar cantidad y “Actualizar”.
  - Ajusta la cantidad en proveedores.pedidos.
  - En caso de ya haber recibido e introducido el pedido en el medicamento no dejará modificarlo.
- **Eliminar Pedido (Gerente):**
  Hacer clic en “Borrar” junto a un pedido. Confirmar.
  - Elimina documento en `pedidos`.
  - Elimina subdocumento en `proveedores`.
  - En caso de ya haber recibido e introducido el pedido en el medicamento no dejará eliminarlo.

---

5. Horario de turnos

- **Consulta de Turnos (Gerente):**
    - En el menú, clic en “Horario”. Se muestran los empleados una paginación por mes.
        - Al clicar en un día de un mes de un empleado saldrá una ventana para asignar el turno:
            - Azul: Turno Mañana
            - Amarillo: Turno Tarde
            - Verde: Libre

.

---

6. Perfil de usuario

- Arriba a la derecha, hacer clic en el correo, se mostrará un menu con botones, clicar en “Configuración”. Se mostrarán los datos del usuario rellenos:
    - Nombre
    - Apellidos
    - Correo
    - Contraseña (vacío, en caso de no querer cambiarla)
    - Rol
- Modificar lo deseado y hacer clic en “Guardar”. Se actualizan los datos en la colección `usuarios`.

6.2. Guía de instalación, configuración y despliegue

Nota: Es imprescindible utilizar **Angular CLI 19.x** para el frontend y **Node.js 18.x** o superior para el backend.

Requisitos previos

- **Node.js (v18.x o superior)**
- **Angular CLI (v19.x)**
- **MongoDB (local o en MongoDB Atlas)**
- **Git (para clonar el repositorio)**
- **Visual Studio Code (todo usando VSC)**

Paso 1: Clonar repositorio

1. Abrir el apartado de git y clonar el repositorio con la url:

   `https://github.com/TFGs-2DAW/PROYECTO-TFG-ANGULAR-NODE.JS.git`

2. Dentro de la carpeta principal encontrarás tres carpetas:
   - `farmaceutica-nodejs` (backend)
   - `farmaceutica-angular` (frontend)
   - `bd-mongo` (archivos .json para la base de datos)

Paso 2: Base de datos (MongoDB)

1. Abre una nueva terminal y pon mongod
2. Una vez iniciado mongo, abrir MongoDBCompass y crear una base de datos llamada farmaceutica
3. Una vez creada la base de datos crear las colecciones medicamentos, pedidos, usuarios, proveedores y empleados.
4. Tras crear las colecciones importar los datos de los archivos .json de la carpeta de bd-mongo a cada colección.
5. Tras realizar todo eso la base de datos estaría configurada.

Paso 3: Configurar y levantar backend

1. Navegar a la carpeta del backend:

   ```
   cd .\farmaceutica-nodejs\
   ```

2. Verificar archivo `.env` con contenido:

   ```
   Asegurarse de que este bien configurado.
   ```

3. Instalar dependencias:

   ```
   npm install
   ```

4. Levantar servidor en modo desarrollo (con Nodemon):

   ```
   npm run dev
   ```

   - El servidor iniciará en `http://localhost:3000`.
   - Debería aparecer en consola: “Conectado a MongoDB” y “Servidor corriendo en puerto 3000”.

Paso 4: Configurar y levantar frontend

1. Abrir nueva terminal y navegar a la carpeta del frontend:

   ```
   cd .\farmaceutica-angular\
   ```

2. Instalar dependencias con **npm**:

   ```
   npm install
   ```

3. Levantar Angular en modo desarrollo:

   ```
   ng serve -o
   ```

   - Se abrirá en `http://localhost:4200`.
   - Ir al login y acceder con uno de los usuarios de la base de datos.
   - Proceder a usar el manual de usuario.
