**Documentación del Proyecto: SpecuLabTeam**

**Sistema de Gestión de Propiedades SpecuLabTeam**

****

**Versión:** 1.0.0

**Autor(es):** Alejandro, Jorge, Tomás y Marco Salas

**Fecha de Creación:** 09 de septiembre de 2025

**Repositorio de GitHub:** https://github.com/alexgn12/SpecuLabTeam.git

**Índice**

1. Resumen del Proyecto
2. Tecnologías y Metodología

2.1. Metodología de Trabajo

2.2. Contribuciones al Equipo

1. Documentación del Backend (.NET API)

3.1. Arquitectura General

3.2. Estructura de Carpetas del Backend

3.3. Entidades y Modelos de Datos (DTOs)

3.4. Controladores (API Endpoints)

3.5. Servicios y Lógica de Negocio

3.6. Persistencia de Datos (Entity Framework Core)

3.7. Autenticación y Seguridad

3.8. Documentación de API (Swagger)

1. Documentación del Frontend (Angular)

4.1. Arquitectura y Tecnologías

4.2. Flujos de Usuario y Componentes Clave

1. Guía de Instalación y Despliegue

5.1. Instalación Local

5.2. Despliegue (CI/CD)

1. Anexos y Recursos Adicionales

**1. Resumen del Proyecto**

**SpecuLabTeam** es una plataforma integral diseñada para optimizar la gestión y el seguimiento de propiedades inmobiliarias y las solicitudes asociadas a ellas. Este sistema proporciona una solución robusta para administrar edificios, apartamentos, transacciones y el ciclo de vida de las solicitudes, ofreciendo una experiencia de usuario intuitiva y eficiente para los gestores de propiedades.

El proyecto utiliza una arquitectura moderna basada en una API monolítica bien estructurada, con un backend desarrollado en .NET 8 y un frontend interactivo en Angular. La integración continua y el despliegue se gestionan a través de GitHub Actions, asegurando un ciclo de desarrollo ágil y eficiente.

**2. Tecnologías y Metodología**

**2.1. Metodología de Trabajo**

El desarrollo de SpecuLabTeam se llevó a cabo utilizando la metodología ágil Scrum. La gestión del proyecto, la planificación de sprints y el seguimiento del backlog se realizaron mediante Azure DevOps. Esta aproximación fomentó la colaboración, permitió una rápida adaptación a los cambios y aseguró una entrega constante de valor.

**2.2. Contribuciones al Equipo**

• [Nombre Miembro 1]: Participó en el desarrollo tanto del backend en .NET 8 como del frontend en Angular, colaborando en el diseño de la API, la implementación de controladores y la creación de interfaces de usuario.

• [Nombre Miembro 2]: Contribuyó al desarrollo integral del sistema, trabajando en la lógica de negocio del backend y en la integración de servicios con el frontend.

• [Nombre Miembro 3]: Encargado de la implementación de funcionalidades clave en backend y frontend, así como en la optimización de la experiencia de usuario y la documentación técnica.

• [Nombre Miembro 4]: Participó activamente en la configuración de CI/CD, el desarrollo de componentes reutilizables y la integración entre backend y frontend.

**3. Documentación del Backend (.NET API)**

El backend de **SpecuLabTeam** es una API RESTful desarrollada con **ASP.NET Core**. Es el cerebro del sistema, manejando la lógica de negocio, la persistencia de datos y exponiendo los servicios para el frontend.

**3.1. Arquitectura General**

El backend sigue un patrón de **arquitectura en capas**, separando claramente las responsabilidades:

* **Capa de Presentación:** Controladores API que gestionan las peticiones HTTP.
* **Capa de Lógica de Negocio:** Servicios que implementan la lógica del negocio.
* **Capa de Acceso a Datos:** Repositorios que interactúan directamente con la base de datos.
* **Capa de Dominio:** Entidades que representan los objetos de negocio.

+---------------------+

| Frontend (Angular)|

+---------------------+

|

v (HTTP/REST API)

+---------------------+

| Backend (.NET 8 API)|

+---------------------+

|

v (ORM/EF Core)

+---------------------+

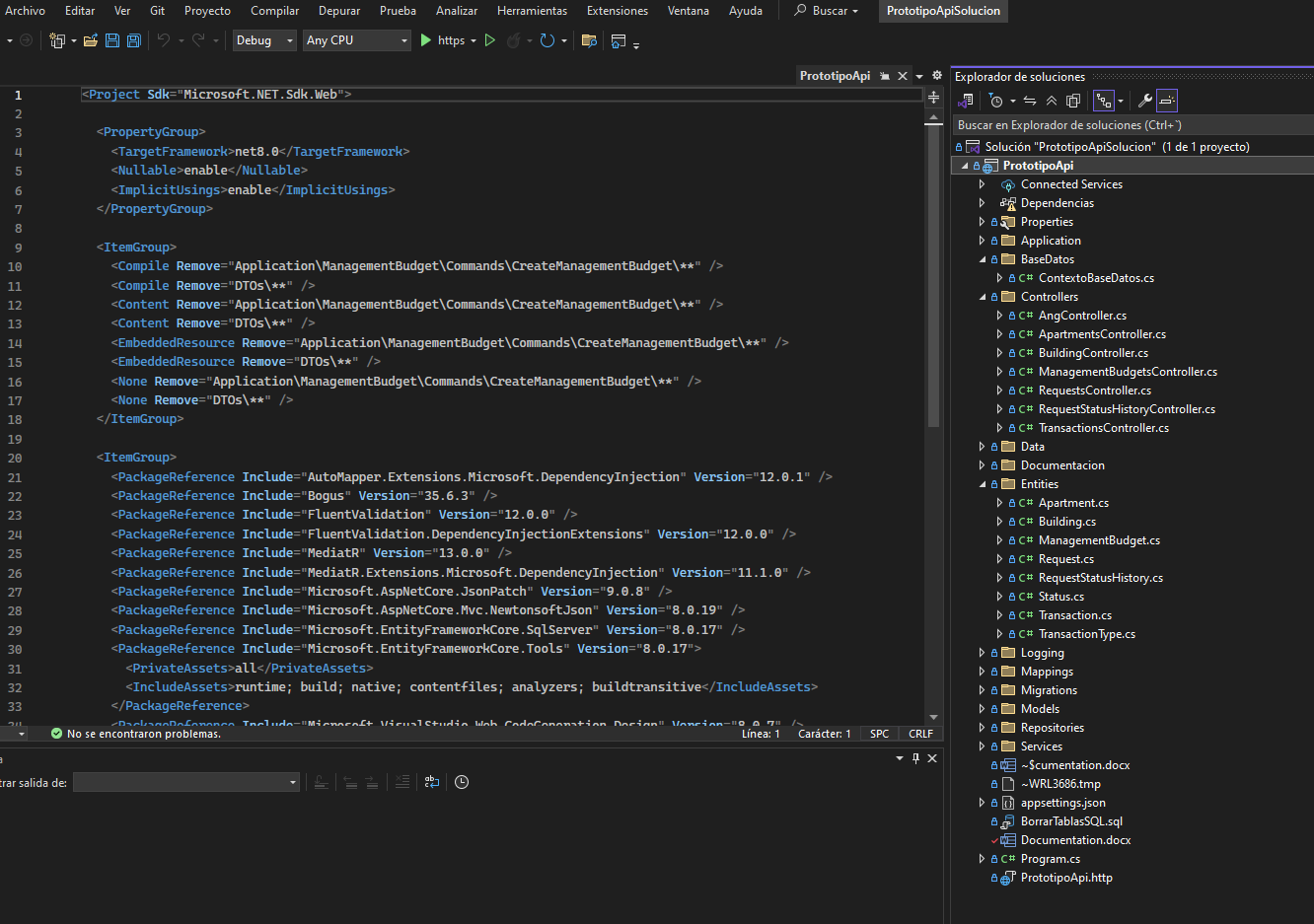
| Base de Datos |

+---------------------+

**3.2. Estructura de Carpetas del Backend**

El proyecto **PrototipoApi** está organizado de la siguiente manera:

* **Application**: Contiene la lógica de negocio principal, DTOs y mapeos.
* **BaseDatos**: Contexto de Entity Framework Core y migraciones.
* **Controllers**: Los controladores de la API que exponen los *endpoints*.
* **Data**: Clases para inicialización de datos (seeders) y ejemplos.
* **Documentacion**: Archivos de documentación interna (PRD, etc.).
* **Entities**: Definición de los modelos de base de datos.
* **Logging**: Implementación de la gestión de logs.
* **Mappings**: Clases de mapeo entre entidades y DTOs (ej. AutoMapper).
* **Migrations**: Archivos generados por Entity Framework Core para la gestión del esquema de la base de datos.
* **Models**: DTOs (Data Transfer Objects) para las peticiones y respuestas de la API.
* **Repositories**: Interfaces e implementaciones para el acceso a datos.
* **Services**: Lógica de negocio específica y operaciones de alto nivel.

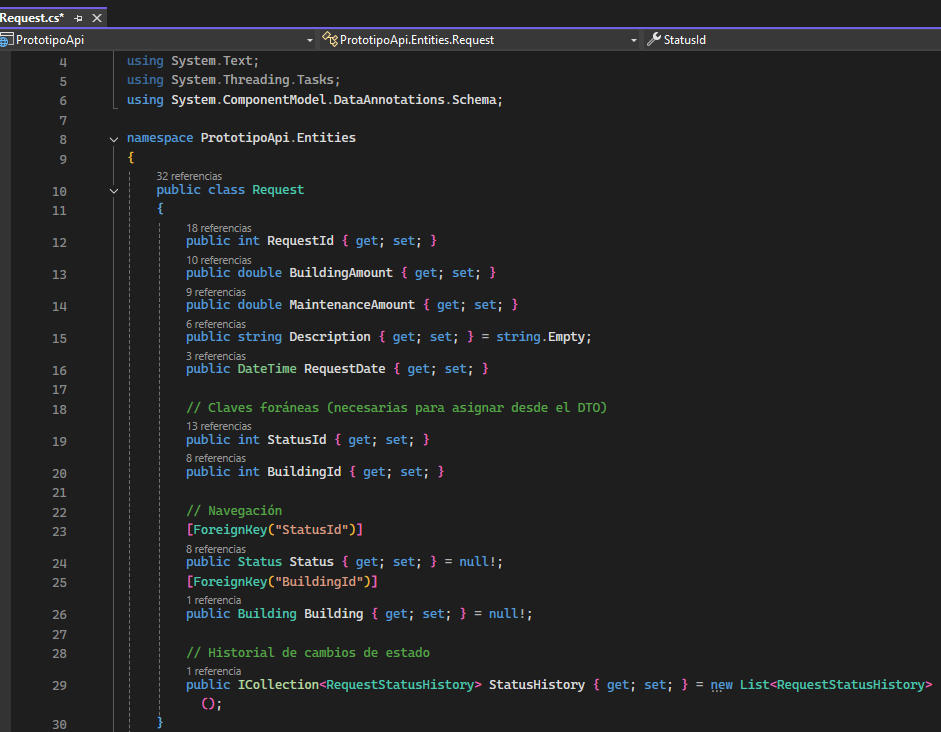


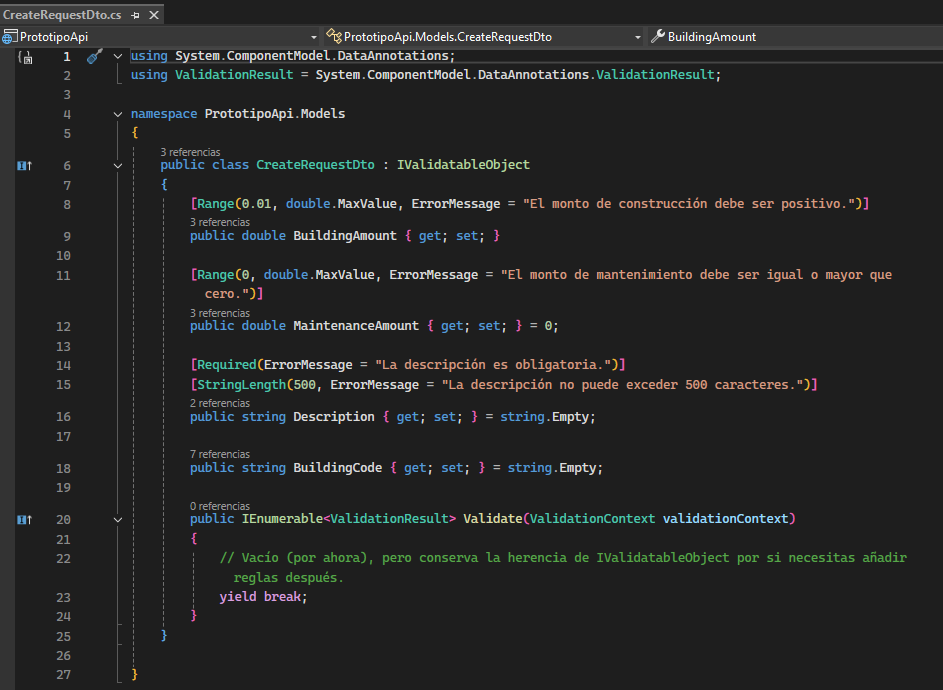
Captura 1: Estructura del proyecto PrototipoApi en el Explorador de soluciones.

**3.3. Entidades y Modelos de Datos (DTOs)**

Las **Entidades** (Entities) representan la estructura de los objetos en la base de datos. Los **DTOs** (Models) son versiones simplificadas o adaptadas de estas entidades, utilizadas para el envío y recepción de datos a través de la API.

* **Entities ejemplos:** Apartment.cs, Building.cs, Request.cs, Transaction.cs.
* **Models (DTOs) ejemplos:** CreateRequestDto.cs, UpdateRequestDto.cs, TransactionDto.cs.

.

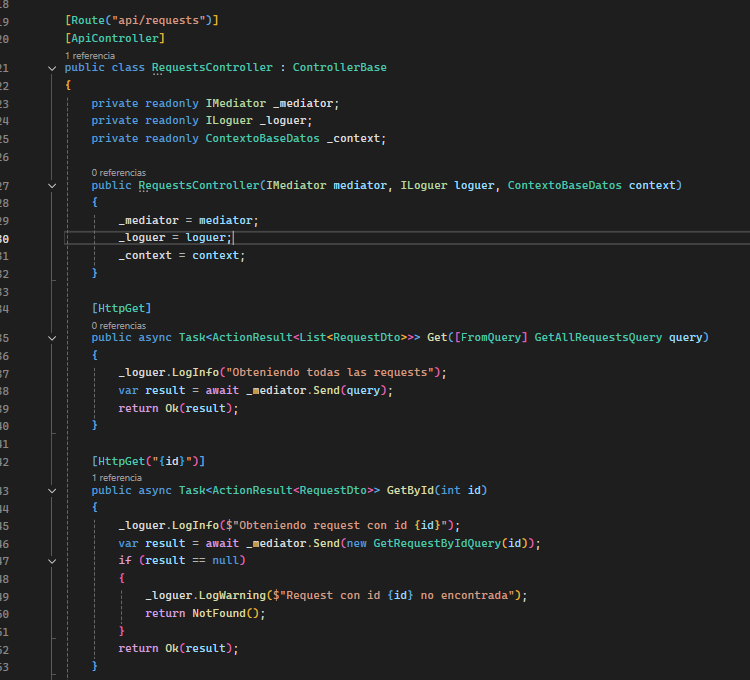


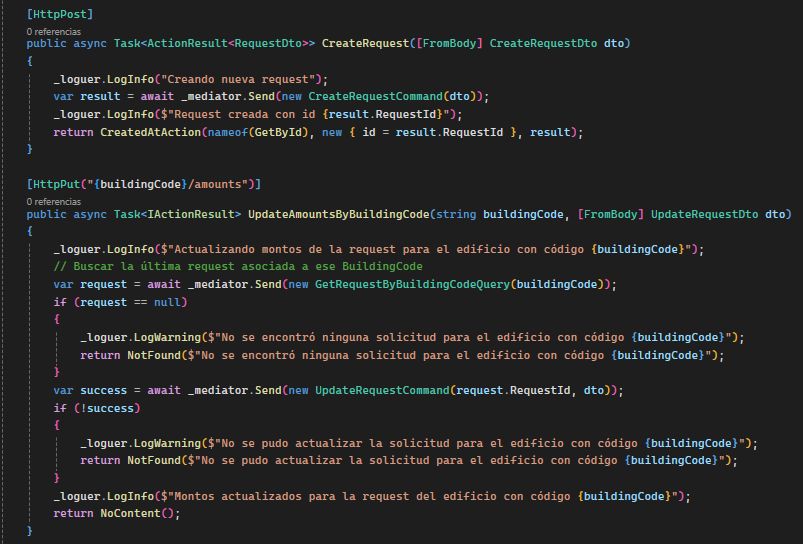
Captura 2: Ejemplo de la entidad Request y su DTO asociado CreateRequestDto.

**3.4. Controladores (API Endpoints)**

Los controladores son la interfaz del backend. Cada controlador agrupa un conjunto de *endpoints* relacionados.

* **AngController.cs**: [Descripción de su funcionalidad, si es específico de Angular.]
* **ApartmentsController.cs**: Gestiona las operaciones CRUD para apartamentos.
* **BuildingController.cs**: Gestiona las operaciones CRUD para edificios.
* **RequestsController.cs**: Administra el ciclo de vida de las solicitudes (creación, actualización de estado, consulta).
* **TransactionsController.cs**: Maneja las transacciones financieras asociadas a las propiedades.





Captura 3: Extracto de código del RequestsController, mostrando endpoints de ejemplo.

**Ejemplo de Endpoint: GET /api/requests**

* **Descripción:** Recupera una lista paginada de solicitudes con posibles filtros.
* **Método:** GET
* **Parámetros de Query (Opcional):** page=1, pageSize=10, status=Pendiente
* **Respuesta Exitosa (Código 200 OK):**

JSON

[

{ "id": 1, "description": "Fuga en el baño", "status": "Pendiente", "buildingId": 101, "dateCreated": "2024-05-20T10:00:00Z" },

{ "id": 2, "description": "Revisión de electricidad", "status": "Aprobada", "buildingId": 102, "dateCreated": "2024-05-18T14:30:00Z" }

]

* **Respuesta de Error (Código 500 Internal Server Error):**

JSON

{ "error": "No se pudieron recuperar las solicitudes." }

**3.5. Servicios y Lógica de Negocio**

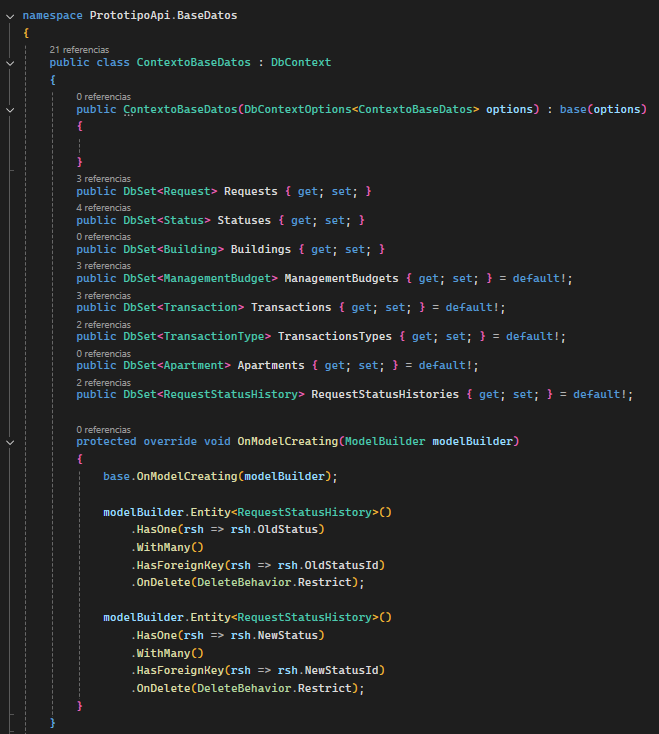
Los servicios (Services) contienen la lógica de negocio principal. Separan la lógica de la interacción directa con los controladores y los repositorios, facilitando la reutilización y el testeo.

* **ExternalApartmentService.cs / ExternalBuildingService.cs**: [Descripción de si estos servicios interactúan con APIs externas o son servicios de negocio específicos de apartamentos/edificios.]
* **Otros servicios:** [Si tienes otros servicios, descríbelos aquí.]

**3.6. Persistencia de Datos (Entity Framework Core)**

Utilizamos **Entity Framework Core** como ORM para la interacción con la base de datos **[Ejemplo: SQL Server]**.

* **ContextoBaseDatos.cs**: Define el contexto de la base de datos y los DbSet para cada entidad.
* **Migraciones:** Se gestionan automáticamente al iniciar la aplicación (MigrateDatabase()).



Captura 4: Definición del Contexto de la Base de Datos con DbSet para las entidades.

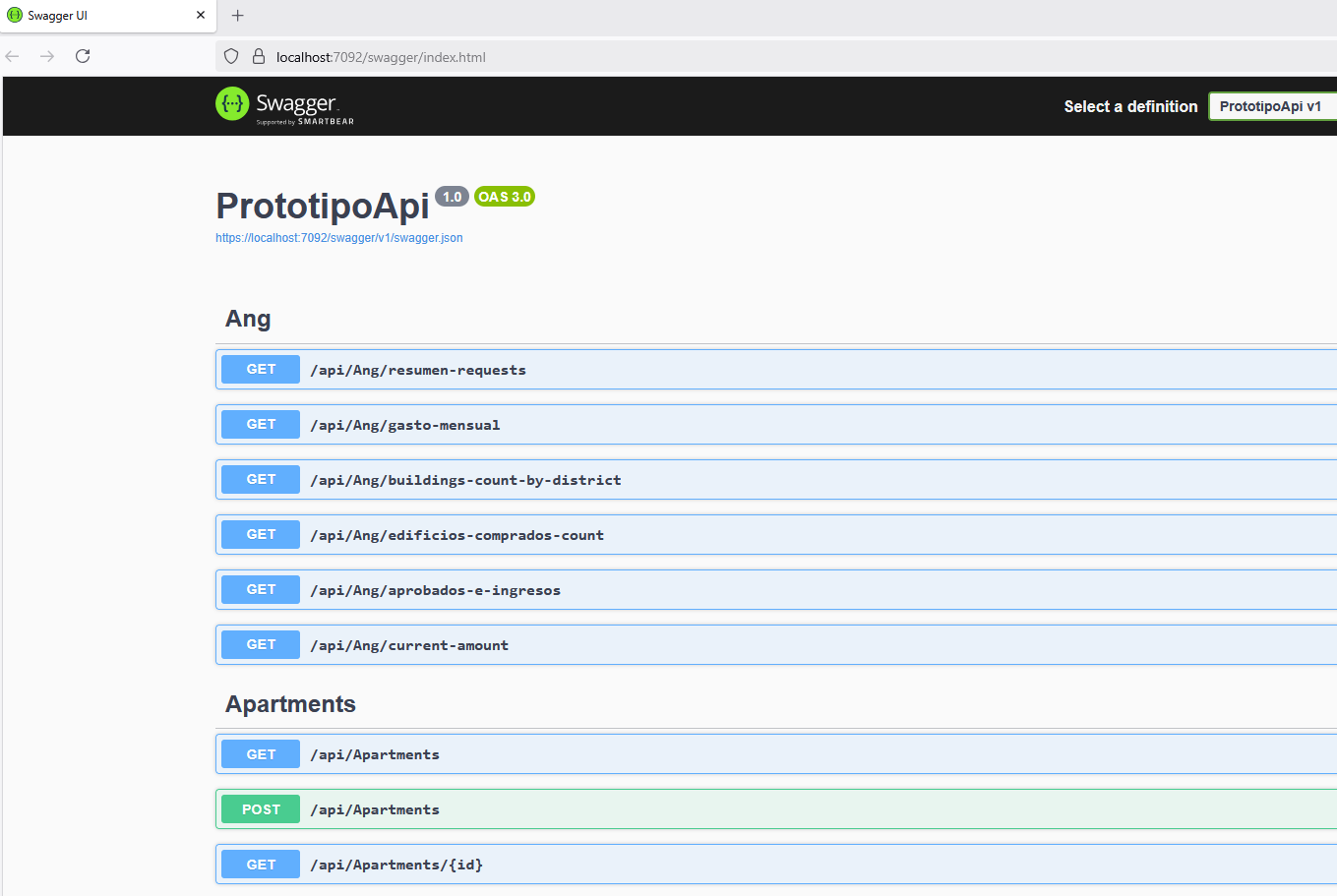
**3.7. Autenticación y Seguridad**

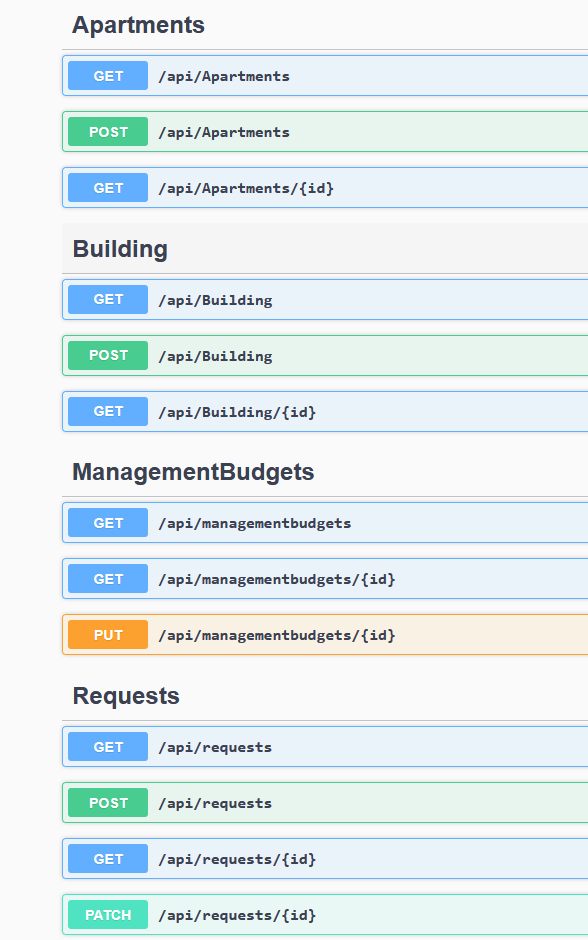
[Describe cómo se maneja la autenticación (ej. JWT, Identity Core) y la autorización (roles/permisos) en tu API. Si aún no está implementado, menciónalo como futura mejora.]

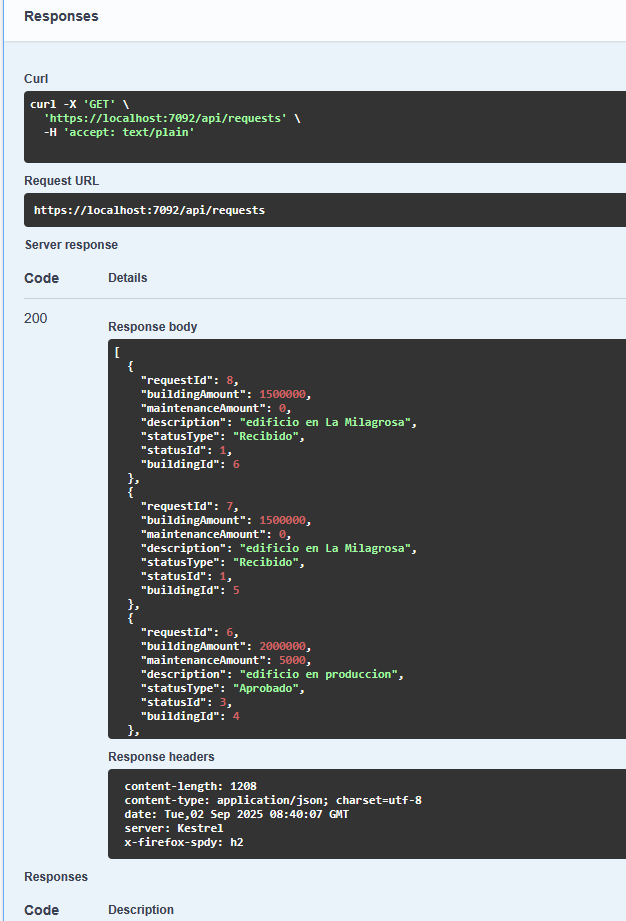
**3.8. Documentación de API (Swagger)**

La API está auto-documentada usando **Swagger/OpenAPI**, lo que permite explorar, entender y probar los *endpoints* de manera interactiva.

[Captura de pantalla sugerida: Una captura de pantalla de la interfaz de Swagger UI de tu proyecto, mostrando algunos endpoints desplegados.]







Captura 5: Interfaz de Swagger UI mostrando la documentación interactiva de la API.

**4. Documentación del Frontend (Angular)**

El frontend de **SpecuLabTeam** es una aplicación web interactiva desarrollada con **Angular**. Proporciona la interfaz de usuario para que los gestores de propiedades interactúen con el sistema.

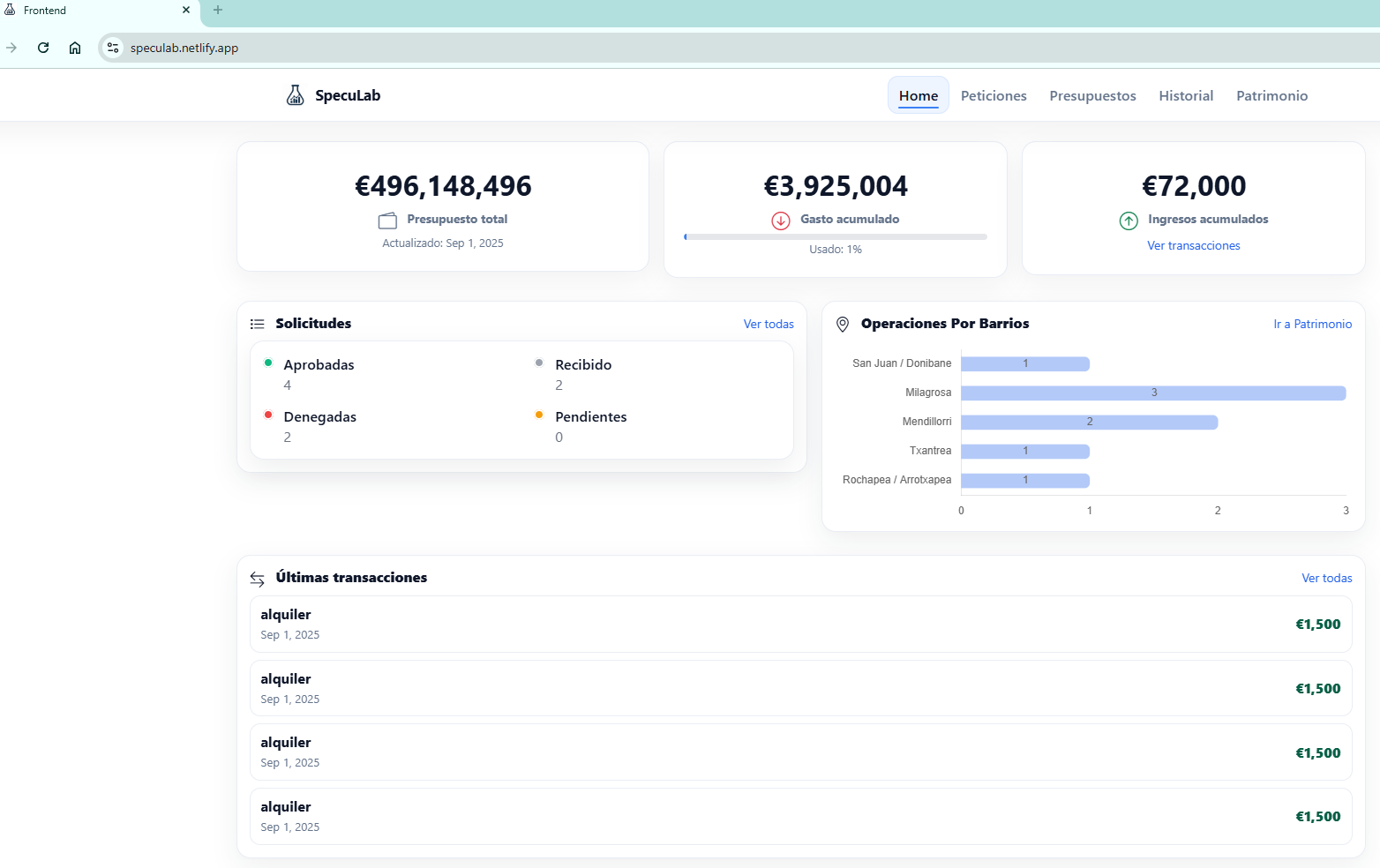
**4.1. Arquitectura y Tecnologías**

* **Framework:** Angular [Versión, si la conoces]
* **Lenguaje:** TypeScript
* **Gestión de Estado:** [Ejemplo: RxJS para la gestión asíncrona, o si usas Ngrx/Akita, menciónalo.]
* **Librerías de UI:** [Ejemplo: Angular Material, Bootstrap, CSS personalizado.]
* **Routing:** Modularizado y protegido mediante [Ejemplo: AuthGuard] para las rutas que requieren autenticación.

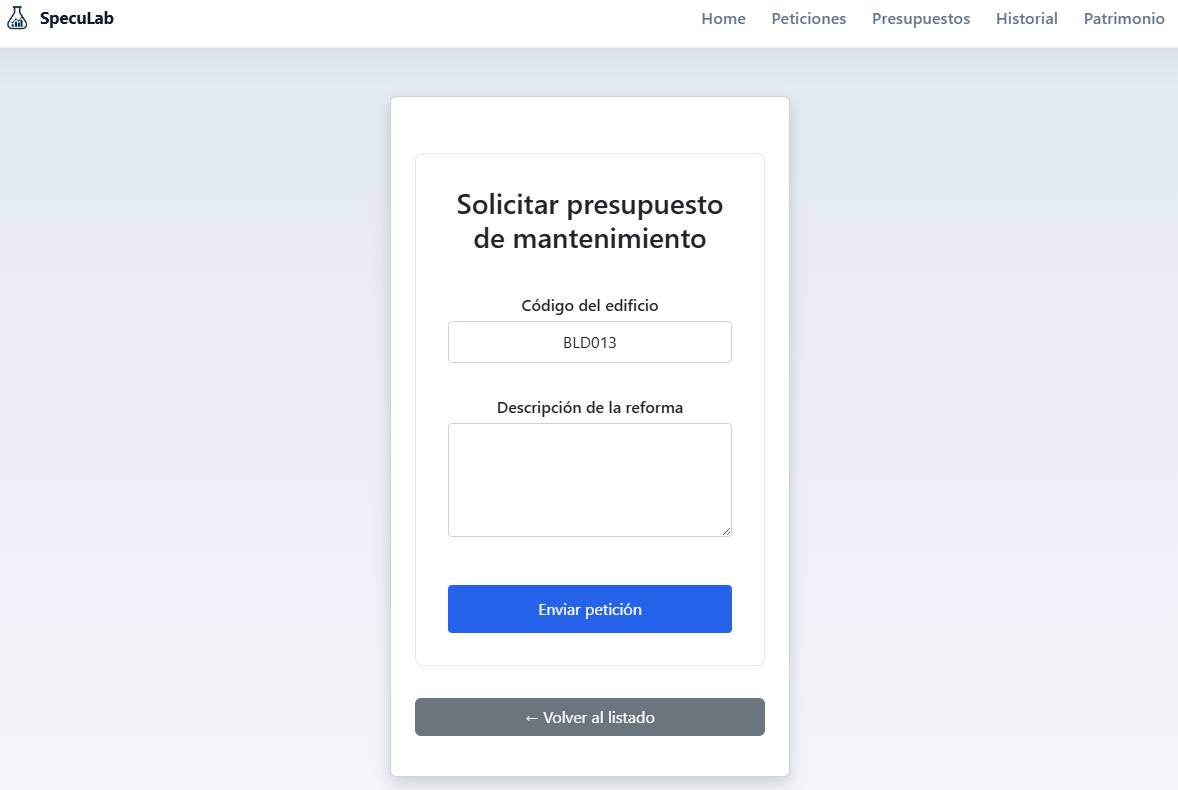
**4.2. Flujos de Usuario y Componentes Clave**

Aquí se detallan los principales flujos de interacción del usuario con la aplicación, acompañados de capturas de pantalla para una mejor comprensión.

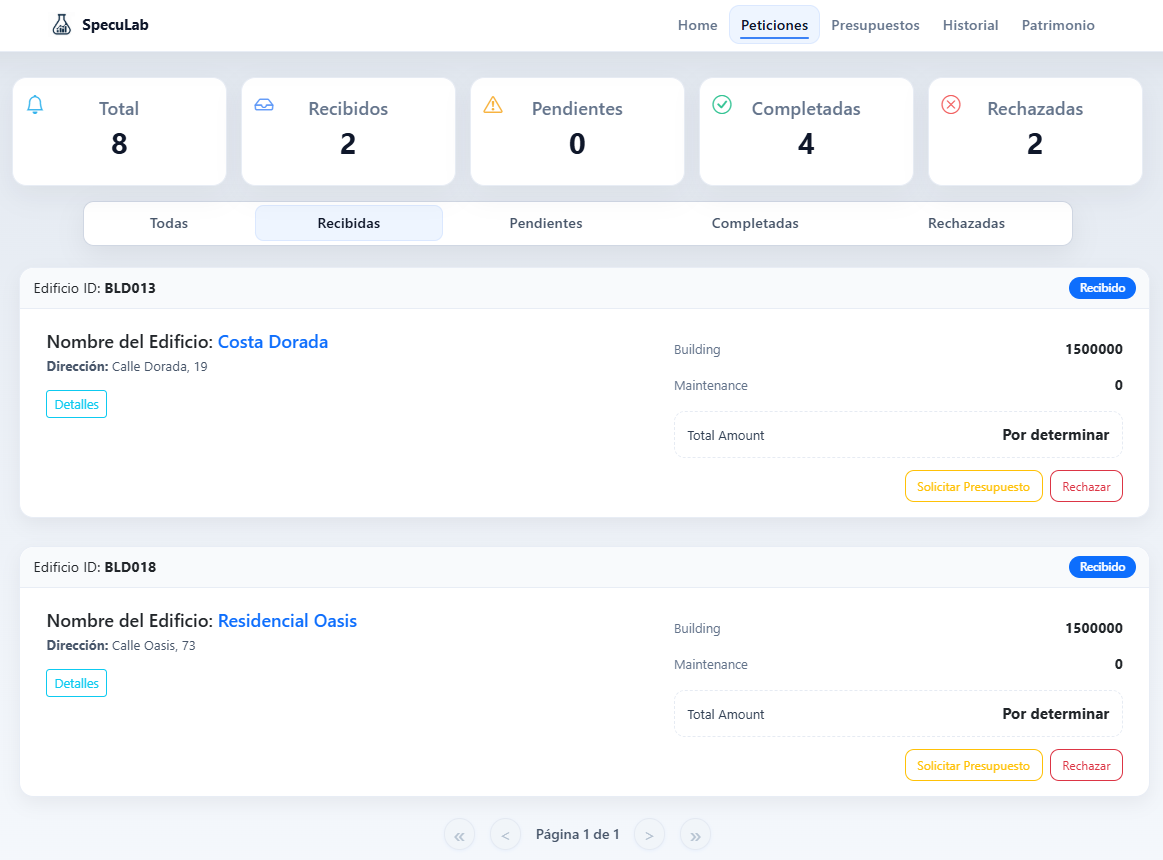
* **4.2.1. Dashboard Principal de Propiedades/Solicitudes**
  + **Descripción:** Tras iniciar sesión, el usuario accede al dashboard principal, que ofrece una visión general de [Describe lo que se ve en el dashboard: "las propiedades activas, solicitudes pendientes o un resumen de las métricas clave."].



* + *Captura 6: Dashboard Principal de SpecuLabTeam.*
* **4.2.2. Gestión de Solicitudes (Creación, Visualización, Actualización)**
  + **Descripción:** Los usuarios pueden crear nuevas solicitudes, ver los detalles de las existentes y actualizar su estado.



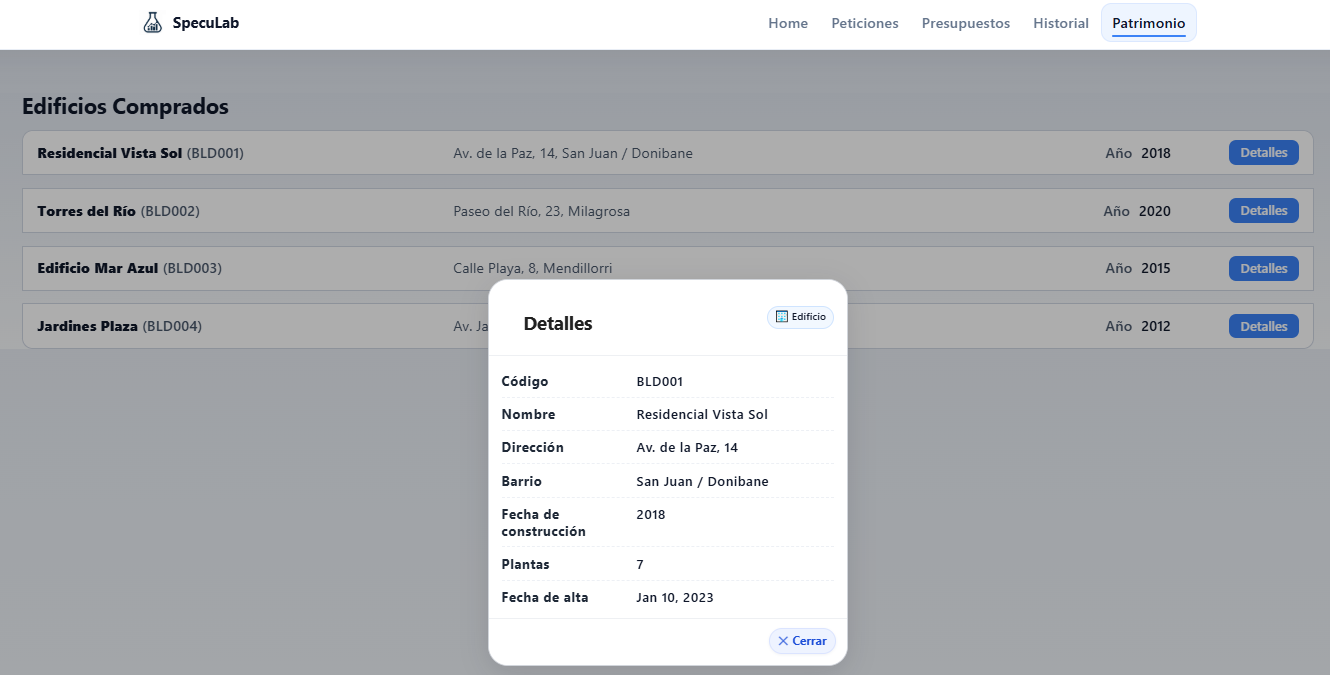
* + *Captura 7: Formulario para crear una nueva solicitud.*



* + *Captura 8: Listado de Solicitudes con opciones de filtro.*
* **4.2.3. Historial de Estados de Solicitudes**
  + **Descripción:** Para cada solicitud, se puede acceder a un historial detallado de los cambios de estado (Ej. Recibido -> Pendiente -> Aprobado).



* + *Captura 9: Historial de estados de una solicitud específica.*
* **4.2.4. Gestión de Edificios/Apartamentos**
  + **Descripción:** Módulos para añadir, editar o visualizar información detallada sobre edificios y apartamentos.



* + *Captura 10: Listado y detalles de Edificios/Apartamentos.*

**5. Guía de Instalación y Despliegue**

**5.1. Instalación Local para Desarrollo**

Para configurar y ejecutar el proyecto **SpecuLabTeam** en un entorno de desarrollo local, sigue estos pasos:

1. **Clonar el Repositorio de GitHub:**

Bash

git clone https://github.com/alexgn12/SpecuLabTeam.git

cd SpecuLabTeam

1. **Configuración del Backend (.NET API):**
   * **Requisitos:** .NET SDK [Versión, ej. 6.0 o 8.0], SQL Server [Versión, si aplica].
   * Navega a la carpeta PrototipoApiSolucion/PrototipoApi.
   * Restaura las dependencias: dotnet restore
   * Configura la cadena de conexión a tu base de datos en appsettings.json.
   * Aplica las migraciones de la base de datos: dotnet ef database update
   * Ejecuta el backend: dotnet run (o inicia desde Visual Studio).
   * La API estará disponible en https://localhost:[puerto] (revisar launchSettings.json).
2. **Configuración del Frontend (Angular):**
   * **Requisitos:** Node.js [Versión], npm/yarn, Angular CLI.
   * Navega a la carpeta del frontend (si está en el mismo repo, ej. SpecuLabTeam/frontend).
   * Instala las dependencias: npm install (o yarn install)
   * Configura la URL de la API del backend en el archivo de entorno (src/environments/environment.ts).
   * Inicia el servidor de desarrollo: ng serve --open
   * El frontend estará disponible en http://localhost:4200.

**5.2. Despliegue (CI/CD)**

[Describe tu estrategia de Integración Continua y Despliegue Continuo. Si usas GitHub Actions, explica brevemente los *workflows*.]

* **GitHub Actions para el Frontend:**
  + **Descripción:** Un *workflow* automatizado construye la aplicación Angular y la despliega en **[Ejemplo: Azure Static Web Apps, Netlify, Vercel]** cada vez que se detectan cambios en la rama main (o master).
  + [**Captura de pantalla sugerida:** Un extracto de tu archivo .github/workflows/frontend-deploy.yml mostrando los pasos clave.]
  + *Captura 11: Workflow de GitHub Actions para el despliegue del Frontend.*
* **GitHub Actions para el Backend:**
  + **Descripción:** Otro *workflow* se encarga de construir la API de .NET y desplegarla en **[Ejemplo: Azure App Service, un servidor VPS]** tras cada actualización en la rama principal.
  + [**Captura de pantalla sugerida:** Un extracto de tu archivo .github/workflows/backend-deploy.yml mostrando los pasos clave.]
  + *Captura 12: Workflow de GitHub Actions para el despliegue del Backend.*

**6. Anexos y Recursos Adicionales**

* **Definiciones PRD:** Puedes incluir los contenidos de tus archivos Definicion\_PRD\_ejemplos.txt y PRD\_historial\_angular.txt aquí o hacer referencia a ellos.
* **Diagramas de Secuencia:** Si tienes diagramas de secuencia para flujos clave (ej. cómo una solicitud es procesada de principio a fin).
* **Notas de Diseño:** Cualquier decisión de diseño importante o consideraciones especiales.