**Instrucciones claras para configurar y ejecutar el proyecto**

**1. Clonar el repositorio**

* Clona el repositorio en tu equipo local utilizando Git.

git clone https://github.com/Andco07/TaskRegistry.git

**2. Configurar la base de datos**

* Crea una base de datos PostgreSQL.
* Actualiza el archivo appsettings.json en el proyecto principal para incluir la cadena de conexión de la base de datos:
* "ConnectionStrings": {
* "DefaultConnection": "Host=localhost;Database=nombre\_base\_datos;Username=usuario;Password=contraseña"
* },
* "Jwt": {
* "Key": "clave\_secreta\_para\_jwt",
* "Issuer": "TaskRegistry”,
* "Audience": "TaskRegistryUsers"

}

**3. Restaurar las dependencias**

* Abre una terminal en la carpeta del proyecto y ejecuta:

dotnet restore

**4. Ejecutar las migraciones**

* Aplica las migraciones a la base de datos para crear las tablas necesarias:

dotnet ef database update

**5. Ejecución de los scripts**

* Si existen scripts SQL adicionales necesarios para inicializar datos o configurar el sistema, ejecútalos utilizando un cliente de base de datos como pgAdmin o psql.
* Asegúrate de ejecutar los scripts en el orden indicado en el archivo README.md o la documentación del proyecto.

**6. Ejecutar el proyecto**

* Inicia la aplicación con:

dotnet run

* La API estará disponible en http://localhost:5000 y Swagger en http://localhost:5000/swagger.

**7. Probar las funcionalidades**

* Accede a Swagger para explorar los endpoints y probar las funcionalidades.
* Usa un cliente como Postman o Insomnia para enviar solicitudes a la API.

**Explicación breve de decisiones técnicas**

**1. Framework y Tecnologías**

* **ASP.NET Core**: Se eligió por su rendimiento, flexibilidad y capacidad para desarrollar APIs RESTful modernas.
* **Entity Framework Core**: Permite un acceso sencillo a la base de datos y facilita la gestión de migraciones y consultas con LINQ.
* **PostgreSQL**: Base de datos robusta y escalable, ideal para aplicaciones de tamaño medio y grande.

**2. Autenticación y Seguridad**

* **JWT (JSON Web Tokens)**: Los tokens JWT se usan para autenticar y autorizar a los usuarios de forma segura y eficiente.
* **Roles y Claims**: Se implementaron para restringir el acceso a ciertos endpoints, asegurando que solo los usuarios autorizados puedan realizar acciones específicas.

**3. Documentación con Swagger**

* Swagger facilita la exploración y prueba de los endpoints para desarrolladores.
* Los comentarios XML aseguran que los endpoints estén documentados con claridad.

**4. Arquitectura**

* **Separación de responsabilidades**:
  + Controladores manejan las solicitudes HTTP.
  + Modelos representan los datos.
  + Servicios (como el servicio JWT) encapsulan la lógica reutilizable.
* Esta estructura mejora el mantenimiento y la extensibilidad del código.

**5. Pruebas Unitarias**

* Se utilizaron pruebas unitarias con **xUnit** para verificar el correcto funcionamiento de los controladores y servicios.
* La base de datos en memoria asegura que las pruebas sean rápidas y no afecten la base de datos de producción.

**6. Interfaz de Usuario**

* La interfaz se diseñó con Bootstrap para proporcionar una experiencia de usuario limpia y funcional.
* Se priorizó la claridad y facilidad de navegación.

Estas decisiones técnicas aseguran un balance entre rendimiento, seguridad y facilidad de uso, tanto para desarrolladores como para usuarios finales.