DOCUMENTACIÓN DEL SISTEMA (DIGITAL SUPPORT)

Nombre del Proyecto: Digital Support (Sistema web de gestión de reportes y solicitudes de Aplicaciones)

Tipo de Aplicación y Rol Desempeñado: App Web Full Stack (Desarrollador Full Stack)

Desarrollado por: Fabrizio J. Cuno Barrios

Fecha de Desarrollo: 2025-2026

Tecnologías Aplicadas y Detalles Extra:

- FRONTEND: Angular CLI 20.3.3, Ts (Typescript), HTML, CSS, RxJS (Manejo de Observables), Angular Forms y Reactive Forms, Consumo de Api Rest (HttpClient)
- BACKEND: .NET Web API, C#, Entity Framework Core, Arquitectura RESTful, Serialización JSON, Documentación de endpoints con Swagger UI.
- BD: SQL SERVER, T-SQL, Diseño bajo Modelado Relacional
- Herramientas: Visual Studio, Visual Studio Code, SQL Server Management, Swagger UI, npm
- Arquitectura: Arquitectura en 3 Capas, Operaciones CRUD, Integración Cliente-Servidor

El sistema permite la creación y administración de usuarios con distintos roles (Administrador, Colaborador y Cliente), así como el registro, gestión y seguimiento de reportes e incidencias generados por los clientes sobre las aplicaciones brindadas por la organización. La plataforma centraliza la atención de solicitudes, permitiendo el control de estados, monitoreo de avances y notificaciones por correo electrónico para garantizar trazabilidad e integridad en la comunicación con el usuario.

Cree esta aplicación porque he visto que en las organizaciones no suelen contar con una plataforma centralizada para registrar, gestionar y dar seguimiento a incidencias reportadas por los clientes, lo que generaba pérdida de información, retrasos en la atención y falta de trazabilidad en la comunicación. Digital Support permite digitalizar este proceso, garantizando control, seguimiento y mejora en los tiempos de respuesta.

Objetivo General: Desarrollar un sistema web que permita gestionar de manera eficiente las solicitudes e incidencias reportadas por los clientes, optimizando la comunicación, seguimiento y control de atención.

- ❖ Funcionalidades por Rol:
  - ➢ Administrador:
    - Visualización global del avance de solicitudes.
    - Gestión de usuarios, roles y clientes (empresa y persona natural).
    - Monitoreo de desempeño y estadísticas de colaboradores.
    - Envío de notificaciones de avance al cliente.
  - ➢ Colaborador:
    - Atención de solicitudes registradas por los clientes.
    - Actualización del progreso de trabajo.

- Asignación colaborativa de tareas con coordinación interna.
- Envío de notificaciones de avance al cliente.

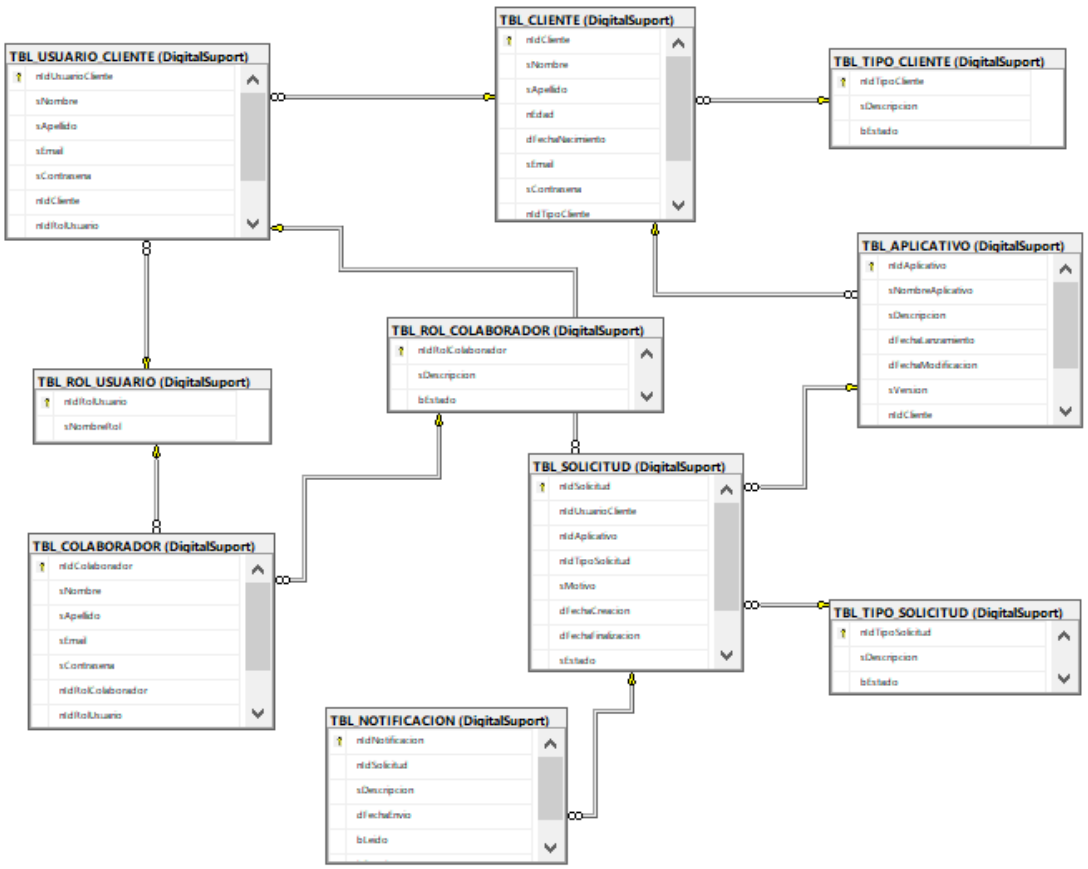
➢ Usuario Cliente:

- Registro de solicitudes de soporte sobre aplicaciones.
- Seguimiento del estado de atención.
- Recepción de notificaciones y comunicación del progreso.
- Consulta de colaboradores asignados.

BASE DE DATOS: La base de datos fue implementada en SQL Server, diseñada bajo un modelo relacional que garantiza la integridad de la información mediante el uso de claves primarias y foráneas. Se adjunta los script de inicialización, los cuales permiten generar la estructura completa del sistema. Ahí se encuentra:

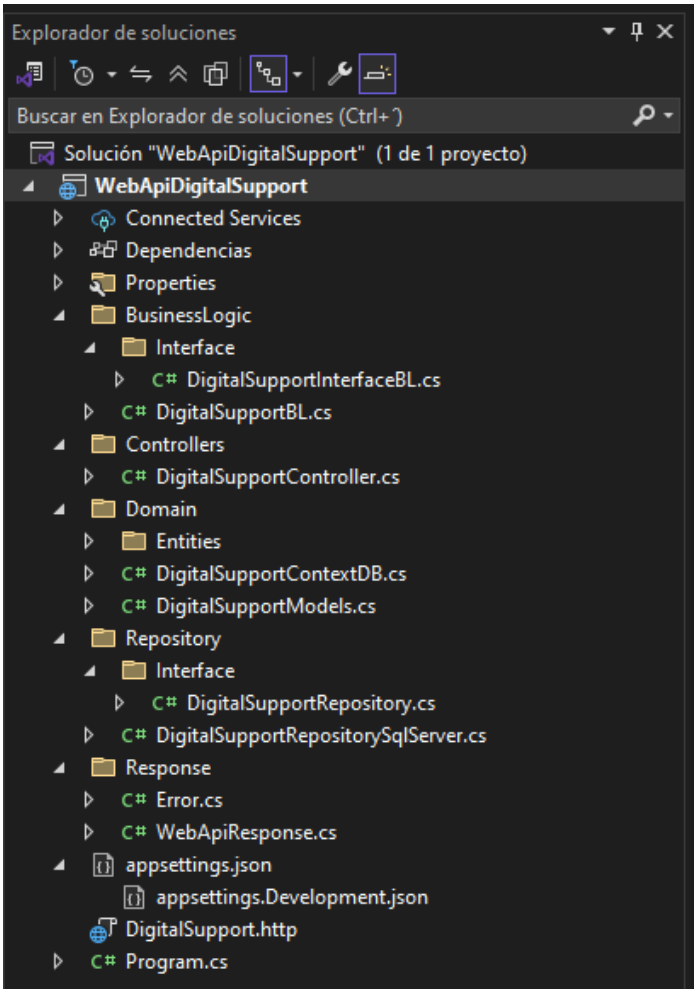
- Creación de tablas.
- Definición de Primary Keys y Foreign Keys.
- Relaciones entre entidades.
- Procedimientos almacenados.
- Funciones de apoyo a la lógica de negocio.

DIAGRAMA DE BD: El diagrama representa la estructura relacional del sistema y la interacción entre las entidades principales utilizadas para la gestión de usuarios, solicitudes, seguimiento y control operativo.



BACKEND: El backend fue desarrollado mediante .NET Web API, implementando una arquitectura en capas para separar responsabilidades y garantizar mantenibilidad, escalabilidad y control de la lógica de negocio.

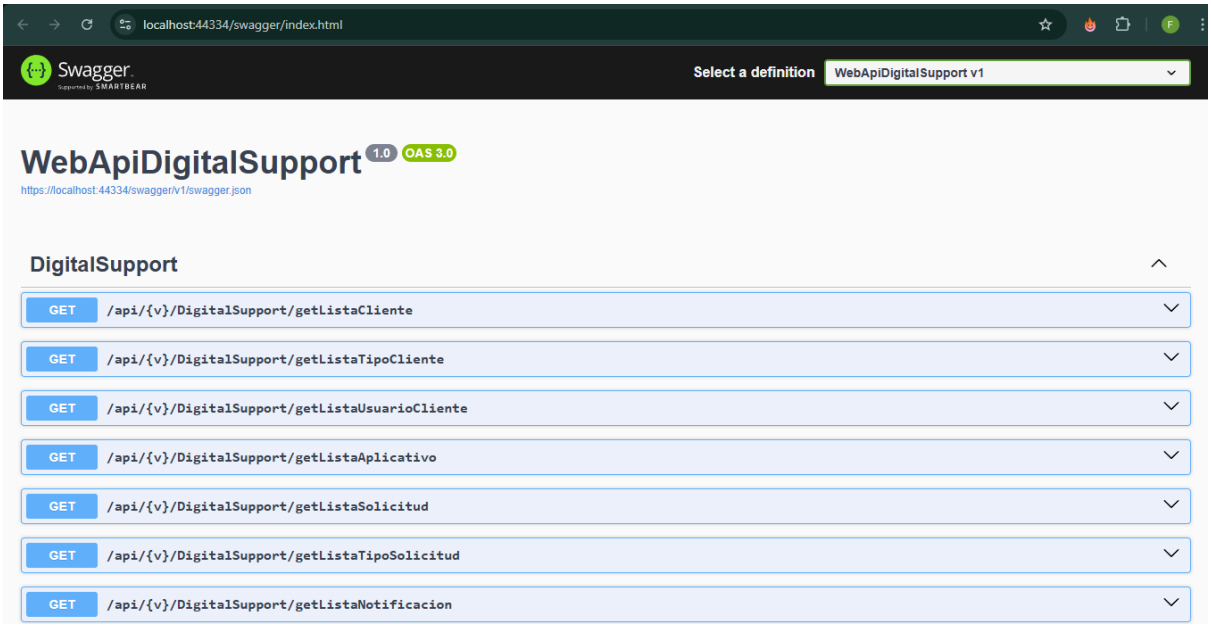
La API permite la comunicación entre el frontend Angular y la base de datos SQL Server mediante servicios REST que procesan solicitudes en formato JSON. El proyecto fue organizado en una arquitectura en capas, separando controladores, lógica de negocio, acceso a datos y modelos de dominio.



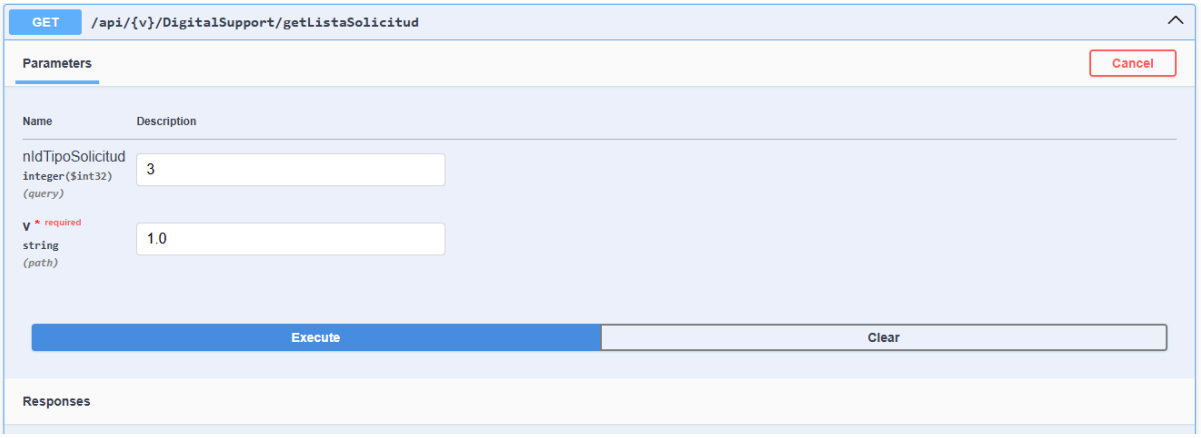
La configuración de la aplicación se gestionó mediante appsettings.json, donde se definieron la cadena de conexión a SQL Server y los parámetros del servicio de notificaciones por correo electrónico (SMTP). “Algunos datos, se deben reemplazar para hacer una prueba óptima”

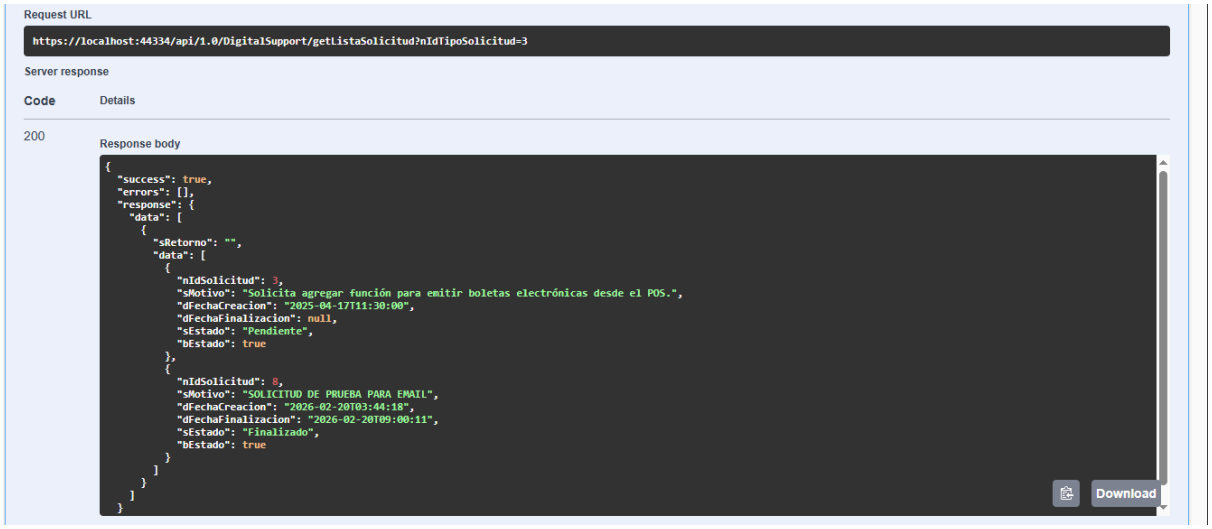
- ❖ Capa de Controladores (Controllers): Recibe las peticiones HTTP provenientes del frontend y gestiona las respuestas de la API.
- ❖ Capa de Lógica de Negocio (BusinessLogic): Contiene las reglas del sistema, validaciones y procesamiento de información antes de ser persistida.
- ❖ Capa de Acceso a Datos (Repositories): Gestiona la interacción con la base de datos mediante consultas y operaciones CRUD.
- ❖ Capa de Dominio (Domain): Define las entidades del sistema y el contexto de datos utilizado por Entity Framework.

La API fue documentada mediante Swagger UI, permitiendo visualizar y probar los endpoints desarrollados para la gestión de usuarios, solicitudes y seguimiento.



EJEMPLO DE PRUEBA DE ENDPOINT PUESTO A PRUEBA:



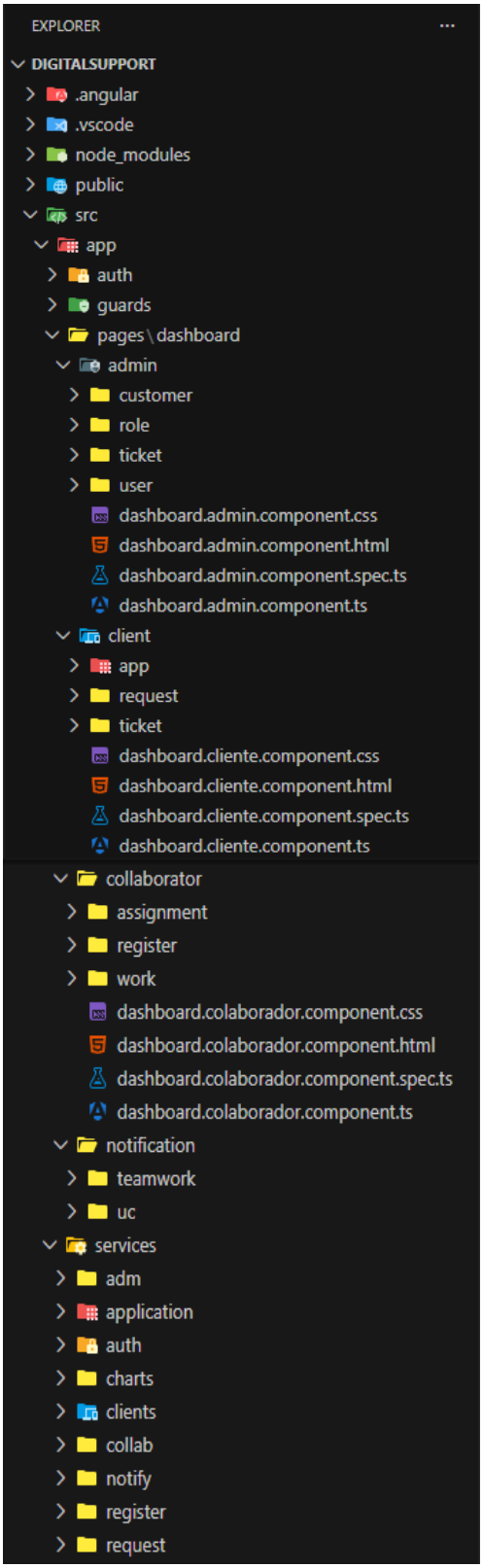


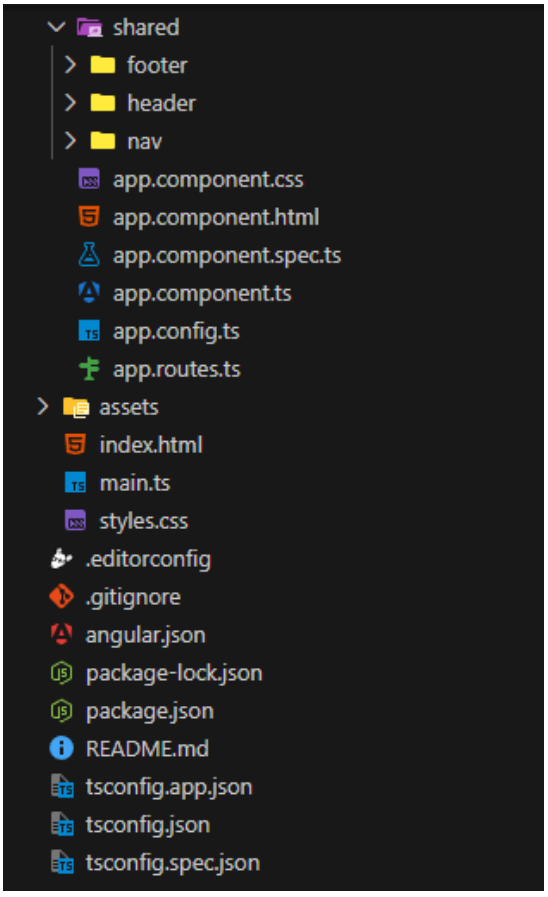
Integración de Notificaciones

El sistema implementa el envío de notificaciones por correo electrónico mediante configuración SMTP, permitiendo informar a los usuarios sobre el estado y avance de sus solicitudes.

FRONTEND: El frontend del sistema fue desarrollado utilizando Angular CLI 20, implementando una arquitectura modular basada en componentes y servicios para garantizar una correcta separación de responsabilidades y facilitar la escalabilidad del sistema. La aplicación consume los servicios REST del backend mediante HttpClient, gestionando la información de manera reactiva con RxJS y formularios validados mediante Reactive Forms.

El proyecto fue organizado en módulos funcionales, separando autenticación, gestión de páginas, servicios de consumo API, validaciones de acceso (guards) y componentes compartidos, permitiendo una estructura mantenible y reutilizable.





Los servicios Angular gestionan la comunicación con la API REST utilizando HttpClient. Se emplea programación reactiva mediante RxJS para el manejo de observables y flujo de datos asíncrono.

```typescript
@Injectable({
  providedIn: 'root'
})
export class AdminService {
    private admins$: BehaviorSubject<Administrador[]> = new BehaviorSubject<Administrador[]>([]);

    constructor(private http: HttpClient) {}

    getListaAdministrador(): Observable<IAdministradorResponse> {
        return this.http.get<IAdministradorResponse>(
            'https://localhost:44334/api/1.0/DigitalSupport/getListaAdministrador'
        ).pipe(
            tap((res) => {
            if (res.success && res.response?.data) {
                this.admins$.next(res.response.data);
            }
            }),
            catchError(this.handleError)
        );
    }
```

El frontend implementa un patrón basado en:

- Componentes: Encargados de la representación visual y la interacción con el usuario.
- Servicios: Responsables de consumir los endpoints de la API y centralizar la lógica de comunicación HTTP.
- Guards: Controlan el acceso a rutas según el rol del usuario autenticado.
- Modelos (Interfaces): Definen la estructura tipada de los datos recibidos desde el backend.
- Routing: Gestiona la navegación interna del sistema sin recarga de página.

Integración con Backend: La comunicación entre Angular y .NET se realiza mediante peticiones HTTP en formato JSON. Las respuestas son procesadas mediante observables, actualizando dinámicamente la interfaz del usuario.

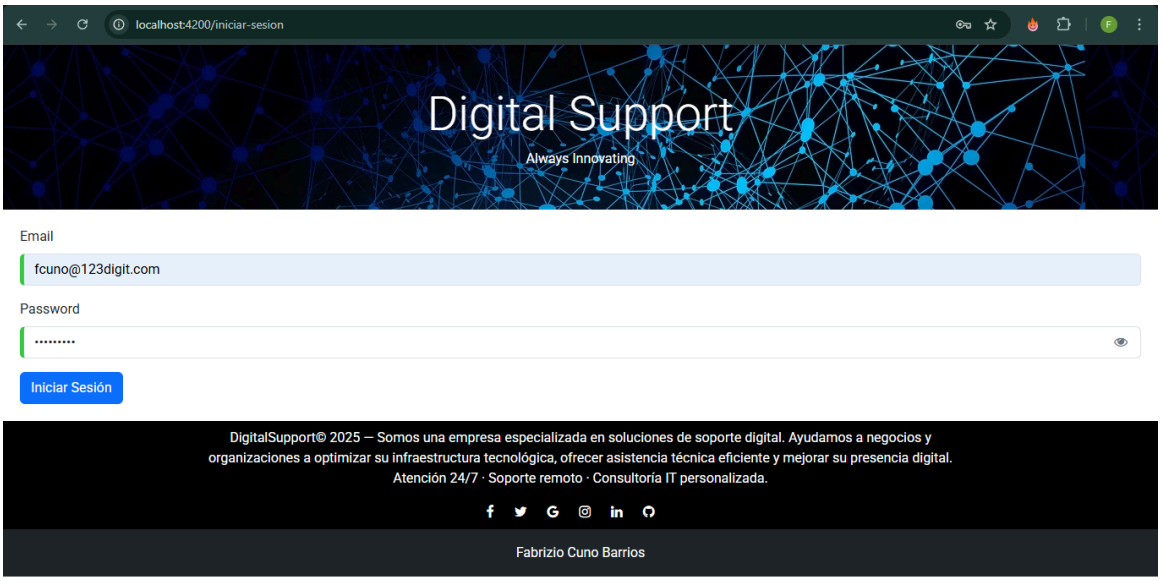
Ejecución del Proyecto: Utilizamos el comando ng serve

```
Microsoft Windows [Versión 10.0.19045.4291]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\desarrolloweb\DigitalSupport>ng serve
[baseline-browser-mapping] The data in this module is over two months old.  To ensure accurate Baseline data, please
 update: `npm i baseline-browser-mapping@latest -D`
Initial chunk files | Names                                 |  Raw size
styles.css          | styles                                | 287.42 kB |
scripts.js          | scripts                               | 180.66 kB |
chunk-EBOIDTFP.js   | -                                     |  63.05 kB |
chunk-3NGDBCWF.js   | -                                     |  57.20 kB |
chunk-K7EVKIPW.js   | -                                     |  35.58 kB |
```

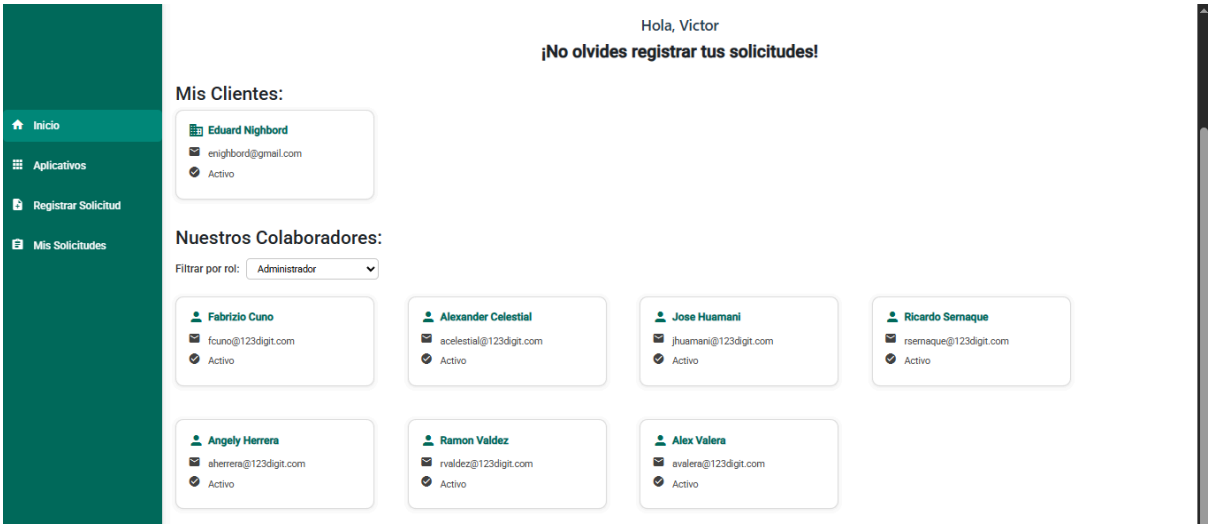
```
Watch mode enabled. Watching for file changes...
NOTE: Raw file sizes do not reflect development server per-request transformations.
  ➜ Local:   http://localhost:4200/
  ➜ press h + enter to show help
```

LOGIN DE LA PLATAFORMA (VALIDACIÓN ROBUSTA DE EMAIL Y PASSWORD OBLIGATORIOS PARA ACCEDER A UNO DE LOS 3 ROLES)



DASHBOARD DE ADMINISTRADOR: PANELES QUE INDICAN DATOS IMPORTANTES DEL FLUJO, 6 TIPOS DE GRÁFICOS ESTADÍSTICOS QUE INDICAN DATA IMPORTANTE DE LOS COLABORADORES. NAVBAR DE REDIRECCIÓN DONDE SE PUEDE CREAR USUARIOS (ADMINS, CLIENTE Y COLABORADORES), REGISTRAR CLIENTES (EMPRESA O PERSONAS NATURAL), CREAR NUEVOS ROLES DE LOS TRABAJADORES Y VISUALIZAR EL AVANCE TOTAL DE SOLICITUDES

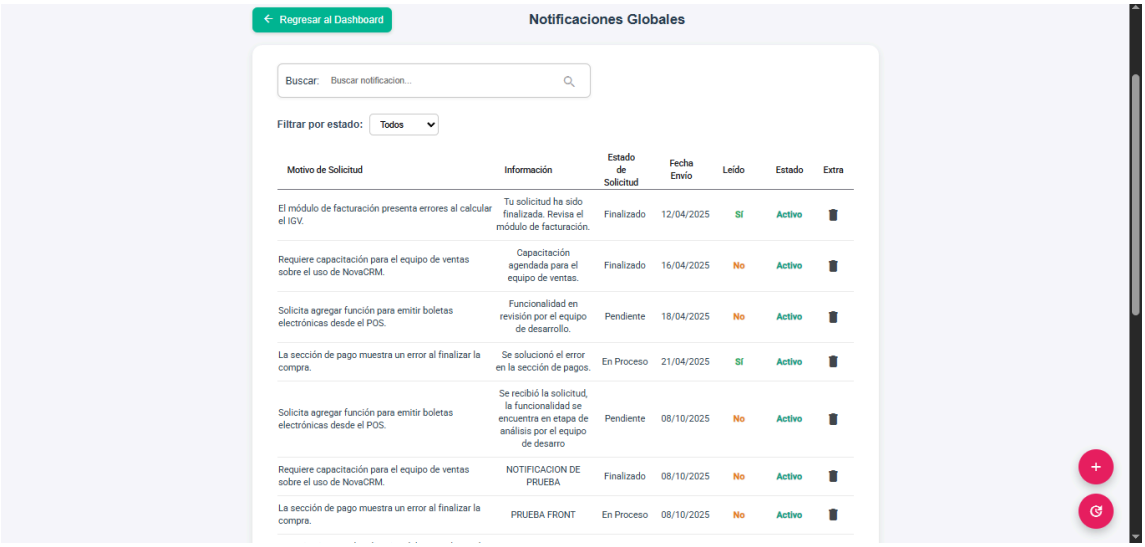
DASHBOARD DE USUARIO CLIENTE: PANEL DONDE PUEDES VISUALIZAR LOS CLIENTES RELACIONADOS AL USUARIO, LOS COLABORADORES DE DIGITAL SUPPORT QUE ESTÁN DISPONIBLES PARA ATENDER TUS SOLICITUDES, NAVBAR DE REDIRECCIÓN PARA PODER REGISTRAR APLICATIVOS, PEDIR NUEVAS SOLICITUDES Y VER EL AVANCE Y ESTADO DE ESTAS.



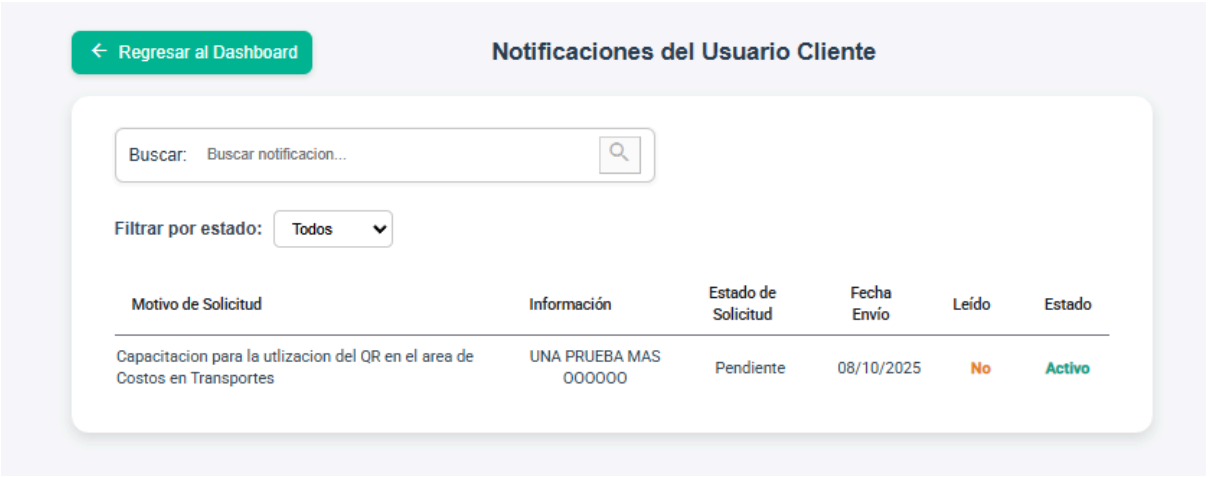
DASHBOARD DE COLABORADOR: PANEL DE VISUALIZACIÓN DE TODAS LAS SOLICITUDES DISPONIBLES, Y SU BOTÓN DE EDICIÓN PARA PODER COLOCAR SU NUEVO ESTADO (PENDIENTE - EN PROCESO - FINALIZADO), NAVBAR DE REDIRECCIÓN PARA REGISTRAR EL AVANCE DE LA SOLICITUDES, CREAR ASIGNACIONES DE TRABAJO COLABORATIVO (CON O SIN COORDINACIÓN) Y VISUALIZAR EL AVANCE REALIZADO

DASHBOARD DE NOTIFICACIONES (ADMINS/COLABORADORES): FILTRADO DE BÚSQUEDA DE NOTIFICACIONES TOTALES, BOTONES FLOTANTES DE LOS CUALES PUEDES CREAR NOTIFICACIONES QUE SE MANDAN AL CORREO DE LOS UC, O EDITAR NOTIFICACIONES YA CREADAS. CON EL APARTADO DE “REGRESAR AL

DASHBOARD” TE REDIRIGE A TU DASHBOARD PRINCIPAL SEGÚN EL ROL QUE LLEVES



DASHBOARD DE NOTIFICACIONES (UC): FILTRADO DE BÚSQUEDA DE NOTIFICACIONES PERSONALES, CON EL APARTADO DE “REGRESAR AL DASHBOARD” TE REDIRIGE A TU DASHBOARD PRINCIPAL (DE USUARIO CLIENTE)



Despliegue del Proyecto: El sistema fue desarrollado bajo un entorno de arquitectura cliente-servidor, por lo que requiere la ejecución independiente del backend (.NET), la base de datos (SQL Server) y el frontend (Angular).

Requisitos del Entorno:
* SQL Server Express o superior.
* .NET SDK instalado.
* Node.js y Angular CLI.
* Navegador web moderno

Configuración de la Base de Datos:

1. Abrir SQL Server Management Studio.
2. Ejecutar los scripts adjuntos, en orden.
3. Verificar la creación de tablas, relaciones y procedimientos almacenados.

Configuración del Backend (.NET API):

1. Abrir la solución en Visual Studio.
2. Configurar y editar la cadena de conexión en `appsettings.json`.
3. Ejecutar el proyecto mediante IIS Express.
4. Verificar la ejecución accediendo a Swagger: https://localhost:{puerto}/swagger

Configuración del Frontend (Angular):

1. Abrir el proyecto en Visual Studio Code.
2. Instalar dependencias: npm install
3. Ejecutar la aplicación: ng serve
4. Acceder desde el navegador: http://localhost:4200

Integración del Sistema:

Una vez ejecutados los tres componentes: Frontend Angular → consume API REST → .NET procesa lógica → SQL Server persiste datos.

El sistema opera mediante intercambio de información en formato JSON, permitiendo una comunicación desacoplada y escalable.

El despliegue local permite validar completamente las funcionalidades del sistema, incluyendo autenticación, gestión de solicitudes, seguimiento y notificaciones.