Manual Técnico

Desarrollo de Software

Primer Año de Bachillerato

Autores:

- Katherine Andrea Ruiz Bonilla
- Abraham Isaac Rodríguez Velázquez
- Noe David Saravia Siliezar
- Rodrigo Alejando Tisnado Corpeño

Sistema de administración de horas sociales Social Clock

Fecha:

Índice

1. Introducción

El sistema *Social Clock* es una aplicación de escritorio desarrollada en C# con conexión a SQL Server, diseñada para la gestión y control de las horas sociales de los estudiantes en instituciones educativas. Su principal objetivo es proporcionar una plataforma que permita registrar alumnos, administrar proyectos, almacenar información en una base de datos relacional y mostrar reportes estadísticos de manera clara para el usuario.

La aplicación proporciona una interfaz clara y organizada que facilita la interacción con el sistema. Entre sus funciones principales se incluyen el registro de estudiantes, la administración de proyectos, la generación de listados y la consulta de estadísticas.

Gracias a su integración con una base de datos relacional, el sistema mantiene la consistencia de los registros y permite un acceso rápido y seguro a la información almacenada. De esta manera, *Social Clock* se convierte en una herramienta que optimiza la gestión de las horas sociales y mejora la organización de los procesos administrativos relacionados.

2. Tecnologías y herramientas

Frontend

- **Visual Studio Code 2022 (Community)**

Version 17.14.13 (August 2025)

Uso: Visual studio code es usado para la creación y estructuración del programa.

- **.Net Framework**

Versión 4.7.2

Uso: …

- **Sql Server Management Studio 21**

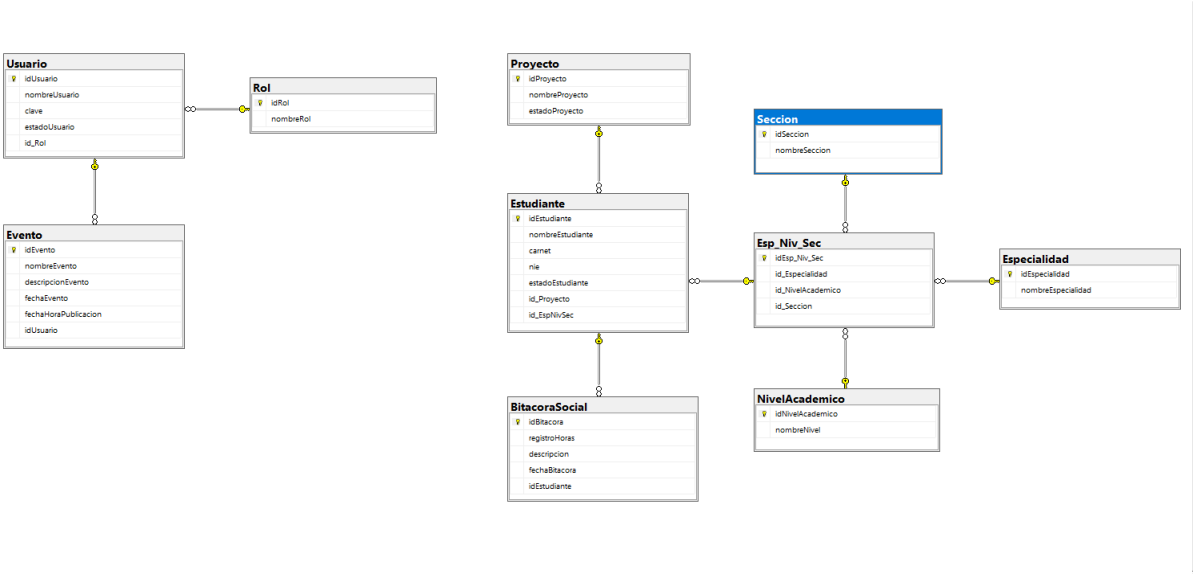
versión 16.0.1000.6

Uso: Gestor de base de datos relacional para almacenar y gestionar datos del proyecto.

3. Estructura de la base de datos

Modelo Relacional

Diagrama de dominio de la base de datos del proyecto PTC.



Nota: Mejor calidad del diagrama en el siguiente enlace

https://drive.google.com/file/d/1LmPrScEq_wH9t6xMSx8Pc2iW9ksbVBF5/view?usp=drive_link

Script de la base de datos

En el siguiente enlace se podrá encontrar el script de la base de datos, procedimientos almacenados, funciones y estructuras de tablas.

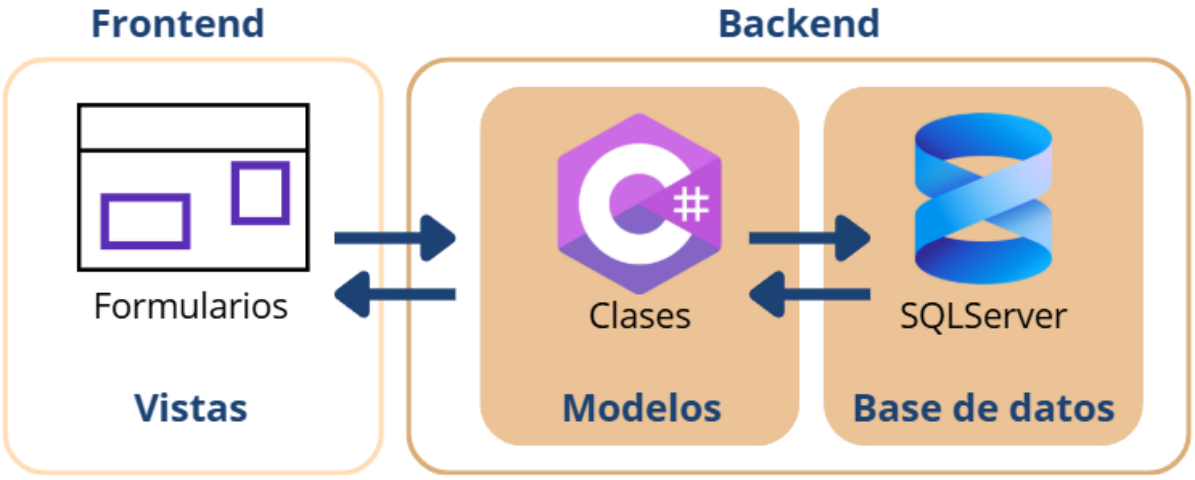
https://drive.google.com/file/d/1LmPrScEq_wH9t6xMSx8Pc2iW9ksbVBF5/view?usp=drive_link

4. Diccionario de datos

En el siguiente enlace podrá encontrar el diccionario de la base de datos:

https://docs.google.com/spreadsheets/d/106OGEzA5CacUHCKg4awIRTdnZJrxOeGU/edit?usp=drive_link&ouid=108985890997782131546&rtpof=true&sd=true

5. Arquitectura de software



1. **Frontend:**
    - Esta es la primera capa, con la que el usuario interactúa y la que desencadena todo el flujo de trabajo. Está constituida por las Views, o Vistas, que corresponden a formularios(Windows Forms) y están programados con lenguaje C#. A partir de estas Vistas los usuarios navegan en la aplicación y envían peticiones a través de eventos; las cuales son transmitidas de las Vistas hacia los Modelos.

2. **Backend:**
    - Modelos: Está conformado por un conjunto de Entidades o Clases. Cada clase corresponde a una tabla de la base de datos, por lo tanto, cuenta con todos los campos que dicha tabla posee como atributos privados. Las Vistas se relacionan con estas Clases a través de las propiedades GET y SET de los atributos. GET, se utiliza cuando el usuario solicita datos, mientras que SET se usa cuando se requiere ingresar información nueva. Las peticiones o datos que envían las Vistas, son procesadas y transmitidas de los Modelos hacia la base de datos en forma de consultas query, esto, a través de una clase Conexión y, del mismo modo, transmiten la respuesta que obtienen de la base de datos hacia las Vistas.

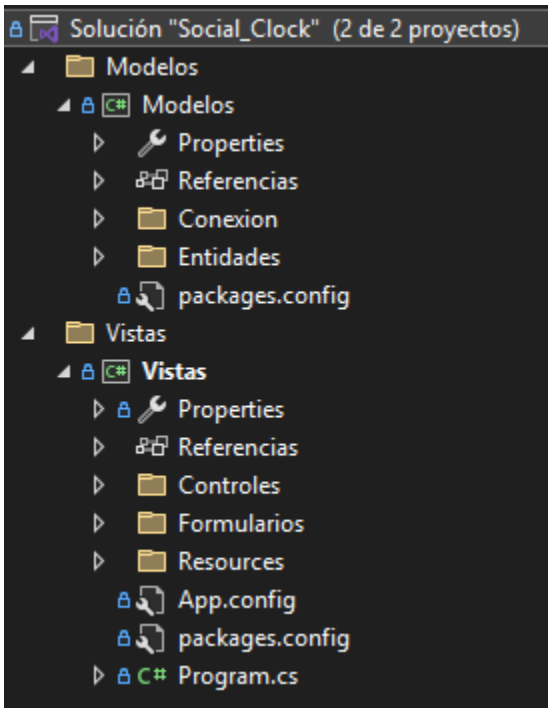
- Base de Datos: Se conforma por tablas con campos, en los que se almacena la información. La base de datos recibe las consultas DML enviadas por los Modelos e ingresa, elimina, actualiza o muestra datos según la petición, para luego transmitir el resultado de vuelta a los Modelos.

.

6. Estructura del proyecto

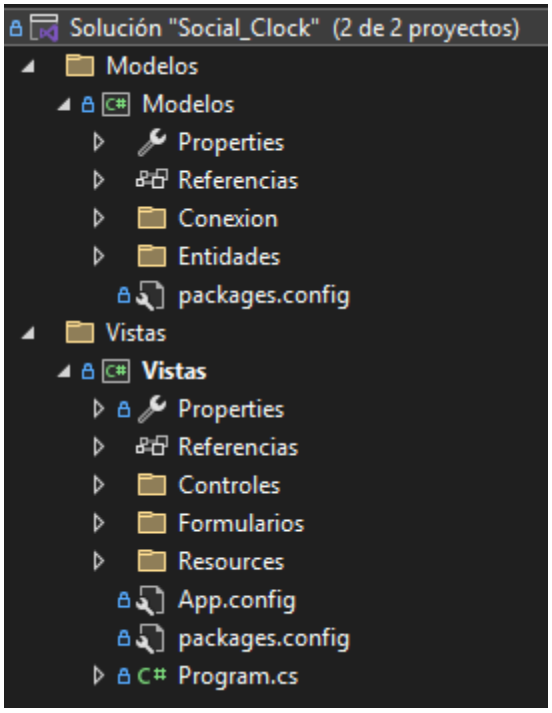
El proyecto se divide en Frontend (Vistas) y Backend (Modelos), los cuales se encuentran organizados mediante carpetas para un mayor orden y facilidad de mantenimiento.

Estructura general:



Refleja claramente un enfoque modular y sigue el patrón arquitectónico MVC (Modelo-Vista-Controlador), en nuestro proyecto está simplificado, ya que no existen “Controladores” incorporados explícitamente.

Estructura del backend:



Tiene un enfoque de arquitectura por capas y refleja el patrón DAO (Data Access Object). En resumen, la función de cada elemento o carpeta es:

- **Properties:** Contiene archivos de configuración internos del proyecto que son gestionados automáticamente por Visual Studio.

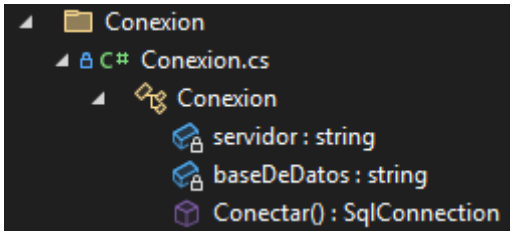
- **Referencias:** Muestra todas las dependencias externas que el proyecto necesita para compilar y funcionar correctamente.

- **Conexión:** Conecta la base de datos con la aplicación.

- **Entidades**: Contiene las Clases que representan los modelos de datos.

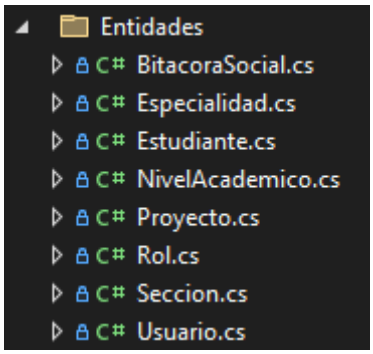
- **Packages**.config: Lista los paquetes NuGet que el proyecto requiere.

Estructura detallada de la carpeta Conexión:



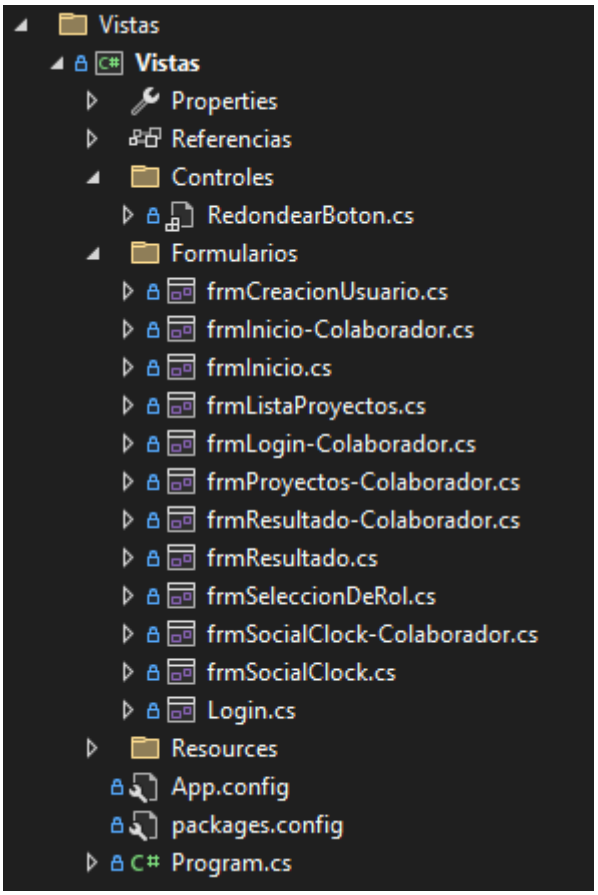
Esta carpeta contiene la Clase: Conexión, que es la responsable de conectar el proyecto con la base de datos. Esta posee los atributos: servidor y baseDeDatos; que se utilizan en el método “Conectar()” para determinar el servidor donde se encuentra la base de datos y el nombre de la misma.

Estructura detallada de la carpeta Entidades:



Esta carpeta almacena las Clases que corresponden a las tablas, o entidades, de la base de datos. Se aplica el patrón DAO (Data Access Object); cada Clase maneja directamente sus validaciones y operaciones CRUDS, esto mediante métodos y funciones que contienen las indicaciones y consultas DML de inserción, eliminación, actualización y búsqueda de datos; las cuales transmiten a la base de datos según sea requerido por el usuario. Se empacan y reciben datos a través de accesores GET y SET a los atributos de cada una de las Clases.

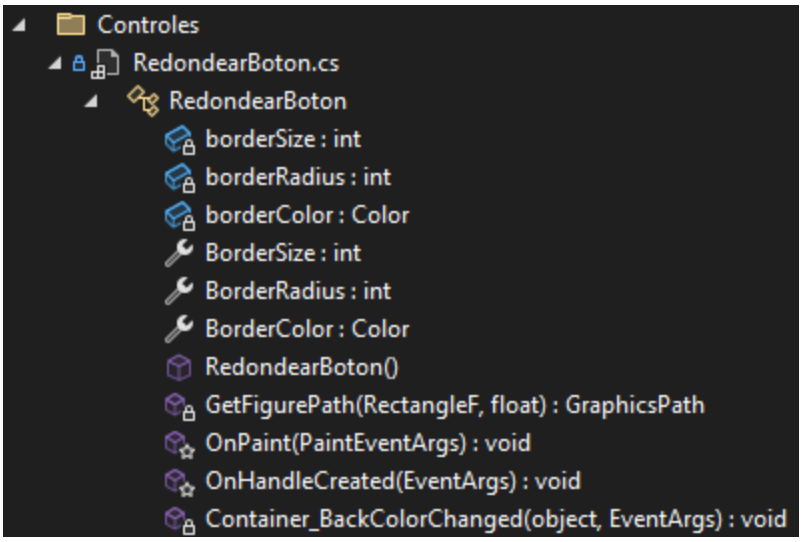
Estructura del Frontend:



Tiene un enfoque modular por capas y sigue un patrón MVC simplificado. En resumen, la función de cada carpeta es:

- **Propierties**: Contiene el archivo y configuraciones del proyecto como versión, cultura, y atributos de ensamblado.
- **Referencias**: Lista las bibliotecas externas y ensamblados que el proyecto utiliza, además de conectar las Vistas con los Modelos.
- **Cont|roles**: Almacena componentes reutilizables o personalizados.
- **Formularios**: Contiene todas las interfaces gráficas (Formularios).
- **Resources**: Guarda archivos de configuración.
- **Program.cs**: Es el punto de entrada de la aplicación; este contiene el método que inicia el formulario principal.

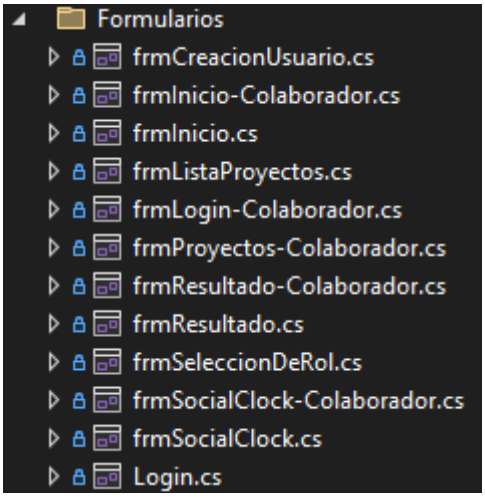
  Estructura detallada de la carpeta Controles:



Esta carpeta contiene un control personalizado, el cual concierne a un botón funcional, pero, con la apariencia de sus bordes modificada, pues dibuja sus esquinas redondeadas.

Este control se emplea y reutiliza en los formularios del proyecto con fines estéticos, sin embargo, mantiene la funcionalidad de un botón estándar

Estructura detallada de la carpeta Formularios:



Esta carpeta almacena las interfaces gráficas (Formularios) de todos los niveles de usuario; que para el proyecto corresponden a: administrador y colaborador.
Cada formulario posee su propio código de programación (archivos .cs), el cual muestra datos y lo hace funcional a partir de la gestión de eventos; conectándolo, al mismo tiempo, a los Modelos.

Cada formulario contiene un archivo Designer.cs y uno .resx (Resource XML). El primero se encarga de generar automáticamente el código que define la interfaz visual del formulario, es decir, contiene la declaración y configuración de los controles visuales, así como sus propiedades. Mientras que el segundo, almacena datos como cadenas de texto, imágenes, íconos, etc; en formato XML.

7. Diseño de aplicación

Paleta de Colores

La aplicación utiliza una combinación de colores pensada para que la experiencia sea clara, agradable y fácil de entender.

Color	Código	Uso
Azul Oxford	**#0A1338**	Color principal. Se utiliza principalmente en fondos y áreas importantes de la pantalla.
Azul Claro	**#99B4D1**	Se aplica en botones y recuadros para resaltar opciones y guiar la atención hacia lo que se puede seleccionar.
Blanco	**#FFFFFF**	Aporta limpieza y hace que todo se vea ordenado.
Arena Suave	**#C4B3A3**	Utilizado en botones de acciones específicas como eliminar, editar y limpiar.
Tonos suaves y degradados	—	Se utilizan como acompañamiento visual, aportando modernidad y frescura al diseño.

Aplicación en los diferentes elementos

Color	Código	Aplicación en la interfaz
Azul Oxford	**#0A1338**	Fondo principal, barras laterales y áreas destacadas. Transmite confianza y seriedad.
Azul Claro	**#99B4D1**	Botones y recuadros de selección. Resalta elementos interactivos y guía la atención del usuario.

Blanco	**#FFFFFF**	Textos, fondos secundarios y áreas de lectura. Aporta limpieza, contraste y orden.
Cafe Caqui	**#C4B3A3**	Botones de acciones específicas como eliminar, editar y limpiar. Se usan más que todo en funciones clave.
Tonos suaves y degradados	—	Fondos decorativos, transiciones y detalles gráficos. Aportan modernidad y estética sin distraer.

Tipos de tamaños y fuentes

La aplicación utiliza únicamente dos tipografías: Microsoft Sans Serif y Century Gothic. Cada una tiene asignados tamaños y estilos específicos según el tipo de información que se quiere resaltar.

Tamaño	**Estilo**	**Uso**
28.2 pt	Regular	Títulos principales y encabezados destacados.
18 pt	Regular	Subtítulos, apartados secundarios.

Century Gothic

Tamaño	**Estilo**	**Uso**
12 pt	Negrita (Bold)	Textos de contenido.
16 pt o más	Negrita (Bold)	Indicaciones dentro del programa (ejemplo: señalar dónde escribir nombres o datos importantes).

Tipos y dimensiones de imágenes utilizadas

En la aplicación se emplean imágenes específicas para reforzar la navegación y la comprensión de cada sección.

Imagen	Dimensiones	Formato	Uso dentro del sistema
Fondo de inicio de sesión	1243 × 620 px	JPG / PNG	Imagen de fondo que aparece en la pantalla de login. Aporta estética visual.
Ícono de usuario	32 × 32 px	PNG con fondo transparente	Representa el campo de *nombre de usuario* en el inicio de sesión.
Ícono de candado	32 × 32 px	PNG con fondo transparente	Representa el campo de *contraseña* en el inicio de sesión.
Ícono de tuerca	64 × 64 px – 128 × 128 px	PNG con fondo transparente	Dirige al apartado de *Creación de Usuario* (agregar administradores o colaboradores al sistema).
Ícono de casa	64 × 64 px – 128 × 128 px	PNG con fondo transparente	Representa el apartado de *Inicio*. Permite agregar estudiantes al sistema.
Ícono de Listado	64 × 64 px – 128 × 128 px	PNG con fondo transparente	Representa la *Lista de Proyectos*. Muestra los proyectos con sus estudiantes y permite agregar proyectos nuevos.
Ícono de estadísticas (barras)	64 × 64 px – 128 × 128 px	PNG con fondo transparente	Representa la sección de *Estadísticas*. Permite visualizar el número de estudiantes en cada proyecto.

Tamaños de pantalla soportadas

La aplicación ha sido diseñada para funcionar correctamente en una resolución de pantalla específica:

Resolución soportada	Observaciones
1261 × 667 px	Es la resolución óptima del sistema. Garantiza que todos los elementos de la interfaz se visualicen correctamente sin deformaciones.

Cuando la aplicación se utiliza en resoluciones mayores o en modo pantalla completa, algunos elementos pueden desajustarse o deformarse. Por este motivo, el sistema ha sido configurado para forzar el uso de la resolución indicada y así mantener la coherencia visual y funcional.

9. Buenas prácticas de desarrollo

En C#

- **Uso de convenciones de nomenclatura:**
    - Clases y métodos con *PascalCase* (ejemplo:RegistrarEstudiante).

Cuando se desarrolla en **C#**, es importante seguir reglas estandarizadas de escritura para nombres de clases, métodos, variables y atributos.Para eso sirve seguir la nomenclatura de PascalCase

PascalCase

- Cada palabra comienza con mayúscula, sin guiones ni guiones bajos.
- Se usa en nombres de **clases**, **métodos** y **propiedades**.
- Ejemplos:
    - RegistrarEstudiante (método)
    - UsuarioActivo (propiedad)
    - GestionProyectos (clase)
    - Variables y atributos con *camelCase* (ejemplo nombreEstudiante).

En SQL Server: **CamelCase**

La primera palabra empieza con minúscula y las siguientes con mayúscula.

Se usa en **variables locales** y **parámetros de métodos**.

Ejemplos:

- nombreEstudiante (variable)
- cantidadHoras (parámetro)
- fechaRegistro (variable)

9. Requerimientos de Hardware y Software

Entorno de desarrollo local

- **Procesador:** Intel Core i5 o superior.
- **Memoria RAM:** 8 GB mínimo.
- **Almacenamiento:** 2 GB disponibles para proyecto y base de datos de pruebas.
- **Sistema operativo:** Windows 10 o superior (64 bits).
- **Software necesario:**
    - Visual Studio 2022 con paquetes C# y .NET.
    - SQL Server 2019 o superior.
    - SQL Server Management Studio (SSMS).
    - .NET Framework 4.8 o .NET 6.0.

Servidor para producción

- **Procesador:** Intel Xeon (mínimo 4 núcleos).
- **Memoria RAM:** 16 GB.
- **Disco:** SSD 500 GB con respaldos automáticos.
- **Sistema operativo:** Windows Server 2019 o superior.
- **Base de datos:** SQL Server 2019 o superior.

10. Instalación y configuración

En fase de desarrollo

- Abrir el proyecto *SocialClock.sln* en Visual Studio.
- Crear la base de datos *Social_Clock* en SQL Server y ejecutar el script de tablas y relaciones.
- Configurar la cadena de conexión en el proyecto.
- Compilar y ejecutar en modo *Debug*.

En fase de implementación

- Distribuir el instalador en formato *.exe*.
- Configurar la conexión al servidor SQL central.
- Realizar pruebas iniciales:
    - Iniciar sesión con un usuario autorizado.
    - Registrar un alumno de prueba.
    - Verificar almacenamiento y consistencia de datos.