

SUNDEO

Desarrollo de WEB / API de medición y control de producción eléctrica solar mediante
terminales de autoconsumo

Daniel Caballero Wawoe

Curso académico: 2 DAM

Tutor del proyecto: Víctor Colomo Gómez

ÍNDICE PAGINADO

Índice

ÍNDICE PAGINADO	2
1. RESUMEN	5
2. JUSTIFICACIÓN DEL PROYECTO	6
3. OBJETIVOS	7
A. OBJETIVO GENERAL	7
B. OBJETIVOS ESPECÍFICOS	7
4. DESARROLLO	8
FUNDAMENTACIÓN TEÓRICA:	8
1.1 Planteamiento del problema e hipótesis	8
1.2 Marco tecnológico	8
1.3 Procedimientos de desarrollo	8
1.4 Tareas realizadas	9
TECNOLOGÍAS UTILIZADAS:	9
ASP.NET Core	9
Entity Framework Core	10
JWT (JSON Web Tokens)	10
API RESTful	11
Arquitectura Modelo-Vista-Controlador (MVC)	11
SQL Server	12
WebGL (Three.js / Babylon.js)	12
React / Blazor WASM	13
Git / GitHub	14
METODOLOGÍAS UTILIZADAS:	15
PLANIFICACIÓN DEL PROYECTO:	16
MATERIALES Y MÉTODOS:	17
Diagrama de la BBDD:	17
Diagrama de Casos de uso:	22
Estructura del Proyecto:	23
Análisis del Código:	24

RESULTADOS Y ANÁLISIS:	26
5. CONCLUSIONES	28
6. LÍNEAS DE INVESTIGACIÓN FUTURAS	30
1. Integración con IoT (medidores inteligentes):	30
2. Incorporación de elementos interactivos avanzados:	30
3.Desarrollo de algoritmos predictivos:	30
4.Aplicación móvil nativa	30
7. BIBLIOGRAFÍAS	31
8. ANEXOS	32
1. Estructura del proyecto:	32
1-Estructura Backend:	32
2-Estructura BBDD:	33
3-Estructura Frontend:	33
2. Planificación del desarrollo del proyecto:	35
Tabla de planificación (Trello):	35
5-Proyecto de GIT:	36
6-Diagrama de Gantt (Contraído):	37
7-Diagrama de Gantt (expandido):	38
3. Análisis del Código:	39
Appsettings:	39
Método de autenticación(logging):	40
Intercepción de axios 1:	41
4. Resultados	42
Inicio de sesión:	42
Dashboard (Vista principal/usuario):	42
Dashboard (Vista principal/administrador):	43
Gráficos Interactivos:	43
Gestión de perfil:	44
Exportación a CSV:	44
Exportación a imagen:	45
Panel de Administración:	45
5. Plataformas de referencia:	45

FusionSolar:	45
Enphase Enlighten:	46
SMA Sunny:	46
SolarEdge:	46

1. RESUMEN

La transición energética global está transformando los modelos de consumo y producción de electricidad. En 2024, se añadieron más de 585 GW de nueva capacidad renovable en todo el mundo, de los cuales 452 GW provinieron de energía solar, consolidándola como la fuente renovable de mayor crecimiento. En este contexto, los usuarios de instalaciones solares de autoconsumo ya sean domésticos, industriales o comunitarios, se enfrentan a la necesidad de tener herramientas efectivas que les permitan visualizar y entender cómo se genera, consume y vierte la energía en tiempo real.

SUNDEO surge como respuesta a esta necesidad. Se trata de una aplicación web interactiva que ofrece una solución avanzada para la monitorización energética. Su objetivo principal es permitir al usuario observar en tiempo real la producción energética mediante una combinación única de métricas numéricas y visualización 3D interactiva.

A diferencia de las plataformas tradicionales que se basan únicamente en gráficos 2D y datos tabulados, SUNDEO introduce una interfaz gráfica basada en tecnologías **WebGL** que permite representar el flujo energético dentro de un modelo arquitectónico 3D. Esto facilita la interpretación espacial de los datos, ayudando a detectar patrones de consumo o pérdidas energéticas y fomentando la toma de decisiones orientadas a la eficiencia y la sostenibilidad.

SUNDEO combina los siguientes pilares tecnológicos:

- Monitorización numérica en tiempo real.
- Análisis de excedentes energéticos para detectar oportunidades de optimización.
- Visualización tridimensional de los flujos energéticos mediante Three.js o Babylon.js.

Con estas funcionalidades, SUNDEO pretende no solo ser una herramienta técnica, sino también educativa y de concienciación energética, contribuyendo a que los usuarios

comprendan el funcionamiento de sus instalaciones y participen activamente en la transición energética.

2. JUSTIFICACIÓN DEL PROYECTO

Pese al crecimiento del mercado de monitorización solar, la mayoría de las soluciones actuales están limitadas a interfaces 2D, paneles numéricos y dashboards con información genérica. Herramientas ampliamente utilizadas como SolarEdge, Huawei FusionSolar, Enphase Enlighten o SMA Sunny Portal ofrecen monitoreo energético en tiempo real, pero presentan diversas limitaciones técnicas y funcionales:

(Consultar anexo 5)

- 1- Interfaz poco personalizable o dependiente de hardware específico.
- 2- Falta de visualización espacial de los componentes de la instalación.
- 3- Escasa orientación educativa sobre eficiencia energética.
- 4- Limitado acceso a datos históricos y opciones de exportación.

En contraste, SUNDEO ofrece una propuesta innovadora centrada en el usuario final y basada en una arquitectura abierta y extensible. Entre sus ventajas destacan:

- 1- Interfaz 3D interactiva mediante **WebGL** (Three.js o **Babylon.js**), que permite visualizar los flujos energéticos en tiempo real sobre un modelo arquitectónico personalizado.
- 2- Representación espacial que facilita la detección de patrones de consumo, pérdidas energéticas o desequilibrios en la producción.
- 3- Acceso independiente de hardware, ideal para instalaciones de autoconsumo con diversidad de componentes.

Además, al estar construido sobre tecnologías modernas como **ASP.NET Core**, **React** y **JWT**, SUNDEO garantiza escalabilidad, seguridad y fácil mantenimiento, posicionándose como una alternativa viable y diferenciadora frente a las soluciones propietarias actuales del mercado.

3. OBJETIVOS

A. OBJETIVO GENERAL

Desarrollar una aplicación web interactiva que permita monitorear en tiempo real la generación energética en instalaciones de autoconsumo solar.

B. OBJETIVOS ESPECÍFICOS

- Diseñar y desarrollar una **API RESTful** con **ASP.NET Core**.
- Implementar autenticación y autorización mediante **JWT**.
- Integrar una interfaz **WebGL** (Three.js/Babylon.js) en **React** o **Blazor WASM**.
- Almacenar datos históricos en **SQL Server**
- Asegurar la escalabilidad, seguridad y mantenibilidad del sistema.

4. DESARROLLO

FUNDAMENTACIÓN TEÓRICA:

El presente trabajo se inscribe en el ámbito de las energías renovables, concretamente en el desarrollo de soluciones tecnológicas orientadas al monitoreo y análisis de la producción solar fotovoltaica. La finalidad principal es diseñar e implementar una aplicación web funcional que permita registrar, visualizar y gestionar datos energéticos, facilitando la toma de decisiones tanto a nivel técnico como operativo.

1.1 Planteamiento del problema e hipótesis

La creciente adopción de sistemas fotovoltaicos conlleva la necesidad de herramientas accesibles y efectivas que permitan supervisar su rendimiento en tiempo real. Actualmente, muchas instalaciones carecen de sistemas de visualización intuitivos o personalizados, lo que dificulta el seguimiento de la producción y la detección de posibles incidencias.

Se parte de la hipótesis de que una aplicación web centrada en la visualización y análisis de datos de producción solar puede contribuir significativamente a optimizar la gestión energética de pequeñas y medianas instalaciones. Dicha solución debe permitir tanto el acceso a datos históricos como la exportación de informes, todo ello a través de una interfaz accesible y adaptada al usuario final.

1.2 Marco tecnológico

El desarrollo se basa en una arquitectura cliente-servidor. En el **backend** se ha utilizado el framework **.NET** para construir la **API** que gestiona la lógica de negocio, las operaciones sobre la base de datos y la autenticación. En el **frontend** se ha empleado **React** para la creación de una interfaz dinámica y modular, junto con librerías especializadas en visualización de datos (como **Recharts** o **Chart.js**).

Además, se ha optado por la centralización de estilos mediante un archivo **CSS** global, lo que favorece la mantenibilidad y coherencia del diseño. El almacenamiento de datos se realiza en una base de datos relacional, accesible desde el servidor.

1.3 Procedimientos de desarrollo

El proceso de desarrollo ha seguido una metodología iterativa, basada en la implementación progresiva de funcionalidades. Entre los procedimientos más relevantes se incluyen:

- Modelado de la base de datos y definición de las entidades principales.

- Desarrollo de endpoints en la **API** para **CRUD** de usuarios y datos de producción.
- Construcción de componentes **React** reutilizables para tablas, gráficos y formularios.
- Implementación de modales para la edición de perfil y visualización detallada de la producción.
- Validación de formularios, especialmente en el proceso de cambio de contraseña.
- Exportación de datos y gráficos en formato CSV e imagen.
- Pruebas funcionales y verificación del flujo completo de datos entre cliente y servidor.

1.4 Tareas realizadas

Durante la ejecución del proyecto se han llevado a cabo las siguientes tareas:

- Análisis de requisitos funcionales y no funcionales.
- Diseño de la estructura del proyecto y arquitectura del sistema.
- Desarrollo del backend con .NET y configuración de rutas, controladores y modelos.
- Implementación del frontend con React y adaptación del diseño visual.
- Integración de librerías para la representación gráfica de datos.
- Configuración de exportaciones, formularios y validaciones.
- Elaboración de la documentación técnica y la memoria del proyecto.

TECNOLOGÍAS UTILIZADAS:

ASP.NET Core

ASP.NET Core es un framework de desarrollo web de código abierto creado por Microsoft. Se basa en el lenguaje C# y permite construir aplicaciones web modernas,

APIs REST y servicios **backend** escalables. Su arquitectura modular, orientada a microservicios, y su compatibilidad multiplataforma (Windows, Linux y macOS) lo hacen ideal para entornos de producción exigentes. Además, está optimizado para alto rendimiento y es ampliamente adoptado en entornos empresariales.

- **Ventajas:** Alto rendimiento, arquitectura modular, integración nativa con servicios de Azure, soporte a largo plazo (LTS).
- **Desventajas:** Curva de aprendizaje pronunciada para nuevos desarrolladores, menor comunidad que Node.js, despliegue más complejo en entornos no Windows.

Entity Framework Core

Entity Framework Core (EF Core) es un mapeador objeto-relacional (ORM) para **.NET** que permite a los desarrolladores interactuar con bases de datos relacionales mediante objetos y clases en lugar de comandos **SQL**. **EF Core** permite definir el modelo de datos mediante código (Code First) o usar una base de datos existente (Database First). También facilita la gestión de migraciones y el versionado del esquema de base de datos.

- **Ventajas:** Integración fluida con **ASP.NET Core**, consultas LINQ, soporte para múltiples motores de bases de datos, automatización de migraciones.
- **Desventajas:** Rendimiento subóptimo en consultas complejas si no se optimiza adecuadamente, mayor uso de recursos frente a consultas SQL manuales, curva de aprendizaje en el modelado avanzado.

JWT (JSON Web Tokens)

JWT es un estándar abierto (RFC 7519) para el intercambio seguro de información entre partes como un objeto **JSON**. Se usa ampliamente para implementar mecanismos de autenticación y autorización en **APIs** modernas. En **SUNDEO**, **JWT** se emplea para la autenticación **stateless**, donde el **backend** no almacena sesiones, sino que confía en el token firmado que envía el cliente en cada petición.

- **Ventajas:** **Stateless** (no necesita almacenamiento de sesión), portabilidad, compatibilidad con OAuth2, facilidad de uso con bibliotecas modernas.

- **Desventajas:** El tamaño del token puede impactar en el rendimiento en aplicaciones móviles, requiere mecanismos adicionales de renovación de tokens (refresh tokens), vulnerabilidad ante ataques XSS si no se almacena de forma segura en el cliente.

API RESTful

Una **API RESTful (Representational State Transfer)** es un conjunto de principios de arquitectura que permite la comunicación entre sistemas mediante el protocolo **HTTP**. En el proyecto **SUNDEO**, la **API** desarrollada con **ASP.NET Core** expone diversos endpoints que permiten al cliente (frontend) realizar operaciones **CRUD** (crear, leer, actualizar y eliminar) sobre los recursos principales: usuarios, producción solar, configuraciones, etc.

Cada recurso se identifica mediante una **URL** única, y las operaciones se determinan por el método **HTTP** empleado (**GET, POST, PUT, DELETE**). Esto permite una separación clara entre el cliente y el servidor, y facilita el desarrollo de aplicaciones distribuidas, escalables y mantenibles.

Por ejemplo:

- **GET /api/Users** → Obtiene una lista de usuarios.
- **POST /api/Production** → Registra nuevos datos de producción.
- **PUT /api/Users/edit/{id}** → Edita los datos de un usuario específico.
- **DELETE /api/Production/{id}** → Elimina un registro de producción.

Arquitectura Modelo-Vista-Controlador (MVC)

La arquitectura Modelo-Vista-Controlador es un patrón ampliamente adoptado en el desarrollo de aplicaciones web. Esta separación de responsabilidades facilita el mantenimiento, la escalabilidad y la reutilización del código. En el contexto de **SUNDEO**, se aplica principalmente en el backend desarrollado con **ASP.NET Core**, aunque también se mantiene una estructura modular en el frontend mediante componentes en **React**.

- **Modelo (Model):** Representa la lógica de negocio y la estructura de los datos. En EF Core, esto se traduce en las clases que definen las entidades (por ejemplo, User, ProductionData) y su relación con la base de datos.
- **Vista (View):** Aunque **ASP.NET MVC** incluye mecanismos de renderizado de vistas, en este proyecto la visualización se gestiona íntegramente desde el frontend con React, que actúa como la capa de presentación.
- **Controlador (Controller):** Son las clases que reciben las solicitudes HTTP, procesan la lógica y devuelven una respuesta. En **ASP.NET Core**, cada controlador expone acciones que interactúan con el modelo y devuelven datos a la vista (o en este caso, a la API).

Esta arquitectura favorece la escalabilidad del sistema y permite una colaboración más eficiente entre diferentes roles del equipo de desarrollo, al separar claramente la lógica de datos, la presentación y el flujo de control.

SQL Server

Microsoft SQL Server es un sistema de gestión de bases de datos relacional (RDBMS) que se utiliza ampliamente en entornos empresariales. SUNDEO utiliza **SQL Server** para almacenar datos históricos y operacionales sobre producción, consumo y excedentes energéticos. Permite ejecutar consultas complejas, generar reportes, y escalar a entornos de alta disponibilidad.

- **Ventajas:** Alta integración con herramientas de Microsoft (Power BI, Azure), transacciones ACID, optimización de consultas mediante índices y planes de ejecución, escalabilidad vertical y horizontal.
- **Desventajas:** Coste de licenciamiento en versiones empresariales, complejidad de administración y tuning de rendimiento.

WebGL (Three.js / Babylon.js)

WebGL es una API JavaScript que permite renderizar gráficos 2D y 3D dentro de cualquier navegador compatible sin necesidad de plugins. SUNDEO utiliza bibliotecas

basadas en WebGL como Three.js y Babylon.js para crear escenas tridimensionales interactivas que representan instalaciones fotovoltaicas.

- **Three.js:** Popular y ligera, ofrece un alto grado de personalización, ideal para desarrolladores que necesitan control sobre cada aspecto del renderizado.
- **Babylon.js:** Orientada a desarrolladores que buscan una solución más completa con motor de físicas, GUI integrada y optimización predeterminada.
- **Ventajas:** Visualizaciones interactivas y realistas en el navegador, sin necesidad de software adicional; mejora la comprensión espacial del sistema energético.
- **Desventajas:** Desarrollo más complejo que interfaces 2D, consumo elevado de recursos en el cliente, dependencia del hardware gráfico del usuario.

React / Blazor WASM

Ambas son tecnologías para el desarrollo del frontend de aplicaciones web modernas.

- **React** es una biblioteca JavaScript desarrollada por Meta que permite construir interfaces de usuario basadas en componentes reutilizables. Utiliza un DOM virtual para mejorar el rendimiento y es altamente modular.
- **Blazor WebAssembly (WASM)** permite a los desarrolladores escribir el frontend en C# en lugar de JavaScript. Ejecuta código .NET directamente en el navegador a través de WebAssembly, lo que permite un desarrollo full-stack con un solo lenguaje.
- **Ventajas:** Interfaces reactivas y escalables, fuerte comunidad, ecosistema de componentes, rendimiento aceptable incluso en dispositivos móviles.
- **Desventajas:** React requiere conocimientos avanzados de JavaScript moderno; Blazor WASM tiene un tamaño de carga inicial mayor y menor madurez que React en algunos entornos productivos.

Git / GitHub

Git es un sistema de control de versiones distribuido, mientras que GitHub es una plataforma de colaboración que utiliza Git como base. En SUNDEO, Git se utilizó para llevar el historial de desarrollo y facilitar la colaboración entre desarrolladores.

- **Ventajas:** Control de versiones eficiente, posibilidad de trabajo colaborativo asincrónico, integración con pipelines de CI/CD y herramientas de automatización.
- **Desventajas:** Requiere conocimientos previos sobre ramas, conflictos y fusiones (merge), especialmente en proyectos con múltiples colaboradores simultáneos.

URL del repositorio: <https://github.com/Azazel033/SUNDEO>

METODOLOGÍAS UTILIZADAS:

El desarrollo del proyecto SUNDEO se llevó a cabo siguiendo una **metodología ágil en cascada**, una combinación estructurada que permite avanzar por etapas claramente definidas, pero con revisiones frecuentes, propias de los métodos ágiles.

Esta metodología fue especialmente útil para un desarrollo secuencial con entregables funcionales en cada fase:

1. **Diseño de la base de datos:** modelado de las entidades, relaciones y claves primarias/foráneas.
2. **Desarrollo del backend:** implementación de la **API REST** con **ASP.NET Core** y autenticación con **JWT**.
3. **Frontend funcional:** desarrollo con **React** y **Blazor WASM** para gestionar usuarios, visualizar datos y realizar peticiones.
4. **Mejora visual del frontend:** integración de visualización 3D, unificación de estilos con **CSS** global y diseño responsivo.
5. **Corrección de errores y pruebas funcionales:** pruebas manuales de funcionalidades, control de errores y validación de endpoints.

Esta metodología permitió un desarrollo iterativo pero ordenado, adecuado al entorno académico y al alcance del proyecto.

PLANIFICACIÓN DEL PROYECTO:

[\(Consultar anexo 2\)](#)

La planificación se estructuró en seis fases principales, distribuidas en semanas y orientadas a objetivos concretos:

Fase	Tareas principales	Duración estimada
1	Análisis de requisitos y diseño inicial	2 semanas
2	Diseño y modelado de la base de datos	1 semana
3	Desarrollo del backend (API + seguridad)	3 semanas
4	Desarrollo funcional del frontend	2 semanas
5	Visualización 3D e integración general	2 semanas
6	Pruebas, corrección de errores y documentación	1 semana

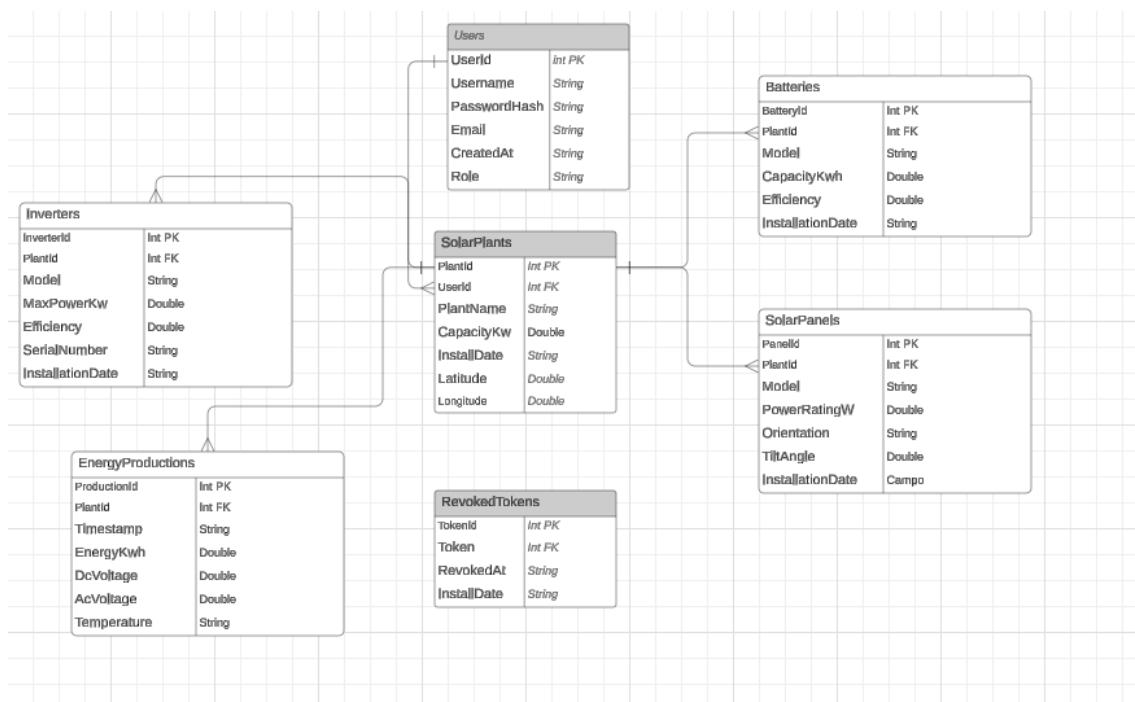
Para la organización se utilizó **Trello** como tablero de tareas, con columnas por fase y **GitHub** para control de versiones, seguimiento de cambios.

El control de las tareas se fue documentando mediante un diagrama de Gantt

MATERIALES Y MÉTODOS:

Diagrama de la BBDD:

Esta base de datos está diseñada para gestionar un sistema de monitoreo y administración de plantas solares. Permite registrar usuarios, plantas solares asociadas a esos usuarios, y los distintos componentes técnicos y datos de producción de energía asociados a cada planta.



Tablas y Descripciones:

1. Users

Contiene la información de los usuarios del sistema.

- **UserId**: Identificador único del usuario (clave primaria).
- **Username**: Nombre de usuario.
- **PasswordHash**: Contraseña almacenada de forma segura.

- **Email:** Dirección de correo electrónico del usuario.
- **CreatedAt:** Fecha de creación del usuario.
- **Role:** Rol del usuario (por ejemplo: administrador, cliente, etc.).

Relación: Un usuario puede tener **múltiples** plantas solares asociadas (SolarPlants).

2. RevokedTokens

Registra los tokens de acceso que han sido revocados, útil para mantener la seguridad del sistema y evitar accesos no autorizados.

- **TokenId:** Identificador único del token revocado (clave primaria).
- **Token:** Cadena del token revocado.
- **RevokedAt:** Fecha y hora en que el token fue revocado.

Relación: Esta tabla **no está relacionada** con otras. Es independiente.

3. SolarPlants

Representa una planta solar registrada por un usuario.

- **PlantId:** Identificador único de la planta (clave primaria).
- **UserId:** Clave foránea que hace referencia al usuario propietario.
- **PlantName:** Nombre asignado a la planta solar.
- **CapacityKw:** Capacidad de generación de energía en kilovatios (kW).
- **InstallDate:** Fecha de instalación.
- **Latitude y Longitude:** Ubicación geográfica de la planta.

Relación: Cada planta pertenece a un solo usuario, y puede tener múltiples dispositivos (baterías, inversores, paneles) y registros de producción de energía.

4. Batteries

Registra información sobre las baterías instaladas en una planta solar.

- **BatteryId:** Identificador único (clave primaria).
- **PlantId:** Clave foránea hacia SolarPlants.
- **Model:** Modelo de la batería.
- **CapacityKwh:** Capacidad de almacenamiento en kilovatios-hora (kWh).
- **Efficiency:** Eficiencia de la batería.
- **InstallationDate:** Fecha de instalación.

Relación: Muchas baterías pueden estar asociadas a una sola planta.

5. Inverters

Contiene información de los inversores eléctricos usados en la planta solar.

- **InverterId:** Identificador único (clave primaria).
- **PlantId:** Clave foránea hacia SolarPlants.
- **Model:** Modelo del inversor.
- **MaxPowerKw:** Potencia máxima en kilovatios.
- **Efficiency:** Eficiencia de conversión del inversor.
- **SerialNumber:** Número de serie.

- **InstallationDate:** Fecha de instalación.

Relación: Un inversor está vinculado a una sola planta.

6. SolarPanels

Información sobre los paneles solares instalados.

- **PanelId:** Identificador único (clave primaria).
- **PlantId:** Clave foránea hacia SolarPlants.
- **Model:** Modelo del panel.
- **PowerRatingW:** Potencia nominal en vatios.
- **Orientation:** Orientación (por ejemplo, norte, sur, este, oeste).
- **TiltAngle:** Ángulo de inclinación del panel.
- **InstallationDate:** Fecha de instalación.

Relación: Múltiples paneles pueden estar instalados en una planta.

7. EnergyProductions

Registra los datos de producción energética de una planta solar.

- **ProductionId:** Identificador único (clave primaria).
- **PlantId:** Clave foránea hacia SolarPlants.
- **Timestamp:** Fecha y hora del registro de producción.
- **EnergyKwh:** Energía generada en kilovatios-hora.

- **DcVoltage:** Voltaje en corriente continua.
- **AcVoltage:** Voltaje en corriente alterna.
- **Temperature:** Temperatura durante la producción (opcional).

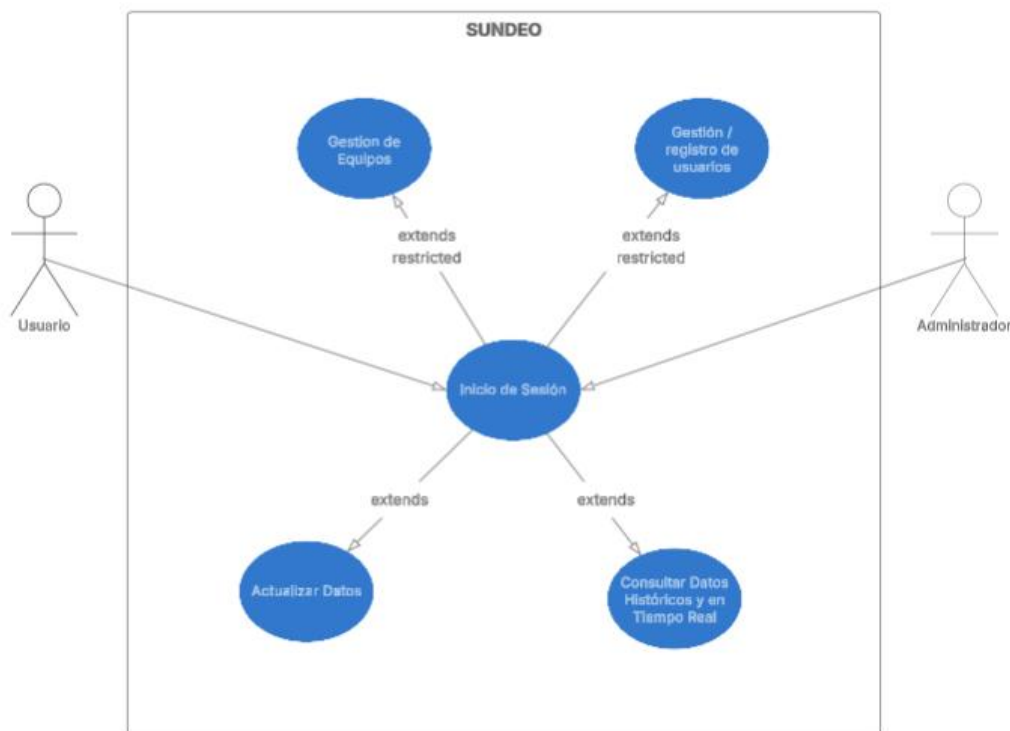
Relación: Muchos registros de producción pueden asociarse a una sola planta.

Diagrama de Casos de uso:

El siguiente diagrama de casos de uso representa de manera gráfica las principales interacciones entre los actores del sistema y las funcionalidades disponibles según su rol. En SUNDEO se distinguen dos perfiles de usuario: el **cliente** y el **administrador**.

- El **cliente** puede iniciar sesión, consultar tanto sus datos históricos como en tiempo real relacionados con la producción solar, y gestionar su información personal.
- El **administrador**, por su parte, tiene acceso a funcionalidades más amplias que incluyen la gestión de usuarios y plantas, así como la supervisión general de los datos de producción, ya sea a nivel global o por cliente específico.

Este diagrama permite visualizar claramente los distintos casos de uso asociados a cada tipo de usuario, facilitando la comprensión de los requisitos funcionales del sistema.



Estructura del Proyecto:

(Consultar anexo 1)

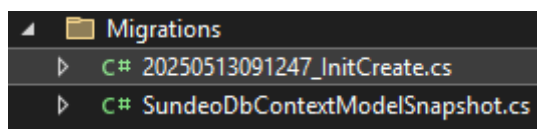
La estructura del proyecto SUNDEO está compuesta por tres capas principales:

Backend:

- Implementado en **ASP.NET Core**.
- Contiene controladores para la autenticación, gestión de plantas y consulta de producción, así como usuarios.
- Seguridad gestionada con **JWT** para sesiones stateless.
- ORM: **Entity Framework Core** con enfoque Code First.

Base de datos:

- SQL Server como motor relacional.
- Estructura normalizada en 3NF.
- Tablas principales: Users, SolarPlants, EnergyProductions, entre otras.



- Migraciones automatizadas desde código C#.

Frontend:

- Desarrollo en **React** con posibilidad de usar **Blazor WASM** en ciertas vistas.
- Integración de **Three.js** para la visualización tridimensional.
- Uso de componentes reutilizables, rutas protegidas y diseño responsive.
- Gestión de estado con hooks y localStorage.

Análisis del Código:

(Consultar anexo 3)

Definición de parámetros mediante appsettings:

La configuración principal del **backend** de la aplicación se establece a través de un archivo JSON, donde se definen parámetros esenciales para su funcionamiento.

En primer lugar, se incluye la configuración de los niveles de registro (**logging**), que permite establecer el nivel de detalle con el que se registrarán los eventos durante la ejecución de la aplicación. Se utiliza el nivel **Information** por defecto, y se establece un filtrado para limitar los mensajes provenientes de los componentes de ASP.NET Core, reduciendo así el ruido en los registros del sistema.

A continuación, se define la cadena de conexión a la base de datos relacional. En este caso, se utiliza un servidor local y una base de datos denominada **sundeo_db**, con una conexión de confianza habilitada y un certificado **auto-firmado** aceptado, lo que permite facilitar el desarrollo local sin necesidad de infraestructura adicional.

Finalmente, se establece la configuración del sistema de autenticación mediante **JWT (JSON Web Tokens)**. Se detallan aspectos como la audiencia prevista del token, el emisor autorizado, la clave secreta utilizada para su firma y la duración de validez del token, en este caso limitada a tres minutos. Esta configuración es crítica para garantizar un mecanismo seguro de autenticación **stateless**, donde el servidor puede validar cada solicitud sin mantener estado de sesión.

Lógica de Autenticación: Login (Controlador en ASP.NET Core):

El sistema de autenticación de la aplicación se implementa mediante un controlador en **ASP.NET Core** que expone un endpoint para el inicio de sesión de los usuarios. Este endpoint se encuentra anotado con metadatos que definen tanto su comportamiento como sus restricciones de acceso.

La anotación **[HttpPost("login")]** indica que el método responderá a solicitudes **HTTP** de tipo **POST** dirigidas a la ruta **/api/auth/login**. Esta convención es coherente con los principios **RESTful**, en los que las operaciones de escritura, como la autenticación, deben realizarse mediante **POST**. Por su parte, la anotación **[AllowAnonymous]** permite que esta acción sea accesible sin necesidad de autenticación previa, algo imprescindible en cualquier punto de entrada para usuarios aún no identificados.

El método comienza extrayendo del cuerpo de la solicitud las credenciales del usuario, encapsuladas en un objeto de tipo **LoginRequest**. Posteriormente, se realiza una consulta a la base de datos para verificar si el nombre de usuario proporcionado existe. Si no es así, se devuelve una respuesta de tipo **401 Unauthorized**, señalando que la autenticación ha fallado.

En caso de que el usuario exista, se procede a validar la contraseña. Para ello, se utiliza la biblioteca **BCrypt**, la cual permite comparar de forma segura la contraseña en texto plano con el hash almacenado en la base de datos. Esta medida incrementa significativamente la seguridad frente a ataques por diccionario o fuerza bruta.

Si las credenciales son correctas, se genera un conjunto de *claims* o afirmaciones de identidad que incluyen información clave del usuario, como su rol, nombre de usuario y su identificador. Estas afirmaciones se integran en un token **JWT (JSON Web Token)**, el cual será firmado criptográficamente utilizando la clave secreta definida en el archivo de configuración.

Antes de emitir el token, el sistema comprueba la validez del tiempo de vida configurado. En caso de que no se pueda interpretar correctamente, se establece un valor por defecto de tres minutos (esto puede cambiarse según corresponda). Finalmente, se genera y firma el token, y este es devuelto al cliente en la respuesta. A partir de ese momento, el usuario podrá incluir este token en la cabecera de autenticación en cada solicitud para acceder a los recursos protegidos del sistema, sin necesidad de mantener una sesión activa en el servidor.

Intercepción de axios:

En el cliente web, se configura una instancia personalizada de **Axios**, el cliente **HTTP** utilizado para interactuar con la **API**. Dicha instancia incluye cabeceras por defecto para aceptar contenido **JSON**, y se complementa con lógica adicional para la gestión automática de tokens **JWT**.

Uno de los aspectos más relevantes es el uso de **interceptores**. Antes de enviar una solicitud, el interceptor verifica si existe un **token** almacenado en el navegador. Si lo hay, intenta decodificarlo y comprueba su validez comparando la hora actual con la fecha de expiración codificada en el propio **token**. En caso de que el **token** esté vencido, se elimina del almacenamiento local, se cancelan todas las solicitudes pendientes y se redirige al usuario a la pantalla de inicio de sesión. Si el token es válido, se añade automáticamente a la cabecera **Authorization** como un **Bearer Token**.

Asimismo, el interceptor de respuestas permite detectar errores **HTTP** relacionados con la autorización (códigos 401 o 403) y aplicar un procedimiento similar: cancelación de solicitudes, notificación al usuario y redirección.

Este enfoque no solo proporciona una autenticación persistente y segura, sino que también garantiza una experiencia de usuario fluida, evitando peticiones innecesarias y gestionando correctamente la sesión sin intervención manual.

RESULTADOS Y ANÁLISIS:

(Consultar anexo 4)

El sistema SUNDEO ha sido implementado de forma satisfactoria, cumpliendo con los objetivos planteados y validando la hipótesis de que una solución centrada en el usuario, con representación 3D interactiva y análisis numérico en tiempo real, mejora la comprensión y gestión de la producción solar en instalaciones de autoconsumo.

1. Inicio de sesión y autenticación

El sistema de autenticación implementado con **JWT** permite el acceso seguro a los usuarios mediante un formulario de inicio de sesión. Tras la autenticación, el token es almacenado en el navegador para autorizar las futuras peticiones.

2. Visualización general de producción

Una vez autenticado, el usuario accede a una vista principal donde se muestran métricas numéricas de producción, consumo y vertido a la red en tiempo real. Esta información se actualiza periódicamente a través de llamadas a la **API**.

3. Representación gráfica de datos

El sistema ofrece gráficos interactivos (usando **Recharts** o **Chart.js**) que representan la evolución temporal de la producción, el consumo y otros parámetros relevantes. Los datos pueden visualizarse por día, semana o mes, lo cual facilita el análisis de tendencias.

5. Gestión de usuarios y perfil

El sistema permite a los usuarios editar su perfil, actualizar su contraseña y visualizar los datos asociados a su cuenta. Esta gestión se realiza a través de formularios con validaciones y modales para una experiencia fluida.

6. Exportación de datos

Los datos energéticos y los gráficos pueden exportarse en formato CSV o imagen, lo cual es útil para el análisis externo, informes o respaldos. Esta funcionalidad mejora la utilidad de la herramienta para técnicos o investigadores.

7. Panel de administración

En caso de estar autenticado como administrador, el sistema permite gestionar todos los usuarios y plantas, así como tener una visión global de la producción por cliente o instalación.

5. CONCLUSIONES

El desarrollo del sistema SUNDEO representa un avance significativo en el ámbito del monitoreo energético para instalaciones solares de autoconsumo. A lo largo del proyecto, se ha identificado una carencia clara en el mercado de soluciones accesibles que combinen tanto el análisis numérico como visualizaciones interactivas en 3D para facilitar la comprensión de los flujos energéticos. SUNDEO responde a esta necesidad mediante una propuesta tecnológica innovadora que unifica backend robusto, almacenamiento histórico eficiente y un frontend inmersivo.

Entre las principales conclusiones destacan:

- **Innovación en visualización:** La integración de tecnologías **WebGL** (**Three.js/Babylon.js**) permite representar, por primera vez en este tipo de plataformas, el flujo energético dentro de un entorno arquitectónico tridimensional. Esta visualización no solo mejora la experiencia del usuario, sino que también facilita una comprensión más intuitiva de los desequilibrios energéticos y oportunidades de mejora.
- **Solidez tecnológica:** La elección de **ASP.NET Core** como núcleo del backend, junto con **Entity Framework Core** y **SQL Server**, proporciona una base sólida, escalable y mantenible. A su vez, la implementación de autenticación basada en **JWT** garantiza la seguridad sin sacrificar el rendimiento ni la experiencia de usuario.
- **Interfaz adaptable:** El uso de **React y Blazor WebAssembly** en el desarrollo del frontend abre la posibilidad de crear interfaces modernas, reactivas y compatibles con distintos dispositivos, facilitando el acceso a usuarios con distintos niveles de experiencia técnica.
- **Relevancia social y ambiental:** SUNDEO no solo actúa como herramienta de control técnico, sino también como medio para promover la conciencia energética entre los usuarios. Su potencial educativo y su capacidad para motivar un consumo más eficiente alinean el proyecto con los objetivos de desarrollo sostenible y la transición energética global.

En conclusión, SUNDEO es una propuesta tecnológica viable, relevante y diferenciadora, capaz de aportar valor tanto a nivel individual como comunitario. El sistema puede escalarse y adaptarse a diferentes contextos, y sienta las bases para futuras ampliaciones en el campo del análisis energético inteligente.

6. LÍNEAS DE INVESTIGACIÓN FUTURAS

1. Integración con IoT (medidores inteligentes):

Una de las proyecciones más relevantes consiste en la integración con dispositivos de Internet de las Cosas (IoT), específicamente medidores inteligentes conectados a la red. Esta integración permitiría capturar datos en tiempo real de consumo y producción energética sin intervención manual. Además, el uso de sensores distribuidos en la instalación permitiría generar una base de datos mucho más rica y precisa, abriendo la puerta a análisis temporales, detección de fallos, y mantenimiento predictivo.

2. Incorporación de elementos interactivos avanzados:

Actualmente, la visualización de datos en SUNDEO es funcional y clara, pero puede evolucionar hacia una experiencia más inmersiva e intuitiva. El desarrollo de nuevos elementos interactivos, como simulaciones personalizadas en 3D, paneles dinámicos o visualizaciones por realidad aumentada, permitiría al usuario comprender mejor la información que se le presenta y fomentar una mayor participación en el control y análisis de su producción energética.

3. Desarrollo de algoritmos predictivos:

La explotación de algoritmos de aprendizaje automático permitiría anticipar tanto el consumo energético del usuario como la generación estimada en función de variables como la meteorología, la época del año, o los patrones de comportamiento del hogar. Esta capacidad predictiva dotaría al sistema de valor añadido, ofreciendo recomendaciones de uso, alertas ante anomalías o estimaciones de ahorro energético.

4. Aplicación móvil nativa

Finalmente, se contempla el desarrollo de una aplicación móvil nativa (para Android y/o iOS) que permita al usuario recibir notificaciones en tiempo real, consultar datos históricos y actuales, y gestionar su perfil energético desde cualquier lugar. Esta evolución es clave para mejorar la accesibilidad y fomentar el uso cotidiano del sistema, favoreciendo un vínculo más estrecho entre el usuario y su consumo energético.

7. BIBLIOGRAFÍAS

1. SolarEdge. (s. f.). *Monitoring Platform* | SolarEdge. <https://monitoring.solaredge.com/>
2. Huawei. (s. f.). *FusionSolar Smart PV Management System*. <https://support.huawei.com/enterprise/en/solar/fusionsolar-pid-20225210>
3. Enphase. (s. f.). *Enlighten – Enphase Energy*. <https://enlighten.enphaseenergy.com/>
4. SMA Solar Technology. (s. f.). *Sunny Portal*. <https://www.sunnyportal.com/>
5. JWT.io. (s. f.). *Introduction to JSON Web Tokens*. <https://jwt.io/introduction>
6. Microsoft. (s. f.). *SQL Server documentation*. <https://learn.microsoft.com/en-us/sql/sql-server/>
7. Microsoft. (s. f.). *Entity Framework Core documentation*. <https://learn.microsoft.com/en-us/ef/core/>
8. Microsoft. (s. f.). *Create a Web API with ASP.NET Core*. <https://learn.microsoft.com/en-us/aspnet/core/web-api/>
9. Three.js. (s. f.). *Three.js – JavaScript 3D library*. <https://threejs.org/>
10. Babylon.js. (s. f.). *Babylon.js: Powerful, beautiful, simple, open – Web rendering engine*. <https://www.babylonjs.com/>
11. React. (s. f.). *React – A JavaScript library for building user interfaces*. <https://react.dev/>
12. Microsoft. (s. f.). *ASP.NET Core Blazor WebAssembly*. <https://learn.microsoft.com/en-us/aspnet/core/blazor/>
13. GanttPRO. (s. f.). *Gantt chart online software*. <https://ganttpro.com/>
14. Trello. (s. f.). *Trello: Manage your team's projects from anywhere*. <https://trello.com/>
15. SolarEdge Technologies Inc. (s. f.). *Monitoring Platform*. <https://monitoring.solaredge.com/>

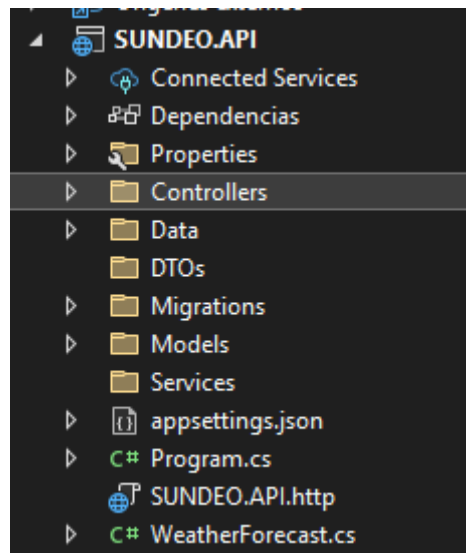
8. ANEXOS

1. Estructura del proyecto:

En esta sección se recogen todas las imágenes que muestran la estructura (en carpetas) del proyecto.

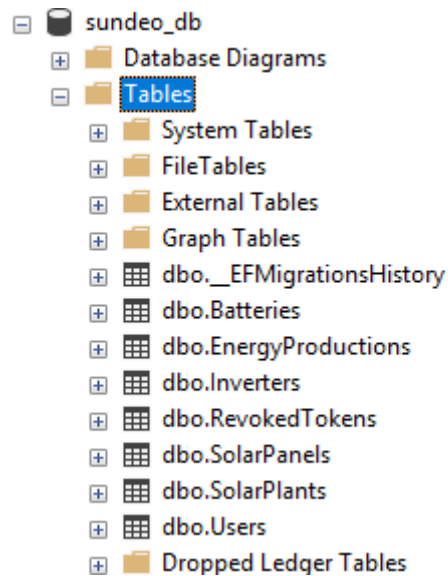
1-Estructura Backend:

El backend (**API**) se divide según sus secciones, con el programa principal (Program.cs) junto al archivo de configuración (appsettings.json) en la raíz de la solución, luego en subcarpetas se dividen los componentes en Controllers que denotan las llamadas a cada entidad de la BBDD, Models que denotan las entidades con sus propiedades y Migrations que contienen los datos para regenerar la BBDD sin necesidad del gestor ni script especializados.



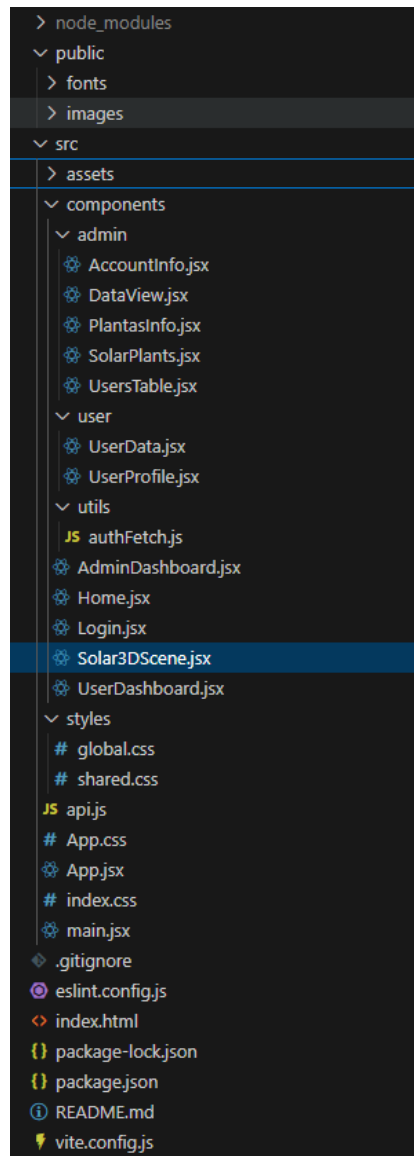
2-Estructura BBDD:

La BBDD la cual se crea a partir de las migraciones encerradas en la **API** se divide en **dbos (data base owners)** que representarían cada una de las tablas, con sus respectivos campos y claves



3-Estructura Frontend:

Esta se divide en las carpetas de recursos: images y fonts, la de estilos Styles y la carpeta de componentes principalmente, dentro de los componentes se encuentran los compartidos en su raíz y los de acceso de administrador o usuario en su respectiva subcarpeta

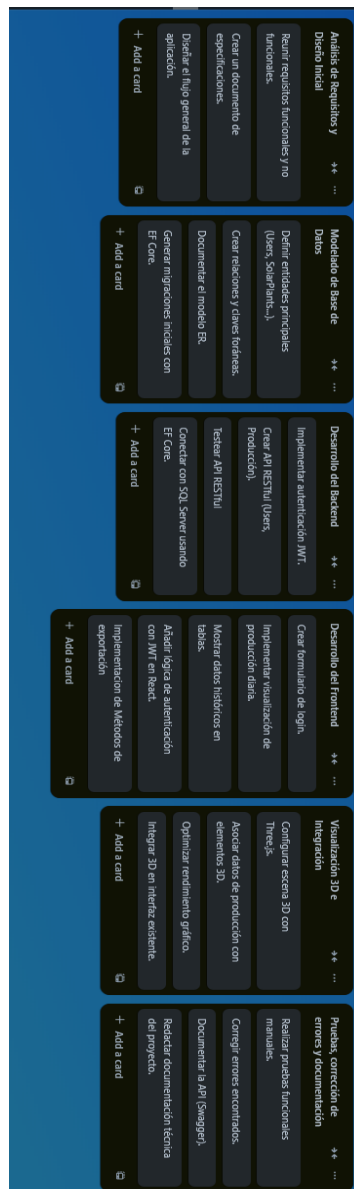


2. Planificación del desarrollo del proyecto:

Este apartado encierra las imágenes detalladas de las distintas fases de planificación y desarrollo del proyecto

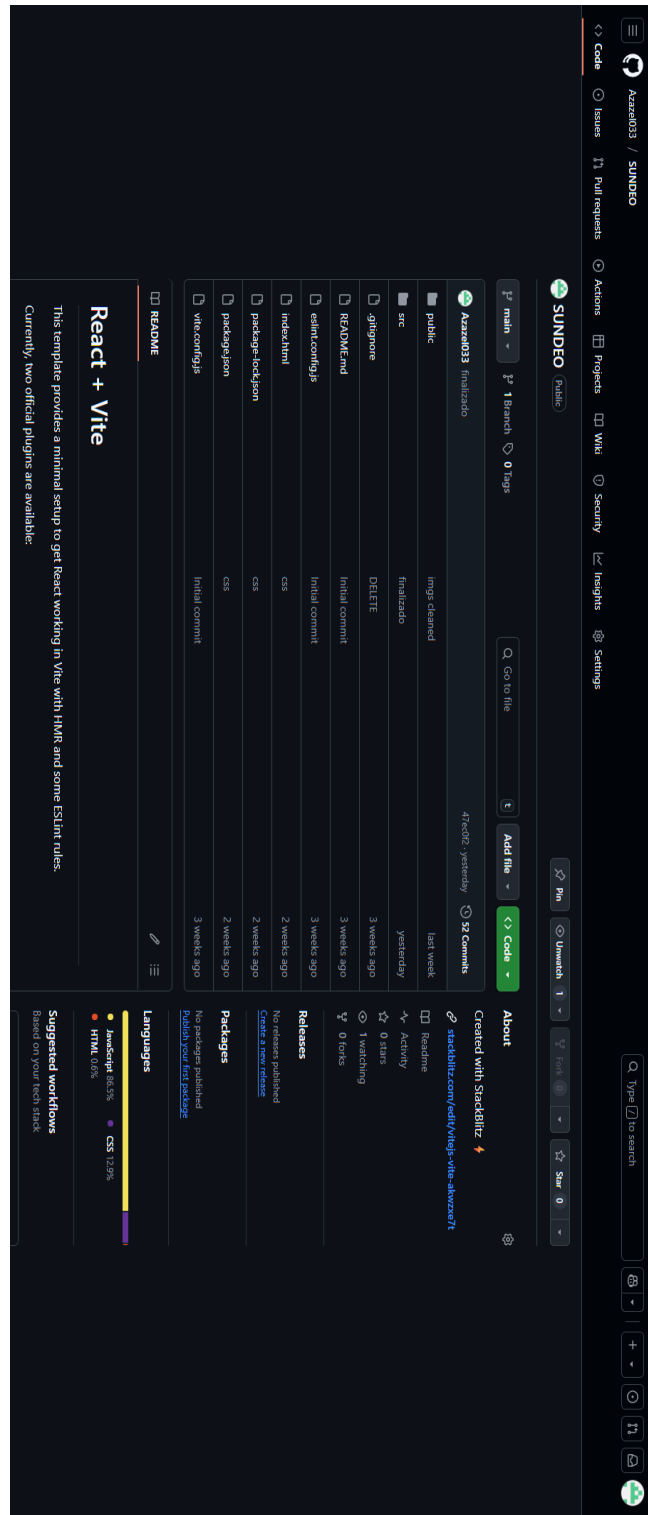
Tabla de planificación (Trello):

El esquema de Trello muestra una primera planificación del desarrollo del proyecto denotando las fases principales y las tareas de cada una.



5-Proyecto de GIT:

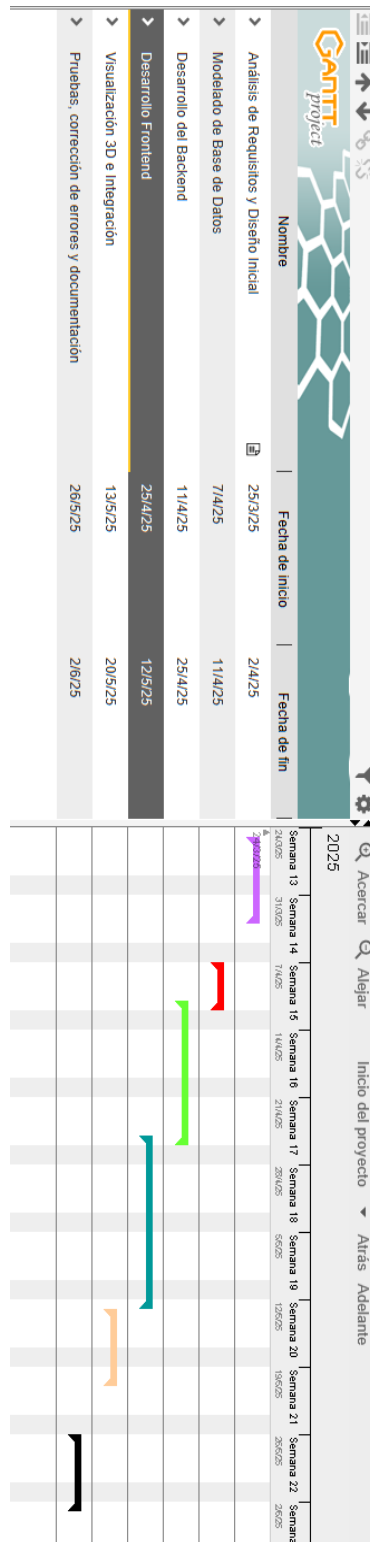
Una captura de la raíz del repositorio de GitHub del Proyecto, el cual es público y accesible desde la URL: <https://github.com/Azazel033/SUNDEO>



The screenshot shows the GitHub repository page for **Azazel033 / SUNDEO**. The repository is public and has 1 branch and 0 tags. The main branch is selected, showing a commit from 47 seconds ago with 52 comments. The repository contains several files and folders, including `public`, `src`, `.gitignore`, `README.md`, `eslint.config.js`, `index.html`, `package-lock.json`, `package.json`, and `vite.config.js`. The repository is created with StackBlitz and includes a README section for **React + Vite**, stating that the template provides a minimal setup to get React working in Vite with HMR and some ESLint rules. The repository also shows a list of languages (JavaScript 88.5%, HTML 0.0%, CSS 12.2%), suggested workflows, and a section for releases and packages.

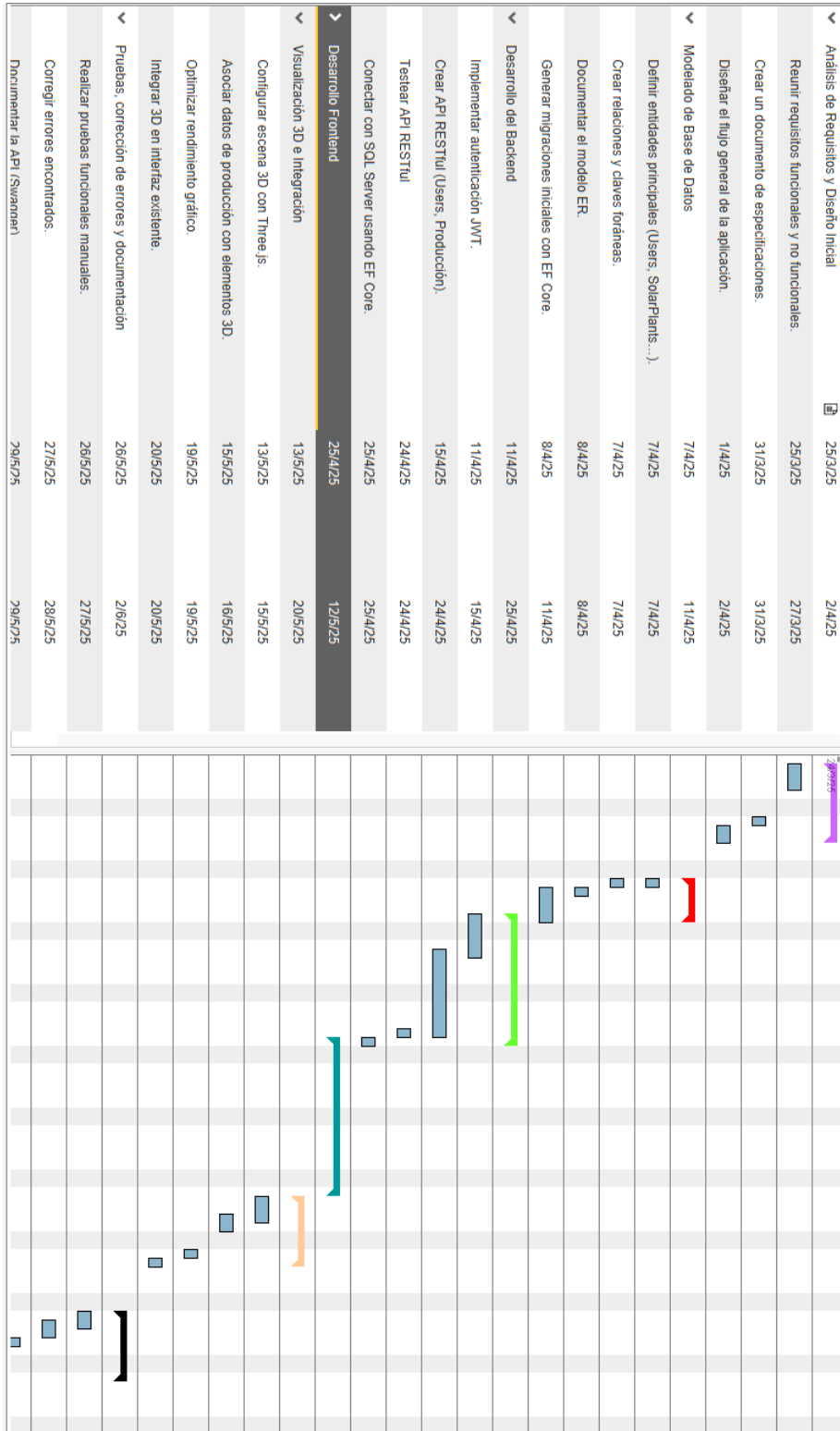
6-Diagrama de Gantt (Contraído):

Aquí se muestra el diagrama de Gantt resultante luego de la realización del proyecto con los días que se le dedico a cada fragmento de manera contraída por faceta.



7-Diagrama de Gantt (expandido):

Por otro lado aquí está el mismo diagrama expandido para apreciar cada tarea.



3. Análisis del Código:

Esta sección muestra capturas de los fragmentos más relevantes del código de la aplicación.

Appsettings:

Es el archivo **.json** en el que se recogen parámetros flexibles necesarios para la ejecución, puede editarse para cambiar los valores sin necesidad de tocar el código.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection": "Server=localhost;Database=sundeo_db;Trusted_Connection=True;TrustServerCertificate=True;"
  },
  "JwtSettings": {
    "Audience": "SundeoUsers",
    "Issuer": "Sundeo",
    "SecretKey": "ClaveSecretaMuySeguraQueTieneMinimo256BitsDeLargo",
    "TokenLifetime": "00:03:00"
  }
}
```

Método de autenticación(logging):

Este fragmento muestra la llamada a la API que verifica los datos del usuario y con ellos crea en caso de ser correctos un token **JWT** que se empleará tanto en la navegación en el frontend como en las consultas restringidas del backend.

```
// Endpoint POST para el login de usuarios
[HttpPost("login")]
[AllowAnonymous] // Permite el acceso sin autenticación previa
0 referencias
public async Task<IActionResult> Login([FromBody] LoginRequest request)
{
    // Busca al usuario en la base de datos por nombre de usuario
    var user = await _context.Set<User>()
        .FirstOrDefaultAsync(u => u.Username == request.Username);

    // Si el usuario no existe, retorna error 401 (No autorizado)
    if (user == null)
    {
        return Unauthorized("Usuario no encontrado");
    }

    // Verifica que la contraseña proporcionada coincida con el hash almacenado
    if (!BCrypt.Net.BCrypt.Verify(request.Password, user.PasswordHash))
    {
        return Unauthorized("Contraseña incorrecta");
    }

    // Se definen los claims (información que se incluirá en el token JWT)
    var claims = new[]
    {
        new Claim(ClaimTypes.Role, user.Role), // Para manejo del backend
        new Claim("Username", user.Username), // Nombre de usuario
        new Claim("Role", user.Role), // Para manejo desde el frontend
        new Claim("UserId", user.UserId.ToString()) // ID del usuario
    };

    // Se obtienen los valores de configuración para el JWT
    var jwtSettings = _configuration.GetSection("JwtSettings");

    // Se genera la clave simétrica a partir de la clave secreta configurada
    var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSettings["SecretKey"]));
    var credentials = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

    // Se define la duración del token; si no se puede parsear, se usa un valor por defecto
    TimeSpan tokenLifetime;
    if (!TimeSpan.TryParse(jwtSettings["TokenLifetime"], out tokenLifetime))
    {
        tokenLifetime = TimeSpan.FromMinutes(3); // Valor por defecto si hay error en la configuración
    }

    // Se genera el token JWT con los claims, emisor, audiencia, expiración y credenciales
    var token = new JwtSecurityToken(
        issuer: jwtSettings["Issuer"],
        audience: jwtSettings["Audience"],
        claims: claims,
        expires: DateTime.Now.Add(tokenLifetime),
        signingCredentials: credentials
    );

    // Convierte el token a string para enviarlo al cliente
    var tokenHandler = new JwtSecurityTokenHandler();
    var jwtToken = tokenHandler.WriteToken(token);

    // Devuelve el token en la respuesta (status 200 OK)
    return Ok(new { Token = jwtToken });
}
```


Intercepción de axios 1:

Parte del código empleado para modificar los axios utilizados en las llamadas a la **API** en estos se define una estructura base para las llamadas, además de que se pasan los parámetros (headers) necesarios como es el token del usuario. Además de que se controlan los resultados de las llamadas y en caso de ser 401 o 403 se devuelve a la ventana de login ya que esto indica que el usuario no está correctamente identificado, o que su tiempo de navegación ha expirado.

```
import axios from 'axios';
import { jwtDecode } from "jwt-decode";

const api = axios.create({
  baseURL: 'http://localhost:5130/api',
  headers: {
    'Accept': 'text/plain',
    'Content-Type': 'application/json',
  },
});

// Interceptor para agregar token y validar si está expirado
api.interceptors.request.use(config => {
  const token = localStorage.getItem('token');
  if (token) {
    // Decodificar el token para verificar su fecha de expiración
    try {
      const decodedToken = jwtDecode(token);
      const expirationTime = decodedToken.exp * 1000;
      const currentTime = Date.now();

      if (currentTime >= expirationTime) {
        // Si el token ha expirado, eliminarlo y redirigir al login
        localStorage.removeItem('token');
        alert('Sesión expirada. Por favor, inicia sesión nuevamente.');
```

```
      setTimeout(() => {
        window.location.href = '/login';
      }, 1500);
    } else {
      config.headers.Authorization = `Bearer ${token}`; // Agregar token a la cabecera
    }
  } catch (error) {
    // Si ocurre algún error al decodificar el token (token mal formado, etc.)
    console.error("Error al decodificar el token:", error);
    localStorage.removeItem('token');
    alert('Token inválido. Por favor, inicia sesión nuevamente.');
```

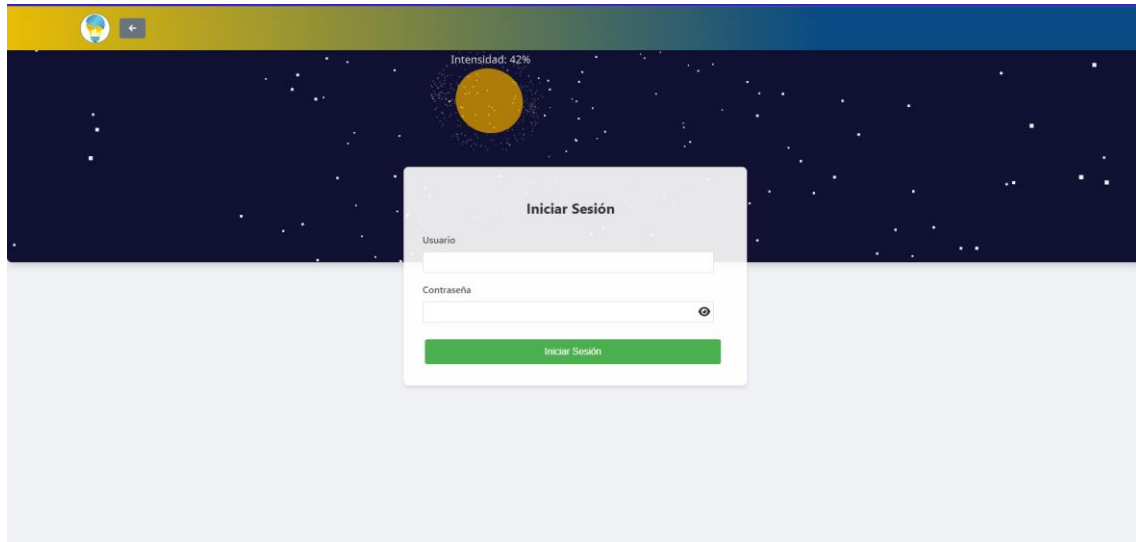
```
    setTimeout(() => {
  }
);

export default api;
```

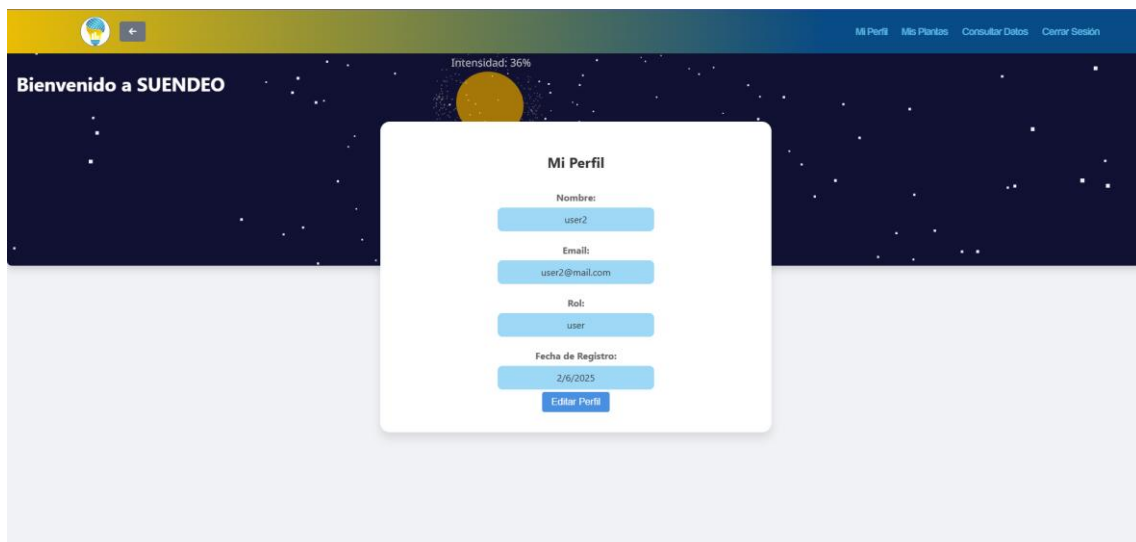
4. Resultados

Imágenes de la web en tiempo de ejecución real.

Inicio de sesión:



Dashboard (Vista principal/usuario):



Dashboard (Vista principal/administrador):

Intensidad: 36%

Gestión de Usuarios

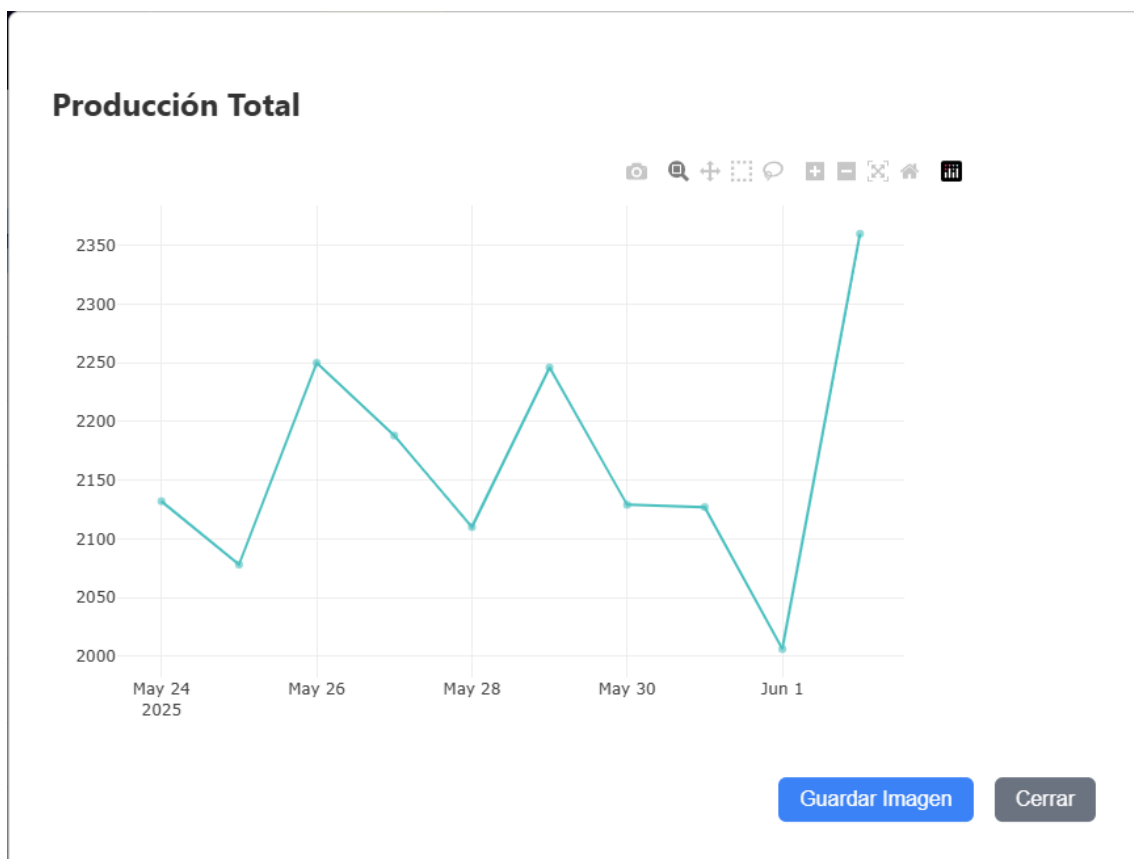
[Agregar Usuario](#)

Filtrar usuarios...

Nombre	Email	Fecha de Registro	Rol	Acciones
Daniel	correo@gmail.com	2/6/2025	admin	Inspeccionar Eliminar
user1	user1@mail.com	2/6/2025	user	Inspeccionar Eliminar
user2	user2@mail.com	2/6/2025	user	Inspeccionar Eliminar
user3	user3@mail.com	2/6/2025	user	Inspeccionar Eliminar
user4	user4@mail.com	2/6/2025	user	Inspeccionar Eliminar

Gráficos Interactivos:

Ejemplo de gráfico interactivo obtenido a partir de los datos recogidos en la BBDD.



Gestión de perfil:

Editar Perfil

Nombre de Usuario:

Email:

Contraseña Actual:

Nueva Contraseña:

Confirmar Nueva Contraseña:

Guardar Cambios

Cancelar

Exportación a CSV:

Opción que permite la obtención de un archivo .csv completamente funcional de los datos registrados.

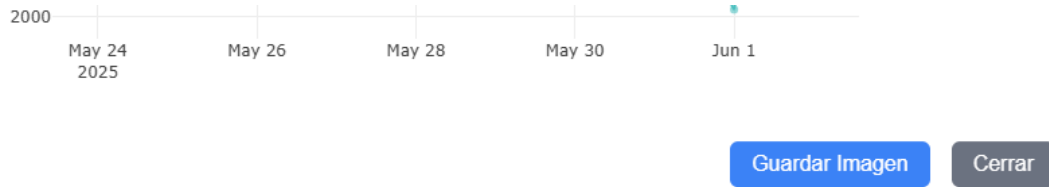
Producción Total Diaria

Ver Gráfico

Exportar CSV

Exportación a imagen:

Opción que permite almacenar la parte de la grafica deseada en una imagen .png



Panel de Administración:

Gestión de Usuarios

[Agregar Usuario](#)

Filtrar usuarios...

Nombre	Email	Fecha de Registro	Rol	Acciones
Daniel	correo@gmail.com	2/6/2025	admin	Inspeccionar Eliminar
user1	user1@mail.com	2/6/2025	user	Inspeccionar Eliminar
user2	user2@mail.com	2/6/2025	user	Inspeccionar Eliminar
user3	user3@mail.com	2/6/2025	user	Inspeccionar Eliminar
user4	user4@mail.com	2/6/2025	user	Inspeccionar Eliminar

5. Plataformas de referencia:

Logotipos de las Plataformas de medición energética utilizadas como referencia.

FusionSolar:



Enphase Enlighten:



SMA Sunny:



SolarEdge:

