MEMORIA FINAL DEL PROYECTO

VIRTUAL ORIGEN: SMART HOME

CICLO FORMATIVO DE GRADO SUPERIOR

DESARROLLO DE APLICACIONES MULTIPLATAFORMA

CURSO 2023-2024

AUTOR:

Sergio de Iscar Valera

TUTOR:

Javier Palacios González

DEPARTAMENTO DE INFORMÁTICA Y COMUNICACIONES

I.E.S. LUIS VIVES

RESUMEN

Virtual Origen: Smart Home:

Se propone el desarrollo de una aplicación enfocada en la optimización del uso de la energía en una finca tecnológicamente modernizada en las afueras de Cebolla, Toledo. Este proyecto surge de la necesidad de gestionar de manera eficiente y automatizada los recursos energéticos de la finca, que se abastece exclusivamente de energía solar almacenada en baterías, debido a la ausencia de suministro eléctrico convencional. La aplicación permitirá el manejo inteligente de dispositivos de alto consumo y sistemas de riego, adaptándose a variables como el clima y el estado de las baterías. Incorporará una interfaz cliente intuitiva que facilitará el control y monitoreo remoto de los dispositivos. Este proyecto no solo aborda una necesidad personal de gestión energética más eficaz, sino que también establece un modelo para aplicaciones similares en contextos donde la autonomía energética es crucial.

Virtual Origen: Smart Home

This project proposes the development of an application focused on optimizing energy usage in a technologically advanced estate located on the outskirts of Cebolla, Toledo. This initiative stems from the need to manage the estate's energy resources efficiently and automatically, which rely solely on solar power stored in batteries due to the lack of conventional electrical supply. The application will facilitate intelligent management of high-consumption devices and irrigation systems, adapting to variables such as weather conditions and battery status. It will incorporate an intuitive client interface that will enable remote control and monitoring of the devices. This project not only addresses a personal need for more effective energy management but also sets a benchmark for similar applications in contexts where energy independence is crucial.

Contenido

Capítulo 1 - Introducción

En nuestra era, la tecnología se ha convertido en una herramienta fundamental para abordar los desafíos globales, particularmente en el ámbito de la sostenibilidad y la eficiencia energética. A través de mi proyecto "Virtual Origen: Smart Home", he buscado aplicar innovaciones tecnológicas para mejorar la gestión de los recursos energéticos en entornos residenciales. Este trabajo, desarrollado como parte de mi formación en el CFGS de Desarrollo de Aplicaciones Multiplataforma, se enfoca en la creación de una solución práctica y sostenible para una finca situada en las afueras de Cebolla, Toledo, un lugar donde la conexión a la red eléctrica convencional es inexistente.

La idea de "Virtual Origen" surgió a raíz de mi propia experiencia tras adquirir una propiedad que carece de acceso a la electricidad tradicional. Esta situación me llevó a instalar un sistema de paneles solares con almacenamiento en baterías, enfrentándome al desafío de optimizar y gestionar eficientemente el consumo energético. Mi respuesta a este reto ha sido el desarrollo de una aplicación multiplataforma integrada en el Internet de las Cosas (IoT), que permite el control y la gestión remota y automatizada de todos los dispositivos eléctricos y sistemas de la finca.

Este proyecto no solo aborda una necesidad específica de gestión energética, sino que también incorpora tendencias emergentes en tecnología y prácticas sostenibles. Mediante una interfaz intuitiva y un backend avanzado, la aplicación facilita la interacción con el entorno de manera inteligente, adaptando el uso de dispositivos de alto consumo y sistemas de riego basándose en datos en tiempo real sobre el clima, el estado de las baterías y otros factores relevantes.

Con "Virtual Origen: Smart Home", mi objetivo ha sido no solo resolver un problema personal de gestión energética sino también contribuir al desarrollo de aplicaciones multiplataforma con una solución replicable que podría ser adaptada en otros contextos similares. Este proyecto representa un esfuerzo significativo hacia la promoción de la autonomía energética y la eficiencia en el uso de recursos naturales, demostrando cómo las tecnologías avanzadas pueden ser aplicadas para mejorar la sostenibilidad y la autonomía en la gestión del hogar.

1.1 Justificación del proyecto

La elección de este proyecto se originó a partir de mi adquisición de un terreno en las afueras de Cebolla, provincia de Toledo, en el año 2021. Esta propiedad, al estar ubicada en una zona sin acceso a la red eléctrica convencional, presentaba un desafío significativo en términos de suministro de energía. Ante esta situación, decidí implementar una solución autónoma de generación eléctrica mediante la instalación de paneles solares y un sistema de almacenamiento en baterías.

Con el tiempo, mientras avanzaba en la modernización tecnológica de la finca, se hizo evidente la necesidad de optimizar el uso de la energía generada y de gestionar diversos procesos de manera automatizada. Esta necesidad se veía acentuada por las limitaciones en la disponibilidad eléctrica y la importancia de mantener una eficiencia energética alta para garantizar la sustentabilidad y la autonomía del sistema.

El desarrollo de una aplicación multiplataforma orientada al Internet de las Cosas (IoT) se planteó como una solución innovadora a estos retos. Este enfoque permite la programación y control remotos de múltiples sistemas, como el riego y la gestión de dispositivos de alto consumo eléctrico (por ejemplo, calderas y estufas), ajustando su funcionamiento a condiciones variables como el clima, el estado de las baterías y otros factores relevantes.

El principal valor añadido de este proyecto radica en su capacidad para brindar una gestión energética eficiente y automatizada en un entorno descentralizado, proporcionando al usuario un control total y acceso a información crucial para la toma de decisiones informadas. Además, "Virtual Origen: Smart Home" no solo responde a una necesidad personal, sino que también sienta las bases para futuras aplicaciones en entornos similares, demostrando la viabilidad y el impacto positivo de las soluciones tecnológicas en la gestión sostenible de recursos.

Figura 1.1-1 : Pequeña smart home con placas solares (AI)

1.2 Objetivos

El propósito fundamental de este proyecto es alcanzar una gestión más eficiente de los recursos en un entorno descentralizado, mediante la implementación del Internet de las Cosas (IoT). Este enfoque busca proporcionar al usuario final una interfaz fácil de usar e intuitiva que le permita configurar y gestionar sus diversos dispositivos inteligentes, posibilitando la administración remota de los mismos.

Objetivos Específicos:

Optimización de Recursos: Implementar estrategias IoT para maximizar la eficiencia en la gestión de recursos, focalizando en la autonomía energética y la automatización de procesos.

Desarrollo de Interfaz de Usuario Intuitiva: Diseñar una interfaz de usuario intuitiva en la aplicación cliente, asegurando una experiencia amigable para el usuario final en la configuración y supervisión de dispositivos inteligentes.

Administración Remota: Posibilitar la administración remota de dispositivos inteligentes a través de la aplicación cliente, permitiendo al usuario controlar y monitorear sus sistemas desde cualquier ubicación.

Figura 1.2-1 : Casa aislada de entornos urbanos (AI)

Aplicación de Conocimientos Adquiridos:

Uso de base de datos no SQL (Firebase): Implementar una base de datos no SQL para almacenar y gestionar la información necesaria.

Utilización de Docker para el Sistema Backend Servidor: Desarrollar y desplegar el sistema Backend servidor utilizando Docker para garantizar portabilidad y eficiencia.

Programación en C# del Servidor: Desarrollar el servidor utilizando el lenguaje de programación C# para asegurar un rendimiento óptimo.

Empaquetado de la Aplicación Servidor: Realizar el empaquetado adecuado de la aplicación servidor para su distribución y despliegue eficientes.

Consumo de Diversas Fuentes de Datos (API): Integrar y consumir datos de diversas fuentes a través de APIs para enriquecer la información disponible.

Incorporación de Nuevos Conocimientos:

Uso de Flutter/Dart para la Aplicación Cliente: Explorar y aplicar el framework Flutter/Dart para el desarrollo de la interfaz cliente, ampliando las habilidades adquiridas en clase.

Tecnología MQTT para Lectura y Envío de Datos: Implementar la tecnología MQTT para la lectura de datos y el envío de comandos a los dispositivos, ampliando el espectro de conocimientos tecnológicos.

Despliegue en Web y PlayStore: Realizar el despliegue de la aplicación tanto en una plataforma web como en la PlayStore, abordando la distribución en distintos entornos.

Instalación y Configuración de Relés Wifi: Adquirir conocimientos sobre la instalación y configuración de relés wifi, así como la configuración de la red MQTT y la obtención de datos del inversor mediante el uso de Python y Bash.

1.3 Alcance

El proyecto "Virtual Origen: Smart Home" abarca el desarrollo y la implementación de una solución integral para la gestión eficiente de energía y dispositivos en una finca aislada. Este proyecto tiene como meta principal la creación de una aplicación multiplataforma que se interconecta con diversos dispositivos inteligentes a través de Internet de las Cosas (IoT), facilitando un control exhaustivo y automatizado tanto de la generación como del consumo energético.

1.3.1 Funcionalidades Principales:

- Registro y Autorización de Usuarios: Desarrollar un sistema robusto para el registro y la autorización de usuarios, garantizando la seguridad y privacidad de la información gestionada.
- Gestión de Propiedades: Implementar funcionalidades para que los usuarios puedan administrar propiedades (casas o fincas), permitiéndoles configurar y controlar múltiples ubicaciones de forma centralizada.
- Compartir Propiedades: Facilitar la gestión compartida de propiedades mediante un sistema de permisos, permitiendo a otros usuarios acceder y controlar dispositivos de manera colaborativa.

- Control de Dispositivos: Permitir la configuración y gestión de dispositivos conectados a cada propiedad, con opciones para personalizar el comportamiento de cada dispositivo según las necesidades específicas del usuario.
- Sistema de Datos y Automatización: Desarrollar un sistema que recolecte datos a través de MQTT y los almacene en una base de datos centralizada, permitiendo la automatización basada en datos en tiempo real, como el estado de las baterías y las condiciones climáticas.
- Visualización Gráfica de Datos: Integrar visualizaciones gráficas en la aplicación cliente para ofrecer a los usuarios una visión clara del estado de las baterías, el clima, el consumo energético actual y la actividad de los dispositivos en tiempo real.
- Configuración Avanzada de Dispositivos: Proporcionar opciones avanzadas para la configuración de dispositivos, incluyendo parámetros como el porcentaje mínimo de batería requerido, consumo máximo de energía, y ajustes basados en la probabilidad de lluvia y el horario.
- Creación y Carga de Escenas: Implementar funcionalidades para crear y cargar "escenas" o configuraciones predefinidas de dispositivos, que pueden ser activadas en cualquier momento según las necesidades del usuario.
- Versión Multiplataforma: Desarrollar la aplicación para ser accesible tanto en dispositivos Android como en formato web, asegurando una amplia compatibilidad y accesibilidad.
- Dockerización del Servidor: Utilizar Docker para el despliegue del sistema backend, garantizando un entorno controlado y fácilmente actualizable.

Figura 1.3.1-1 : Supuesta casa con una instalación fotovoltaica descentralizada (AI)

1.3.2 Tecnologías Utilizadas:

- Flutter: Para el desarrollo de la aplicación cliente, facilitando su uso tanto en web como en dispositivos móviles.
- Firebase: Como solución para la base de datos no SQL y servicios adicionales como la autenticación de usuarios y hosting.
- C#: Seleccionado para el desarrollo del sistema backend, aprovechando sus librerías para integraciones con MQTT y Firebase.
- Docker: Para la gestión del servidor, proporcionando un método eficiente y seguro para su despliegue y mantenimiento.

1.3.2.1 Frontal

Para el desarrollo de la aplicación cliente busque un framework que me permitiera realizar una aplicación intuitiva y visualmente atractiva al usuario final.
Algunas de las opciones fueron estas:

Flutter:

Flutter es un framework de UI desarrollado por Google que permite desarrollar aplicaciones nativas compiladas para móviles, web y escritorio desde una única base de código.

Ha sido mi opción por mi experiencia con este framework, por su agradable estética, por la cantidad de librerías de la comunidad y por la capacidad de compilar tanto para dispositivos móviles y web. Además, al ser una tecnología desarrollada con Google ofrece mejor integración con sistemas como Firebase o servicios de Google Maps lo cual favorece el desarrollo de esta aplicación.

Pros:

- Alto rendimiento y compilación a código nativo.
- Amplia comunidad y soporte continúo de Google.
- Rico ecosistema de widgets y fácil personalización de la UI.

Contras:

- Curva de aprendizaje relativamente alta para desarrolladores sin experiencia en Dart.
- El tamaño de la aplicación puede ser relativamente grande comparado con soluciones nativas.

React Native:

React Native es un framework de Facebook que permite desarrollar aplicaciones móviles usando JavaScript y React.
Aunque React Native es una excelente opción para desarrollo rápido y eficiente de aplicaciones móviles multiplataforma, no fue seleccionado para este proyecto debido a que buscaba una solución que ofreciera un rendimiento más cercano al nativo y una integración más profunda con dispositivos IoT. Flutter, al compilar a código nativo y tener un soporte robusto para interfaces ricas y personalizadas, se adaptó mejor a las necesidades específicas del proyecto en términos de rendimiento y experiencia de usuario. Además de que no tengo tanta experiencia desarrollando con esta tecnología.

Pros:

- Permite a los desarrolladores que ya conocen JavaScript aprovechar sus conocimientos.
- Gran comunidad y muchas librerías disponibles.
- Integración con código nativo para optimización de rendimiento.

Contras:

- Rendimiento menor comparado con las aplicaciones completamente nativas.
- Gestión de dependencias a veces complicada con actualizaciones frecuentes.

Kotlin para Android:

Kotlin es un lenguaje de programación que puede ser usado para desarrollar aplicaciones nativas para Android. Es un lenguaje moderno y poderoso para desarrollo Android nativo, sin embargo, su uso habría limitado la aplicación a una única plataforma, lo cual contradice el objetivo de crear una solución verdaderamente multiplataforma. Al optar por Flutter, se aseguró que la aplicación podría funcionar tanto en Android como en iOS, además de ofrecer capacidades para una futura expansión a plataformas web o de escritorio.

Pros:

- Integración directa y soporte completo en el ecosistema Android.
- Más seguro y conciso que Java, con muchas características modernas de lenguaje.
- Buen soporte de la comunidad y de JetBrains.

Contras:

- Limitado a la plataforma Android, sin soporte directo para iOS o web.
- Menor cantidad de recursos y ejemplos comparados con tecnologías más antiguas como Java.

1.3.2.2 Sistema servidor:

C#:

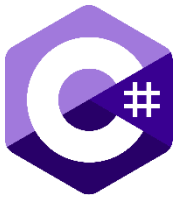
C# es un lenguaje de programación moderno, orientado a objetos, desarrollado por Microsoft como parte de .NET. Ofrece una integración rápida y sencilla con Docker y tengo experiencia desarrollando en el ecosistema de NET en diversos ámbitos. Además de contar con librerías para el uso de MQTT y para conectarme a la base de datos de Firebase, lo cual es esencial en este proyecto.

Pros:

- Fácil integración con Docker.
- Experiencia previa con el lenguaje en otros proyectos.
- Lenguaje robusto con buen soporte para programación asincrónica y concurrencia.
- Amplia gama de bibliotecas y marcos de trabajo.

Contras:

- Menor flexibilidad en entornos que no son de Microsoft.
- Puede ser más pesado en términos de recursos comparado con otros lenguajes como Python.



Python:

Python es un lenguaje de alto nivel que es popular en la ciencia de datos, desarrollo web, y automatización.

Python lo he usado en otras partes de este mismo proyecto, pero para el sistema servidor quiero un lenguaje del que me pueda fiar y tenga más experiencia, ya que es una parte critica de la infraestructura.

Pros:

- Sintaxis sencilla y fácil de aprender.
- Amplia variedad de bibliotecas y marcos de trabajo como Django y Flask.
- Fuerte en tareas de scripting, automatización y análisis de datos.

Contras:

- Menor rendimiento comparado con lenguajes compilados como C#.
- Manejo de dependencias a veces complejo.

Java:

Java es un lenguaje de programación orientado a objetos que es ampliamente utilizado en empresas y aplicaciones Android.
Al igual que Python Java es una buena opción para el sistema de servidor, ya que cuenta con librerías para hacer uso de la tecnología MQTT pero no me decidí por este ya que no tenía tanta experiencia desarrollando aplicaciones servidor como con C#. Además, que me parece que su sintaxis en ciertos casos es mejorada por C#.

Pros:

- Plataforma cruzada gracias a la máquina virtual Java (JVM).
- Amplia biblioteca estándar y soporte de múltiples marcos de trabajo y herramientas.
- Fuerte en aplicaciones empresariales con requisitos de escalabilidad y robustez.

Contras:

- Sintaxis más verbosa comparada con otros lenguajes modernos.
- Requiere un tiempo de arranque y recursos de sistema significativos.

1.3.2.3 Base de datos:

Firebase:

Firebase es una plataforma de desarrollo de aplicaciones desarrollada por Google que proporciona servicios de backend como base de datos en tiempo real, autenticación, y hosting.

Me decidí por Firebase ya que cuenta con una rápida y fácil integración con aplicaciones Flutter, y además de la base de datos no SQL ofrece una gran cantidad de servicios en un principio a un precio gratuito.

Pros:

- Integración fácil y rápida con aplicaciones móviles y web Flutter.
- Gestión simplificada de la autenticación y la seguridad.
- Escalabilidad automática y mantenimiento reducido.
- Una amplia variedad de servicios.

Contras:

- Costos pueden escalar rápidamente con el uso.

PostgreSQL en Amazon RDS:

PostgreSQL es un sistema de gestión de bases de datos relacional objeto (ORDBMS) avanzado, conocido por su estabilidad, robustez y cumplimiento de estándares. Amazon RDS (Relational Database Service) permite la fácil configuración, operación y escalado de bases de datos relacionales en la nube, incluyendo PostgreSQL.

PostgreSQL es una base de datos relacional muy potente y Amazon RDS proporciona una excelente solución de gestión de bases de datos. Sin embargo, para este proyecto, se necesitaba una base de datos que pudiera escalar automáticamente y manejar solicitudes en tiempo real de manera eficiente sin la complejidad del escalado manual. Firebase ofreció un modelo más sencillo y directo de base de datos NoSQL, que era más adecuado para manejar las necesidades dinámicas y en tiempo real del IoT.

Pros:

- Fiabilidad y Seguridad: PostgreSQL es conocido por su alta conformidad con los estándares SQL y su robusta gestión de transacciones, lo cual es esencial para aplicaciones críticas.
- Escalabilidad y Mantenimiento: Amazon RDS facilita la escalabilidad con pocos clics y maneja tareas de mantenimiento como backups y parches automáticamente.
- Compatibilidad con Herramientas: Amplio soporte para herramientas de desarrollo y plataformas de análisis debido a su naturaleza SQL estándar.

Contras:

- Costo: Mientras que RDS maneja muchas operaciones de mantenimiento, puede ser más caro que manejar tu propio servidor si la carga de uso es predecible y constante.
- Limitaciones de Configuración: Aunque RDS ofrece mucha flexibilidad, ciertas configuraciones de bajo nivel no están disponibles para el usuario.

MongoDB Atlas:

MongoDB es una base de datos NoSQL orientada a documentos que ofrece una gran flexibilidad para almacenar y manipular datos. MongoDB Atlas es el servicio oficial de base de datos como servicio (DBaaS) ofrecido por MongoDB Inc., que permite desplegar, operar y escalar instancias de MongoDB en la nube de manera automatizada.

Aunque MongoDB Atlas ofrece escalabilidad y flexibilidad excepcionales para manejar grandes volúmenes de datos no estructurados, el modelo de datos y las transacciones de Firebase fueron considerados más adecuados para este proyecto debido a su integración nativa con otros servicios de Google y su enfoque más simplificado para la autenticación y sincronización de datos en tiempo real. Además, la estructura de costos y la gestión de la complejidad operativa de MongoDB Atlas no se alineaban completamente con las restricciones del proyecto.

Pros:

- Flexibilidad de Esquema: Ideal para aplicaciones que necesitan almacenar datos con estructuras variadas sin un esquema fijo.

- Escalabilidad Automática: MongoDB Atlas ofrece escalabilidad horizontal automática, lo cual es ideal para aplicaciones con crecimiento variable o impredecible.
- Integraciones y Herramientas: Amplio soporte de herramientas de desarrollo y fácil integración con otras plataformas y servicios gracias a APIs bien documentadas.

Contras:

- Costo en Escalabilidad: Aunque ofrece escalabilidad automática, los costos pueden aumentar significativamente con el uso intensivo de datos.
- Optimización de Consultas: Las consultas en MongoDB pueden requerir optimización manual para asegurar el rendimiento, especialmente en colecciones con grandes volúmenes de datos.

Este amplio alcance asegura que el proyecto no solo atienda a las necesidades inmediatas de gestión energética eficiente, sino que también establezca una base sólida para futuras expansiones y mejoras, con potencial para ser adaptado y aplicado en otros contextos similares.

Capítulo 2 - Desarrollo del proyecto

2.1 Análisis de la aplicación

Virtual Origen busca la eficiencia y el control remoto de entornos descentralizados, para lograr este objetivo la aplicación permite crear múltiples propiedades, que sirven para agrupar dispositivos, saber de dónde tomar los datos climatológicos y los datos del inversor. También permite la creación de múltiples dispositivos inteligentes, que identifica cada uno de los dispositivos domotizados reales, estos dispositivos permiten la siguiente configuración:

- Delimitar las franjas horarias en las que se va a encender.
- Establecer los días de la semana que se podrá activar el dispositivo.
- Marcar el rango de porcentaje de batería o/y watios de producción o/y watios de consumo o/y temperatura o/y probabilidad de precipitación.

Virtual Origen también permita la visualización de gráficos sobre la previsión climática del día y gráficos sobre los datos del inversor (porcentaje de batería, watios de producción y watios de consumo) a lo largo de las horas.

Las propiedades pueden ser compartidas entre usuarios de la app, estableciendo un sistema de permisos y garantizando la integridad de los datos aun cuando múltiples usuarios accedan y editen una misma información en tiempo real.

También permite la creación de escenas, una escena es una agrupación de configuración de diversos dispositivos, la utilidad de las escenas es la capacidad de almacenar diversas configuraciones que aplican a múltiples dispositivos y poder reestablecer esa configuración de manera sencilla. Un ejemplo podría ser una configuración de verano y otra de invierno.

Las nombradas son las funcionalidades con las que cuenta la aplicación cliente, pero esta no serviría sin el sistema servidor. Allí donde se quiera implementar este sistema abría que implementar una red en la que habría un servidor, que puede ser un ordenador monoplaca (en mi caso una Raspberry Pi 4), y múltiples placas Arduino conectadas a la red escuchando una cola MQTT.
A continuación muestro la solución empaquetada que permite conectar cualquier tipo de dispositivo a la solución Virtual Origen.

Figura 2.1-1 : Arte vectorial de una reunión analítica (AI)

En estas dos imágenes pueden observar una placa Arduino, capaz de estar conectada a la red WiFi, conectada a un relé y un interruptor manual. Con esta pequeña caja se puede domotizar casi cualquier dispositivo, ya que lo único que hay que hacer es proveer de corriente a la placa y pasar los cables que enciendan el dispositivo por el relé. El Arduino tiene un código que se encarga de escuchar la cola MQTT, si a dicha cola llega un uno esta cerrará el relé encendiendo el dispositivo, a su ver, si a la cola le llega un cero esta abrirá el relé cortando la corriente del dispositivo.

Figura 2.1-1 y Figura 2.1-2 : Ensamblado Arduino

En la primera imagen pueden ver el sistema servidor de alta disponibilidad, consta de dos Raspberry Pi 3 Model B conectadas a unos relés, una de ellas cuenta con las aplicaciones dockerizadas, la otra se encarga de leer por USB los datos que provee el inversor, si uno de los ordenadores monoplaca se apagase o perdiese conectividad el otro ordenador lo detectaría y se encarga de reiniciarla a través de los relés.

Figura 2.1-3 : Servidor alta disponibilidad. Figura 2.1-4 : Inversor de placas fotovoltaicas

En la siguiente figura pueden observar lo que es un resumen de la arquitectura completa de la aplicación, desde la parte cliente hasta los relés encargados de la domotización de dispositivos.

Con este resumen se busca que el lector entienda en su totalidad la arquitectura de la aplicación, el cliente puede acceder tanto de manera web como usando la aplicación Android, la aplicación lo que acaba haciendo es editar una serie de campos en la base de datos Firebase Firestore, que a su vez es compartida por el sistema servidor, brindando de esta manera la configuración editada por el usuario.

El sistema backend servidor cuenta con un total de cuatro aplicaciones:

- InversorNow: Solución encargada de escuchar una cola MQTT que brinda el inversor y guardar el último dato en la base de datos, en mi caso concreto no es directamente el inversor, ya que el inversor da los datos por USB, por lo que hay otro ordenador encargado de leer y transmitir los datos por MQTT. Los datos se emiten cada treinta segundos.
- InversorYesterday: Aplicación encargada de escuchar una cola MQTT que da la media de los datos del inversor cada quince minutos.
- Weather: Esté se encarga de tomar los datos climáticos pertinentes de la API OpenWeather, este se ejecuta una vez al día y guarda la predicción para el día de hoy.
- SmartDevice: Está solución es la encargada de encender o apagar los dispositivos dependiendo de su configuración y de los datos brindados por InversorNow y Wather. Este se ejecuta cada minuto. Por seguridad si no ha de encender siempre envía un cero, esto se hace ya que si por algún motivo de red el mensaje MQTT anterior no se ha escuchado el dispositivo seguirá encendido, de esta manera me aseguro de que los dispositivos solo se encienden cuando la configuración lo permita.

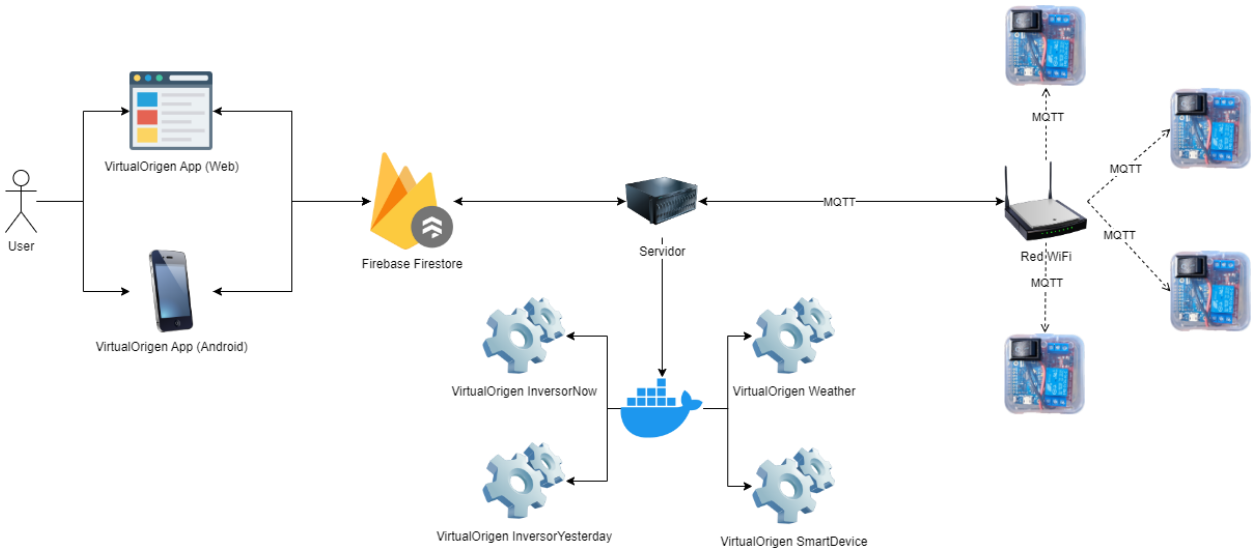


Figura 2.1-4 : Arquitectura Básica Completa

2.1.1 Diagrama E/R

El diagrama Entidad-Relación (ER) representa la estructura de datos y las relaciones entre las diferentes entidades involucradas en el proyecto "Virtual Origen: Smart Home". Este diseño es fundamental para organizar y gestionar la información de manera eficiente en la base de datos NoSQL de la aplicación. A continuación, se detalla la función de cada entidad y sus atributos principales:

Property:

Representa las diferentes propiedades o fincas gestionadas por la aplicación.

Atributo	Descripción
ID (PK)	Identificador único de la propiedad
Name	Nombre de la propiedad
Location	Ubicación de la propiedad
Color	Color asociado a la propiedad

Tabla 2.1.1-1 : Atributos de las propiedades

Smart Device:

Almacena información sobre los dispositivos inteligentes instalados en cada propiedad.

Atributo	Descripción
Property_ID (FK)	Identificador de la propiedad a la que pertenece el dispositivo
ID (PK)	Identificador único del dispositivo
Name	Nombre del dispositivo
Type	Tipo de dispositivo (e.g., riego, caldera)
Color	Color asociado al dispositivo
TimeZones	Franja horaria de operación
Days	Días de operación
BatteryRange	Rango de batería para la operación
ProductionRange	Rango de producción de energía para la operación
TemperatureRange	Rango de temperatura para la operación
RainRange	Rango de lluvia para la operación
IsManualMode	Indica si el dispositivo está en modo manual
IsOn	Indica si el dispositivo está encendido

Tabla 2.1.1-2 : Atributos de los dispositivos inteligentes

Scene:

Representa configuraciones predefinidas de dispositivos que pueden ser activadas en cualquier momento.

Atributo	Descripción
Property_ID (FK)	Identificador de la propiedad asociada a la escena
ID (PK)	Identificador único de la escena
Name	Nombre de la escena
Color	Color asociado a la escena

Tabla 2.1.1-3 : Atributos de las escenas

Day Weather:

Almacena datos meteorológicos diarios relevantes para la operación de los dispositivos.

Atributo	Descripción
Property_ID (FK)	Identificador de la propiedad asociada
DateTime (PK)	Fecha y hora de la información meteorológica
Temperature	Temperatura actual
TemperatureMin	Temperatura mínima
TemperatureMax	Temperatura máxima
Humidity	Humedad
Clounds	Estado de nubosidad
WindSpeed	Velocidad del viento
RainProbability	Probabilidad de lluvia
WeatherIconUrl	URL del icono meteorológico

Tabla 2.1.1-4 : Atributos del clima asociado a una propiedad

Inversor Now:

Descripción: Almacena datos en tiempo real del inversor de energía de la propiedad.

Atributo	Descripción
Property_ID (FK)	Identificador de la propiedad asociada
Battery	Estado actual de la batería
Consumption	Consumo actual de energía
Gain	Ganancia actual de energía

Tabla 2.1.1-5 : Atributos del último dato dado por el inversor

Inversor Yesterday:

Almacena datos históricos del inversor de energía del día anterior.

Atributo	Descripción
Property_ID (FK)	Identificador de la propiedad asociada
Datetime	Fecha y hora del registro
Battery	Estado de la batería
Consumption	Consumo de energía
Gain	Ganancia de energía

Tabla 2.1.1-6 : Atributos de un dato del inversor

Invitation:

Gestiona las invitaciones enviadas a otros usuarios para compartir el acceso a las propiedades.

Atributo	Descripción
FromEmail	Correo electrónico del remitente
FromProfileImage	Imagen de perfil del remitente
OwnerId	Identificador del propietario de la propiedad
PropertyName	Nombre de la propiedad
Property_ID (FK)	Identificador de la propiedad asociada
State	Estado de la invitación
IsNew	Indica si la invitación es nueva

Tabla 2.1.1-7 : Atributos de una invitación a una propiedad

Resumen del Diagrama ER

El diagrama ER de "Virtual Origen: Smart Home" muestra una estructura bien definida para gestionar propiedades, dispositivos inteligentes, escenas predefinidas, datos meteorológicos y estados del inversor en tiempo real e históricos. Cada entidad está diseñada para almacenar información específica que facilita la gestión y automatización de los recursos energéticos y dispositivos en una finca tecnológicamente avanzada. Las relaciones entre las entidades aseguran que los datos estén interconectados, permitiendo una gestión eficiente y optimizada dentro de la aplicación.

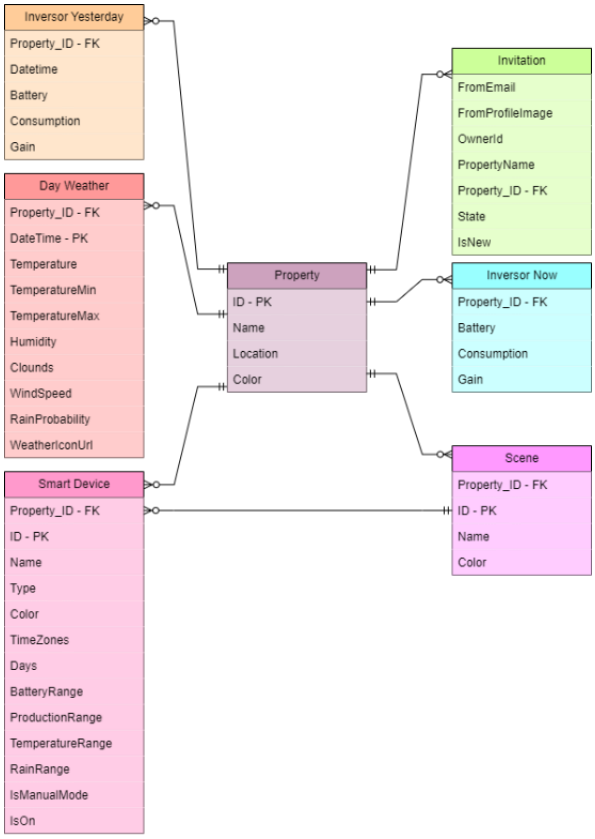


Figura 2.1.1-1 : Diagrama entidad relación

2.1.2 Diagrama de clases

El diagrama de clases del proyecto "Virtual Origen: Smart Home" representa la estructura de las clases que conforman la aplicación y las relaciones entre ellas. Este diseño es crucial para la implementación de la lógica de la aplicación, asegurando que cada componente interactúe correctamente con los demás. A continuación, se describe cada clase y sus principales atributos:

Property:

Representa las propiedades o fincas que son gestionadas por la aplicación.

Atributo	Tipo	Descripción
ID	String	Identificador único de la propiedad
Name	String	Nombre de la propiedad
Location	String	Ubicación de la propiedad
Color	String	Color asociado a la propiedad

Tabla 2.1.2-1 : Atributos de la clase propiedad

Smart Device:

Almacena información sobre los dispositivos inteligentes instalados en cada propiedad.

Atributo	Tipo	Descripción
ID	String	Identificador único del dispositivo
Name	String	Nombre del dispositivo
Type	String	Tipo de dispositivo (e.g., riego, caldera)
Color	String	Color asociado al dispositivo
TimeZones	List<DateTimeRange>	Lista de franjas horarias de operación
Days	List<Boolean>	Lista de días de operación
BatteryRange	String	Rango de batería para la operación
ProductionRange	String	Rango de producción de energía para la operación
TemperatureRange	String	Rango de temperatura para la operación
RainRange	String	Rango de lluvia para la operación
IsManualMode	Boolean	Indica si el dispositivo está en modo manual
IsOn	Boolean	Indica si el dispositivo está encendido

Tabla 2.1.2-2 : Atributos de la clase dispositivos inteligentes

Scene:

Representa configuraciones predefinidas de dispositivos que pueden ser activadas en cualquier momento.

Atributo	Tipo	Descripción
ID	String	Identificador único de la escena
Name	String	Nombre de la escena
Color	String	Color asociado a la escena

Tabla 2.1.2-3 : Atributos de la clase escena

Day Wather:

Almacena datos meteorológicos diarios relevantes para la operación de los dispositivos.

Atributo	Tipo	Descripción
DateTime	DateTimeRange	Fecha y hora de la información meteorológica
Temperature	Double	Temperatura actual
TemperatureMin	Double	Temperatura mínima
TemperatureMax	Double	Temperatura máxima
Humidity	Int	Humedad
Clounds	Int	Estado de nubosidad
WindSpeed	Double	Velocidad del viento
RainProbability	Double	Probabilidad de lluvia
WeatherIconUrl	String	URL del icono meteorológico

Tabla 2.1.2-4 : Atributos de la clase clima día

Inversor Now:

Almacena datos en tiempo real del inversor de energía de la propiedad.

Atributo	Tipo	Descripción
Battery	Int	Estado actual de la batería
Consumption	Int	Consumo actual de energía
Gain	Int	Ganancia actual de energía

Tabla 2.1.2-5 : Atributos de la clase que representa el último dato del inversor

Inversor Yesterday:

Almacena datos históricos del inversor de energía del día anterior

Atributo	Tipo	Descripción
DateTime	DateTime	Fecha y hora del registro
Battery	Int	Estado de la batería
Consumption	Int	Consumo de energía
Gain	Int	Ganancia de energía

Tabla 2.1.2-6 : Atributos de la clase que representa un dato del inversor

Invitation

Gestiona las invitaciones enviadas a otros usuarios para compartir el acceso a las propiedades.

Atributo	Tipo	Descripción
FromEmail	String	Correo electrónico del remitente
FromProfileImage	String	Imagen de perfil del remitente
OwnerId	String	Identificador del propietario de la propiedad
PropertyName	String	Nombre de la propiedad
State	Boolean	Estado de la invitación
IsNew	Boolean	Indica si la invitación es nueva

Tabla 2.1.2-7 : Atributos de la clase invitación

Resumen del Diagrama de Clases

El diagrama de clases de "Virtual Origen: Smart Home" ilustra la organización y estructura de las diferentes clases que componen la aplicación. Cada clase representa un componente crucial del sistema, desde la gestión de propiedades y dispositivos inteligentes hasta la administración de datos meteorológicos y estados del inversor. Las relaciones entre las clases aseguran una integración coherente y efectiva, permitiendo que la aplicación funcione de manera óptima. Este diagrama es esencial para guiar el desarrollo de la aplicación, proporcionando una visión clara de cómo cada parte del sistema debe interactuar con las demás.

Figura 2.1.2-1 : Diagrama de clases

2.1.3 Casos de uso

El diagrama de casos de uso del proyecto "Virtual Origen: Smart Home" proporciona una visión clara y estructurada de las interacciones entre los usuarios y las funcionalidades principales de la aplicación. Este diagrama es crucial para entender cómo los diferentes componentes del sistema se relacionan entre sí y cómo los usuarios pueden interactuar con la aplicación para gestionar eficientemente sus propiedades y dispositivos inteligentes. A continuación, se detalla cada uno de los módulos principales y sus respectivos casos de uso:

1. **Gestión de Scenes**

Este módulo permite a los usuarios gestionar escenas predefinidas, que son configuraciones específicas de dispositivos que pueden activarse en conjunto. Los casos de uso incluidos son:

- **Nueva Scene:** Permite crear una nueva escena.
- **Editar Scene:** Permite modificar una escena existente.
- **Borrar Scene:** Permite eliminar una escena.
- **Añadir Smart Device:** Permite agregar un dispositivo inteligente a una escena.
- **Editar Smart Device:** Permite modificar los detalles de un dispositivo inteligente dentro de una escena.
- **Borrar Smart Device:** Permite eliminar un dispositivo inteligente de una escena.
- **Aplicar Scene:** Permite activar una escena, aplicando todas las configuraciones de dispositivos asociadas.

2. **Gestión de Property**

Este módulo se encarga de la administración de las propiedades o fincas gestionadas por la aplicación. Los casos de uso incluidos son:

- **Nueva Property:** Permite agregar una nueva propiedad.
- **Editar Property:** Permite modificar los detalles de una propiedad existente.
- **Borrar Property:** Permite eliminar una propiedad.
- **Información Property:** Proporciona información detallada de una propiedad.
- **Obtener Properties:** Permite recuperar una lista de todas las propiedades del usuario.
- **Información Climática:** Proporciona información meteorológica relevante para la propiedad.
- **Información de Inversor:** Proporciona datos del inversor de energía de la propiedad.
- **Enviar Invitación:** Permite enviar una invitación a otro usuario para compartir el acceso a una propiedad.

3. Autentificación

Este módulo maneja la autenticación y gestión de usuarios. Los casos de uso incluidos son:

- **Iniciar sesión:** Permite a un usuario iniciar sesión en la aplicación.
- **Crear usuario:** Permite registrar un nuevo usuario.
- **Borrar usuario:** Permite eliminar una cuenta de usuario.
- **Cerrar sesión:** Permite a un usuario cerrar su sesión.
- **Editar perfil:** Permite modificar los detalles del perfil del usuario.

4. Gestión de Smart Device

Este módulo se encarga de la administración de los dispositivos inteligentes. Los casos de uso incluidos son:

- **Nuevo Smart Device:** Permite agregar un nuevo dispositivo inteligente.
- **Editar Smart Device:** Permite modificar los detalles de un dispositivo inteligente.
- **Borrar Smart Device:** Permite eliminar un dispositivo inteligente.
- **Información Smart Device:** Proporciona información detallada de un dispositivo inteligente.

Resumen del Diagrama de Casos de Uso

El diagrama de casos de uso de "Virtual Origen: Smart Home" ilustra de manera efectiva las principales funcionalidades de la aplicación y cómo los usuarios interactúan con estas funcionalidades. Los módulos de gestión de escenas, propiedades, dispositivos inteligentes y autenticación cubren todas las operaciones necesarias para una gestión integral y eficiente de un entorno de smart home. Cada caso de uso está claramente definido, asegurando que los requisitos del usuario se cumplan y proporcionando una base sólida para el desarrollo y la implementación de la aplicación.

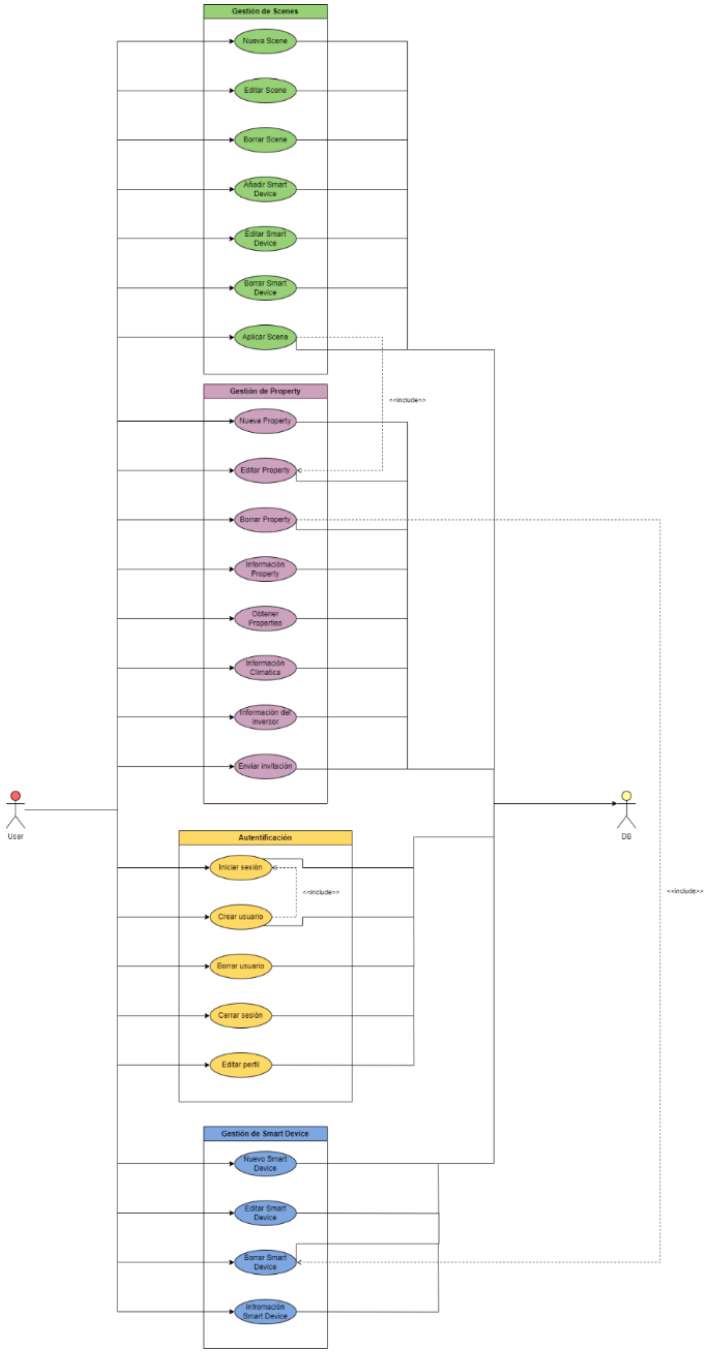


Figura 2.1.3-1 : Diagrama casos de uso

2.1.4 Diagrama de secuencia

Este diagrama detalla el proceso que se siguen en cada vez que se ejecuta el servicio de SmartDevice, el encargado de determinar si hay que encender o pagar un dispositivo teniendo en cuenta tolas las condiciones, finalmente si pasa todas las condiciones que tenga configuradas se encenderá, por el contrario si no se cumple alguna condición enviara la señal de apagado y no continuara procesando los datos de ese dispositivo, ya que ha de cumplir todas las condiciones para encenderse, a no ser que este en modo manual, en ese caso encenderá siempre.

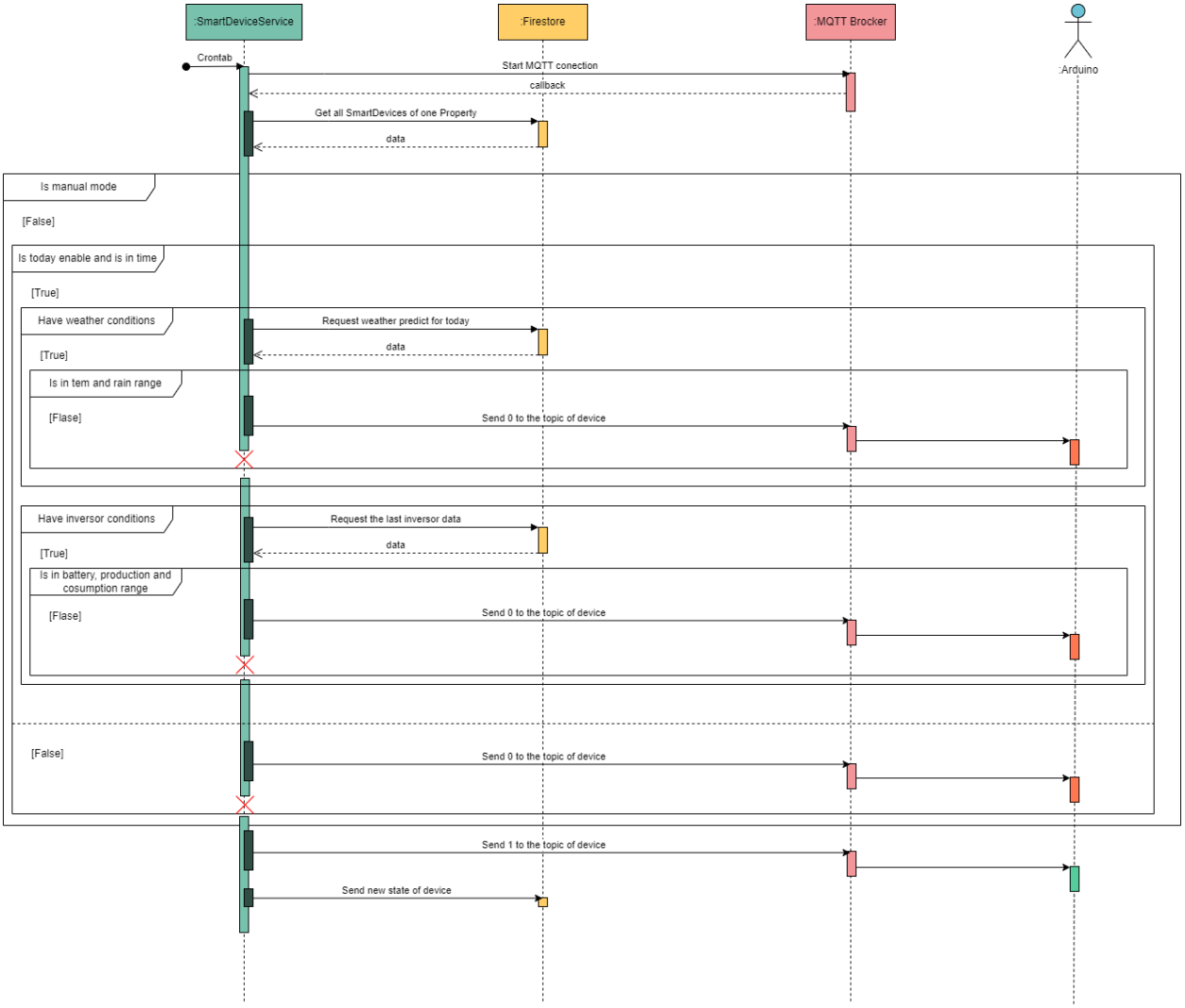


Figura 2.1.4-1 : Diagrama de secuencia SmartDevice Service

2.2 Diseño

2.2.1 Prototipado

El diseño prototipado de la aplicación "Virtual Origen: Smart Home" ha sido desarrollado utilizando Figma, una herramienta de diseño de interfaces de usuario altamente versátil. El enfoque principal del diseño es la simplicidad y que sea intuitiva para los nuevos usuarios, asegurando que los usuarios puedan interactuar con la aplicación de manera eficiente y sin complicaciones.

Aspectos Clave del Diseño

1. **Pantalla de Inicio:**

La pantalla de inicio da la bienvenida al usuario y muestra un resumen de todas las propiedades gestionadas. Cada propiedad está representada por una tarjeta que incluye información clave como el estado del clima, el estado de la batería y el consumo de energía.

Elementos Destacados:

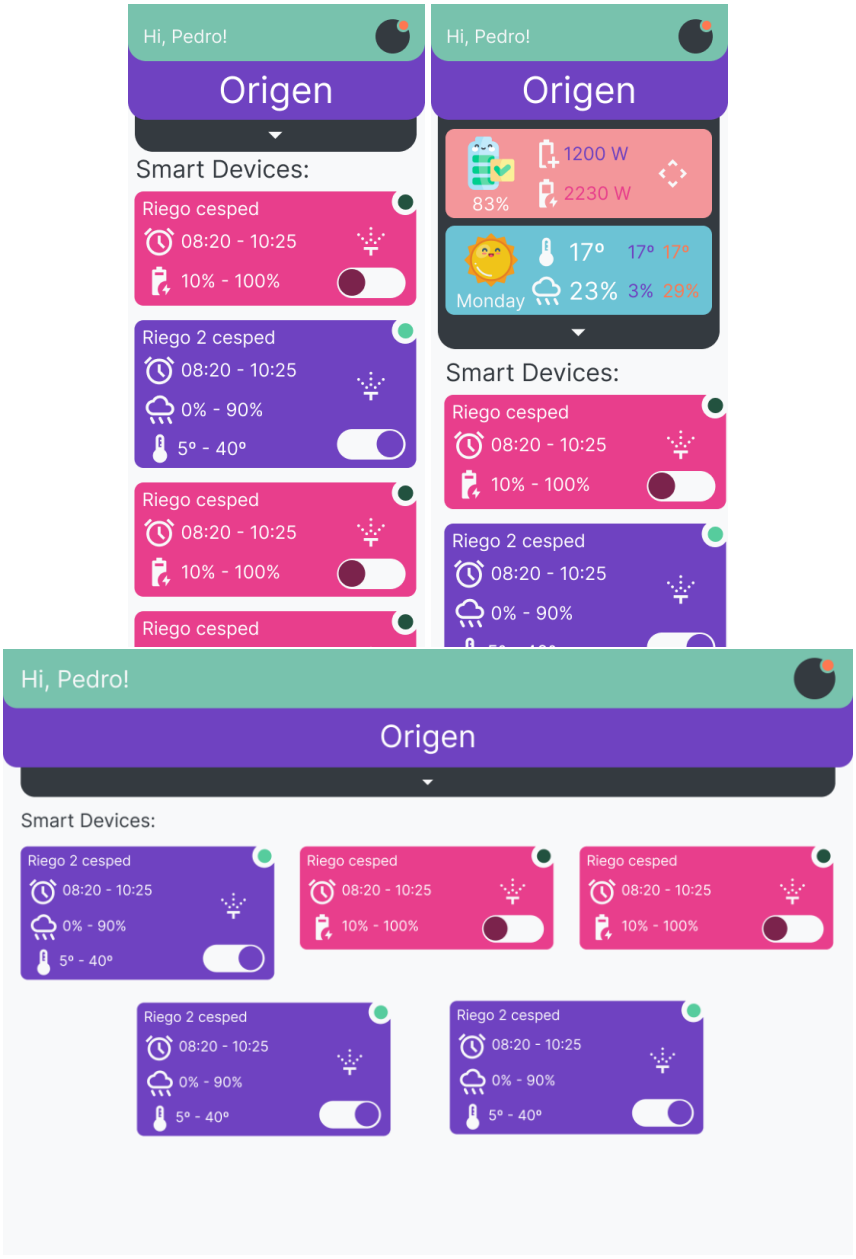
- Saludo personalizado ("Hi, Pedro!").
- Barras de progreso visuales que indican el porcentaje de batería.
- Iconos claros y concisos que representan la temperatura, la probabilidad de lluvia y el consumo de energía.

2. Pantalla de Propiedad:

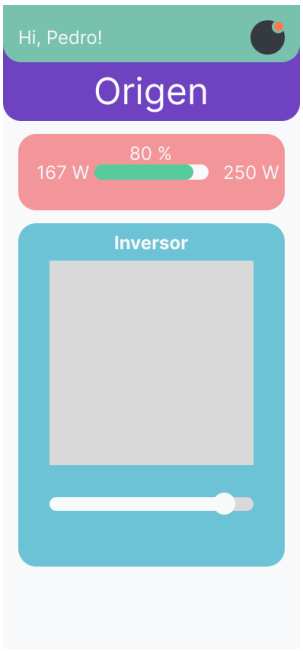
Esta pantalla proporciona una visión detallada de una propiedad específica. Los usuarios pueden ver y gestionar todos los dispositivos inteligentes asociados a esa propiedad.

Elementos Destacados:

- Información detallada sobre el clima actual y el estado del inversor de energía.
- Tarjetas individuales para cada dispositivo inteligente, mostrando sus configuraciones y estado actual (encendido/apagado).
- Uso de colores diferenciados para dispositivos activos e inactivos, facilitando la identificación rápida.



3. Pantalla de Inversor:



Esta pantalla muestra información detallada sobre el rendimiento del inversor de energía de la propiedad y muestra un gráfico en el que se puede observar el rendimiento a lo largo del tiempo.

Elementos Destacados:

- Barra de progreso que indica el estado de carga de la batería.
- Indicadores visuales del consumo y la producción de energía.
- Gráfico de barras para observar la progresión de los datos del inversor a lo largo del tiempo.

4. Diseño Responsivo:

La aplicación ha sido diseñada para ser completamente responsivo, asegurando una experiencia de usuario consistente y optimizado tanto en dispositivos móviles como en escritorio (web).

Elementos Destacados:

- La disposición de los elementos se adapta según el tamaño de la pantalla, manteniendo la funcionalidad y la claridad.
- Los mismos principios de diseño visual se aplican en todas las plataformas, proporcionando una experiencia cohesiva.

Enfoque en la Sencillez e Intuitividad

El diseño de Virtual Origen se centra en ofrecer una interfaz clara y fácil de usar, permitiendo a los usuarios gestionar sus propiedades y dispositivos inteligentes sin necesidad de un conocimiento técnico profundo. Los siguientes principios de diseño fueron fundamentales en la creación de la interfaz:

- **Uso de Iconos Claros:** Los iconos universales ayudan a los usuarios a comprender rápidamente la función de cada elemento, reduciendo la necesidad de texto explicativo.
- **Colores Diferenciados:** La paleta de colores se utiliza estratégicamente para resaltar información importante y guiar la atención del usuario.
- **Interacción Intuitiva:** Los elementos interactivos, como botones y deslizadores, están diseñados para ser fácilmente accesibles y comprensibles, mejorando la experiencia de usuario.
- **Consistencia Visual:** La consistencia en el diseño visual en todas las pantallas y dispositivos asegura que los usuarios puedan navegar por la aplicación sin confusión.

En resumen, el diseño prototipado de la aplicación "Virtual Origen: Smart Home" logra una combinación efectiva de simplicidad y funcionalidad, garantizando que los usuarios puedan gestionar sus sistemas de energía y dispositivos inteligentes de manera eficiente y agradable.

2.2.2 Base de datos

Firebase Firestore es una base de datos en tiempo real y en la nube ofrecida por Google como parte de la plataforma Firebase. Diseñada para satisfacer las necesidades de aplicaciones modernas y escalables, Firestore ofrece una solución flexible y altamente disponible para el almacenamiento y la sincronización de datos en tiempo real. A continuación, se exploran algunas de las características clave de Firebase Firestore y sus ventajas:

Características Principales

1. **Base de Datos en Tiempo Real:**

Firebase Firestore permite la sincronización en tiempo real de datos entre dispositivos clientes y la base de datos en la nube. Esto significa que cualquier cambio realizado en la base de datos se refleja automáticamente en todos los clientes conectados, garantizando una experiencia de usuario fluida y reactiva.

2. **Patrón Listener para Cambios:**

Una de las características más poderosas de Firebase Firestore es su capacidad para implementar un patrón de listener que permite a los clientes recibir notificaciones instantáneas sobre cambios en los datos. Esto permite construir aplicaciones reactivas que pueden responder dinámicamente a los cambios en la base de datos sin necesidad de solicitudes constantes al servidor.

3. **Escalabilidad y Disponibilidad:**

Firestore está diseñado para escalar automáticamente según las necesidades de la aplicación, garantizando un rendimiento óptimo incluso en aplicaciones con un gran volumen de usuarios y datos. Además, Firebase ofrece una disponibilidad del 99.95% para todas las bases de datos, lo que asegura la fiabilidad y la continuidad del servicio.

Ventajas de Firebase Firestore

- Firestore ofrece una interfaz intuitiva y fácil de usar que permite a los desarrolladores crear y gestionar bases de datos sin necesidad de experiencia previa en administración de bases de datos.
- Firebase proporciona herramientas de autenticación y autorización integrada que permiten proteger los datos de la aplicación y controlar el acceso de los usuarios de manera granular.
- Firestore se integra perfectamente con otras funcionalidades de Firebase, como la autenticación de usuarios, la mensajería en la nube y el análisis de aplicaciones, lo que facilita la creación de aplicaciones completas y potentes.

Firebase ofrece un plan gratuito limitado llamado Spark y un plan de pago llamado Blaze, a continuación, detallare el límite relevante para la aplicación de la solución Virtual Origen:

Plan Spark:

Cloud Storage:

Se utiliza para almacenar las fotos de los usuarios, las fotos suelen ser de unos 500 KB cada una y solo hay una foto por usuario.

Nombre	Limite	Uso (10.000 usuarios)
Bytes almacenados	5GB	0,5 (500 KB)

Tabla 2.2.2-1 : Plan Spark Firebase Cloud Storage limites

Cloud Firestore:

Se utiliza como base de datos principal para la mayoría de las entidades, propiedades, dispositivos, datos del inversor…

Nombre	Limite	Uso (100 usuarios, 5 propiedades)
Operaciones de escritura	20 K/Día	26.445 uso/Día
Operaciones de lectura	50 K/Día	37.150 uso/Día
Operaciones de eliminación	20 K/Día	5.005 uso/Día

Tabla 2.2.2-2 : Plan Spark Firebase Cloud Firestore limites

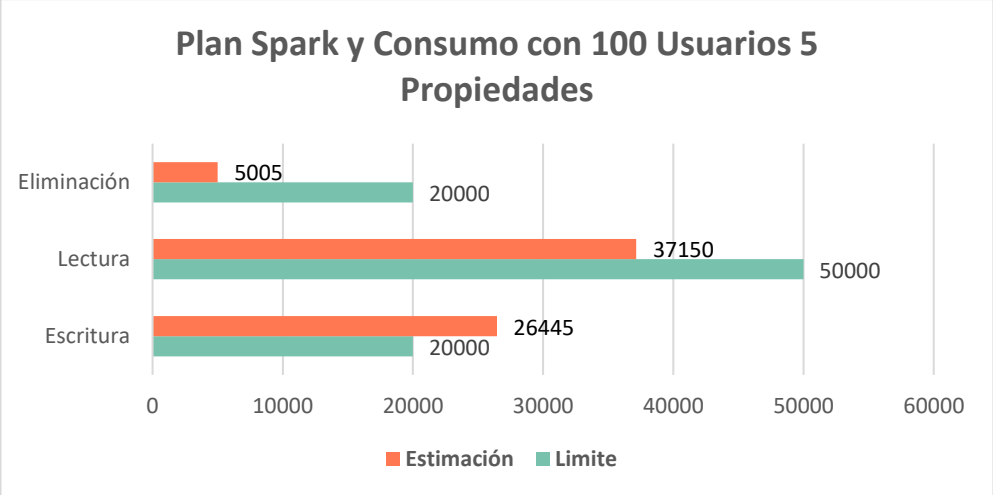


Figura 2.2.2-1 : Grafica los limites plan Spark Firebase y la estimación de usos diarios

Operaciones de escritura:

Los cálculos para una propiedad son los siguientes, una propiedad consta de cuatro entidades que hacen uso de la base de datos:

- Virtual Origen InversorNow: Dos veces por minuto escribe, 2880 usos al día (24*60*2).
- Virtual Origen InversorYesterday: Una vez cada hora, pero escribe cuatro datos, 96 usos al día (24*4).
- Virtual Origen Weather: Se ejecuta una vez al día, pero hace unas 25 escrituras (normalmente es menos, pero he añadido un 30% de margen hacia arriba), 25 usos al día.
- Virtual Origen Smart Device: Se ejecuta una vez por minuto, pero no siempre escribe, solo cuando hay un cambio de estado en el dispositivo, por lo que, generosamente, he considerado que el 20% de las llamadas se ejecutara un cambio de estado, 288 usos al día (24*60*20/100).

Ha este uso hay que añadirle el consumo que pueden hacer los usuarios, basándome en el uso que han estado haciendo los testers he determinado que un usuario en un día hace unas 100 escrituras, lo que nos da un total de 10.000 operaciones de escritura al día con 100 usuarios.

Ahora si sumamos lo que gastan cinco propiedades totalmente instaladas y lo que gastan los 100 usuarios nos da un total de 26.445 operaciones de escritura al día.

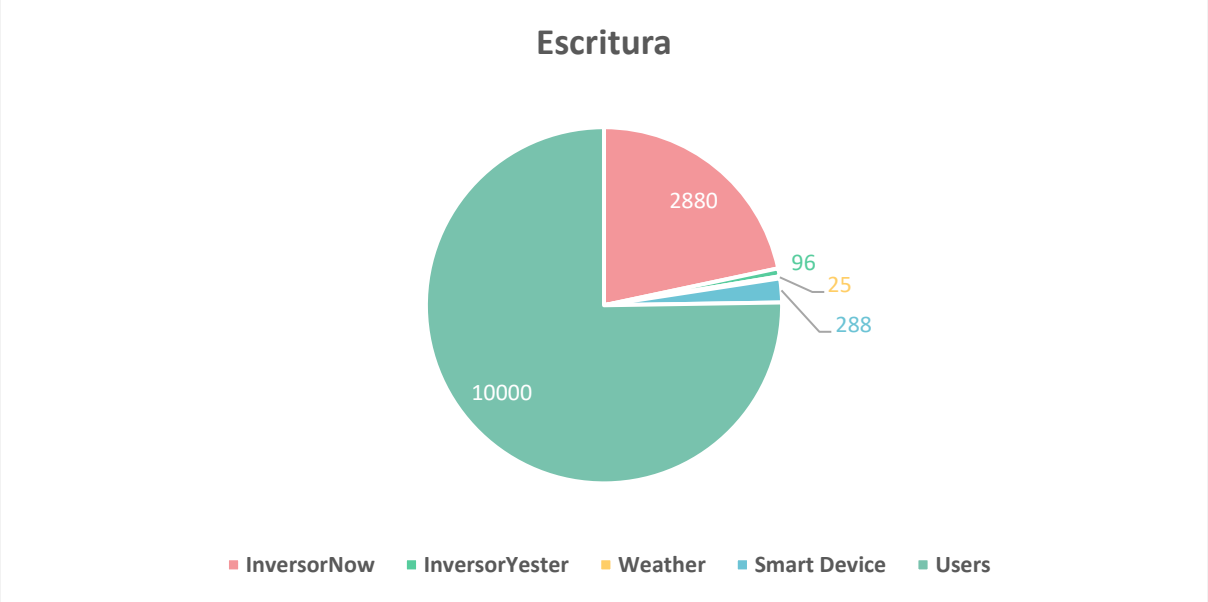


Figura 2.2.2-1 : Grafica de escrituras al día estimadas / quien las gasta

Operaciones de lectura:

Para calcular cuantas operaciones de lectura suceden en un día hay que calcular lo siguiente:

- Virtual Origen InversorNow:No tiene lectura.
- Virtual Origen InversorYesterday: Se ejecuta cada hora, hace 960 operaciones de lectura en un día (24*40).
- Virtual Origen Weather: Se ejecuta una vez al día, y realiza unas 30 operaciones de lectura.
- Virtual Origen Smart Device: Se ejecuta una vez por minuto, por lo que hace uso de 1440 llamadas (24*60).

Por la parte de los usuarios podemos calcular que, cada usuario hace unas 250 operaciones de lectura, extrapolándolo a 100 usuarios nos da unas 25.000 operaciones de lectura.

Al igual que en la escritura, si calculamos con cinco propiedades y 100 usuarios nos da un total de 37150 operaciones de lectura totales.

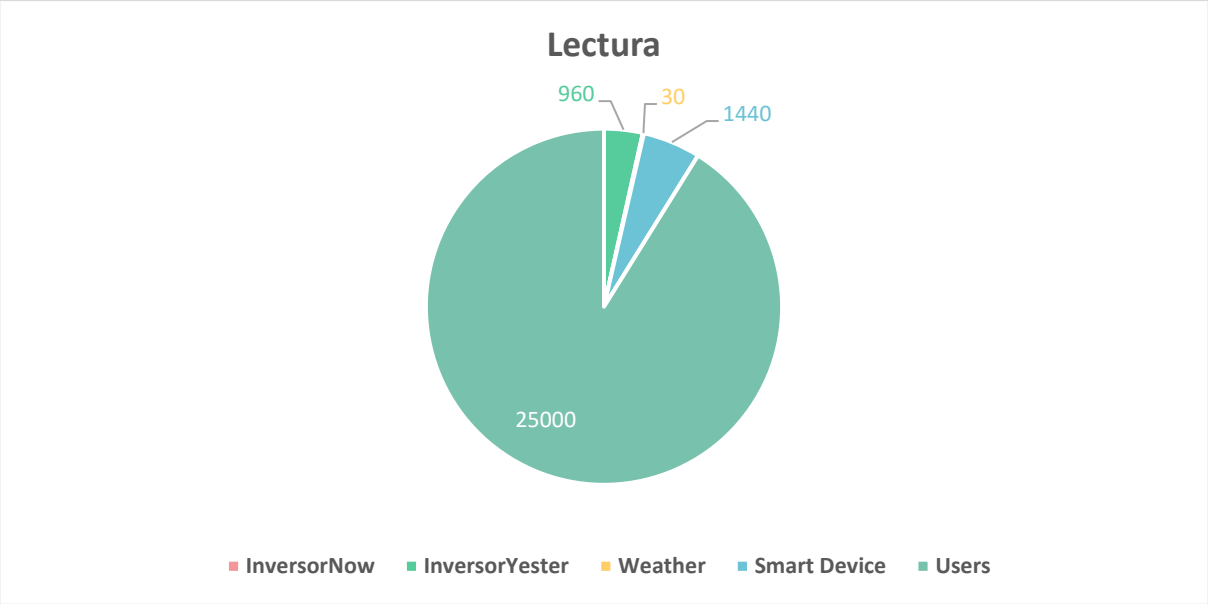


Figura 2.2.2-2 : Grafica de lecturas al día estimadas / quien las gasta

Operaciones de eliminación:

Las operaciones de lectura son más sencillas de calcular, ya que solo hacen uso de las eliminaciones “Virtual Origen Weather”, que hace una única eliminación al día y las eliminaciones que hacen los usuarios, que cada usuario hace unas 50 operaciones de eliminación al día lo que da un total de 5.000 con 100 usuarios.

En total teniendo en cuenta las cinco propiedades y los cien usuarios son unas 5.005 operaciones de eliminación.

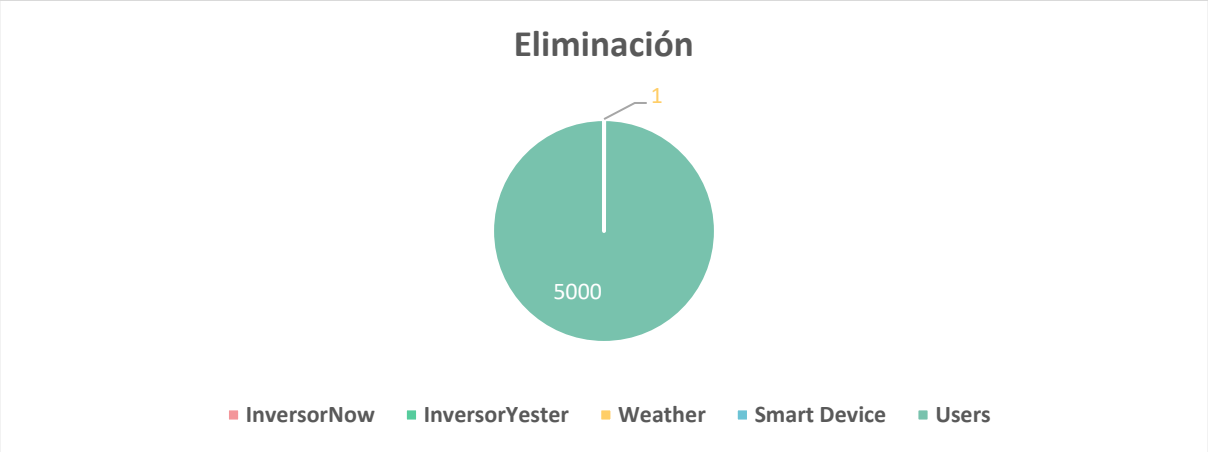


Figura 2.2.2-1 : Grafica de eliminaciones al día estimadas / quien las gasta

2.2.3 Arquitectura

Arquitectura del Software frontend

La arquitectura del software en la aplicación "Virtual Origen: Smart Home" se basa en el uso de GetX para implementar una interfaz reactiva a los cambios. Se ha adoptado el patrón de diseño Modelo-Vista-Update (MVU) para separar claramente la lógica de presentación de los datos y la lógica de negocio. Esta elección proporciona una arquitectura limpia y modular, facilitando el mantenimiento y la escalabilidad del código.

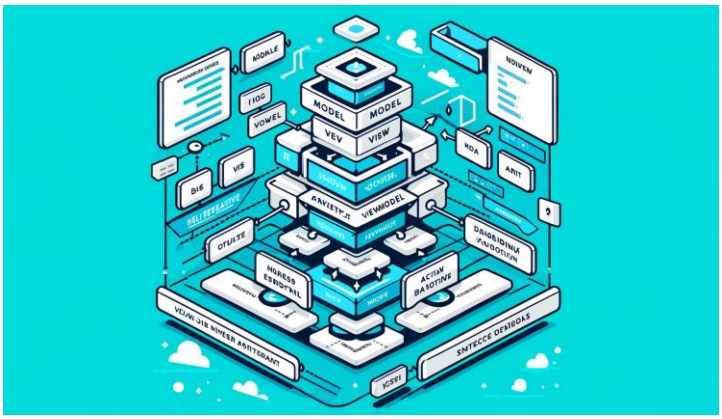


Figura 2.2.3-1 : Arte vectorial representado la arquitectura MVU (AI)

Además, se aprovechan los listeners proporcionados por Firebase para la base de datos en tiempo real. Estos listeners permiten que la aplicación reaccione automáticamente a los cambios en los datos almacenados en Firebase, garantizando una experiencia de usuario integrada y reactiva en tiempo real. Esta combinación de GetX y Firebase ofrece una arquitectura robusta y eficiente, que cumple con los requisitos de una aplicación moderna y dinámica.

Arquitectura del Software backend

El sistema servidor cuenta con cuatro aplicaciones, todas ellas están organizadas en una arquitectura por capas con bajo acoplamiento y alta cohesión, teniendo una capa de modelos, repositorios y si el servició lo necesita otros servicios como MQTT o Request HTTP…

Arquitectura del Sistema VirtualOrigen

En cuanto a la arquitectura del sistema VirtualOrigen, se ha diseñado una estructura organizada y modular para facilitar la gestión inteligente de dispositivos. La imagen adjunta muestra una representación visual de esta arquitectura, que se centra en la capacidad de encender y apagar relés de manera inteligente. Las entidades clave están organizadas de manera coherente, permitiendo un control eficiente y flexible de los dispositivos.

La arquitectura del sistema se basa en la interacción entre diferentes componentes, como el propio usuario, el servidor y los relés MQTT. Cada entidad está diseñada para desempeñar un papel específico en la gestión y control de los dispositivos, asegurando que la aplicación cumpla con los requisitos de funcionalidad y rendimiento establecidos.

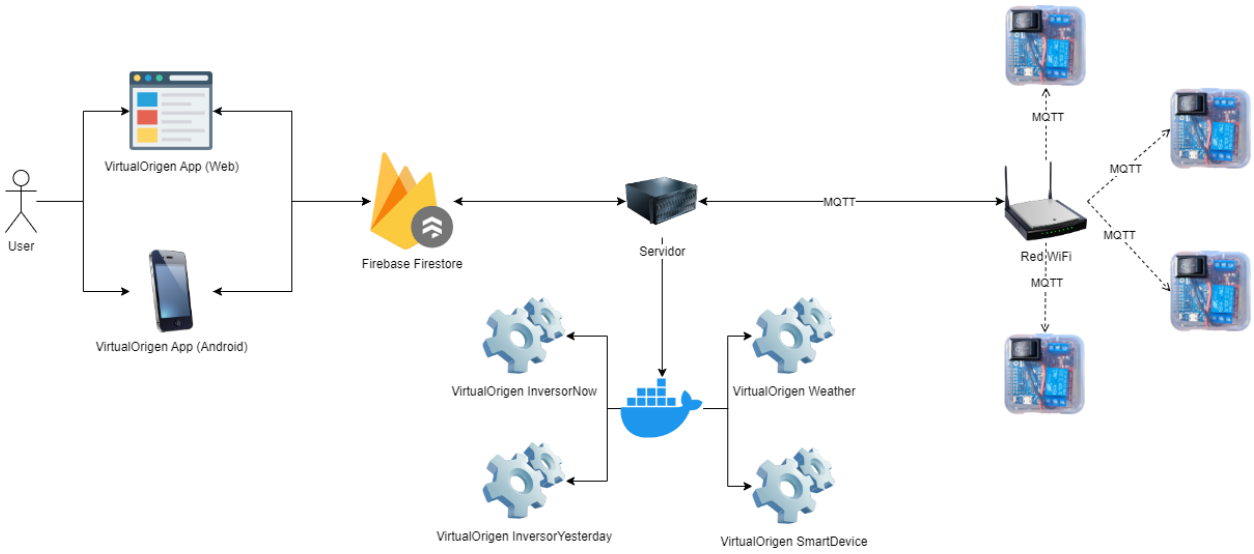


Figura 2.2.3-1 : Arquitectura Básica Completa

2.3 Implementación

Arduino:

Arduino es una plataforma de hardware de código abierto y fácil de usar que se utiliza para crear prototipos de dispositivos electrónicos interactivos. En el proyecto "Virtual Origen: Smart Home", Arduino se utiliza para controlar los relés que activan y desactivan los dispositivos físicos en el hogar, como luces, electrodomésticos y sistemas de riego. La programación de Arduino se realiza utilizando el lenguaje de programación Wiring, que es similar a C/C++, lo que permite a los desarrolladores crear código personalizado para controlar el comportamiento de los dispositivos conectados.

Este archivo contiene el código utilizado en las cajas Arduino para escuchar la cola MQTT y controlar los relés. Implementa la lógica para recibir mensajes MQTT y controlar los relés para encender o apagar dispositivos conectados dependiendo del mensaje recibido.

Cada dispositivo tiene una cola para él, como se ve en la imagen la variable “Rele” contiene el UUID que corresponde con el dispositivo en la base de datos.

Es esencial para la integración de dispositivos físicos en la aplicación, permitiendo la comunicación entre la aplicación y los dispositivos en el hogar.

```
const char *ssid = "LSIR_2G_Cebollera";
const char *password = "****";
const char *mqtt_server = "192.168.1.199";

const int PinRele = 5; // D1 en la placa

char Rele[48] = "cb05e53b-f541-4b0b-b52d-93cacf35a126";

WiFiClient RacisClient;
PubSubClient client(RacisClient);

void setup()
{
  pinMode(PinRele, OUTPUT);
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}

void setup_wifi()
{

  delay(10);
  // We start by connecting to a WiFi network
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
  }
}
```

```
void callback(char *topic, byte *payload, unsigned int length)
{
  // Open or close the relay
  if ((char)payload[0] == '0')
  {
    digitalWrite(PinRele, LOW);
  }
  else
  {
    digitalWrite(PinRele, HIGH);
  }
}

void reconnect()
{
  // Loop until we're reconnected
  while (!client.connected())
  {
    // Attempt to connect
    if (client.connect("ESP8266Client"))
    {
      // Once connected, publish an announcement...
      char MsgConectado[64] = "Conectado ";
      strcat(MsgConectado, Rele);
      client.publish("cebolla/reles", MsgConectado);
      // ... and resubscribe
      char TopicSuscripcion[64] = "cebolla/reles/";
      strcat(TopicSuscripcion, Rele);
      client.subscribe(TopicSuscripcion);
    }
    else
    {
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}
```

Figura 2.3-1 : Programa Arduino MQTT

Csharp:

C# es un lenguaje de programación desarrollado por Microsoft que se utiliza principalmente para crear aplicaciones de escritorio, aplicaciones web y aplicaciones móviles utilizando el framework .NET. En el contexto de "Virtual Origen: Smart Home", C# se utiliza en el backend de la aplicación para implementar la lógica empresarial, gestionar la comunicación con la base de datos y controlar la lógica de negocio de la aplicación. Con C#, los desarrolladores pueden escribir código limpio y estructurado que garantiza un rendimiento óptimo y una gestión eficiente de los recursos del sistema.

Este archivo, parte del backend de la aplicación, contiene la lógica para determinar si un dispositivo inteligente debe ser encendido o apagado en función de ciertos criterios predefinidos.

Se encarga de tomando los datos del clima y/o del inversor decidir si se ha de encender o apagar dependiendo de la configuración del dispositivo.

Figura 2.3-2 : Programa C# que determina el estado de un dispositivo inteligente

Bash:

Bash es un intérprete de comandos de Unix ampliamente utilizado que se utiliza para automatizar tareas repetitivas y administrar sistemas operativos basados en Unix/Linux. En el proyecto "Virtual Origen: Smart Home", Bash se utiliza para crear scripts de automatización que recopilan datos del inversor de energía y los exponen utilizando el protocolo MQTT. Estos scripts permiten integrar datos en tiempo real sobre el rendimiento de la energía solar en la aplicación, lo que proporciona a los usuarios información valiosa para gestionar eficientemente su consumo de energía.

Lectura_Inversor.bat

```
#!/bin/bash

#set -x

Log=/home/salviati/LeoVoltronic/LOGs/LeoVoltronic.log.$(date +%Y-%m-%d)
Procesos=$(ps -ef | grep LeoVoltronic.sh | grep -v grep | wc -l)
if [[ $Procesos -le 3 ]]; then

for i in {1..2}
do

        Lectura=""
        while (( $(echo "$Lectura"|wc -l) <= 4 )); do
                Lectura=$(/usr/local/bin/mpp-solar -p /dev/hidraw0 -c QPIGS)
                sleep 2
        done
        while (( $(echo "$Lectura"|grep "USB open error"|wc -l) > 0 )); do
                #Hay que reiniciar, el dispositivo USB ha desaparecido
                /usr/bin/touch "/home/salviati/LeoVoltronic/LOGs/Reinicios/Reinicio_$(date +%F"_"%R)"
                /usr/sbin/reboot
                exit 1
        done

        Consumo=$(echo "$Lectura" | grep "ac_output_active_power" | awk '{print $2}')
        Baterias=$(echo "$Lectura" | grep "battery_capacity" | awk '{print $2}')
        Placas=$(echo "$Lectura" | grep "pv_input_power" | awk '{print $2}')

        printf "$(date +%F" "%R" ") " >> $Log
        printf "$Consumo " >> $Log
        printf "$Baterias " >> $Log
        echo $Placas >> $Log

        /usr/bin/mosquitto_pub -h 192.168.1.199 -t cebolla/inversor/consumo -m $Consumo
        /usr/bin/mosquitto_pub -h 192.168.1.199 -t cebolla/inversor/baterias -m $Baterias
        /usr/bin/mosquitto_pub -h 192.168.1.199 -t cebolla/inversor/placas -m $Placas
        /usr/bin/mosquitto_pub -h 192.168.1.199 -t cebolla/inversor/todo -m "$Consumo,$Baterias,$Placas"

        if [[ $i -ne 2 ]]; then
                sleep 30
        fi
done

fi

exit 0
```

Figura 2.3-3 : Archivo Bash que lee los datos del inversor y los expone por MQTT

Flutter/Dart:

Flutter es un framework de desarrollo de aplicaciones multiplataforma nativa creado por Google, mientras que Dart es el lenguaje de programación utilizado para escribir aplicaciones en Flutter. Flutter permite el desarrollo rápido y eficiente de interfaces de usuario atractivas y reactivas. En el proyecto "Virtual Origen: Smart Home", Flutter se utiliza para construir la interfaz de usuario de la aplicación.

Este archivo, ubicado en el frontend de la aplicación, contiene un repositorio genérico para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en una entidad cualquiera utilizando Firebase como base de datos. Permite interactuar con la base de datos Firebase para realizar operaciones de lectura de datos de manera genérica en toda la aplicación. Proporciona una capa de abstracción para la gestión de datos en el frontend, facilitando la integración con Firebase y la implementación de funcionalidades de lectura de datos.

GenFirebaseRepositoryOnlyGet.dart

```dart
abstract class GenFirebaseRepositoryOnlyGet<T, ID>
    extends IListenerRepositoryOnlyGet<T, ID, String> {
  String get collectionName;
  String get listName;
  String get idName;

  T fromJson(Map<String, dynamic> json);

  StreamSubscription<DocumentSnapshot>? listenerStream;
  StreamSubscription<DocumentSnapshot>? singleListenerStream;

  Future<MapEntry<DocumentReference<Object?>, DocumentSnapshot<Object?>?>>
      getRefAndSnapshot({required String idc}) async {
    try {
      DocumentReference docRef =
          FirebaseFirestore.instance.collection(collectionName).doc(idc);
      DocumentSnapshot<Object?> snapshot = await docRef.get();
      return MapEntry(docRef, snapshot.exists ? snapshot : null);
    } catch (e) {
      throw Exception("Error getting data of $idc from $collectionName");
    }
  }

  bool checkIfDocExists(DocumentSnapshot<Object?>? snapshot) {
    return snapshot == null || !snapshot.exists || snapshot.data() == null;
  }

  @override
  void addSingleListener({
    required ID id,
    required String idc,
    required Function(T?) listener,
  }) {
    singleListenerStream = FirebaseFirestore.instance
        .collection(collectionName)
        .doc(idc)
        .snapshots()
        .listen((DocumentSnapshot snapshot) {
      if (checkIfDocExists(snapshot)) {
        listener(null);
        return;
      }
      var data = snapshot.data() as Map<String, dynamic>;
      var entity = (data[listName] as List<dynamic>)
          .firstWhere((task) => task[idName] == id, orElse: () => null);
      if (entity == null) {
        listener(null);
        return;
      }
      listener(fromJson(entity as Map<String, dynamic>));
    });
  }

  @override
  void removeSingleListener({required ID id, required String idc}) {
    singleListenerStream?.cancel();
  }

  @override
  void addListener({
    required String idc,
    required Function(List<T>) listener,
  }) {
    listenerStream = FirebaseFirestore.instance
        .collection(collectionName)
        .doc(idc)
        .snapshots()
        .listen((DocumentSnapshot snapshot) {
      if (checkIfDocExists(snapshot)) {
        listener([]);
        return;
      }
      var data = snapshot.data() as Map<String, dynamic>;
      var entities = (data[listName] as List<dynamic>)
          .map((e) => fromJson(e as Map<String, dynamic>))
          .toList();
      listener(entities);
    });
  }

  @override
  Future<bool> existsById({
    required ID id,
    required String idc,
  }) async {
    return await findById(id: id, idc: idc) != null;
  }

  @override
  Future<List<T>> findAll({required String idc}) async {
    var docRef = await getRefAndSnapshot(idc: idc);
    if (checkIfDocExists(docRef.value)) return [];
    var data = docRef.value!.data() as Map<String, dynamic>;
    return (data[listName] as List<dynamic>)
        .map((e) => fromJson(e as Map<String, dynamic>))
        .toList();
  }

  @override
  Future<T?> findById({
    required ID id,
    required String idc,
  }) async {
    var docRef = await getRefAndSnapshot(idc: idc);
    if (checkIfDocExists(docRef.value)) return null;
    var data = docRef.value!.data() as Map<String, dynamic>;
    var entity = (data[listName] as List<dynamic>)
        .firstWhere((task) => task[idName] == id, orElse: () => null);
    if (entity == null) return null;
    return fromJson(entity as Map<String, dynamic>);
  }

  @override
  void removeListener({required String idc}) {
    listenerStream?.cancel();
  }
}
```

Figura 2.3-4 : Repositorio genérico para Firebase Firestore

2.4 Implantación

Para poder hacer uso de la solución completa de “Virtual Origen : Smart Home” y tener todas las funcionalidades que hacer una gestión más eficiente y eficaz de los recursos energéticos, es necesario hacer una implantación física de un servidor en la finca que requiera de los servicios de domotización, esto se puede hacer con un ordenador monoplaca como por ejemplo una Raspberry Pi 3 Model B, también haría falta hacer una instalación del software backend de Virtual Origen, esto es realmente sencillo ya que todas las soluciones están dockerizadas, lo que permite una instalación isolada al sistema operativo del servidor. Incluso es viable una distribución propietaria que ya incluya, en el ordenador monoplaca, docker con las soluciones backend de Virtual Origen.

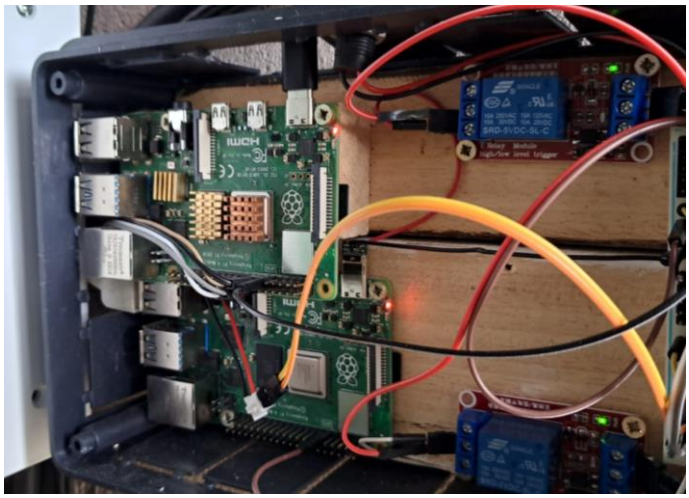


Figura 2.4-1 : Instalación Rasberry Pi alta disponibilidad

También es necesario hacer uso de los relés WiFi que se activan atreves del protocolo MQTT, estas cajas incluyes una placa Arduino conectado a un relé, para que este funciona necesita una reprogramación para que contenga el identificador único que corresponde con su dispositivo inteligente en la base de datos, por lo demás la instalación es sencilla ya que solo necesita corriente y que pases los cables por el relé, como si fuera un interruptor.

Figura 2.4-2 : Arduino WiFi conectada a un relé

En cuanto a la parte física eso sería la implementación, ya solo quedaría que los usuarios que vayan a usar la app se registren o bien mediante el portal web o usando la aplicación Android en la app de Virtual Origen, creen una propiedad y configuren los dispositivos inteligentes, también pueden invitar a otros usuarios de la app para compartir una finca y poder hacer un seguimiento en tiempo real multiusuario.

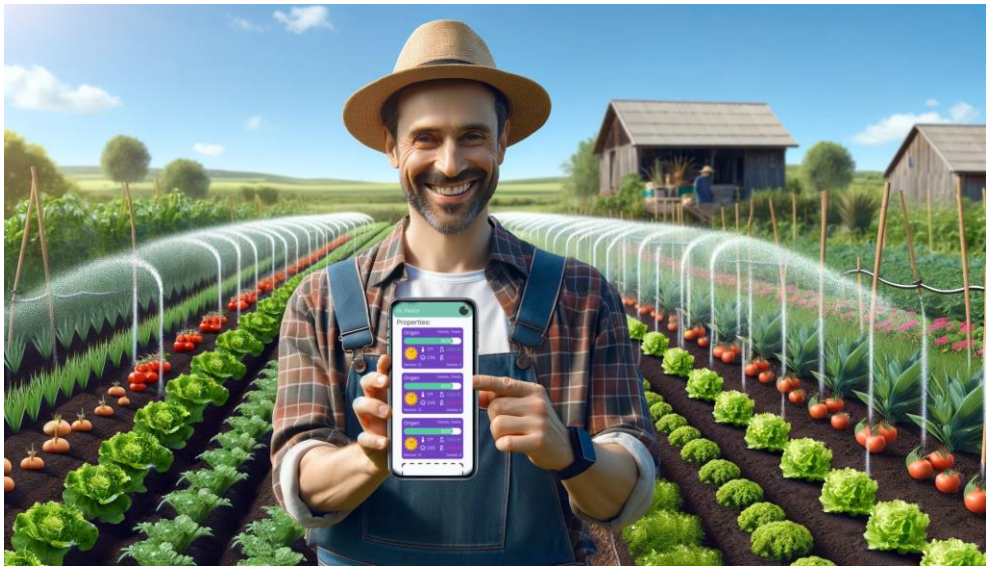


Figura 2.4-3 : Supuesto granjero usando la app en un entorno descentralizado (AI)

En cuanto a los costos de la implantación depende del numero de dispositivos a domotizar y si se implanta un sistema servidor de alta disponibilidad, pero estos son los cálculos para cinco propiedades cada una con una veintena de dispositivos domotizados:

Nombre	Unidades	PVP
Raspberri PI 4 Model B 4GiB	1	89,90 €
Arduino D1 Mini V3 4MiB	5	18,99 €
Caja de plástico pequeña	24	9,99 €
Relé 10A	10	14,99 €
Interruptores 6A	35	8,99 €

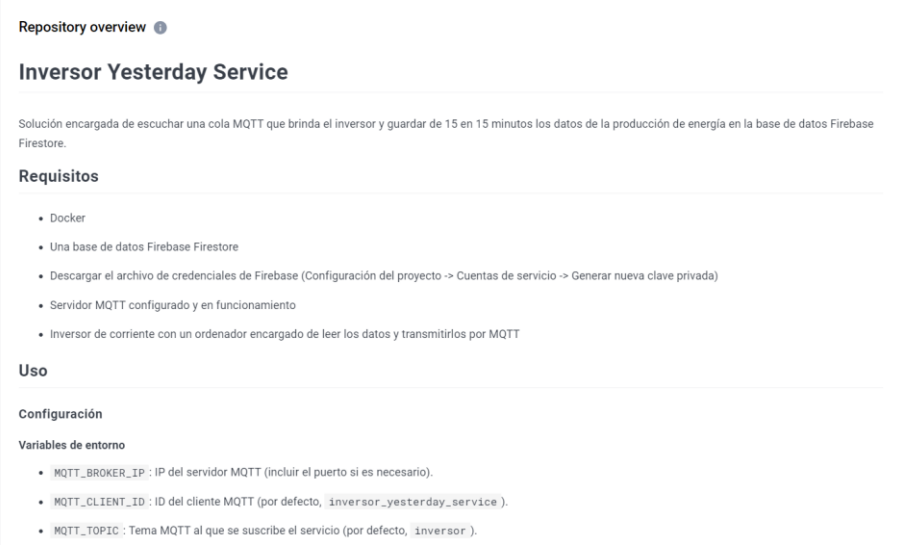
Con estos datos podemos calcular que una propiedad y veinte dispositivos cuestan un total de **214,82€** (89,90 + (18,99 *4) + 9,99 + (14,99 * 2) + 8,99).

Por lo que los costes de instalar cinco propiedades y en cada propiedad veinte dispositivos seria de **1.074,10 €** (214,82 * 5).

2.5 Documentación

Docker Hub:

Documentación técnica de cada uno de los servicios del servidor que están subidos al repositorio de imágenes de Docker oficial Docker Hub. En esta documentación se detalla la finalidad de cada servició junto con los requisitos y la configuración previa que hay realizar para poder hacer uso del servicio.



Repository overview

Inversor Yesterday Service

Solución encargada de escuchar una cola MQTT que brinda el inversor y guardar de 15 en 15 minutos los datos de la producción de energía en la base de datos Firebase Firestore.

Requisitos

- Docker
- Una base de datos Firebase Firestore
- Descargar el archivo de credenciales de Firebase (Configuración del proyecto -> Cuentas de servicio -> Generar nueva clave privada)
- Servidor MQTT configurado y en funcionamiento
- Inversor de corriente con un ordenador encargado de leer los datos y transmitirlos por MQTT

Uso

Configuración

Variables de entorno

- `MQTT_BROKER_IP` : IP del servidor MQTT (incluir el puerto si es necesario).
- `MQTT_CLIENT_ID` : ID del cliente MQTT (por defecto, `inversor_yesterday_service`).
- `MQTT_TOPIC` : Tema MQTT al que se suscribe el servicio (por defecto, `inversor`).

GitHub:

Capítulo 3 - Resultados Y Discusión

En cuento al desarrollo de la aplicación con respecto a la planificación inicial del anteproyecto no solo es que se ha podido seguir, si no que siempre se ha estado más avanzado, teniendo la aplicación en un estado de madurez funcional dos semanas antes de que terminase la fase de desarrollo.

Para lograr un seguimiento eficaz que me ha permitido tener una planificación adecuada he utilizado Notion, es una herramienta todo-en-uno que combina la funcionalidad de un editor de notas, gestor de tareas, base de datos y wiki para mejorar la productividad y organización personal o de equipos.

Finalmente, la aplicación cuenta con todas las funcionalidades que se planteó en el anteproyecto, pudiendo gestionar dispositivos inteligentes de manera remota y de forma inteligente, pudiendo compartir las propiedades y hacer de la experiencia de usuario agradable.

Durante el desarrollo las mayores dificultades fueron el uso de Docker en la Raspberry ya que cuenta con una arquitectura de sistema diferente a un ordenador convencional, también ubo dificultades con Docker porque los servicios dejaban de funcionar, al principio era porque la Raspberry leía los datos del inversor y eso la dejaba totalmente colapsada y muchas veces se quedaba totalmente inoperativa y había que reiniciarla físicamente, esto se resolvió haciendo un sistema de alta disponibilidad con dos Raspberry, de esta manera siempre están activos los servicios cuando tienen que estar.

Por lo demás el desarrollo fue bien, ya tenía experiencia con el uso de Flutter por lo que la interfaz no costo mucho, quizás lo que tuvo más dificultad fueron los gráficos y la que fuese responsiva.

En cuanto al backend en C# fue también cómodo, al tener varias aplicaciones que cada una solucionan un problema no se complicaba excesivamente la solución, aun siendo soluciones pequeñas se ha seguido los principios de código limpio y se ha seguido una arquitectura organizada por capas y con un acoplamiento bajo.

La instalación física no tiene mucha complicación gracias a la caja con el Arduino y el relé, con esa solución empaquetada se puede domotizar muchos aparatos de manera sencilla y rápida.

Ha sido un proyecto muy enriquecedor en cuanto a conocimientos nuevos y se ha demostrado mi capacidad para implementar una arquitectura completa de una aplicación compleja.

Figura 3-1 : Supuesta casa con placas solares y huerto domotizada (AI)

Capítulo 4 - Trabajo Futuro

La solución Virtual Origen cuenta con todas las funcionalidades propuestas en el anteproyecto, no obstante, siempre se puede mejorar he implementar nuevas funcionalidades, algunas de ellas pueden ser estas:

- Integración de notificaciones de estado para un determinado dispositivo.
- Tener la posibilidad de enlazarlo al calendario del dispositivo o el calendario de Google.
- Mayor complejidad en las condiciones de encender, por ejemplo, dependencia de otros dispositivos.
- Menor acoplamiento a la base de datos, aislando los clientes de la base de datos implementada por detrás.
- Poder integrar sistemas de seguridad como alarmas y cámaras IP de seguridad.

Todas estas funcionalidades son nuevas o pequeñas mejoras en alguna ya existente, la aplicación tal y como esta es totalmente funcional y cumple con lo necesario para mejorar la eficiencia energética de hogares.

También en un futuro entraría la implementación de esta solución en otras fincas descentralizadas que necesiten de una mejor eficiencia energética.

Figura 4-1 : Supuesto técnico instalando la solución IOT VirtualOrigen (AI)

Capítulo 5 - Conclusiones

El proyecto "Virtual Origen: Smart Home", desarrollado durante el curso 2023-2024 en el marco del CFGS de Desarrollo de Aplicaciones Multiplataforma, ha logrado implementar una solución innovadora para la gestión de la energía en entornos descentralizados mediante el uso de tecnologías IoT.

1. **Logro de Objetivos:**

Se ha alcanzado una gestión eficiente y automatizada de los recursos energéticos de una finca que opera exclusivamente con energía solar. Esto se ha logrado mediante el desarrollo e implementación de una aplicación multiplataforma que permite un control remoto y adaptativo de dispositivos de alto consumo y sistemas de riego.

La aplicación ha demostrado ser intuitiva y fácil de usar, facilitando al usuario la configuración y supervisión de dispositivos en tiempo real, lo que refleja una integración exitosa de las interfaces de usuario con la funcionalidad del backend.

2. **Innovación Tecnológica:**

La incorporación de tecnologías como Flutter para el desarrollo del frontend y C# para el backend, junto con la utilización de Firebase y MQTT, ha permitido crear una arquitectura robusta que soporta la recopilación, almacenamiento y análisis de datos en tiempo real.

La solución ha destacado por su capacidad para adaptar el consumo de dispositivos en función de variables críticas como el estado de las baterías y las condiciones climáticas, asegurando así una operación eficiente y sostenible.

3. **Desafíos y Aprendizajes:**

El proyecto enfrentó desafíos técnicos significativos, especialmente en la integración de diferentes tecnologías y en la gestión de la comunicación entre el frontend y el backend. La superación de estos desafíos ha enriquecido la experiencia de aprendizaje, ampliando el conocimiento técnico y práctico en el desarrollo de aplicaciones multiplataforma.

4. **Impacto y Aplicabilidad Futura:**

"Virtual Origen: Smart Home" no solo responde a una necesidad personal de gestión eficiente de la energía, sino que también establece un modelo replicable que podría ser adaptado para otros entornos que requieran soluciones de autonomía energética.

El proyecto ha demostrado el potencial de las soluciones basadas en IoT para contribuir significativamente a la sostenibilidad energética, abriendo caminos para futuras investigaciones y desarrollos en el área de smart homes y gestión de recursos energéticos.

El desarrollo del proyecto "Virtual Origen: Smart Home" ha representado un hito significativo en mi trayectoria formativa dentro del CFGS de Desarrollo de Aplicaciones Multiplataforma. A través de este proyecto, he tenido la oportunidad de aplicar y profundizar en conocimientos técnicos adquiridos en el aula, especialmente en el campo emergente del Internet de las Cosas (IoT). La experiencia de diseñar una solución completa y funcional que integra de manera efectiva frontend, backend y manejo de bases de datos ha reforzado mi comprensión de las arquitecturas de software y los sistemas de bases de datos en tiempo real. Asimismo, este proyecto me ha permitido desarrollar habilidades cruciales como la resolución de problemas, el pensamiento crítico y la gestión de proyectos, preparándome de manera integral para afrontar los desafíos del mundo profesional y tecnológico actual.

En conclusión, el proyecto ha cumplido y superado las expectativas iniciales, proporcionando una solución funcional de un sistema IoT para entornos descentralizados y ofreciendo un marco sólido para futuras innovaciones en el campo de la eficiencia energética.

Figura 5-1 : Barrio de casas domotizadas con la solución Virtual Origen (AI)

Capítulo 6 - Bibliografía

ALOTAIBI, N. J. (s.f.). Obtenido de https://gptstore.ai/gpts/ScxItyf8pT-image-generator

Blance, J. (s.f.). *Github*. Obtenido de https://github.com/jblance/mpp-solar/wiki

Docker. (s.f.). *Docker docs*. Obtenido de https://docs.docker.com/engine/install/raspberry-pi-os/

Flutter. (s.f.). *Youtube*. Obtenido de https://www.youtube.com/watch?v=PkPAtfNNJX8

Freepik. (s.f.). *Flaticon*. Obtenido de https://app.flchart.dev/

Khoshabi, I. (s.f.). *FL Chart*. Obtenido de https://app.flchart.dev/

Koko, M. (s.f.). *Youtube*. Obtenido de https://www.youtube.com/watch?v=8j6iw33bJU0

Llamas, L. (27 de Oct de 2021). *Luisllamas*. Obtenido de https://www.luisllamas.es/en/how-to-use-mqtt-protocol-with-c-and-mqttnet-library/

Open-Meteo. (s.f.). *Open-Meteo*. Obtenido de https://open-meteo.com/

Otterlord. (24 de Jun de 2022). *Medium*. Obtenido de https://enlear.academy/run-amd64-docker-images-on-an-arm-computer-208929004510