# KIOSKO DIGITAL: DASHBOARD DE RECOLECCIÓN Y PROCESAMIENTO DE DATOS DE EDIFICIOS INTELIGENTES

## DIGITAL KIOSK: DATA COLLECTION AND PROCESSING DASHBOARD FOR SMART BUILDINGS


Trabajo Fin de Grado
Curso 2022-2023

**AUTORES**
Carlos Murcia Morilla
Clara Fernández Forte

**DIRECTORA**
Maria Belén Díaz Agudo

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

# KIOSKO DIGITAL: DASHBOARD DE RECOLECCIÓN Y PROCESAMIENTO DE DATOS DE EDIFICIOS INTELIGENTES

## DIGITAL KIOSK: DATA COLLECTION AND PROCESSING DASHBOARD FOR SMART BUILDINGS

Trabajo de Fin de Grado en Ingeniería Informática


**AUTORES**

Carlos Murcia Morilla

Clara Fernández Forte

**DIRECTORA**

Maria Belén Díaz Agudo


**Convocatoria:** mayo 2023


Grado en Ingeniería Informática

Facultad de Informática

Universidad Complutense de Madrid


29 de MAYO de 2023

# AGRADECIMIENTOS

En primer lugar, queremos agradecer especialmente a Belén, la directora de nuestro proyecto, por acompañarnos dándonos las pautas y correcciones que necesitábamos y confiar en nosotros para desarrollar esta idea.

Además, nos gustaría agradecer también a todos los profesores que nos han ayudado durante nuestro periodo de aprendizaje en la facultad, por transmitirnos sus conocimientos y experiencias.

Tampoco nos queríamos olvidar de agradecer a Fernando Pozuelo (Siemens) por permitirnos utilizar la tecnología de Siemens para esta aplicación así como el acceso a los datos del edificio inteligente de Siemens Tres Cantos.

También queremos agradecer a todos nuestros compañeros, alumnos de la Facultad de Informática, por darnos apoyo moral, momentos divertidos y muchas historias que recordar de nuestros años como universitarios.

Por último pero no menos importante, agradecer a nuestras familias por darnos la oportunidad de acceder a esta universidad y apoyarnos lo mejor que han podido para ayudarnos a completar la carrera exitosamente.

# RESUMEN

Este trabajo de fin de grado consiste en el desarrollo de un dashboard digital basado en la tecnología Low Code de Mendix para la visualización y procesamiento de datos de dispositivos IoT (sensores, detectores, etc.) de edificios inteligentes.

La aplicación muestra de forma gráfica e intuitiva información que recoge del edificio inteligente de Siemens en Tres Cantos y permite la conexión a otras infraestructuras adaptándose a los nuevos datos que puedan ofrecer.

Al permitir tanta flexibilidad, este dashboard está enfocado como un marco genérico donde se puede acceder a la conexión del edificio que se precise, por ejemplo, es posible conectarlo a un museo para mostrar el índice de ruido y ocupación para personas con sensibilidad acústica o trastornos del espectro autista, en la facultad de Informática mostrando la ocupación de la cafetería o información sobre eventos importantes y en hoteles para conocer servicios disponibles o comentarios y calificaciones de los huéspedes, etc. Es por ello que también se detalla el desarrollo de diversos casos de uso adicionales al implementado.

Como valor añadido, este dashboard se ha desarrollado permitiendo al usuario personalizarlo de forma sencilla con la modificación del tema, cambiando la imagen del edificio o el mensaje que se muestra.

**Palabras clave**

Low Code,

Mendix,

Datos,

Dashboard,

APIs,

Eficiencia,

Análisis

# ABSTRACT

This final degree project consists of the development of a digital dashboard based on Mendix Low Code technology for the visualization and processing of data from IoT devices (sensors, detectors, etc.) of smart buildings.

The application displays in a graphic and intuitive way information gathered from the Siemens intelligent building in Tres Cantos and allows the possibility to connect to other infrastructures, adapting to the data they may offer.

As a consequence of its flexibility, this dashboard is intended to be a generic framework where you can access the building connection you need, for example, it is possible to connect it to a museum to show the noise and occupancy rate for people with acoustic sensitivity or disorders within the autism spectrum, in a university campus showing the occupation of the cafeteria or information about important events and in hotels to know available services or comments and ratings of the guests, etc. Due to this, the development of various additional use cases to the one implemented is also detailed.

Furthermore, as an added value this dashboard has been developed allowing the user to easily customize it by modifying the theme, changing the image of the building or setting the message that is displayed.

**Keywords**

Low Code,

Mendix,

Data,

Dashboard,

APIs,

Eficiency,

Analysis

# ÍNDICE DE CONTENIDOS

# ÍNDICE DE FIGURAS

# Capítulo 1 - Introducción

## 1.1 Motivación

Las empresas y organizaciones que disponen de grandes edificios cada vez se preocupan más por el ahorro de energía y la sostenibilidad de éstos. A día de hoy, muchos edificios cuentan con sensores inteligentes que permiten realizar una medición constante del consumo energético con el fin de verificar que se cumplen los objetivos así como de estudiar maneras de mejorar la eficiencia energética.

Tomando como punto de partida esta idea, surge este dashboard digital donde se muestra de forma clara e intuitiva en las zonas más concurridas de los edificios, un resumen visual de la información que recogen los sensores del edificio como el consumo, la ocupación o la temperatura.

Además, el aspecto más destacable sobre este dashboard es que cuenta con una implementación que hace posible la fácil adaptación de cualquier tipo de sensor y dato, por lo que puede resultar de interés para una amplia variedad de casos de uso (hospitales, museos, campus universitarios, hoteles, etc).

## 1.2 Contexto

La idea de este trabajo surgió dentro de la empresa Siemens (véase bibliografía [13]) para desarrollar una solución específica para uno de sus edificios. Posteriormente, después de la toma de requisitos, vimos el potencial que esto podía conllevar y de la amplia variedad de casos de uso que se podrían implementar para diversos edificios. Por ello, nosotros de forma independiente y sin aportaciones de Siemens más allá de la idea original comentada antes, decidimos adaptar y ampliar (a la par que cubrimos los requisitos originales) el concepto original a un marco más genérico para poder proporcionar este servicio con otras finalidades (por ejemplo, presentar datos del índice de ruido y ocupación para personas con sensibilidad acústica o trastornos del espectro autista).

## 1.3 Objetivos del trabajo

El objetivo de este proyecto es mostrar a los usuarios de manera clara, sencilla y visual la información de un edificio inteligente, de manera que, dependiendo del caso de uso y las necesidades del usuario; se puedan comprobar datos como la ocupación, temperatura, nivel de ruido, eficiencia de las instalaciones en las que se encuentran, etc.

Adicionalmente, surge el objetivo de operar y procesar la información que se recibe de forma que el propietario pueda analizar los datos de cualquier intervalo de tiempo y comprobar cuáles son los puntos clave a mejorar sobre su consumo, en qué momentos del día la ocupación es mayor, etc.

## 1.4 Plan de trabajo

El desarrollo de este dashboard se ha dividido en tres fases:

1. **Fase de Investigación**

   Centrada en conocer qué datos se encontraban disponibles a partir de los diferentes sensores del edificio (y escoger los que resultaban más interesantes para el usuario) y comprobar de qué manera pudieran ser recogidos desde Mendix. Fueron necesarias también múltiples sesiones de aprendizaje (sobre conexiones a APIs y traducción de datos XML y JSON a objetos en Mendix) adicionales a nuestros previos conocimientos de esta herramienta.

   Llegados a este punto ya contábamos con los datos necesarios en Mendix para el desarrollo visual de la aplicación, por tanto el desarrollo del back-end tuvo lugar de forma paralela a la investigación.

2. **Fase de Planificación**

   Realizamos numerosas sesiones de brainstorming para la elaboración de varios conceptos de diseño y fuimos modificándolo con la intención de acercarnos a las necesidades del usuario final. También se tuvieron en cuenta otras funcionalidades más allá del diseño, como la configuración y personalización de la aplicación.

3. **Fase de Implementación**

   Una vez escogidas unas bases sobre el diseño y la información a mostrar, pasamos al desarrollo final de la aplicación.

4. **Diseño e investigación sobre casos de uso adicionales**

   Se procede a investigar y detallar nuevas soluciones en las que Kiosko Digital encajaría satisfactoriamente haciendo uso de datos de otras naturalezas y con propósitos diversos. Para el diseño detallado se utiliza la herramienta InkScape (véase bibliografía [9]) donde se muestran los gráficos e información adaptados a cada caso de uso. Esta herramienta es un software de código libre de vectores gráficos utilizado para la edición de imágenes, diagramas y gráficos (similar a photoshop) cuyo formato principal es SVG (Scalable Vector Graphics) que cumple con los estándares XML y CSS2.

## 1.5 Estructura del trabajo

Este trabajo está dividido en cinco capítulos principales con la estructura que se comenta a continuación:

En el Capítulo 1 se explican diferentes aspectos de interés como la motivación que originó esta solución, el contexto en el que se desarrolla, los objetivos a los que se llegan y el plan y estructura del trabajo.

El Capítulo 2 trata de las tecnologías y herramientas que se utilizan. Este capítulo comienza con una introducción de la tecnología Low Code, su historia e importancia actual, además de un breve apartado para la historia de la plataforma utilizada: Mendix. También se realiza una comparativa con la programación tradicional mostrando los beneficios de este entorno de desarrollo.

Tras esta introducción al Low Code, nos centramos en explicar la plataforma con la que hemos trabajado: Mendix. Esta explicación abarca temas de importancia como

su historia, su metodología ágil (*Agile*), componentes, tecnologías, conexiones y funcionalidades utilizadas.

El Capítulo 3 detalla información sobre el dashboard que hemos desarrollado: Kiosko Digital; así como la investigación que hemos realizado, la implementación de conexiones a APIs, el uso de gráficos y las sesiones de diseño de la interfaz.

En el Capítulo 4 reunimos los diferentes casos de uso en los que se puede utilizar este dashboard que extienden y adaptan el ámbito de la solución a distintos requerimientos que se precisen según las necesidades de los usuarios y los usos del edificio en cuestión.

Por último, en el Capítulo 5 se detallan las conclusiones finales del trabajo, recordando los objetivos iniciales cumplidos además del trabajo futuro, donde se valoran distintas funcionalidades para aumentar y escalar el valor de la solución actual.

Los capítulos posteriores contienen las traducciones al inglés de la Introducción y Conclusiones Finales así como las contribuciones personales de cada miembro del equipo de desarrollo de este proyecto.

# Capítulo 2 - Estado del arte

## 2.1 Historia del Low Code

**Años 70 - 90: 4GL**

El Lenguaje de Programación de Cuarta Generación (4GL) es el precursor de las plataformas de desarrollo low-code, un concepto que comenzó entre los años 70 y 90, análogamente con la mayor parte del desarrollo del 3GL (véase bibliografía [2]).

Los lenguajes 4GL (ABAP, Unix Shell, SQL) son más intuitivos a nivel de programación que los 3GL (C, C++, Java, Python), pues tienen una sintaxis más cercana al lenguaje natural. Además, suelen incluir interfaces gráficas y capacidades de gestión más avanzadas.

**Años 90: Desarrollo Rápido de Aplicaciones (RAD)**

El Desarrollo Rápido de Aplicaciones (RAD) cobró impulso cuando se extendió el concepto de "montar" visualmente las aplicaciones de escritorio utilizando herramientas como Visual Basic, Delphi y Oracle Forms.

Las herramientas RAD destacan por ser entornos fáciles de aprender, ya que tanto la interfaz de usuario como la lógica pueden almacenarse en una paleta de componentes.

**2000 en adelante**

A partir de la década del 2000, la demanda de aplicaciones empresariales personalizadas y la necesidad de acelerar los tiempos de desarrollo para la posterior comercialización, hizo que el término "low code" comenzara a popularizarse. En este contexto, estas herramientas se convirtieron en una gran alternativa para empresas y desarrolladores que buscaban crear aplicaciones más rápido, con menos recursos y sin la necesidad de escribir mucho código a mano.

Hoy en día, el low code se utiliza en una gran variedad de aplicaciones y se espera que su popularidad siga creciendo en los próximos años.

**Historia de Mendix**

La empresa se fundó en Rotterdam , Países Bajos en 2005 por Derek Roos (véase figura 2.1), Roald Kruit y Derckjan Kruit y posteriormente trasladó su sede a los Estados Unidos a principios de 2012.

Rápidamente, se convirtió en una de las plataformas líderes de desarrollo low code y para 2011 ya había facturado más de 13 millones de dólares gracias a la financiación recibida por Prime Ventures. Posteriormente, en 2014 anunció 20 socios entre los que se encontraban Accenture (véase bibliografía [15]) y Capgemini (véase bibliografía [14]). Finalmente, en 2018, gracias a la popularidad de la plataforma, captó la atención de la empresa de Siemens que la adquirió por valor de 730 millones de dólares.

Actualmente, Mendix (véase figura 2.2) sigue formando parte de Siemens pero actúa como una empresa subsidiaria independiente con más de 500 empleados.

Figura 2.1: Derek Roos

Figura 2.1: Logo de Mendix

## 2.2 Beneficios del Low Code

El desarrollo Low Code (véase bibliografía [8]) presenta una serie de ventajas muy características, diferenciándolo muy significativamente de otros tipos de programación más tradicionales (véase bibliografía [19]).

- El tiempo de desarrollo y modificaciones se reduce considerablemente al dejar de lado una gran parte de la codificación manual. Esto hace que se aceleren los tiempos de desarrollo de aplicaciones y que aumente la productividad permitiendo a los desarrolladores dedicarle más tiempo a tareas de mayor criticidad.
- Al reducir la cantidad de código manual que se debe generar en una aplicación se abaratan los costes del desarrollo y el mantenimiento, además de ser más sencilla la depuración y modificaciones de la solución.

- Los desarrolladores menos experimentados son capaces de construir aplicaciones básicas por la sencillez de este tipo de desarrollo, lo que supone un punto de partida más accesible a la programación.

## 2.3 Introducción a la tecnología Mendix

Mendix(véase bibliografía [7]) es una plataforma de desarrollo Low Code para desarrollar soluciones y aplicaciones de forma sencilla e intuitiva. Ofrece una simplificación de la programación manual con código expresado de forma gráfica y visual para acelerar el tiempo de desarrollo reduciendo así la carga de trabajo del equipo.

Esta plataforma tiene una arquitectura basada en Cloud, lo que permite reducir la dificultad de subir las aplicaciones a la nube, pudiendo solicitar un nodo de AWS (*Amazon Web Services,* véase bibliografía [17]) o *Google Cloud* dentro del portal de Mendix. Amazon Web Services (AWS) (vease bibliografia [20]) es un proveedor basado en computación en la nube (*cloud computing*) que permite, entre otros, el almacenamiento y computación de recursos cuyo objetivo es que los usuarios paguen exclusivamente los servicios que consumen (*pay-by-usage*), siendo este coste variable y mensual en función de las necesidades de cada usuario o empresa. La idea principal es que se pueda acceder a cualquiera de los servicios a través de APIs lo que permite que la capacidad contratada y recursos se configuren automáticamente sin necesidad de que el usuario esté siempre pendiente de configurarlo cuando necesite algún ajuste en su infraestructura (véase bibliografía [21]).

Además de la conexión automatizada a infraestructuras externas para el almacenamiento dinámico de datos, Mendix también presenta una amplia gama de herramientas, características y servicios para los desarrolladores, entre ella se encuentran:

**Plantillas**: los usuarios pueden elegir de entre una gran variedad de plantillas para comenzar a crear sus aplicaciones y personalizarlas para que se adecuen a su caso de uso.

**Módulos**: Mendix tiene integrado un Marketplace donde se pueden encontrar muchísimos módulos útiles para realizar conexiones con bases de datos externas, sistemas CRM, ERP y otros adaptándose a las necesidades del usuario.

**Seguridad**: también se ofrecen herramientas y facilidades de seguridad para garantizar el buen uso de la aplicación, permitir o denegar permisos de acceso, cumplir los estándares de seguridad del mercado, protección contra amenazas, autenticación avanzada, etc.

**Control de versiones**: la plataforma tiene integrada una herramienta de control de versiones basada en Git permitiendo la colaboración de los desarrolladores de manera efectiva, eficiente y rápida.

### 2.3.1 Portal de Mendix

Mendix ofrece un portal online a los desarrolladores para la gestión de las soluciones, donde se encuentran diferentes funcionalidades de utilidad (véase figura 2.3):

- **Requisitos (*Stories*)**: en esta pestaña se puede crear, modificar y completar la lista de requerimientos del usuario (*user stories)*, estos son la recopilación de requisitos de la aplicación, donde se define la tarea y la estimación de esfuerzo. En Mendix se utiliza una metodología SCRUM (metodología *Agile* que se explica en la sección 2.3.2).

- **Equipo (*Team*)**: permite invitar a desarrolladores, usuarios y organizadores de la aplicación.

- **Realimentación (*Feedback*)**: Mendix cuenta con la posibilidad de añadir en los desarrollos una herramienta de realimentación (*feedback*), donde el usuario es capaz de comunicar un error o la falta de algún componente con un mensaje y una captura de la pantalla donde se encuentra. Este mensaje aparece en la pestaña de realimentación y permite a los desarrolladores guardar un listado de errores, marcarlos como solucionados para que se le notifique al usuario, o comentarlos para discutir sobre ese problema.

- **Entornos (*Environments*)**: esta pestaña maneja el despliegue en la nube de la aplicación (Mendix utiliza servicios de AWS). Dentro de Mendix se ofrece la posibilidad de solicitar un nodo donde subir cualquier commit que se desee, los desarrolladores pueden seleccionar que commits se quieren convertir a un paquete de despliegue para subirlos al entorno web como una nueva versión. Normalmente se cuenta con dos entornos en la nube, *acceptance* (entorno de test) y *productive* (donde estará alojada la versión final de la aplicación).

- **Métricas (*Metrics*)**: esta pestaña muestra un gráfico visual sobre el estado y desempeño actual del nodo, donde se muestran datos de interés como el almacenamiento utilizado, el número de conexiones, log-ins, rendimiento…

- **Registros (*Logs*)**: desde el portal de Mendix es posible revisar y almacenar los registros de actividad (*logs*) tanto a tiempo real como en los ficheros generados diariamente para cada uno de los entornos de despliegue de la aplicación.

- **Copias de seguridad (*Backups*)**: permite la generación, descarga e importación de copias de seguridad de la base de datos, con la posibilidad de obtenerlos solo de objetos, ficheros o ambos.

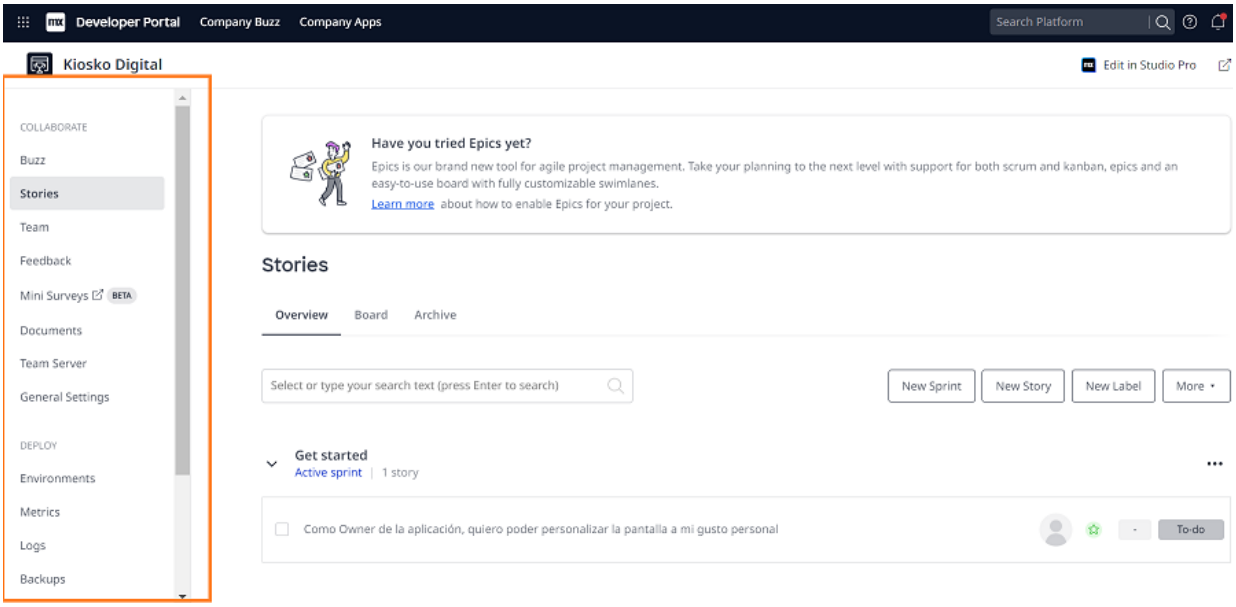



Figura 2.3: Visión general del portal web de Mendix

### 2.3.2 Metodología Agile

La tecnología Mendix está basada en la colaboración entre desarrolladores y usuarios finales, por ese motivo se orienta a la metodología ágil (*agile*) llamada SCRUM (véase bibliografía [18]).

Scrum se basa en un ciclo de trabajo iterativo e incremental dividido en intervalos de desarrollo (*sprints*) de una a cuatro semanas. La pestaña de Requisitos (*Stories)* facilita la toma de funcionalidades y el planteamiento de dichos *sprints*, pudiendo añadir las tareas que se consideren oportunas al *sprint* actual. Tras terminar todas las tareas de un

*sprint* en específico se organiza una reunión con los desarrolladores (*Business Engineers*), el organizador (*SCRUM Master*) y el cliente (*Product Owner*) en la que los desarrolladores muestran el incremento de la aplicación al cliente y a los interesados y se discute sobre los cambios realizados y las modificaciones o requerimientos a implementar.

Las responsabilidades de cada rol son las siguientes (véase bibliografía [22]):

- **Organizador (*SCRUM Master*)**: su principal función es la de establecer una comunicación fluida y satisfactoria entre los desarrolladores y el cliente, así como de la resolución de problemas que obstaculicen el desarrollo de la aplicación.

- **Desarrolladores (*Business Engineers*)**: deciden cómo se van a lograr los requerimientos de la aplicación y son los encargados de elaborar y presentar esta solución. Antes de comenzar el desarrollo, detallan la estimación de esfuerzo, costes y organización de las distintas funcionalidades de la solución (proceso que se repite de forma iterativa con cada nueva funcionalidad que precise el cliente, llamadas *change requests*).

- **Cliente (*Product Owner*)**: se trata del principal encargado de definir los requisitos de la aplicación y mostrarlos al equipo de desarrollo para su posterior estimación de esfuerzo y división de las funcionalidades en intervalos (*sprints*). Cualquier decisión necesaria sobre los detalles de las funcionalidades recaen sobre este rol.

### 2.3.3 Front end

Una de las características más importantes de Mendix es la facilidad del desarrollo de la aplicación. En lo que respecta al *front end*, Mendix ofrece una serie de herramientas y funcionalidades que permiten moldear la visualización, estas son:

- **Páginas**: dentro de Mendix Studio Pro, se puede configurar la distribución e información que se muestra en las páginas de la aplicación (esto se traduce a una página .html). En estas páginas es posible añadir componentes como layouts (columnas, filas y estructura global de la página), containers (los div en HTML), listas o tablas con información de objetos, botones con funcionalidades, textos, elementos de entrada de datos (textos, checkbox, date picker, selectores de objetos, etc), imágenes, menús y muchos otros.

- **Widgets y gráficos**: adicionalmente, Mendix cuenta con un mercado gratuito online (*Marketplace*) donde se pueden obtener componentes de interés para completar la funcionalidad, y un apartado de gráficos configurables y personalizables donde se pueden reflejar análisis de objetos e información de la base de datos.

- **Estilo**: dada la posibilidad de desarrollar páginas configurando su distribución e información, también se le ofrece al programador la versatilidad de poder aplicar estilos CSS o SCSS a la página en general o a sus distintos componentes individualmente, pudiendo aplicar estilo a cualquier elemento de la página (textos, botones, widgets, fondos, etc.).

### 2.3.4 Back end

Trás comentar las características del desarrollo del front end en Mendix pasamos a comentar la lógica y procesos internos que el usuario no visualiza. Los elementos del back end que ofrece esta plataforma son:

- **Gestión de acceso de roles de usuario**: en Mendix Studio Pro se le ofrece al programador la capacidad de definir roles de usuario para cada tipo de usuario que vaya a acceder a la aplicación. Estos roles de usuario sirven para definir posteriormente el acceso y permisos de cada uno de forma que no puedan obtener información que no deben, o ver páginas, datos o botones que no deberían. Estos accesos son completamente configurables de forma que se puede especificar qué páginas pueden ver, qué microflows pueden ejecutar, y qué objetos pueden leer, escribir, crear o eliminar. Esta funcionalidad permite realizar una gestión de la seguridad de la aplicación muy sólida, la privacidad de la información y gran capacidad de modificación de permisos.

- **Microflows**: permiten expresar la lógica funcional de la aplicación y procesos internos en un diagrama de actividades. Se trata de una forma visual e intuitiva del flujo de ejecución de una funcionalidad o acción. A pesar de que Mendix se trate de una plataforma low-code, dentro de estos microflows es necesario el conocimiento de bucles, condiciones, programación orientada a objetos… Ya que en las decisiones es necesario realizar cálculos y en los bucles y manejo de objetos se puede necesitar acceder a datos mediante código, utilizar variables de tipos primitivos o entidades para procesar, calcular y almacenar la solución que requiera esta lógica.

Dentro de los microflows se pueden ejecutar acciones como:

- Crear y actualizar objetos o ficheros
- Mostrar páginas
- Tomar decisiones y realizar bucles
- Llamar a código Java, REST APIs u otros microflows
- Generar mensajes o *logs*

- **Base de datos (*Domain Model*)**: La plataforma Mendix cuenta con un sistema de bases de datos integrado en las propias aplicaciones basado en SQL donde el diseño e implementación destacan por su sencillez. El *Domain Model* es un modelo que describe la información y los datos utilizados por la aplicación en forma de entidades con atributos tipados y relaciones con otras entidades representadas por asociaciones. También cuenta con la posibilidad de crear entidades con herencia y entidades no persistentes (no permanecen almacenados en la base de datos).

## 2.4 Introducción a los Dashboards

Un dashboard es una herramienta para procesar información a tiempo real y mostrarla de forma visual y resumida a través de gráficos y valores agregados. Esta interfaz ofrece una gran utilidad en diversos campos como en finanzas de las empresas, rendimiento del marketing, índice de ocupación, Big Data, etc. Por lo que está aumentando su demanda y popularidad de este tipo de soluciones.

En el mercado existen múltiples dashboards muy útiles, pero hay una en particular que destaca entre las demás: ClickData (véase bibliografía [6]) .

ClickData es un dashboard de código libre capaz de conectarse a múltiples bases de datos y a APIs para recoger información a tiempo real.

A diferencia de ClipData, Kiosko Digital posee la ventaja de adaptarse a cualquier información nueva de forma rápida y eficiente. Explotando la flexibilidad de la tecnología Low Code de Mendix, hemos podido obtener un dashboard de marco general para cualquier conexión IoT a edificios inteligentes, no solo con información energética y de ocupación, sino de cualquier tipo de datos que obtengan sus sensores.

De forma adicional, nuestra aplicación también ofrece una capacidad de personalización más allá de cambiar los gráficos. Los usuarios son capaces de personalizar las imágenes que aparecen, los colores y los mensajes que deseen, lo que supone un valor añadido sobre otros dashboards del mercado.

# Capítulo 3 - Kiosko Digital

La idea de Kiosko Digital surge con el objetivo de mostrar datos de interés del edificio inteligente de Siemens, con una interfaz visual agradable e intuitiva para permitir a cualquier usuario consultar toda la información de un vistazo.

## 3.1 Análisis de requisitos e investigación

Con el fin de alcanzar el objetivo deseado, el primer paso consiste en la búsqueda de las fuentes de información más prometedoras sobre el consumo del edificio. Realizando una breve exploración, se valora que la opción más atractiva es la del Sistema de Gestión de Edificios (SGE) Enlighted (véase bibliografía [1]) .

### 3.1.1 Enlighted

Enlighted Inc. es una empresa de alcance mundial que ofrece una solución sostenible y eficiente de gestión de edificios que monitorea y controla infraestructuras inteligentes. Algunas de las ubicaciones más destacadas en las que se ha implantado este sistema son la la universidad Long Beach de California, la Universidad de Birmingham, AT & T, Loom Barcelona, etc.

La solución se basa en la instalación de sensores y dispositivos avanzados que permiten la recopilación y procesamiento de datos a tiempo real sobre el entorno y las

condiciones que rodean dichos sensores. La información recogida es posteriormente analizada con la ayuda de algoritmos para obtener un informe de la situación en la que se encuentra y actuar en consecuencia aplicando una serie de medidas para optimizar el consumo y ahorro de energía (p.ej. regulando la intensidad de la iluminación en base a la cantidad de luz natural).

En el caso del edificio de Siemens en Tres Cantos, estos sensores se encuentran principalmente alojados en los componentes de la luminaria y enchufes (plugins) de las distintas estancias de trabajo (despachos, salas de reuniones, áreas comunes, etc.) y recogen información de diversa naturaleza como la ocupación, humedad, iluminación, temperatura y consumo. El proceso funciona de forma que los sensores envían a tiempo real sus datos a los dispositivos gateway que, a su vez, vuelcan la información a la plataforma que se encarga de la gestión. Esta plataforma es la que cuenta con un repertorio de APIs por las que es posible obtener dicha información.

### 3.1.2 APIs

Dentro de la plataforma de Enlighted, existe un espacio donde los usuarios pueden investigar y documentarse sobre los servicios API REST mediante los que se pueden obtener todos los datos de los sensores del edificio (véase bibliografía [3] y figura 3.1).

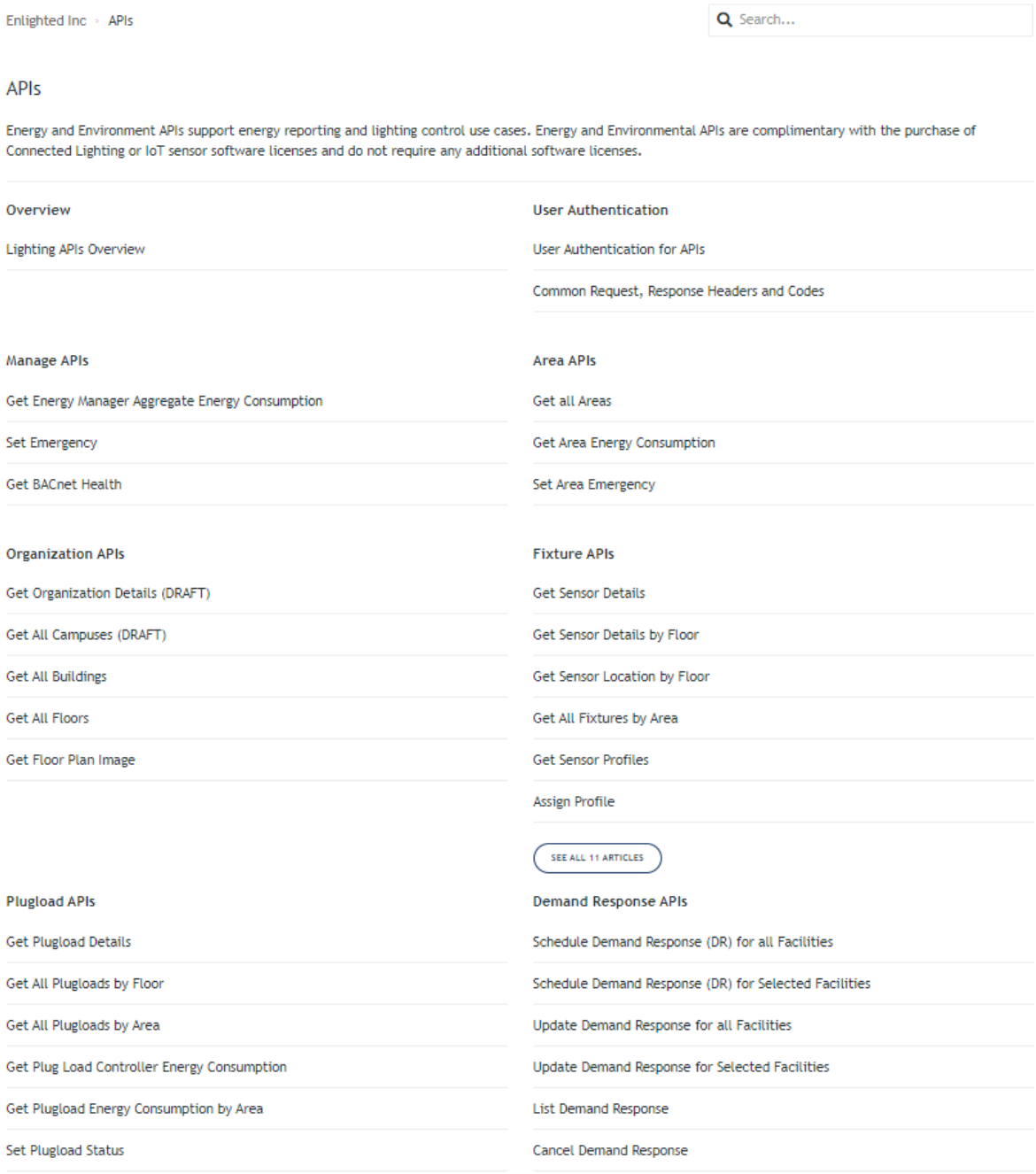



Figura 3.1: Índice de la sección de APIs que ofrece Enlighted.

Para realizar cualquier conexión a una API de Enlighted, es necesaria la obtención previa de los siguientes valores:

- El nombre de usuario: Para poder usar cualquier API es necesario tener un usuario registrado en la plataforma, así como tener permisos de administración del edificio del que se quieran obtener los datos.

  En el caso de nuestro proyecto se nos confirieron los permisos de administración de la planta 5ª del edificio D de Siemens Tres Cantos en Madrid.

- La API Key asociada a dicho usuario: se debe solicitar una API Key que solo se obtiene dentro de la plataforma con la sesión iniciada. Por motivos de seguridad, esta API Key tiene un periodo de validez de un mes, después del cual se debe volver a solicitar una nueva.
- La fecha y hora actuales: para poder obtener la información actualizada hasta el momento en el que se llama a la API. La fecha y hora se convierte a un tipo *long* para obtener lo que se llama *time stamp*.
- El ID de la planta. Lo conseguimos mediante la llamada a la API "Get All Floors" (*https://{em_ip_address_or_hostname}/ems/api/org/floor/v1/list*), siendo el hostname *siemens1.emc.enlightedinc.com* . Esta API devuelve la lista de todos los pisos del edificio de la organización. Esta llamada la realizamos una sola vez a través de la plataforma de *Postman Agent* y comprobamos que el id de la planta 5 era el 35.

Todos estos valores se añaden en las cabeceras, aunque a diferencia del nombre de usuario y time stamp, la API Key se debe encriptar mediante SHA1 junto con los otros dos valores, es decir, se aplica SHA1 a ("nombre de usuario" + "API Key" + "Time stamp").

#### 3.1.2.1 APIs utilizadas

Las APIs que hemos utilizado para el desarrollo de Kiosko Digital son las siguientes:

- Get Aggregate Sensor Energy Data by Floor:

  *https://{em_ip_address_or_hostname}/ems/api/org/facility/v2/facilityEnergyStatsOfFloor/15min/floor/{floorID}/{from_date}/{to_date}*

  Esta API devuelve el consumo total de energía en vatios/hora para todos los sensores y enchufes de la planta en intervalos de quince minutos (96 registros por día). Los datos relativos al consumo de energía se pueden obtener con un rango máximo de cuarenta y cinco (45) días entre el inicio y el final del intervalo de tiempo (*from_date} y {to_date*).

  Más específicamente, la API devuelve los siguientes datos agregados:

  - Energía Base (*Base Energy*): energía que consumiría el dispositivo si estuviera trabajando al 100% de su rendimiento durante el periodo completo del intervalo de tiempo indicado. En otras palabras, es la energía consumida en el peor de los escenarios posibles en el intervalo de tiempo.
  - Energía (*Energy*): referente a la energía real consumida por todos los sensores una vez realizada la optimización de ahorro.
  - Energía ahorrada (*Saved Energy*): representa el ahorro total de los sensores. Se corresponde a la diferencia entre la energía base y la energía consumida. Asimismo, esta energía es el resultado del sumatorio del ahorro energético por ocupación, ambiente, *tuneup* y manual.
  - Ahorro por ocupación: se trata de la energía ahorrada relativa a los registros de ocupación de los sensores. Los sensores están programados de forma que en caso de que detecten un índice de ocupación bajo o

inexistente, reducen en consecuencia el nivel de luminosidad de los dispositivos en los que se encuentran instalados; reduciendo así el consumo.

- Ahorro por la luminosidad del ambiente: Los sensores cuentan también con un sistema de detección de nivel de brillo del ambiente. Si el sensor detecta buena luminosidad en la zona en la que se encuentra debido a la luz solar, reduce su luminosidad ahorrando energía.
- Ahorro por eficiencia *tuneup*: además de las medidas ya mencionadas, los sensores son capaces de tener en cuenta la luminosidad de otros sensores adyacentes para en consecuencia ajustar su propio nivel de brillo.

  En posteriores capítulos nos referiremos a este tipo de ahorro como *Eficiencia Enlighted*.

- Ahorro manual: la energía ahorrada en el caso en el que los dispositivos se apaguen o reduzcan su luminosidad manualmente.

- Get Floor Plan Image:

  *https://{em_ip_address_or_hostname}/ems/api/org/floor/{floor_id}*

  Esta API devuelve una imagen que resume la distribución de la planta especificada.

- Get Detailed Fine Grain Sensor Data:

  https://{em_ip_address_or_hostname}/ems/api/org/sensor/stats/floor/{floor_id}/{from date}/{to date}

Devuelve los datos (con especial detalle en el valor de la ocupación) de todos los sensores en la planta indicada en intervalos de cinco minutos durante el período de tiempo especificado (con un intervalo total máximo de una hora). Los valores que devuelve esta API son los siguientes:

- Energía (*power*): el consumo en vatios del sensor durante los últimos cinco minutos del intervalo.
- Temperatura (*temperature*): la temperatura de los sensores en *Fahrenheit*.
- Ocupación (*occupancy*): se obtiene un número en formato decimal que equivale a un valor binario de 64 bits donde cada bit indica la ocupación correspondiente a cinco segundos del intervalo de cinco minutos (siendo 1 si detecta ocupación y 0 en caso contrario). Por ejemplo, el valor en decimal 11528601896536801 28 convertido a binario sería 00001111 1111111111001000 00111011 11111111 11111111 01000000 00000000.

- Open Weather API:

  Esta API forma parte del repertorio de APIs de código abierto de Open Weather (no de Enlighted) relacionadas con el clima y la temperatura (véase bibliografía [4]) .

  Devuelve los datos meteorológicos actuales para cualquier ubicación en el mundo, incluidas más de 200,000 ciudades. Se recopilan y procesan datos meteorológicos de diferentes fuentes, como modelos meteorológicos globales y locales, satélites, radares y una amplia red de estaciones meteorológicas.

  https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={API key}

Los argumentos necesarios para la llamada a esta API son los siguientes:

- latitud y longitud de la ubicación de la que se desea obtener los datos (conseguida a través de Google Maps).
- API Key: obtenida desde la página oficial de Open Weather tras registrarse de forma gratuita.

#### 3.1.2.2 APIs en Mendix

Tras comprobar que se podrían realizar todas estas conexiones a las APIs desde *Postman Agent* (plataforma online para pruebas de funcionamiento de APIs, véase bibliografía [16]) comenzó la investigación para realizar la conexión desde la aplicación de Mendix (véanse bibliografías [5] y [12]) .

En la plataforma Mendix existe una gran cantidad de información útil para el aprendizaje de conexiones API (foro, *learning paths*, wikis…). La fuente de información más completa es la wiki *Consume a REST Service*, donde se explica todo el proceso de conexión a las APIs, así como la adaptación de la información a objetos de la base de datos de la aplicación.

Antes de realizar la conexión, es necesario definir una estructura para los mensajes JSON y XML (véase figura 3.2 y 3.3), de forma que la aplicación pueda procesar adecuadamente dichos mensajes y obtener la información (clave y valor) respectiva sin errores en el formato.

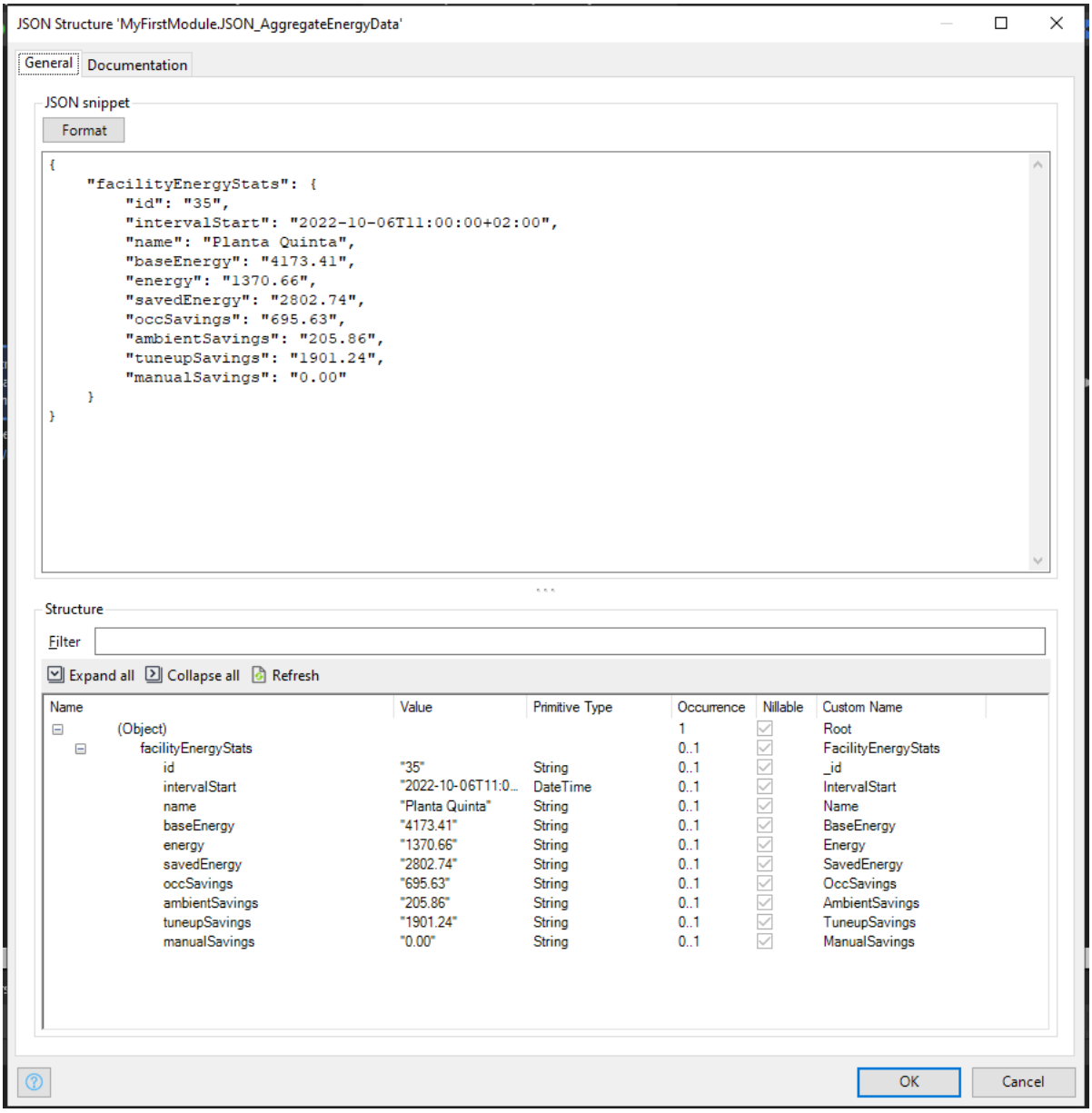



Figura 3.2: JSON structure

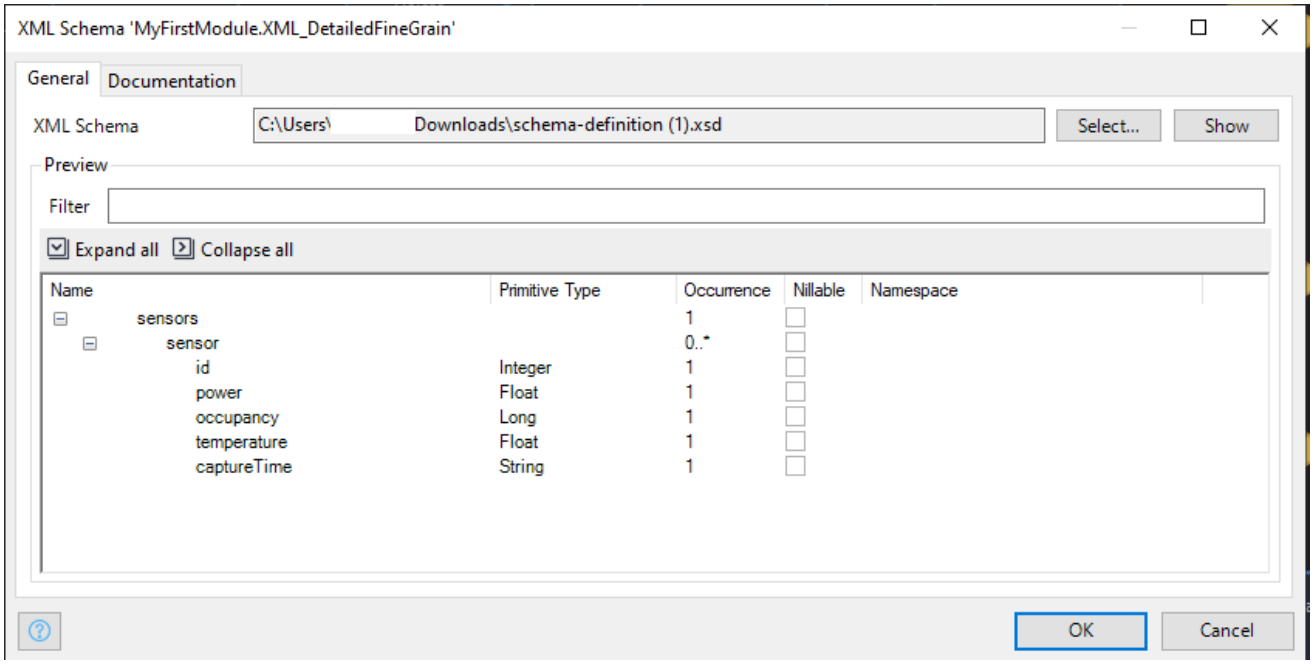



Figura 3.3: XML Structure

Posteriormente, para realizar la transformación de los datos en formato JSON o XML, en Mendix se utiliza una funcionalidad llamada *Import Mapping* (véase figura 3.4). Esta herramienta necesita conocer la estructura del mensaje JSON o XML y sus respectivos objetos en la aplicación para pasar cada dato particular a su atributo especificado. De forma más específica, los *Import Mapping* asocian los valores (clave, valor) de un mensaje a un atributo con el mismo tipado de un objeto de Mendix.

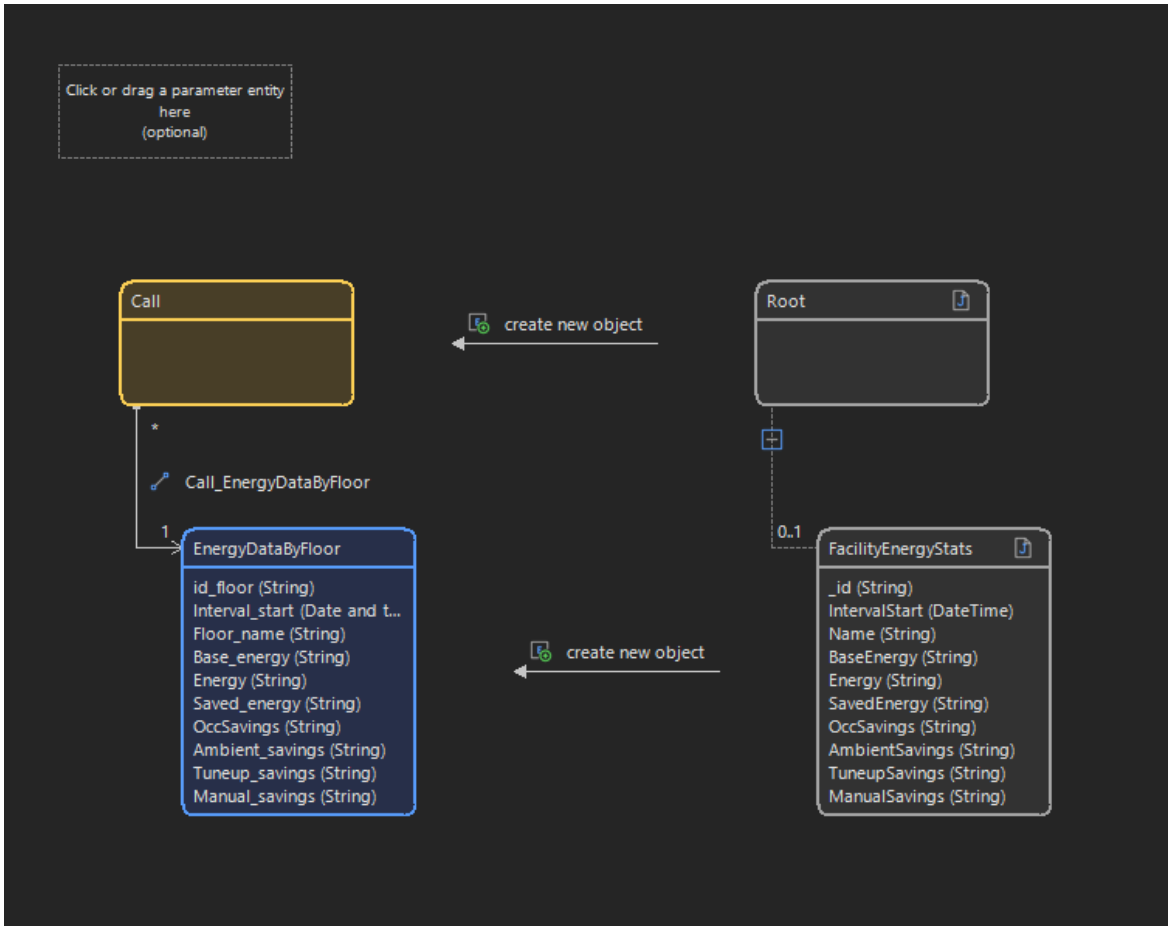



Figura 3.4: Ejemplo de Import Mapping

Una vez se tiene el *Import Mapping* específico de la API REST a la que se desea conectar, se debe construir una lógica de conexión en un Microflow, donde se opere con la información necesaria antes de la llamada a la API, y tras esas operaciones añadir una actividad *Call REST Service* (véase figuras 3.5, 3.6 y 3.7). Ésta permite conectarse al REST Services de tipo GET o POST (para conectarse a un POST REST Service se realiza antes un *Export Mapping*). Para cumplimentar correctamente toda la información el desarrollador debe conocer la url de la conexión, el método HTTP (GET o POST), las cabeceras necesarias y el Import o Export Mapping dependiendo del método HTTP especificado.

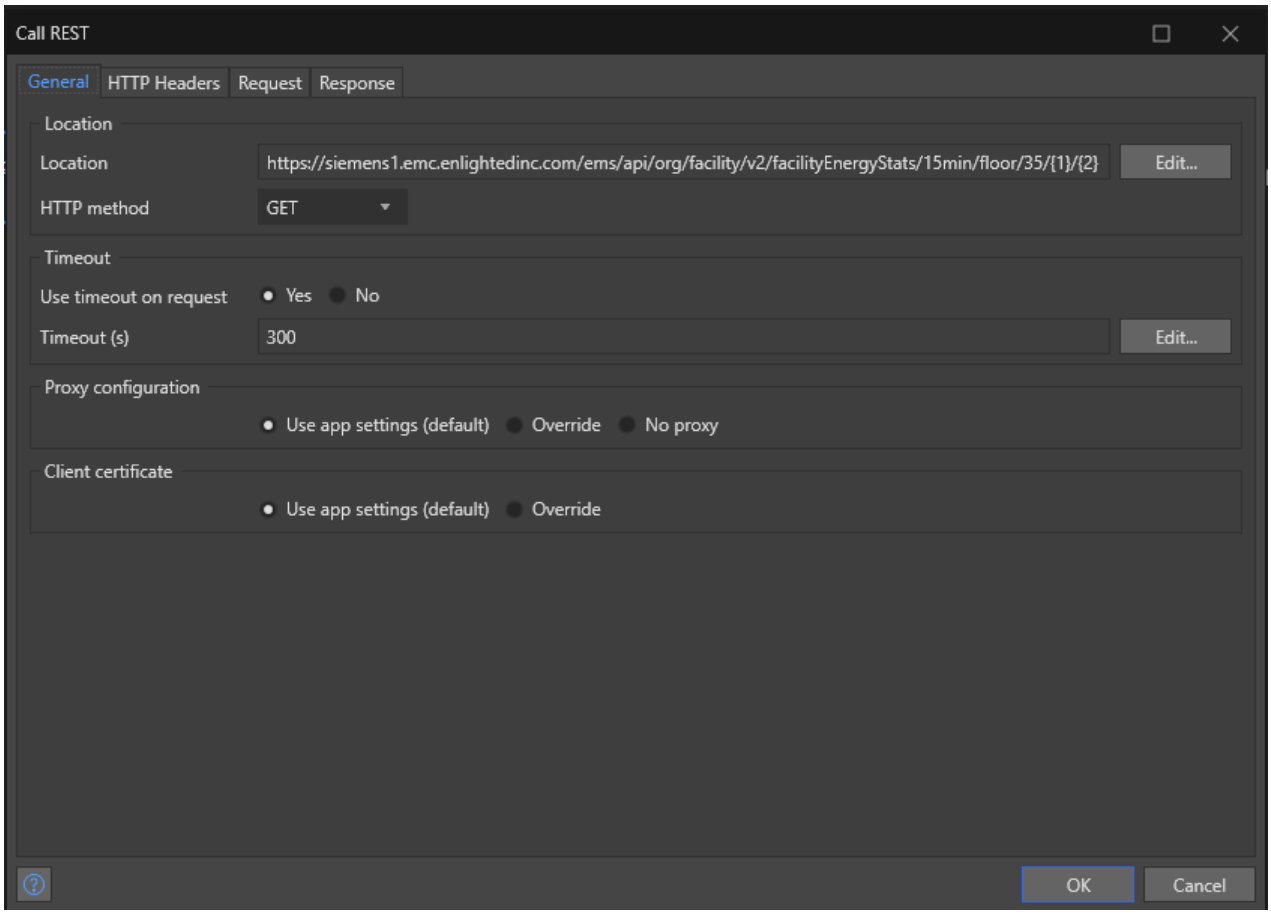



Figura 3.5: Call REST Service Activity, General

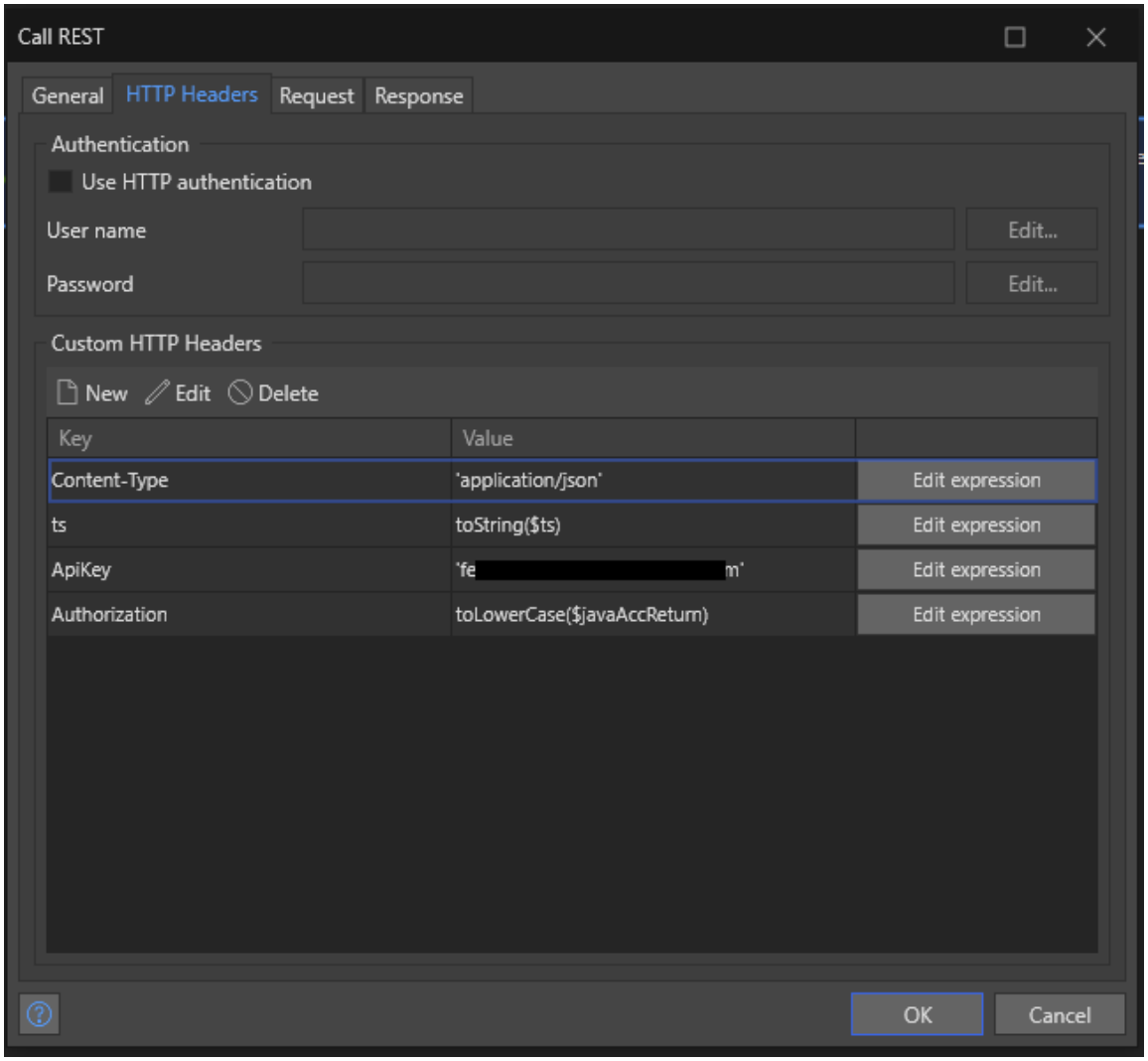



Figura 3.6: Llamada a un servicio REST, Cabeceras HTTP

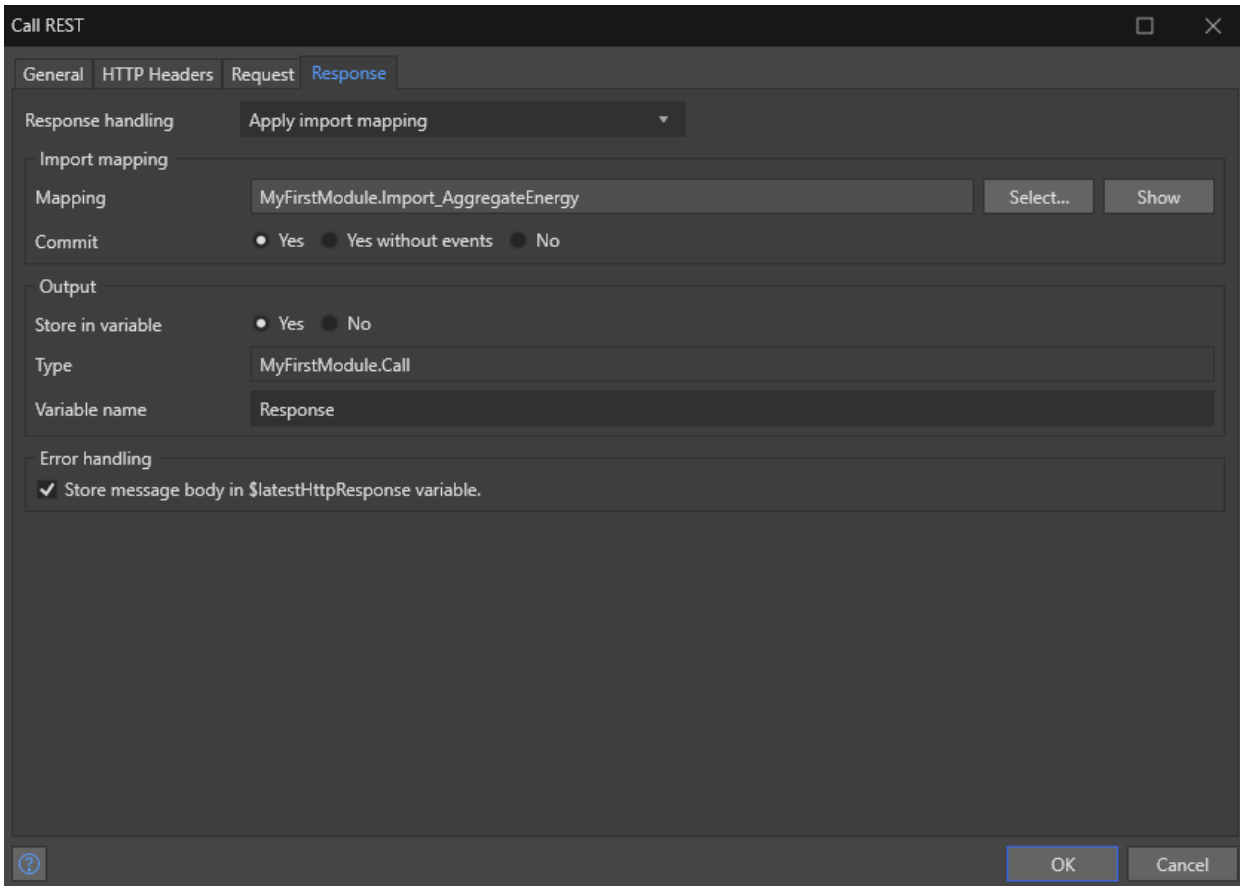



Figura 3.7: Call REST Service, Response

## 3.2 Implementación

Tras un largo proceso de investigación, tanto de la información a utilizar como de las fuentes disponibles y de su acceso desde Mendix, pasamos a implementar este caso de uso: Kiosko digital.

La implementación de la solución de este proyecto se compone de diferentes secciones. Por un lado tenemos el back-end de la aplicación, donde se encuentran la base de datos (también llamado *domain model* en Mendix) y la lógica interna de la

aplicación tanto para conexiones como para el tratamiento de los datos adquiridos (*microflows*); mientras que por el otro contamos con el front-end, que se trata de un aspecto de gran importancia en esta solución.

### 3.2.1 Domain Model

La base de datos en Mendix se define a través de un Domain Model, que no es más que un modelo gráfico entidad-relación donde es posible crear el diagrama de la topología completa de las asociaciones y atributos de los distintos objetos de Mendix. La Base de datos de este dashboard almacena toda la información necesaria de las conexiones del edificio inteligente elegido dependiendo de las APIs utilizadas en el momento. En las imágenes e información que se muestran a continuación se utiliza el ejemplo del edificio de Siemens en Tres Cantos.

El modelo de Kiosko Digital cuenta con la siguiente estructura de entidades (véase figura 3.8).

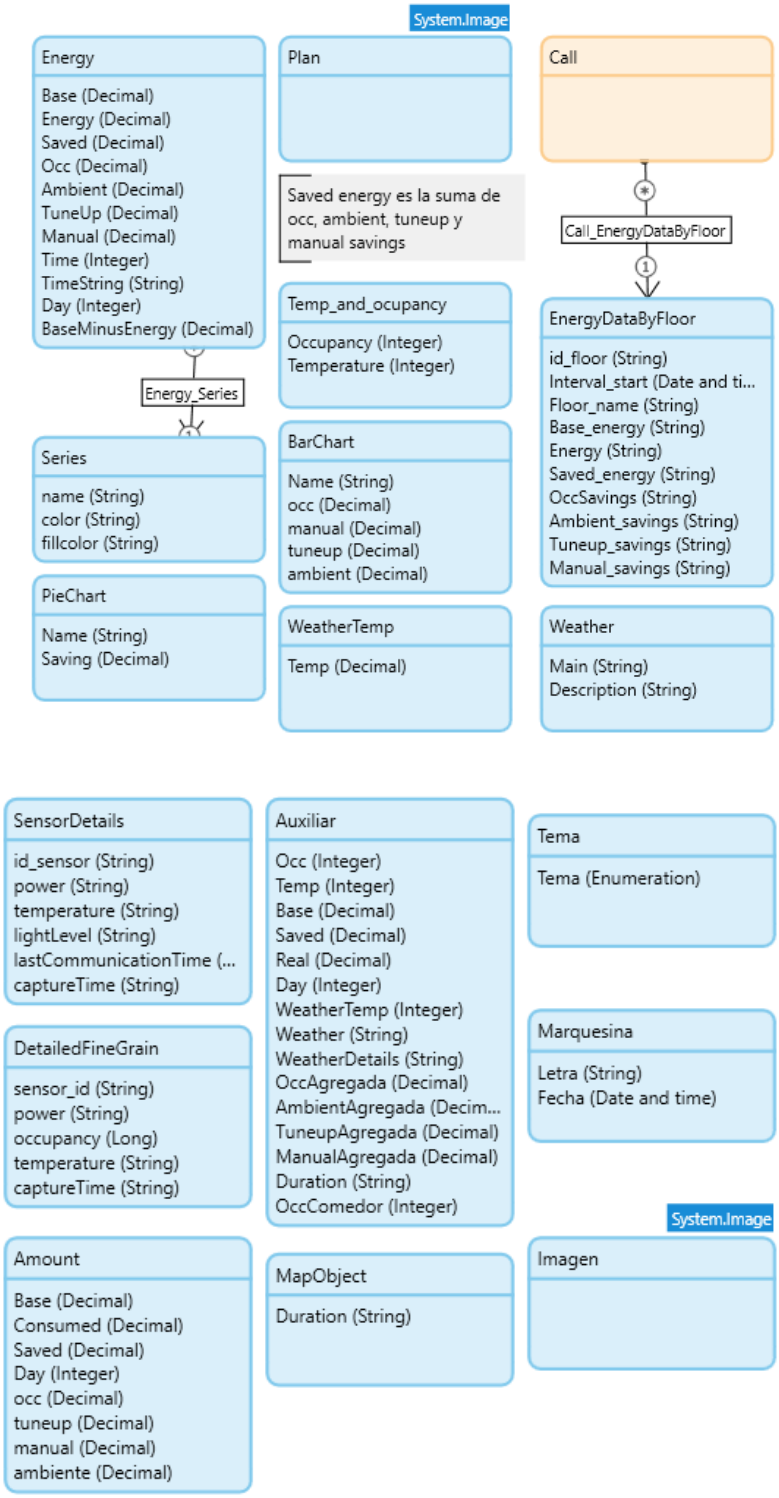



Figura 3.8:Domain Model de Kiosko Digital

Donde las clases más importantes del proyecto son:

- EnergyDataByFloor: recibe la información obtenida a través de la API Get Aggregate Sensor Energy data by floor (capítulo 3.1.2.1).
- DetailedFineGrain: comprende la información obtenida a través de la API Detailed fine grain sensor data(capítulo 3.1.2.1).
- SensorDetails: almacena la información obtenida a través de la API Sensor Details by floor(capítulo 3.1.2.1).
- Energy: transforma todos los datos recibidos como String a los tipos concretos de cada atributo (Double, Date and time, Integer) para facilitar las futuras operaciones y mostrar el gráfico de ahorro con información numérica.
- Amount: recibe toda la información energética acumulada en el periodo de tiempo especificado, en este caso se procesan los datos para almacenar los del día. Actualiza la información de la energía y ahorros acumulados del día según los recibe la aplicación.
- BarChart: accede a los datos de amount y crea objetos para el gráfico de barras situado bajo el consumo total del día.
- Plan: entidad que hereda de System.Image, es la que guarda la imágen con la estructura general de la planta.
- Weather y WeatherTemp: procesan la información recibida mediante la API de OpenWeather

También existen 3 entidades que se encargan de dar posibilidad de personalización: la entidad Tema almacena los temas configurados en la app; la entidad Marquesina almacena el texto que se muestra en el mensaje deslizante de la parte inferior permitiendo su modificación; Imagen almacena la foto del edificio de Siemens, con posibilidad de cambiarla a otra diferente.

### 3.2.2 Lógica interna y procesamiento de datos

Toda la lógica de la conexión a las APIs y procesamiento se encuentra reunida en microflows (explicado en el capítulo 2.3.4). Éstos microflows se dividen en distintos paquetes los cuales se explican a continuación:

3.2.2.1 Paquete de APIs: contiene todos los microflows cuya funcionalidad es la de preparar y realizar la lógica de la conexión a las REST APIs que reúnen la información de los sensores del edificio inteligente que se quiere mostrar. Se encargan de recibir toda la información de los sensores, realizar la llamada con las cabeceras y credenciales necesarias y guardar la respuesta en objetos para su uso posterior en operaciones de interés. Estos microflows son:

- ACT_GetAggregateSensorEnergyDataByFloorInOneHour

  Este microflow (véase figura 3.9) prepara la conexión a la API GetAggregateSensorEnergyData, el primer paso para realizar la conexión es determinar en qué intervalo de tiempo queremos obtener los datos (intervalos de quince minutos porque esta API trabaja con éstos intervalos). Obtenemos la fecha y hora actuales para determinar el intervalo de tiempo del que se quiere recoger la información con esta API (debe ser un intervalo de 15 minutos). El intervalo que se determina es el que comienza media hora antes de la llamada, para evitar posibles retrasos en la subida de los datos. Por ejemplo, si la hora actual son las 12:15, este microflow crea el intervalo 11:45-12:00 del que se recopilan los datos de las APIs.

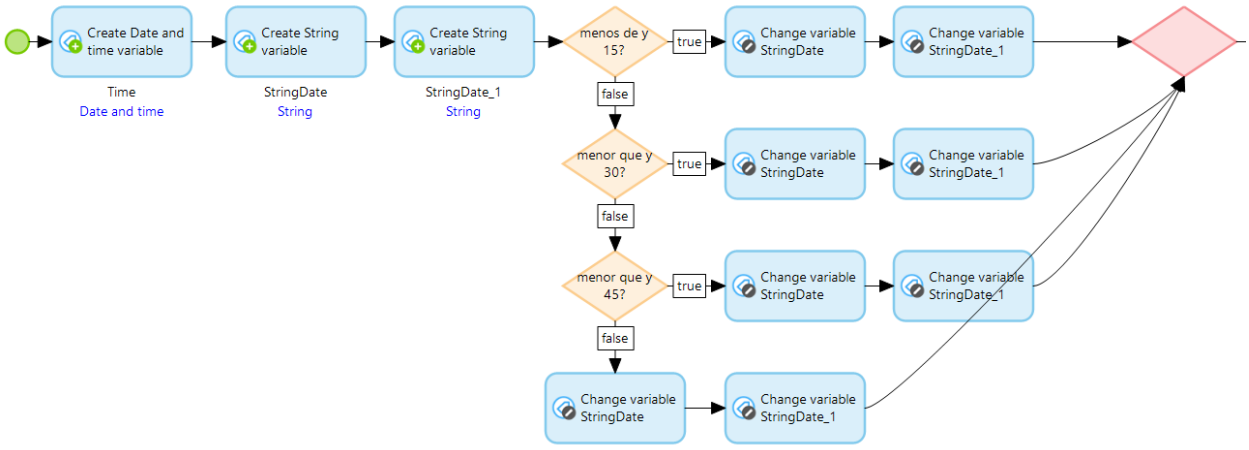



Figura 3.9: Microflow Aggregate Sensor Energy Data

A continuación, se obtiene el *time stamp* de la fecha actual (se realiza un Java action) y se codifica en SHA1 "nombre de usuario" + "API Key" + "Time stamp" (también mediante un Java action) para obtener todos los datos de la cabecera de la llamada. Por último, se llama a la API con todos los datos obtenidos, se utiliza el Import Mapping *Import_Aggregate Energy* y se llama al sub-microflow *Sub_EnergyToDecimal* para operar con los datos obtenidos (véase figura 3.10).

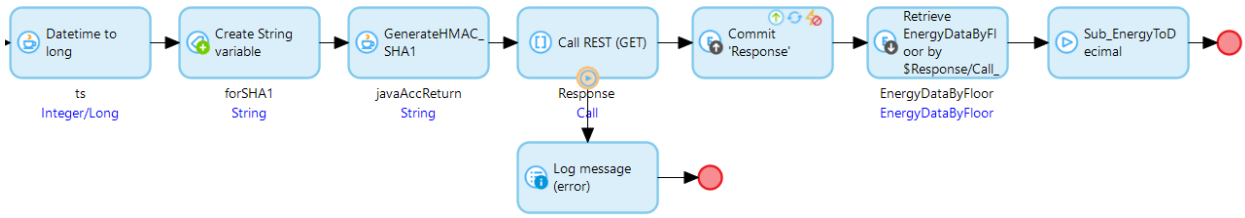



Figura 3.10: Microflow Aggregate Sensor Data 2

- ACT_GetDetailedFineGrainSensorData

  Este microflow (véase figura 3.11) realiza la conexión a la API *Get Detailed Fine Grain Sensor Data*, donde se calcula primero el intervalo de tiempo del que se obtienen los datos. Este cálculo se realiza de forma similar al microflow anterior, con la diferencia de que los datos están agrupados cada cinco minutos, por lo que se obtienen los datos para un intervalo de tiempo menor que el anterior.

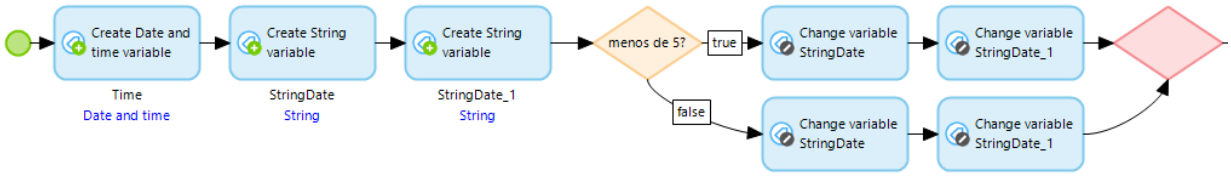



Figura 3.11: Microflow Detailed FIne Grain Data

Trás obtener el intervalo de cinco minutos se obtienen los datos necesarios para la llamada a la API anteriormente explicada, llamando a la API y aplicando el Import Mapping *Import_DetailedFineGrain*. Por último se llama al microflow DS_TempYOcup para recolectar, mostrar y operar la temperatura y ocupación obtenidas (véase figura 3.12).

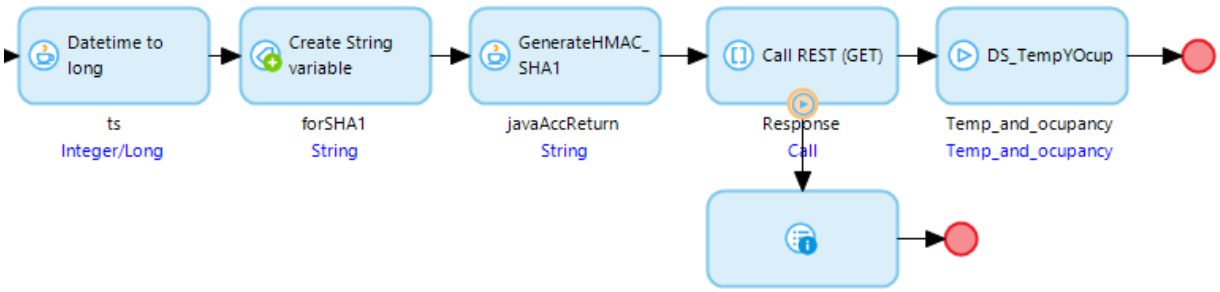


Figura 3.12: Microflow Detailed FIne Grain Data 2

- ACT_GetFloorPlanImage

  Este microflow (véase figura 3.13) realiza la conexión a la API Get Floor Plan Image, donde se necesita toda la información que necesitaban las anteriores llamadas a excepción del intervalo de tiempo, esta llamada devuelve una imagen que se guarda como un objeto *Plan*.

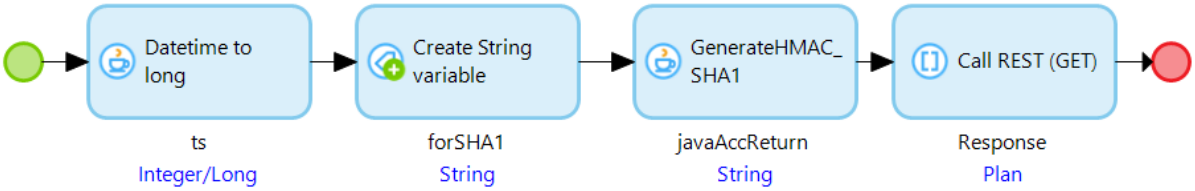



Figura 3.13: Microflow Plan Image

- ACT_GetSensorDetailsByFloor

  Este microflow (véase figura 3.14) realiza la conexión a la API Get Sensor Details By Floor, donde se calcula primero el intervalo de tiempo del que se obtienen los datos. Como esta API recopila información en intervalos de 5 minutos se asemeja mucho al microflow ACT_GetDetailedFineGrainSensorData. Sin embargo, al llamar a la API este microflow utiliza el Import Mapping *Import_SensorDetails*.

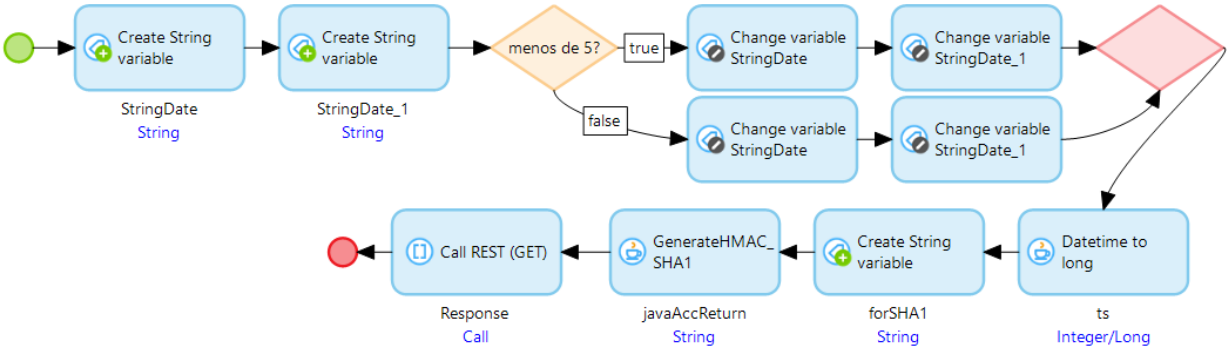



Figura 3.14: Microflow Sensor Details

- ACT_OpenWeatherApiCall

  Este microflow (véase figura 3.15) realiza la conexión a la API Open Weather, esta API no necesita ningún cálculo antes de la llamada, basta con especificar en la url de la llamada la latitud y longitud de las que se quiere obtener los datos meteorológicos. En la llamada se aplica el Import Mapping llamado *Import_Weather* y se actualiza la información en la base de datos.

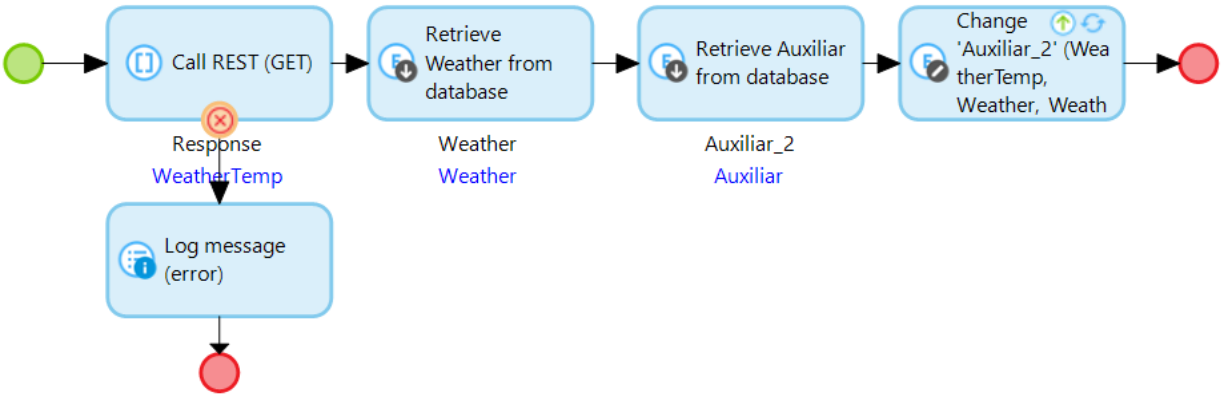



Figura 3.15: Microflow Open Weather API

3.2.2.2 Paquete de cálculos: contiene todos los microflows que realizan los cálculos necesarios sobre los datos que se reciben de las APIs. Estas operaciones pueden ser: obtener los porcentajes de cada tipo de ahorro, los datos agregados de un día, mes o año, los puntos de ocupación máximos del día, etc.

Estos microflows son:

- Sub_AggregateSavings

  Este microflow (véase figura 3.16) calcula el porcentaje de ahorro acumulado del día de cada tipo de ahorro. Busca si el gráfico de barras del día está creado y en el caso en el que no lo esté lo crea calculando los porcentajes de los primeros datos del día, en caso en el que el gráfico ya exista actualiza los valores para poner el nuevo porcentaje teniendo en cuenta todo el ahorro del día.

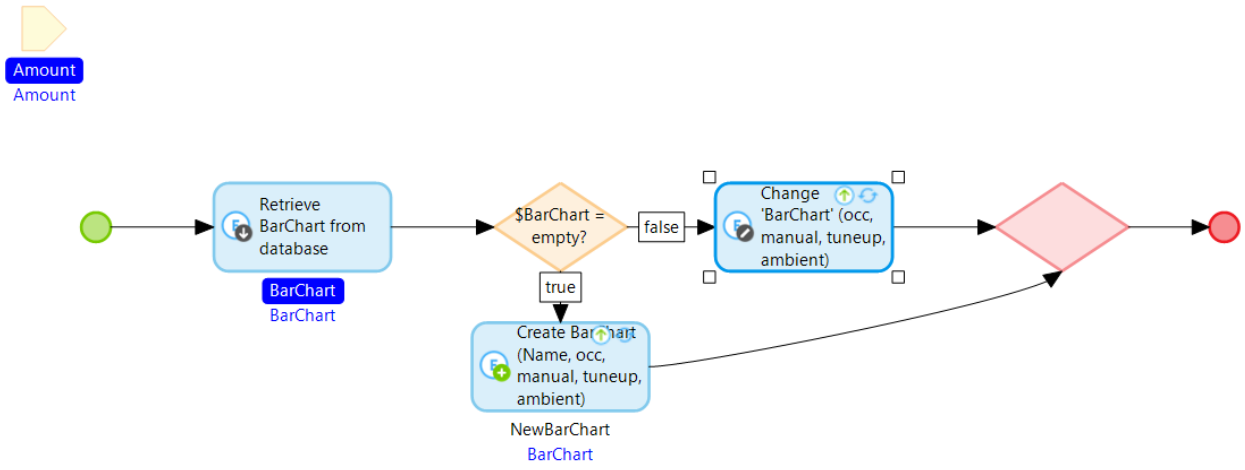



Figura 3.16: Microflow Aggregate Savings

- Sub_EnergyToDecimal

  Este microflow (véase figura 3.17) procesa los datos obtenidos de la llamada Get Aggregate Sensor Energy Data By Floor. Ya que los datos llegan en tipo String, para operar con ellos se obtiene el Integer, Decimal o Date and time dependiendo del dato y se crea el gráfico circular de los datos inmediatos de ahorro, a la par que una nueva entrada para actualizar el gráfico lineal del consumo diario. Al terminar, el microflow llama a *Sub_SumaTotal*.

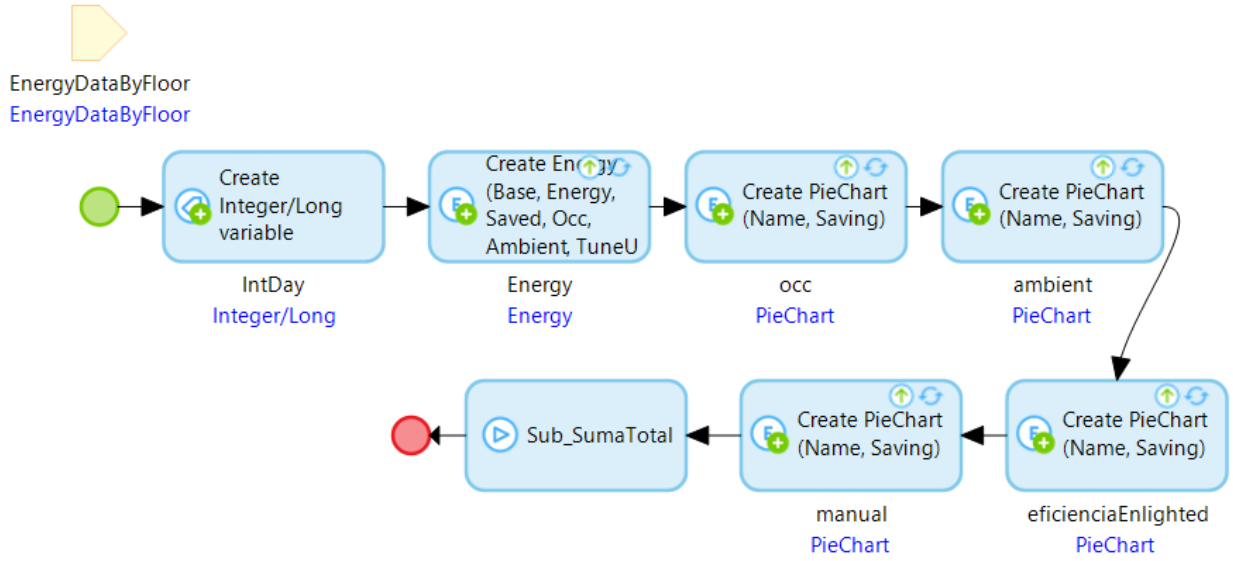



Figura 3.17: Microflow Energy to Decimal

- Sub_SumaTotal

  Este microflow (véase figura 3.18) actualiza todos los valores agregados del día con la nueva información que se ha recibido (si no había ningún Amount el día actual, se crea). Este microflow llama a *Sub_AggregateSavings* para terminar con todos los cálculos agregados.

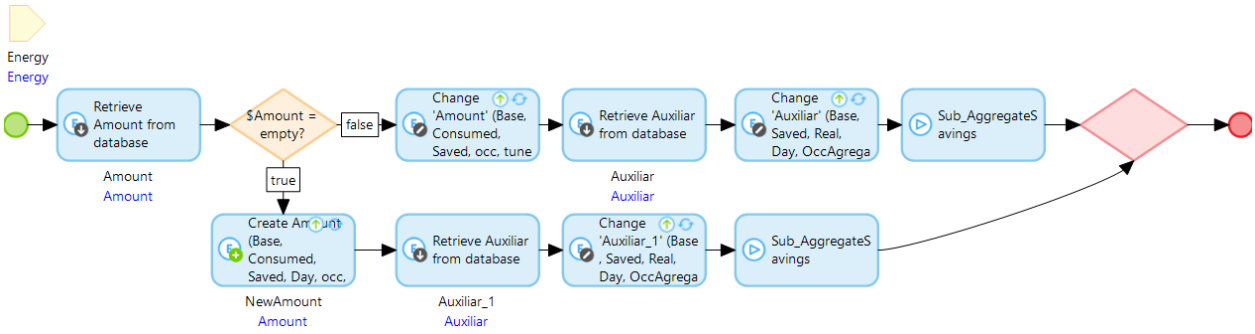



Figura 3.18: Microflow Suma Total

3.2.2.3 Paquete de personalización: contiene todos los microflows referentes a la personalización del diseño visual, tanto la marquesina, como el tema y la imagen del dashboard están almacenados en la base de datos como objetos para que el administrador del sistema sea capaz de realizar modificaciones a su gusto. Estos microflows son:

- ACT_CambiaMarquesina

  Este microflow (véase figura 3.19) actualiza la marquesina que aparece en la parte inferior de la pantalla con el nuevo texto que introduce el usuario.

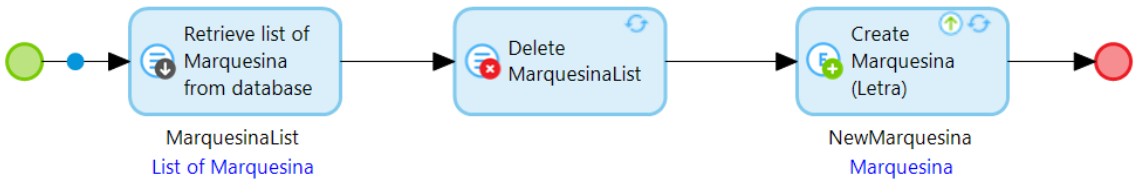



Figura 3.19: Microflow Cambia Marquesina

- DS_Imagen

  Este microflow (véase figura 3.20) devuelve la imágen que posteriormente aparece en pantalla, si el usuario modifica esa imagen este microflow actualiza su vista en el dashboard.

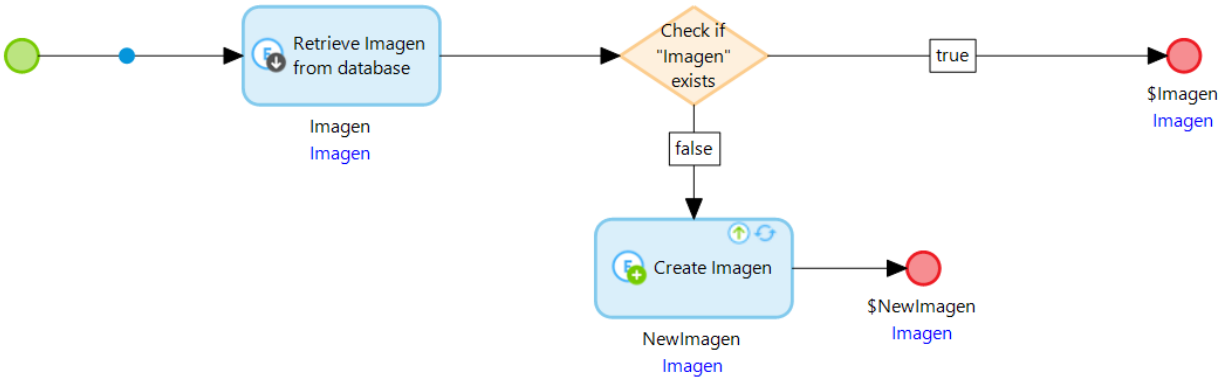



Figura 3.20: Microflow Imagen

- DS_Tema

  Este microflow (véase figura 3.21) devuelve el tema actual del dashboard. El usuario puede escogerlo en cualquier momento desde la página de configuración.

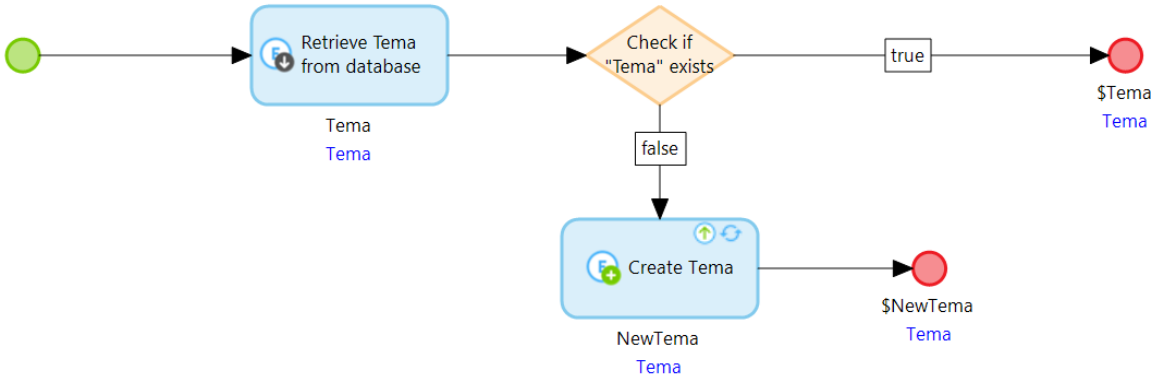



Figura 3.21: Microflow Tema

- ACT_AbrirDashboard

  Este microflow (véase figura 3.22) hace la selección de tema según la que haya escogido el usuario, en el caso en el que no exista el tema escogido se muestra el tema por defecto (tema azul).

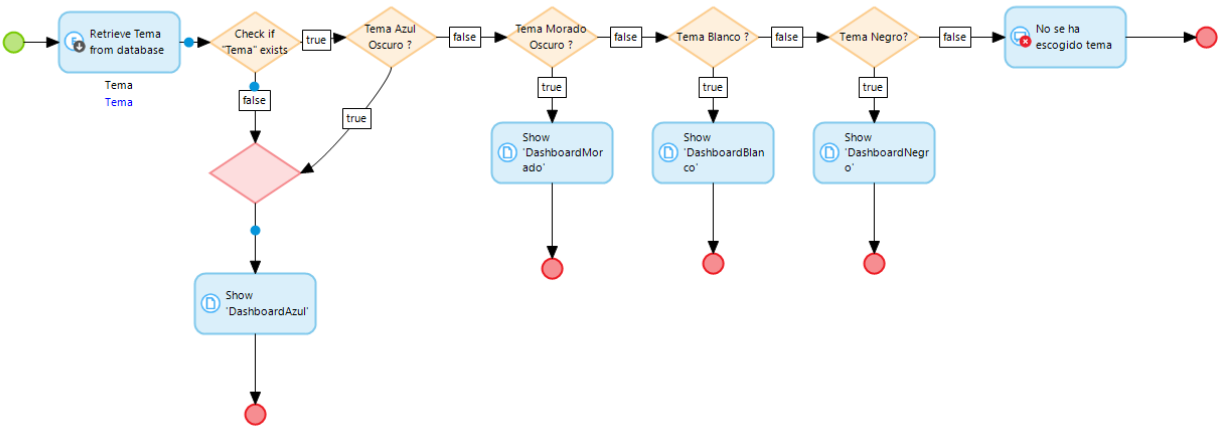



Figura 3.22: Microflow Abrir Dashboard

Aparte de los 3 paquetes de microflows ya nombrados, existe un grupo grande de microflows más sencillos encargados de la creación, eliminación y obtención de objetos.

Tras comentar la lógica principal de la aplicación (los microflows), cabe destacar otros dos aspectos importantes:

- Import Mappings: son elementos que se encargan de la traducción de los mensajes JSON o XML a objetos Mendix, de la forma (véase figura 3.23 y 3.24):

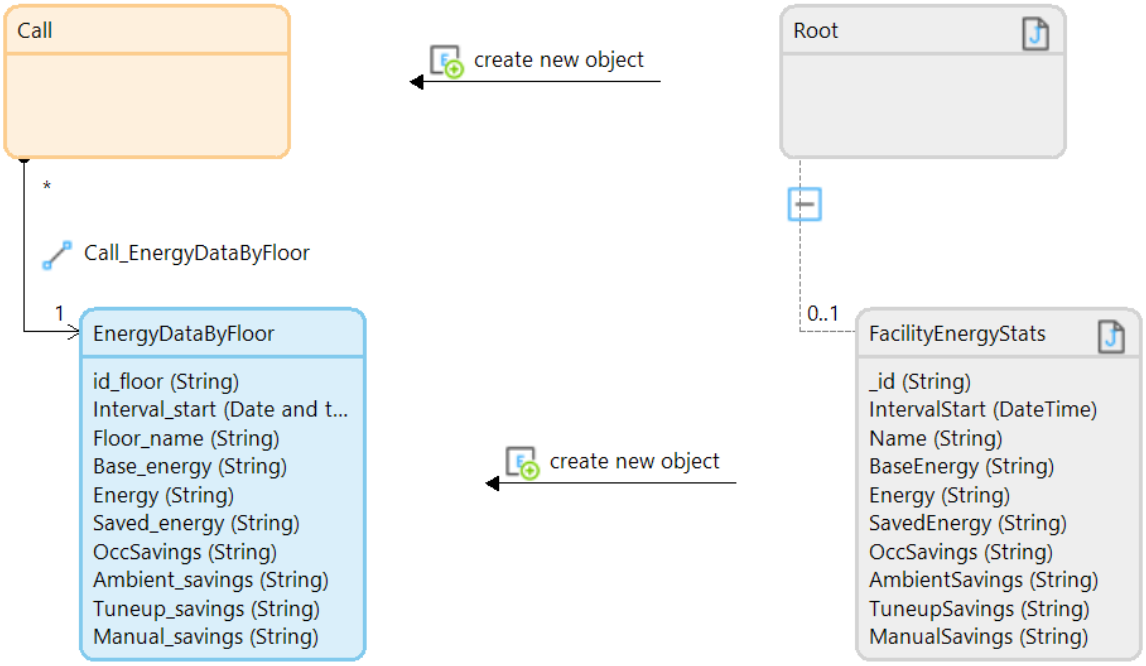



Figura 3.23: Import Mapping Energy Data

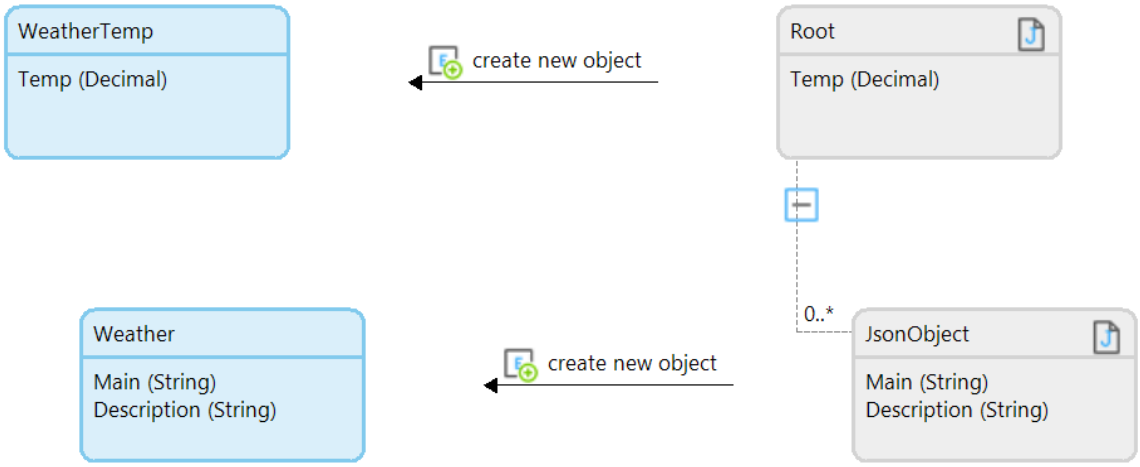



Figura 3.24: Import Mapping Weather

Para configurar estos Import Mapping son necesarios objetos donde se pueda pasar toda la información del mensaje, y asociar cada clave de éste a un atributo de dicho objeto, de tal forma que al hacer *Call REST Service Action* cree una nueva instancia siguiendo este patrón.

- Scheduled Events: permiten ejecutar de forma automática un microflow cuya ejecución ocurre cada cierto tiempo especificado por el desarrollador. En el ejemplo mostrado (véase figura 3.25) más abajo se ejecuta el microflow ACT_GetDetailedFineGrainSensorData cada 5 minutos (véase figura 3.26).

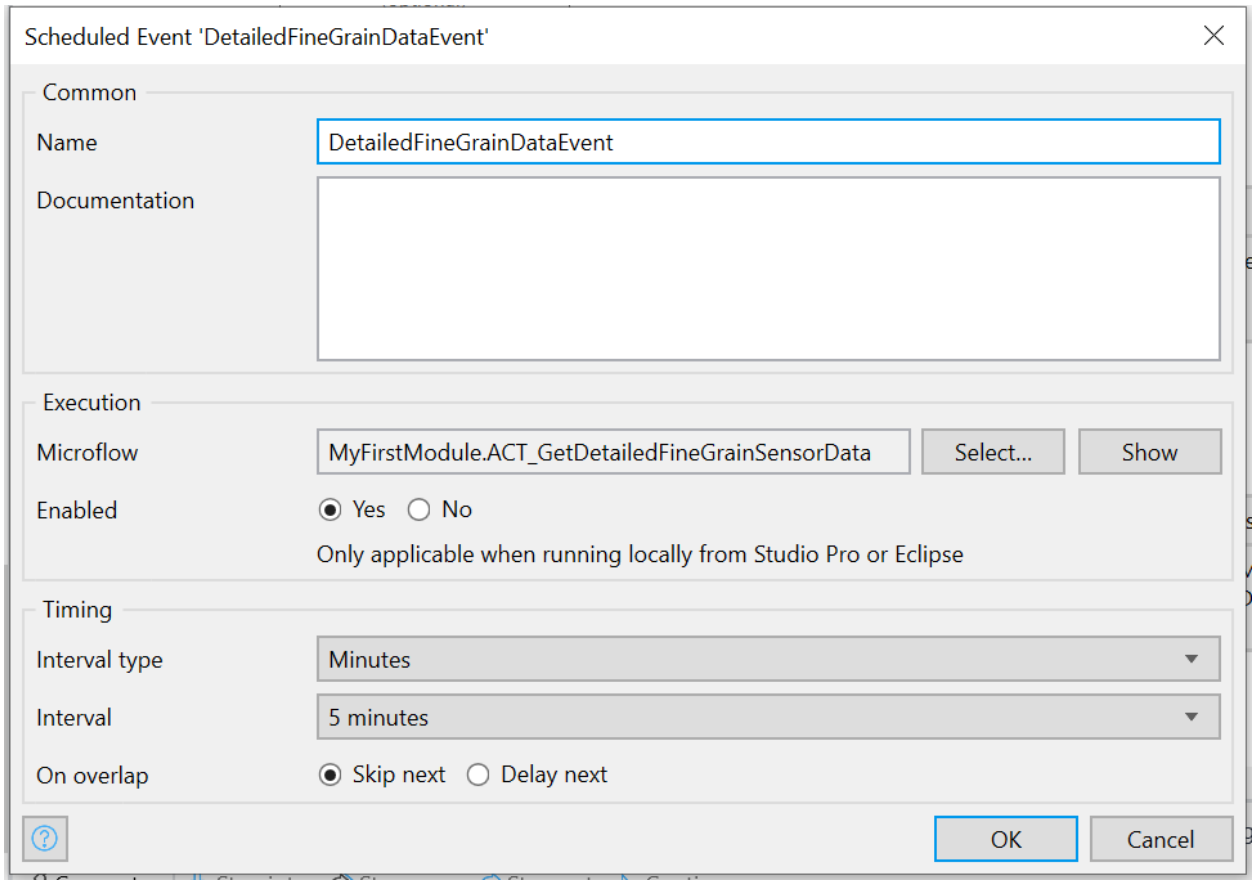



Figura 3.25: Scheduled Event Detailed Fine Grain Data

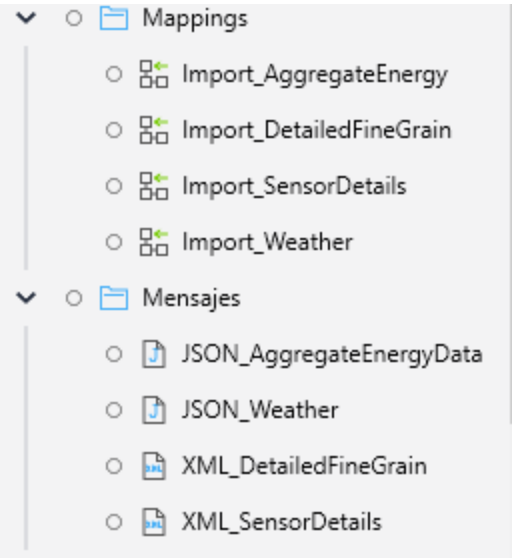



Figura 3.26: Import Mappings y estructuras de los mensajes

### 3.2.3 Diseño de la interfaz

La estructura de la página está modificada tanto con los widgets de Mendix Studio Pro como con programación css para resultados personalizados (véase figura 3.27).

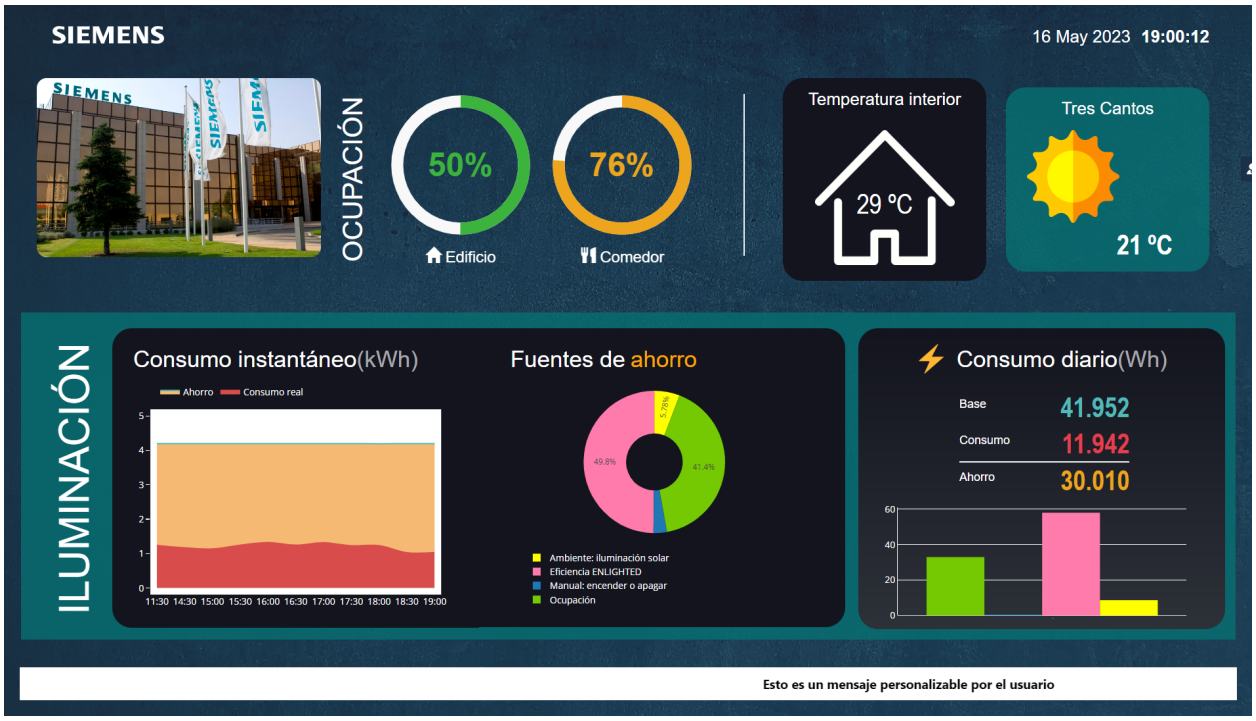



Figura 3.27: Dashboard Kiosko Digital Azul

- El primer elemento de la pantalla es una imagen que representa el edificio inteligente del que se están recopilando y mostrando los datos. Esta foto es completamente personalizable por el usuario, desde la pestaña de ajustes es capaz de subir cualquier imagen que tenga almacenada en su dispositivo lo que provoca una actualización de esta en el dashboard.

- Seguidamente tenemos los diagramas relativos a la ocupación en los que se procesa la ocupación del Edificio (obtenidos vía API) y del comedor de Siemens

Tres Cantos; y cuyos colores varían en ambos en base al porcentaje total; siendo verde si el porcentaje es inferior al 60%, naranja si se encuentra entre un 60% y un 80% y rojo si es superior al 80%.

- En la esquina superior derecha, se encuentra la información de temperatura, donde se hace una comparación de la temperatura exterior (dado por la Weather API en base a la ubicación del edificio) e interior (estableciendo una media de los datos obtenidos de los sensores).

  La API de temperatura exterior nos da también información sobre el tipo de clima, por lo que diseñamos mediante ajustes de visibilidad un conjunto de símbolos para representar cada uno de los estados del tiempo.

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| few clouds | thunderstorm | snow | rainy | clouds | clear |

En la parte inferior se muestran los gráficos relativos a la información del consumo de los sensores en cuanto a iluminación. Donde se puede consultar (de izquierda a derecha) de información más general a más particular. Todos los elementos han sido estilizados y personalizados utilizando SCSS y JavaScript con el fin de darle un toque más profesional al dashboard.

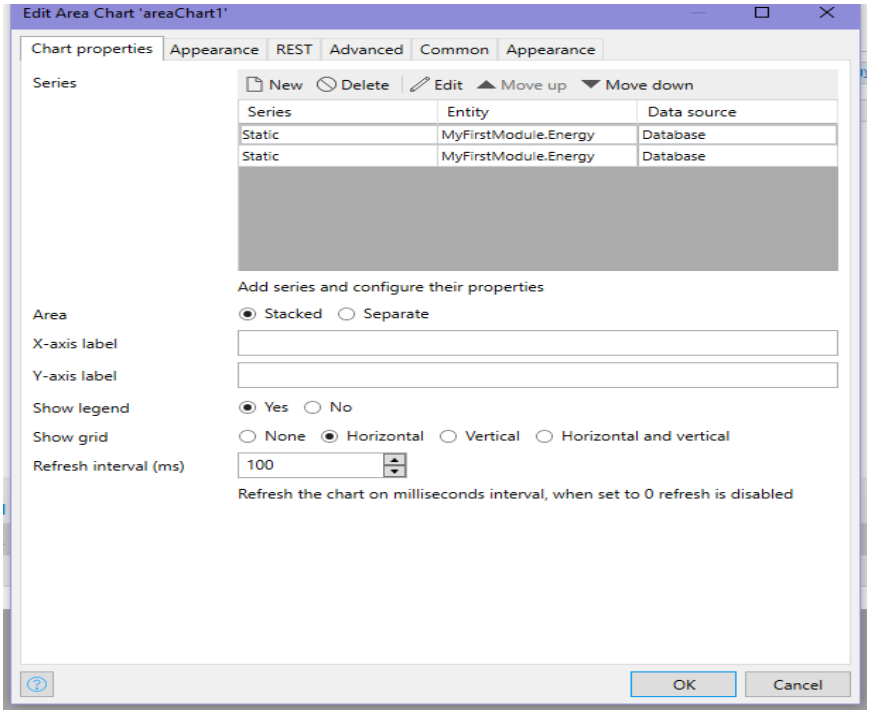
- Comenzando por la izquierda, tenemos un gráfico donde se muestran los datos de consumo y ahorro energético generales a lo largo del día. Este gráfico se

actualiza conforme se van recogiendo los datos del día (cada 15 minutos) y a medida que se van alcanzando intervalos de media hora aparecen los límites del intervalo como valores del eje de abscisas. Hemos configurado una pequeña leyenda explicativa donde el usuario puede interpretar que la recta azul paralela al eje de abscisas comprende el valor del consumo energético sin ajustes de eficiencia (es decir en el caso peor en que los sensores utilicen toda la potencia posible), por ello se trata de una línea recta sin variaciones durante el dia. Por otro lado, la zona roja representa los datos reales de consumo de los sensores y adicionalmente, la zona naranja representa implícitamente el ahorro energético de dichos sensores. Este código de colores será consistente durante esta zona inferior del dashboard para facilitar la interpretación de los valores mostrados.

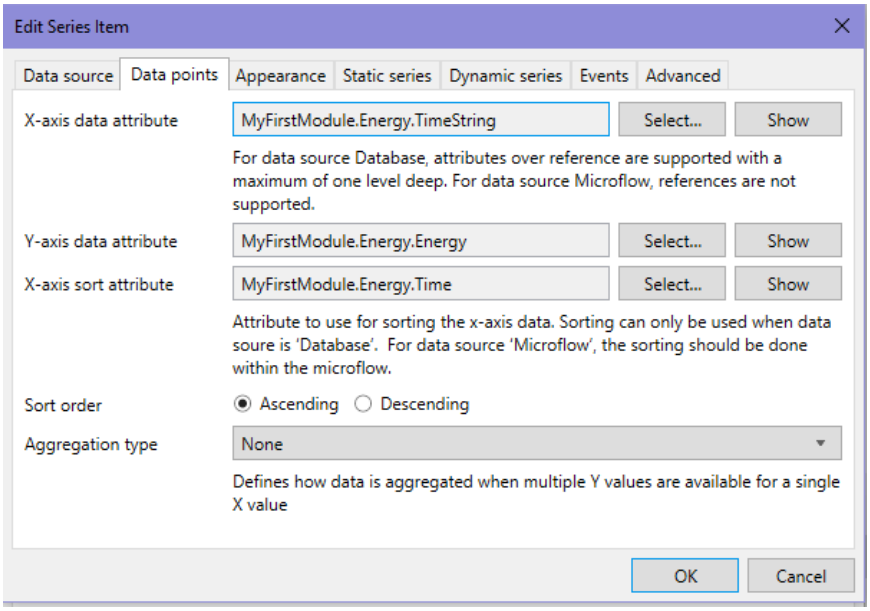
Para conseguir estos datos, se realizan consultas periódicas en la base de datos mediante el microflow *DS_Energy*, que coge el último objeto de tipo Energy (pues se trata del último valor conseguido de la API). Una vez conseguidos los datos, estos se pasan al gráfico (véase figuras 3.28, 3.29, 3.30) creando una serie por cada tipo de atributo mostrado (en este gráfico tenemos la serie de la energía real consumida y del consumo base).

```
{
"grid": {
"yside": "left"
},
"hovermode": false,
"yaxis":{
"range": [0, 5],
"color": "white",
"ticklen": 6
},
"xaxis":{
"color": "white"
},
"font": {
"color": "white"
},
"legend":{
"font":{
        "color": "white"
},
"orientation": "h",
    "y": "1.2"
},
"paper_bgcolor": "transparent",
"plot_bgcolor": "white"
}
```


3.28: JSON de estilo de gráfico




3.29: Propiedades del gráfico de Area

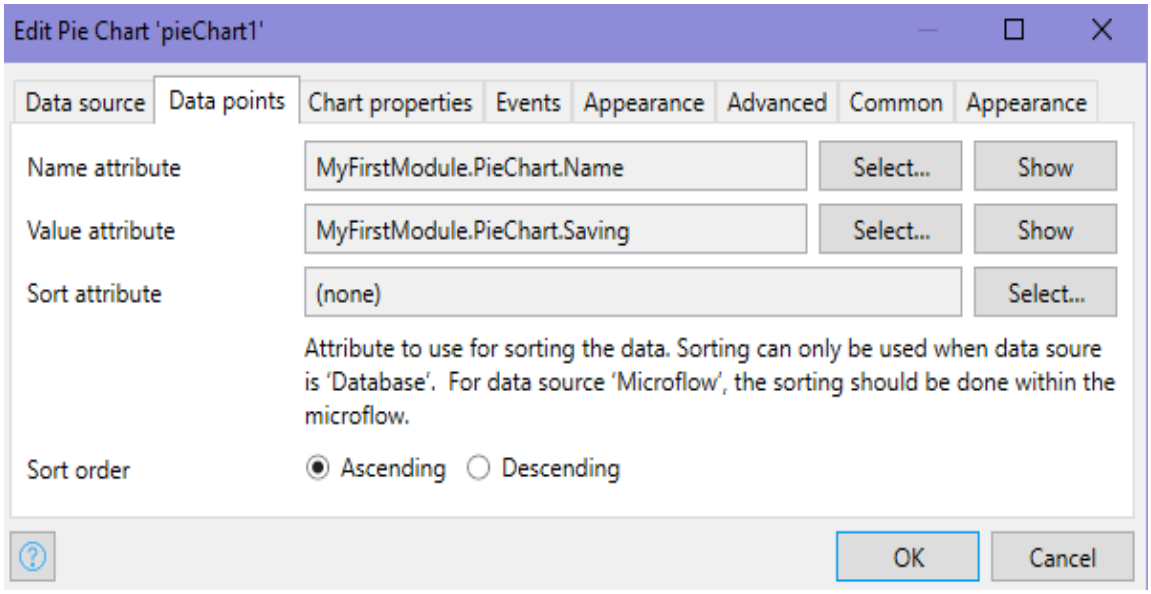



3.30: Datos del gráfico de Area

- Seguidamente, el diagrama en forma de "donut" que se encuentra a la derecha del gráfico anterior contiene información exclusiva sobre el ahorro energético a tiempo real, es decir, no son datos agregados, sino que varían constantemente a lo largo del día. Con el fin de facilitar la cohesión de todos los diagramas, se ha optado por destacar la palabra "ahorro" con el mismo color naranja que en el primer gráfico. La naturaleza de estos datos viene de la misma entidad que el anterior gráfico, con la particularidad de que este se centra solo en los atributos de ahorro recibidos por la API, además de que para configurar un gráfico de sectores (*pie chart*) es necesario tener una entidad específica y de ella crear un objeto para cada tipo de ahorro. Esta entidad es la que en el *Domain Model* nombramos como *PieChart*, cuyos objetos tienen un atributo "name" para agruparlos en el gráfico, y otro atributo "saving" donde se almacena el valor de ahorro (véase figuras 3.31, 3.32).

```
{
"hovermode": false,
"font": {
"color": "transparent"
},
"yaxis": {
"title": "%",
"color": "white"
},
"legend":{
"font":{
        "color": "white"
},
"orientation": "h",
    "y": "auto"
},
"paper_bgcolor": "transparent"
}
```

3.31: Estilo del gráfico de queso




3.32: Puntos del gráfico de queso

- A modo de conclusión, el dashboard cuenta también con una visualización de los datos agregados tanto de consumo como de ahorro, donde el consumo base y real se encuentra en la zona superior de este último cuadro (manteniendo el código de colores del primer gráfico) y el ahorro se muestra desglosado en un gráfico de barras en la zona inferior (que al igual que el diagrama por sectores anterior, cuenta con una entidad *BarChart* dedicada en la base de datos donde se crea un objeto por cada uno de los tipos de ahorro agregados de los sensores).

  Tanto los datos de consumo como el diagrama de ahorro se reinician cada día y a medida que transcurren las horas van aumentando con los nuevos datos a tiempo real que se reciben de la API.

Finalmente, el último elemento del dashboard es una marquesina dinámica completamente personalizable por el usuario donde es posible elegir el mensaje que se mostrará desde la pestaña de configuración.

### 3.2.4 Personalización y administración

La información del dashboard y las opciones de personalización se encuentran en dos vistas adicionales que cuentan con la siguiente estructura:

- La **página de personalización** (véase figura 3.33) cuenta con un diseño sencillo en el que se puede subir la foto que el usuario desee mostrar en el dashboard (pulsando el botón de "Browse…") , cambiar mediante un desplegable el tema (véase figuras 3.27, 3.34, 3.35 y 3.36) deseado (entre los que se encuentra el azul, morado, negro y blanco) y escribir el mensaje de la marquesina.

  Tanto la foto como el resto de aspectos de la página se pueden cambiar en cualquier momento sin afectar al funcionamiento del dashboard ni a la recogida de datos ni actualización (el usuario simplemente deberá volver a la página principal y verá los cambios aplicados además de la información recogida en su ausencia).


Figura 3.33: Área de Personalización

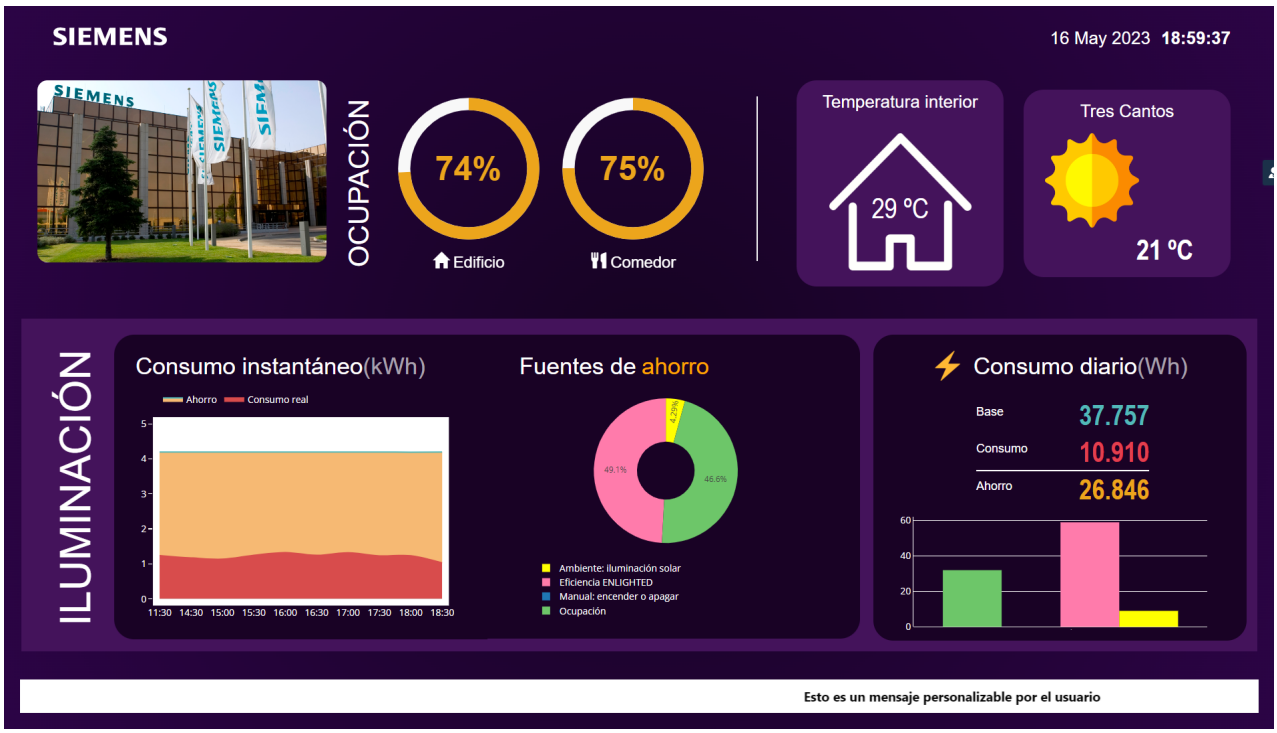



Figura 3.34: Dashboard Kiosko Digital Morado

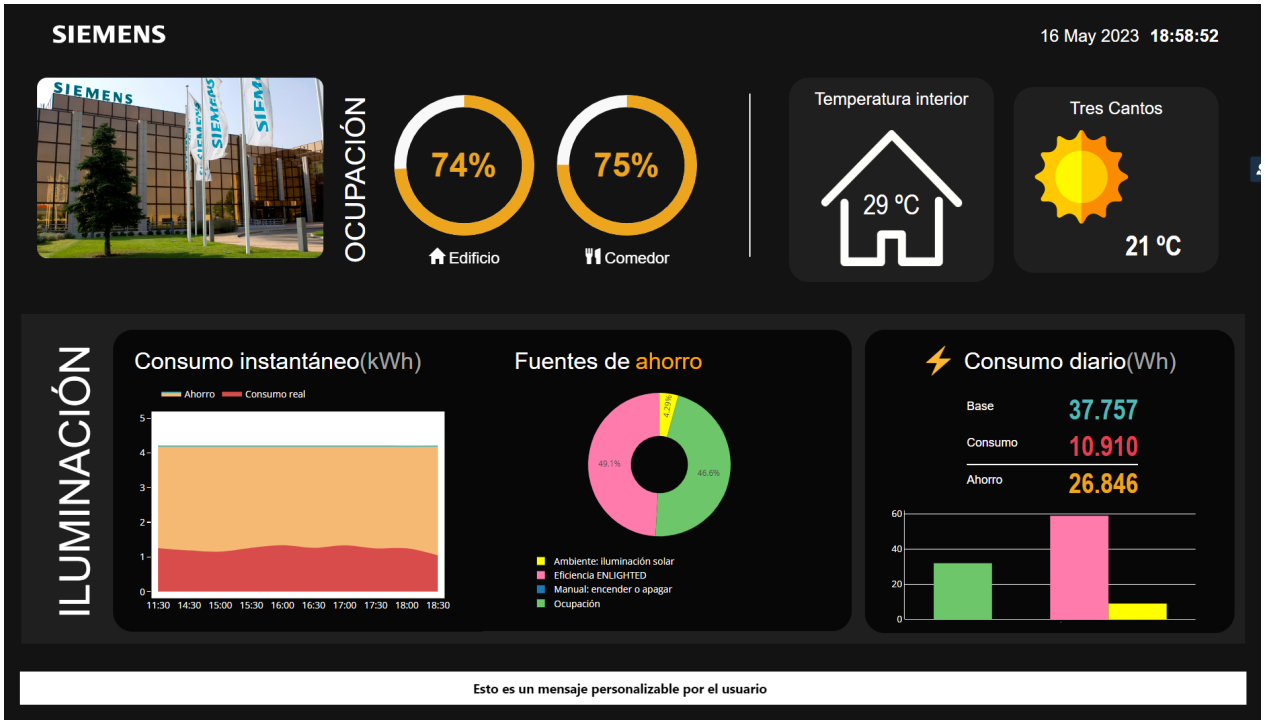



Figura 3.35: Dashboard Kiosko Digital Negro

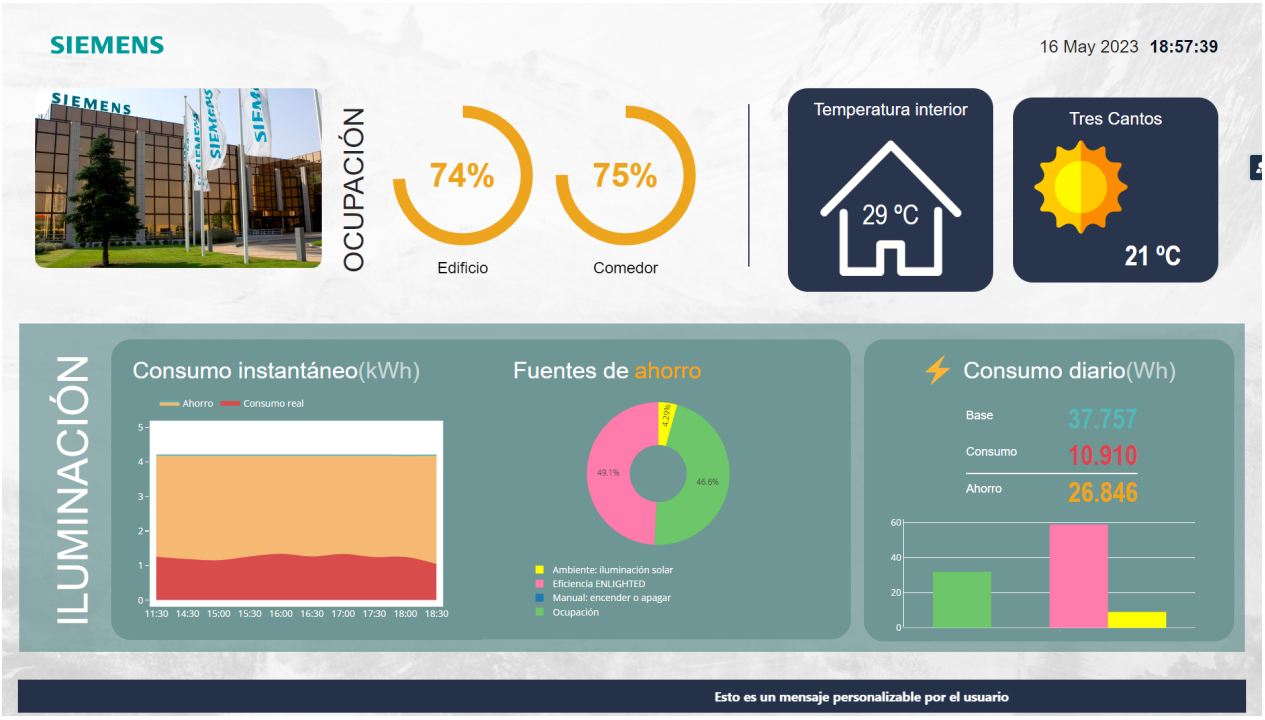
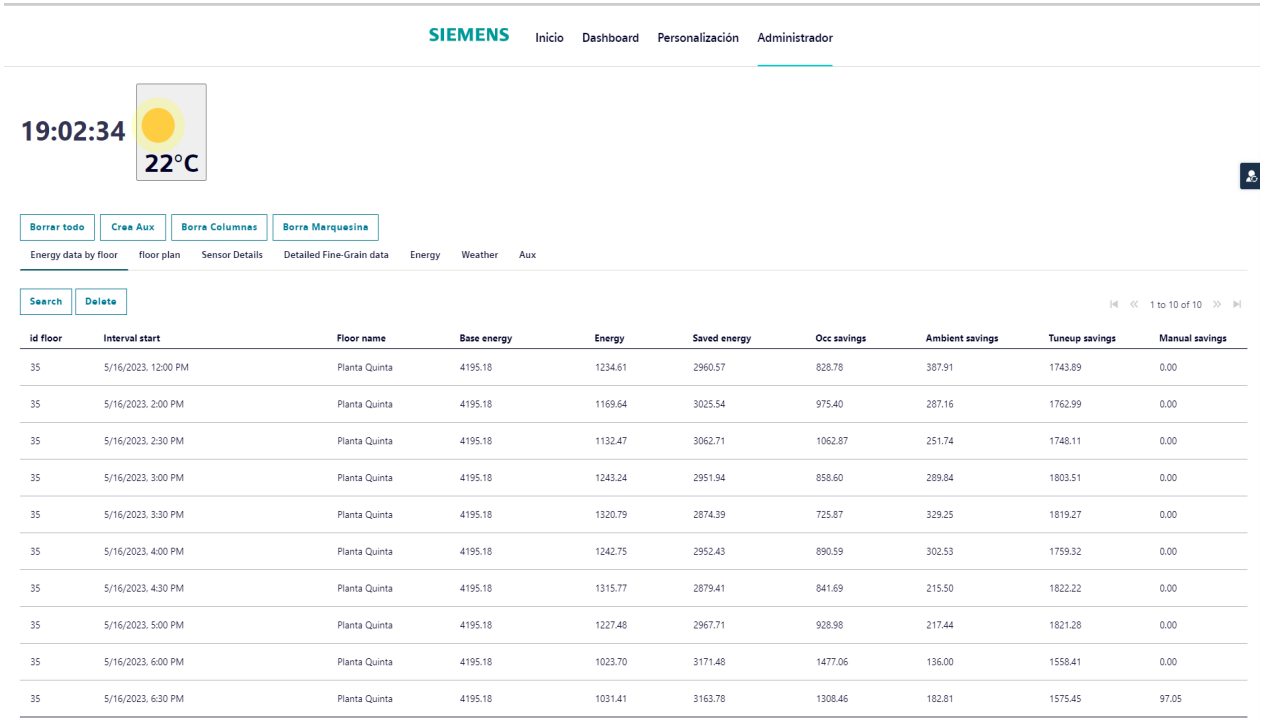



Figura 3.36: Dashboard Kiosko Digital Blancos

- Finalmente, la **página de administración** de datos cuenta con varias pestañas donde se muestra una tabla de la información recogida a tiempo real con el fin de permitir a los administradores de la aplicación explotar estos datos para realizar seguimientos y análisis para realizar predicciones o comprobaciones empíricas de la efectividad de la configuración de sus sensores. Todas las listas cuentan con filtros específicos para facilitar la labor de búsqueda de los administradores (véase figuras 3.37, 3.38 y 3.39).




| id floor | Interval start | Floor name | Base energy | Energy | Saved energy | Occ savings | Ambient savings | Tuneup savings | Manual savings |
|---|---|---|---|---|---|---|---|---|---|
| 35 | 5/16/2023, 12:00 PM | Planta Quinta | 4195.18 | 1234.61 | 2960.57 | 828.78 | 387.91 | 1743.89 | 0.00 |
| 35 | 5/16/2023, 2:00 PM | Planta Quinta | 4195.18 | 1169.64 | 3025.54 | 975.40 | 287.16 | 1762.99 | 0.00 |
| 35 | 5/16/2023, 2:30 PM | Planta Quinta | 4195.18 | 1132.47 | 3062.71 | 1062.87 | 251.74 | 1748.11 | 0.00 |
| 35 | 5/16/2023, 3:00 PM | Planta Quinta | 4195.18 | 1243.24 | 2951.94 | 858.60 | 289.84 | 1803.51 | 0.00 |
| 35 | 5/16/2023, 3:30 PM | Planta Quinta | 4195.18 | 1320.79 | 2874.39 | 725.87 | 329.25 | 1819.27 | 0.00 |
| 35 | 5/16/2023, 4:00 PM | Planta Quinta | 4195.18 | 1242.75 | 2952.43 | 890.59 | 302.53 | 1759.32 | 0.00 |
| 35 | 5/16/2023, 4:30 PM | Planta Quinta | 4195.18 | 1315.77 | 2879.41 | 841.69 | 215.50 | 1822.22 | 0.00 |
| 35 | 5/16/2023, 5:00 PM | Planta Quinta | 4195.18 | 1227.48 | 2967.71 | 928.98 | 217.44 | 1821.28 | 0.00 |
| 35 | 5/16/2023, 6:00 PM | Planta Quinta | 4195.18 | 1023.70 | 3171.48 | 1477.06 | 136.00 | 1558.41 | 0.00 |
| 35 | 5/16/2023, 6:30 PM | Planta Quinta | 4195.18 | 1031.41 | 3163.78 | 1308.46 | 182.81 | 1575.45 | 97.05 |

Figura 3.37: Tabla 1 de Datos

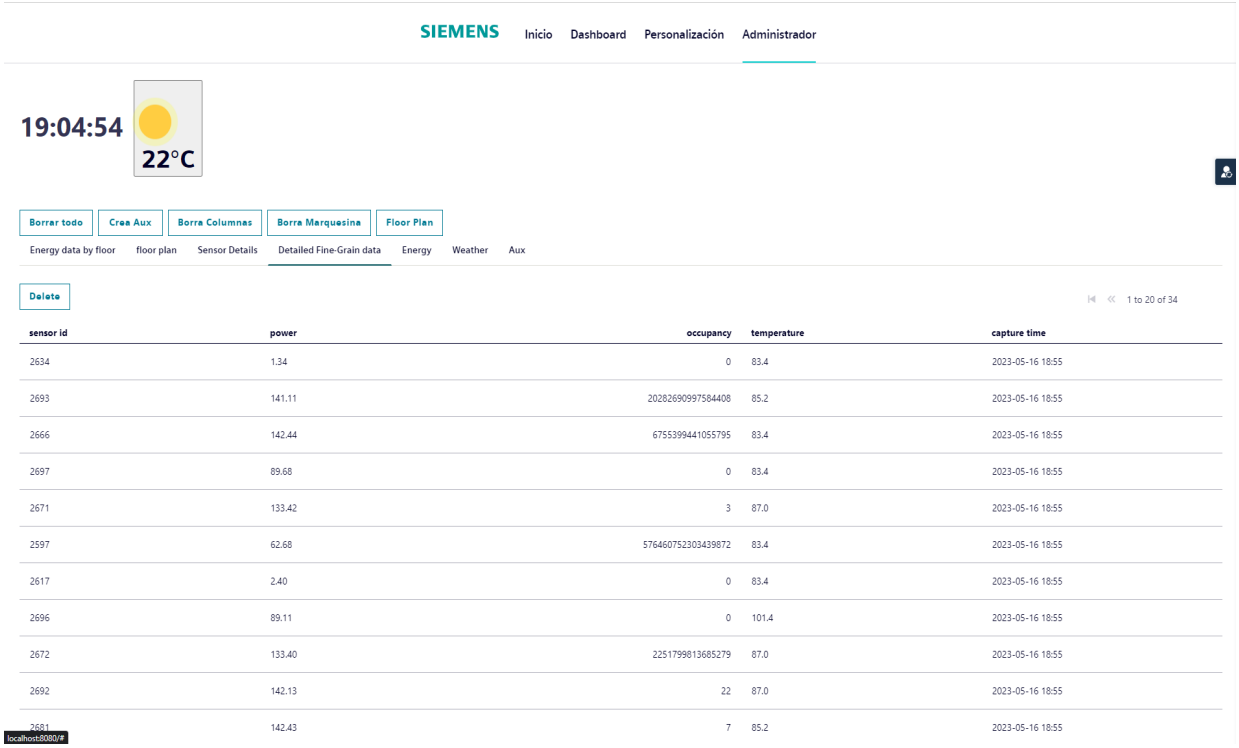



Figura 3.38: Tabla 2 de Datos

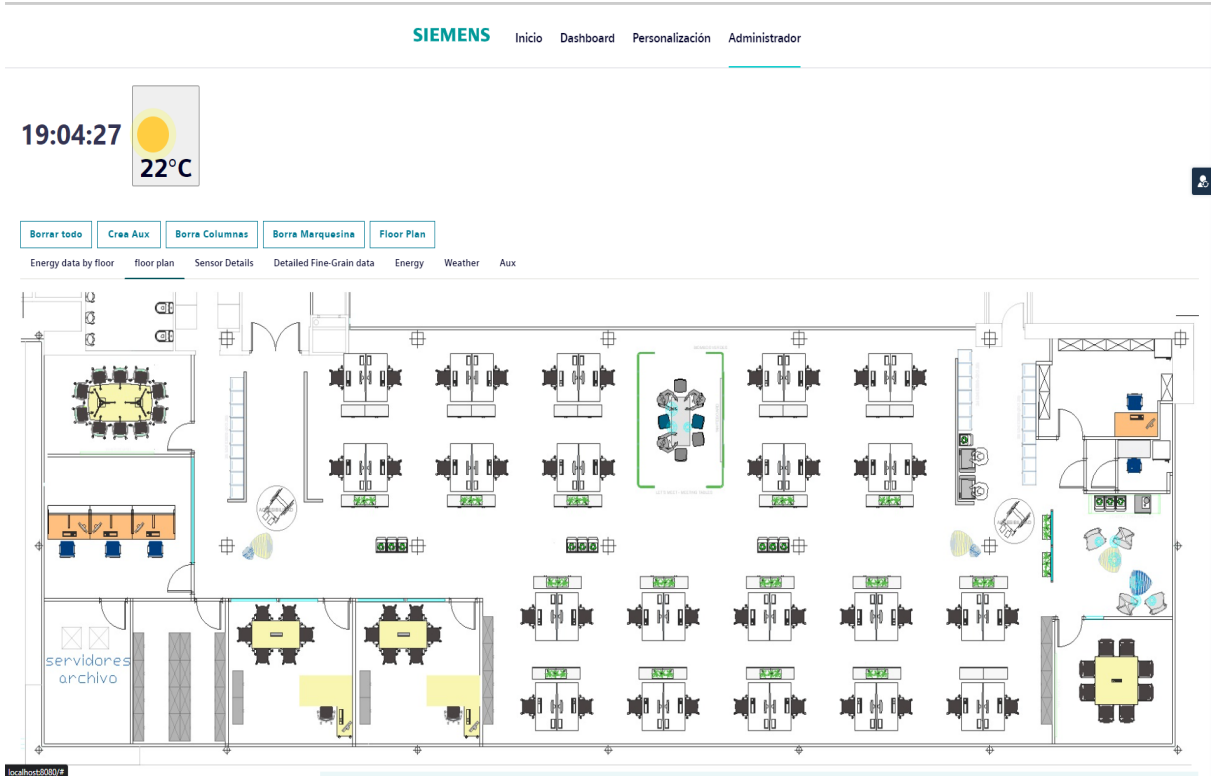



Figura 3.39: Imagen de la disposición de la planta

# Capítulo 4 - Otros casos de uso

Como se ha estado explicando a lo largo de la memoria, Kiosko Digital está preparado para adaptarse a múltiples casos de uso.

## 4.1 Procedimiento para la adaptación a nuevos casos de uso

Para poder modificar el caso de uso actual que se está utilizando en el dashboard se han desarrollado los microflows de forma general permitiendo procesar la información de cualquier fuente. El despliegue de un nuevo caso de uso se realiza con una sencilla modificación en los datos de conexión a las nuevas REST APIs, si estas requieren unas cabeceras diferentes, nuevas credenciales u otras propiedades solo sería necesario añadirlas en el microflow y cambiar la url de la llamada. Este proceso de cambio de caso de uso (cambio de edificio inteligente o de información del mismo o diferente edificio) se puede realizar en cuestión de minutos gracias a la herramienta de desarrollo elegida (Mendix, Low Code) y la previsión por nuestra parte de posibles cambios.

Adicionalmente, es posible que este cambio requiera un mayor almacenamiento de información al aumentar la cantidad de líneas (clave y valor) que se reciben con los mensajes de datos de los sensores. Para este caso bastaría con añadir un nuevo atributo al objeto en el que se almacena la información y asociarlo a la respectiva

línea. Los microflows mencionados actualizan la información guardada en la base de datos cuando reciben nuevas entradas, tras esto operan con ellos obteniendo los datos agregados del día y las tendencias más remarcables de máximos o mínimos del día.

Al realizar la modificación de caso de uso, en cuanto se preparan los microflows y la base de datos, la parte visual de la aplicación también se actualiza con la nueva información, es decir, los gráficos irán generando y mostrando la nueva información de forma automática y sin modificar nada. El único cambio que se debería hacer sería poner el título correcto a los gráficos, ya que los casos de uso pueden tener gráficos de nivel de ruido, ocupación del parking…

Como se puede observar, el uso de tecnología Low Code es conveniente para flexibilizar y agilizar el desarrollo de forma que, con unos pocos minutos para cambiar el caso de uso, sea capaz de adaptarse de forma eficaz.

Los datos que recolecta, procesa y muestra son los del edificio inteligente de Siemens en Tres Cantos para mostrar el marco general desarrollado con un ejemplo, no obstante, han sido valoradas muchas otras opciones que se van a comentar a continuación y para los que se ha hecho un diseño detallado de lo que sería el resultado utilizando la herramienta InkScape (véase bibliografía [9]). In

## 4.2 Posibles casos de uso

### 4.2.1 Nivel de ruido en museos

Se quiere recolectar y mostrar información referente al nivel de ruido en museos, hospitales u otros edificios para ayudar a personas con sensibilidad acústica o con trastornos del espectro autista que puedan verse afectados por el ruido. Se podría incluir esa información en sustitución a la energía e informar a los usuarios de las horas del día que acumulan más nivel de ruido.

Para el caso de un museo, se ha elaborado un dashboard de ejemplo que cuenta con la siguiente estructura (véase figura 4.1):


Figura 4.1: Dashboard de nivel de ruido

Este dashboard conservaría la estructura original, pudiendo cambiar la fotografía por la del museo y contando esta vez con unos diagramas de la ocupación por plantas en la zona superior de la pantalla. Para la zona inferior, una posible solución sería mostrar un gráfico global de líneas que vaya actualizando sus valores a tiempo real a lo largo de las horas del día, actualizando el mensaje superior con la hora con el mayor índice de ruido registrado de ese día, de esta forma el usuario con un solo vistazo puede tener una visión general y observar la tendencia que toma la línea. Seguidamente, y a la derecha del gráfico general, se mostrarían tanto los decibelios actuales como la franja horaria del día anterior en el que hubo un mayor nivel de ruido para así permitir al usuario tomar decisiones sobre cuándo visitar el museo.

### 4.2.2 Ocupación de instalaciones en facultades universitarias y hoteles

También se puede realizar la conexión a hoteles u hostales con información de la ocupación de zonas comunes (piscina, comedor, recibidor…) permitiendo a los clientes tomar la decisión de acceder a esa zona o no. De forma paralela esa información también ayudaría en bibliotecas (como la Biblioteca de la Facultad de Informática) mostrando la ocupación de ésta y de las zonas de trabajo en grupo en una pantalla en la entrada (véase figura 4.2).


Figura 4.2: Dashboard de nivel de ocupación en la Facultad

Para este ejemplo se ha optado por poner el logo de la Complutense en la esquina superior izquierda seguido de una foto de la facultad de Informática. Los gráficos de ocupación podrían mostrar el porcentaje de ocupación de la cafetería y biblioteca a tiempo real, y en la zona inferior se entraría en detalle en la biblioteca mostrando un diagrama de barras del índice de ocupación a lo largo del día, así como la disponibilidad actual de las distintas salas de trabajo en grupo y la hora de finalización del estado en el que se encuentran (libres u ocupadas). Finalmente, para facilitar la consulta de información y reserva de salas se podría mostrar un código QR que dirija a la página web de la Biblioteca donde se pueden consultar horarios, salas y eventos (el que se encuentra en el ejemplo es completamente funcional, conseguido a través de qr.io, bibliografía [10]).

Se estuvo valorando la posibilidad de acceder a la información de los sensores de la Facultad de Informática (UCM) obteniendo los datos que se precisen como por ejemplo: energéticos, de ocupación (de la cafetería u otras zonas), temperatura, CO2 (aunque tras el COVID-19 tiene menos valor), etc para los que también se podría desarrollar una solución parecida.

### 4.2.3 Disponibilidad de plazas de parking y cargadores eléctricos

Por último, una gran cantidad de edificios cuentan con aparcamientos que tienen automatizado el cálculo de la ocupación de los parkings y las plazas con cargador eléctrico. Si se obtuviera la conexión a esa información los usuarios podrían ver las zonas horarias donde aumenta considerablemente la ocupación, la disponibilidad actual tanto de plazas convencionales como de las plazas con cargador eléctrico.

De esta forma el usuario podrá conocer con anterioridad la información necesaria para planificar sus trayectos de la forma más eficiente posible, teniendo en cuenta estos horarios y reduciendo costes en gasolina y tiempo.

# Capítulo 5 - Conclusiones y trabajo futuro

El objetivo principal de este proyecto era desarrollar una plataforma en forma de dashboard de carácter general para la muestra y procesamiento de datos que recogen los sensores de edificios inteligentes. Este objetivo se ha conseguido completar gracias al desarrollo orientado a la posibilidad de adaptarse a información de diferentes índoles y a la simplicidad que conlleva debido al desarrollo Low Code.

El siguiente objetivo consistía en permitir a los usuarios personalizar los aspectos visuales de la aplicación. Este objetivo también se ha completado con una pestaña de personalización simple en la que el usuario puede cambiar la imagen que aparece en la esquina superior izquierda, el texto deslizante que figura en la parte inferior y el tema de la aplicación a 4 temas básicos (blanco, negro, azul y morado).

Con el desarrollo de este dashboard se ha conseguido una herramienta de marco general y personalizable que destaca entre las demás herramientas similares por la conexión a edificios inteligentes con cualquier información que estos recojan, lo que nos deja como trabajo futuro los siguientes puntos:

- **Desarrollar una interfaz de este dashboard para su visualización en dispositivos móviles o tablets:** esta aplicación está desarrollada con tecnología web y preparada para subirse a la nube, por lo que cualquier tipo de dispositivo capaz de conectarse a Internet podría acceder a esta página. Actualmente, el dashboard está diseñado para mostrarse en pantallas grandes en las entradas y

zonas comunes de los edificios, sin embargo, con una interfaz diseñada de forma agradable para la vista en dispositivos móviles, permitiría un aumento considerable de accesibilidad para cualquier usuario que quisiera consultar la información que muestra incluso fuera del edificio.

- **Aumentar la capacidad de personalización:** se podría implementar funcionalidades adicionales para que los usuarios de la aplicación pudieran colocar bloques de gráficos con la disposición que ellos quisieran, pudiendo eliminar aquellos que no quieran, aumentar o disminuir su tamaño, cambiar el tipo de gráfico para la muestra de datos concretos (p. ej. el gráfico de ahorro cambiarlo a linear), modificar los colores de esos gráficos y bloques, y más personalización que pudieran necesitar y solicitar los usuarios interesados.

- **Realizar la conexión a más edificios inteligentes:** la aplicación es capaz de conectarse a múltiples redes de sensores ubicadas en edificios inteligentes, un valor añadido para demostrar la utilidad de este marco general sería conectarse a más edificios.

- **Implementar una Inteligencia Artificial para realizar predicciones:** añadir una Inteligencia Artificial a la aplicación para que pudiera recolectar y procesar los datos con Machine Learning. Con ella se podría obtener un análisis más preciso de los resultados y aumentar la probabilidad de éxito e índice de acierto de las predicciones (p. ej. predecir cuándo habrá más ocupación).

# Introduction

## Motivation

Companies and organizations that have large buildings are increasingly concerned about energy savings and their sustainability. Today, many buildings have smart sensors that allow constant measurement of energy consumption in order to verify that the objectives are met as well as to study ways to improve energy efficiency.

Taking this idea as a starting point, this digital dashboard came to light, where a visual summary of the information collected by the building sensors such as consumption, occupancy or temperature is displayed in a clear and intuitive way in the busiest areas of the buildings.

In addition, the most remarkable aspect of this dashboard is that it has an implementation that makes it possible to easily adapt any type of sensor and data, so it can be of interest for a wide variety of use cases (hospitals, museums, campuses). universities, hotels, etc.).

## Goals

The objective of this project is to show users in a clear, simple and visual way the information of an intelligent building, so that, depending on the use case and the user's needs; data such as occupancy, temperature, noise level, facilities´ efficiency in which they are located, etc. can be checked.

Additionally, the objective of operating and processing the information that is received arises so that the owner can analyze the data of any time interval and check what are the key points to improve on its consumption, at what times of the day the occupation is greater , etc.

## Work plan

The development of this dashboard has been divided into 3 phases:

1. **Investigation Phase**

   Focused on finding out what data was available from the different sensors in the building (and choosing the ones that were most interesting for the user) and checking how they could be collected from Mendix. Multiple learning sessions (about connections to APIs and translation of XML and JSON data to objects in Mendix) were also necessary in addition to our previous knowledge of this tool.

   At this point we already had the necessary data in Mendix for the visual development of the application, therefore the development of the back-end took place in parallel to the research.

2. **Planning Phase**

   We carried out numerous brainstorming sessions for the development of various design concepts and we modified it with the intention of getting closer to the needs of the end user. Other functionalities beyond the design were also taken into account, such as the configuration and customization of the application.

3. **Implementation Phase**

Once the bases on the design and the information to be displayed have been chosen, we proceed to the final development of the application.

4. **Design and research on additional use cases**

We proceed to investigate and detail new solutions in which Kiosko Digital would fit satisfactorily making use of data of other natures and for various purposes. For the detailed design the InkScape tool is used where the elements are realistically displayed.

# Conclusions and future work

The main objective of this project was to develop a platform in the form of a general dashboard for the display and processing of data collected by smart building sensors. This objective has been achieved thanks to the development aimed at the possibility of adapting to information of different kinds and the simplicity that it entails due to Low Code development.

The next goal was to allow users to customize the visual aspects of the application. This goal has also been completed with a simple customization tab where the user can change the image displayed at the top left corner, the scrolling text displayed at the bottom and the theme of the app to 4 basic colors (white , black, blue and purple).

With the development of this dashboard, a general and customizable framework tool has been achieved that stands out among other similar tools due to the connecting to smart buildings with the capability of acquiring any information available, which leaves us with the following points as future work:

- **Develop an interface for mobile devices or tablets:** this application is developed with native web technology and ready to be uploaded to the cloud, so any type of device capable of connecting to the Internet could access this page. Currently, the dashboard is designed to be displayed on large screens in the entrances and common areas of the buildings, however, with an interface designed to be easy on the eyes on mobile devices, it would result in a considerable increase in accessibility for any user wanting to access the application even outside the building.

- **Increase the customization:** additional functionalities could be implemented so that the end users of the application could place graphic blocks with the arrangement that they wanted, being able to eliminate those that they do not want, increase or decrease their size, change the type of graphic for the display of specific data (eg change the savings graph to linear), modify the colors of those graphs and blocks, and more customization that interested users may need and request.

- **Connect to more smart buildings:** the application is able to connect to multiple sensor networks located in smart buildings, an added value to demonstrate the usefulness of this general framework would be to connect to additional buildings.

- **Implement Artificial Intelligence to make predictions:** Add Artificial Intelligence to the application so that it could collect and process data with Machine Learning. By virtue of this, a more precise analysis of the results could be obtained in order to increase the chances of success and success rate of the predictions (for example, forecasting when there will be more occupation).

# CONTRIBUCIONES PERSONALES

A continuación, se detallan las contribuciones personales de cada uno de los alumnos implicados en este trabajo de fin de grado:

## Clara Fernández Forte

Inicialmente, en la fase de investigación, Clara Fernandez se encargó de buscar el método para realizar una conexión a una API desde Mendix, donde encontró una extensa documentación en Mendix Docs y un curso básico de conexiones a APIs en Mendix Academy y Youtube. Con toda esa información obtenida se puso manos a la obra para aprender a conectarse y desarrollar APIs en Mendix para su posterior implementación en el proyecto de Kiosko Digital.

Trás dominar de forma satisfactoria la conexión y el desarrollo de APIs dentro de la plataforma de Mendix se dedicó a buscar diferentes fuentes de información para recolectar datos del edificio de Siemens, encontrándose con la plataforma Enlighted y documentándose de la información que se podría obtener a través de sus APIs.

Al comprobar que la opción de conexiones a Enlighted era factible para este desarrollo, se puso en contacto con el soporte técnico de esta plataforma y a través de correos electrónicos, reuniones y preguntas por el chat de la plataforma web consiguió una cuenta de Enlighted con acceso a la planta quinta del edificio de Siemens en Tres Cantos.

Posteriormente, se incorporó al proceso de búsqueda de la conexión de las APIs a través de la plataforma Postman Agent que estaba realizando Carlos y juntos obtuvieron que cabeceras eran necesarias para la conexión, la forma de encriptar la

API Key y la obtención del *time stamp* para recoger los datos más recientes de los sensores.

A continuación, Clara buscó otras fuentes de información de interés para mostrar y procesar en el dashboard, tras un tiempo de investigación encontró la API de OpenWeather, se informó de los datos que ofrece y de los requisitos para la conexión. Teniendo toda la información de esta API creó un usuario de OpenWeather y solicitó una API Key para, posteriormente, probar la conexión en Postman Agent.

Después de toda la fase de investigación Clara pasó a la fase de planificación, donde se reunió junto a Carlos en varias sesiones para diseñar un boceto del dashboard con la intención de que sea user friendly, sencillo, intuitivo y capaz de mostrar toda la información útil sin sobrecargar a los usuarios de información.

Trás obtener un diseño final del dashboard, Clara pasó a la fase de implementación desarrollando las páginas del dashboard con el estilo y distribución especificados, documentandose e investigando sobre el repertorio de gráficos que debían implementarse. Durante este proceso estuvo apoyando a Carlos en el desarrollo de las conexiones a las APIs. También implementó la funcionalidad de personalización con todos los campos que el usuario puede modificar, estos incluyen la muestra de la imagen en la esquina superior izquierda y el cambio de esta, el cambio de temas con sus respectivos estilos y paleta de colores, y la generación del texto deslizante de la parte inferior de la pantalla permitiendo al usuario modificar el mensaje mostrado.

Todas estas nuevas funcionalidades las preparó en la página de personalización que los usuarios pueden ver al acceder a la aplicación, esta página sencilla ofrece una interfaz muy simple para realizar los cambios que se deseen en esos tres campos.

## Carlos Murcia Morilla

Inicialmente, en la fase de investigación, Carlos Murcia realizó una búsqueda y estudio del método para conectarse a una o varias APIs desde Mendix, Clara encontró una extensa documentación y recursos que Carlos pudo utilizar. Con toda esa información Carlos realizó una investigación exhaustiva sobre las conexiones y desarrollos de APIs en Mendix ya que ambos iban a necesitar conocer este aspecto para la codificación del dashboard.

Trás dominar de forma satisfactoria la conexión y el desarrollo de APIs dentro de la plataforma de Mendix buscó junto a Clara diferentes fuentes de información para recolectar datos del edificio de Siemens, y al encontrar la plataforma Enlighted se documentó de las diferentes APIs que ofrecía y los datos que recolectaba cada una de ellas.

Al acabar la investigación de las APIs de Enlighted, mientras Clara conseguía permisos de acceso, él investigó sobre plataformas de pruebas de conexiones a APIs y encontró Postman Agent, donde comenzó preparando la prueba de diferentes APIs para su posterior testeo en cuanto se tuvieran permisos.

Los permisos los consiguió Clara al poco tiempo y se incorporó al testeo de las APIs, obtuvieron las cabeceras que requiere la conexión, el tipo de encriptación de la API Key y el *time stamp* para recoger los datos más recientes de los sensores.

Trás comprobar que las APIs devolvían información sin problemas de acceso se puso manos a la obra para desarrollar la funcionalidad de conexiones a las APIs dentro de la aplicación en Mendix. Codificando la traducción de la información del mensaje JSON o XML a objetos en Mendix (Import Mapping) y mostrando los datos en tablas sencillas a las que solo pueden acceder los administradores.

Después, pasó junto con Clara a la fase de planificación, donde se reunieron en varias sesiones para diseñar un boceto del dashboard con el fin de ofrecer a los usuarios una vista sencilla, simple y accesible mostrando la información relevante recogida y procesada por la aplicación.

Trás obtener un diseño final, Clara pasó a desarrollar las páginas del dashboard y Carlos acabó con toda la funcionalidad de conexión a las APIs de Enlighted, comenzando posteriormente con la conexión en Mendix a la nueva API OpenWeather que encontró y testeó Clara.

Con todas las conexiones realizadas y transportadas a objetos en la solución, comenzó con el procesamiento de los datos obtenidos para conseguir el ahorro y consumo agregado del día, organizando las llamadas en los distintos microflows y preparando las entidades necesarias para la obtención de los valores utilizados en los gráficos del dashboard. Adicionalmente, implementó una funcionalidad para los desarrolladores que permite añadir de forma manual valores procesados por los gráficos implementados por Clara para facilitar y agilizar la visualización del dashboard.

De forma paralela durante las tareas de conexión y procesamiento de datos de las APIs, ayudó a Clara con algunas tareas sencillas del diseño del dashboard de forma simétrica con las ayudas que ella le ofreció.

# Bibliografia


[1] https://www.enlightedinc.com/ Consultado el 2/12/2022. Página oficial de Enlighted Inc.

[2] https://velneo.com/blog/una-breve-historia-del-desarrollo-low-code Consultado el 12/3/2023. Blog con información de la historia del Low Code

[3] https://support.enlightedinc.com/hc/en-us/categories/360001347034-APIs Consultado el 20/12/2022. Página de información técnica de las llamadas API de Enlighted Inc

[4] https://openweathermap.org/current Consultado el 20/1/2023. Página de información técnica de las llamada API de OpenWeather

[5] https://docs.mendix.com/howto/integration/consume-a-rest-service/ Consultado el 12/1/2023. Página donde se consultaron los aspecto técnicos de implementación de APIs en Mendix.

[6] https://www.clicdata.com/es/ Consultado el 12/4/2023. Página oficial de ClickData, Dashboard de gran interés para realizar una comparativa con Kiosko Digital.

[7] https://www.mendix.com/ Consultado el 1/10/2022. Página oficial de la plataforma de desarrollo Low Code Mendix. Tecnología utilizada en el desarrollo de este dashboard.

[8] https://es.wikipedia.org/wiki/Plataforma_de_desarrollo_sin_c%C3%B3digo Consultado el 15/3/2023. Blog de información sobre la tecnología Low Code o No Code.

[9] https://inkscape.org/es/ Consultado el 24/3/2023. Plataforma de diseño de los dashboards de los nuevos casos de uso (de la facultad, del museo, etc.).
Para información adicional, véase el siguiente enlace: https://inkscape.org/es/acerca-de/

[10] https://qr.io/ Consultado el 18/4/2023. Página para generar un código qr utilizado en el dashboard de muestra de ocupación de la facultad y la biblioteca de la FDI.

[11] https://hmong.es/wiki/Mendix Consultado el 23/2/2023. Blog donde se reúne información de la historia de la plataforma Mendix.

[12] https://www.youtube.com/watch?v=mqRbwXRA49A Consultado el 20/12/2022. Video donde se explican los aspectos básicos de la conexión a REST Services desde Mendix.

[13] https://www.siemens.com/es/es.html Consultado el 9/4/2023. Página de Siemens S.A útil para poner en contexto del entorno y el edificio inteligente utilizados.

[14] https://www.capgemini.com/es-es/ Consultado el 17/3/2023. Fue uno de los principales Sponsors de la plataforma Mendix.

[15] https://www.accenture.com/es-es Consultado el 17/3/2023. Fue uno de los principales Sponsors de la plataforma Mendix junto a Capgemini.

[16] https://www.postman.com/product/what-is-postman/ Consultado el 17/1/2023. Página de explicación de la plataforma para la realización de pruebas de APIs PostMan.

[17] https://aws.amazon.com/es/what-is-aws/ Consultado el 20/3/2023. Página web oficial de AWS donde se explica qué es y sus características.

[18] https://www.atlassian.com/es/agile/scrum Consultado el 20/03/2023. Contiene una explicación detallada de la metodología de trabajo SCRUM y sus aspectos principales para su aplicación.

[19] Laura Pelkonen (2022). Bachelor's Thesis: Implementing a Mendix low-code application using best practices. Laurea University of Applied Sciences. https://www.theseus.fi/bitstream/handle/10024/785999/Pelkonen_Laura.pdf?sequence=2&isAllowed=y

[20] Zadka, M. (2022). Amazon Web Services (chapter 13). In: DevOps in Python. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-7996-0_13 ISBN: 978-1-4842-7995-3

[21] Sajee Mathew (2014). Overview of Amazon Web Services https://www.sysfore.com/Assets/PDF/aws-overview.pdf (pp 8-10)

[22] Rossberg, J. (2014). Introduction to Scrum and Agile Concepts. In: Beginning Application Lifecycle Management. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4302-5813-1_4