UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Diseño de una aplicación web para el control domótico de una vivienda.

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Pelegrín Garcés, Alex

Tutor/a: Albert Albiol, Manuela

Cotutor/a: Torres Bosch, María Victoria

CURSO ACADÉMICO: 2022/2023

Resumen

Se ha diseñado e implementado una aplicación web responsiva que permite al usuario visualizar y controlar los dispositivos inteligentes de una vivienda. La aplicación muestra los dispositivos de los que dispone la vivienda y ofrece funcionalidades para manejar estos dispositivos.

Estas funcionalidades abarcan tanto el control individual de los dispositivos como la agrupación y edición de varios dispositivos a la vez. Además proporciona herramientas para crear una lógica que se aplique a los dispositivos y sus atributos.

La aplicación está construida siguiendo una serie de guías y patrones para el desarrollo de interfaces web con el objetivo de conseguir una interfaz sencilla y fácil de utilizar para el usuario. Todo esto se ha conseguido utilizando un framework especializado en el desarrollo web y su patrón de diseño Modelo-Vista-Template.

Palabras clave: Django, aplicación, web, MVT, dispositivo, inteligente, smart home

Abstract

It has been designed and developed a responsive web application which allows the user to view and control the smart devices in a house. The application shows the devices of the smart home and provides some functionalities to manage them.

Those functionalities range from the individual management of the devices to the possibility of grouping and editing some devices at once. Moreover, it provides tools to create a logic which applies to the devices and its attributes.

The application is built following some guidelines for web design, aiming to achieve a simple and easy-to-use interface for the user. To reach this it has been used a web development framework featuring the Model-View-Template architecture.

Keywords: Django, application, web, MVT, device, smart, smart home

Tabla de contenidos

1. Introducción

1.1. Motivación

En los últimos años la tecnología ha avanzado a pasos gigantes en prácticamente todos los ámbitos.

Desde los dispositivos móviles, que se han convertido en una parte indispensable de nuestro día a día, a la tecnología y nuevas funcionalidades de los automóviles con los que nos desplazamos, la sociedad ha ido utilizando más y más las nuevas tecnologías para hacer su día a día más sencillo y cómodo.

El hogar también ha sufrido esta automatización. En los últimos años se ha incrementado el interés en los dispositivos del hogar inteligentes, y por lo tanto, el número de casas provistas de algún tipo de tecnología domótica en ellas.

De hecho, según un estudio realizado por "Statsia", se prevé una subida importante (casi del triple desde 2017) en la cantidad de viviendas que implementarán esta tecnología a lo largo de los años tal y como se muestra en la Figura 1.

En concreto, según el estudio, hoy en día contamos con un 12,17% de viviendas "inteligentes", y esperan que, en 2024, ese porcentaje aumente hasta el 18,4% y en 2025 lo haga hasta un 21,09%.

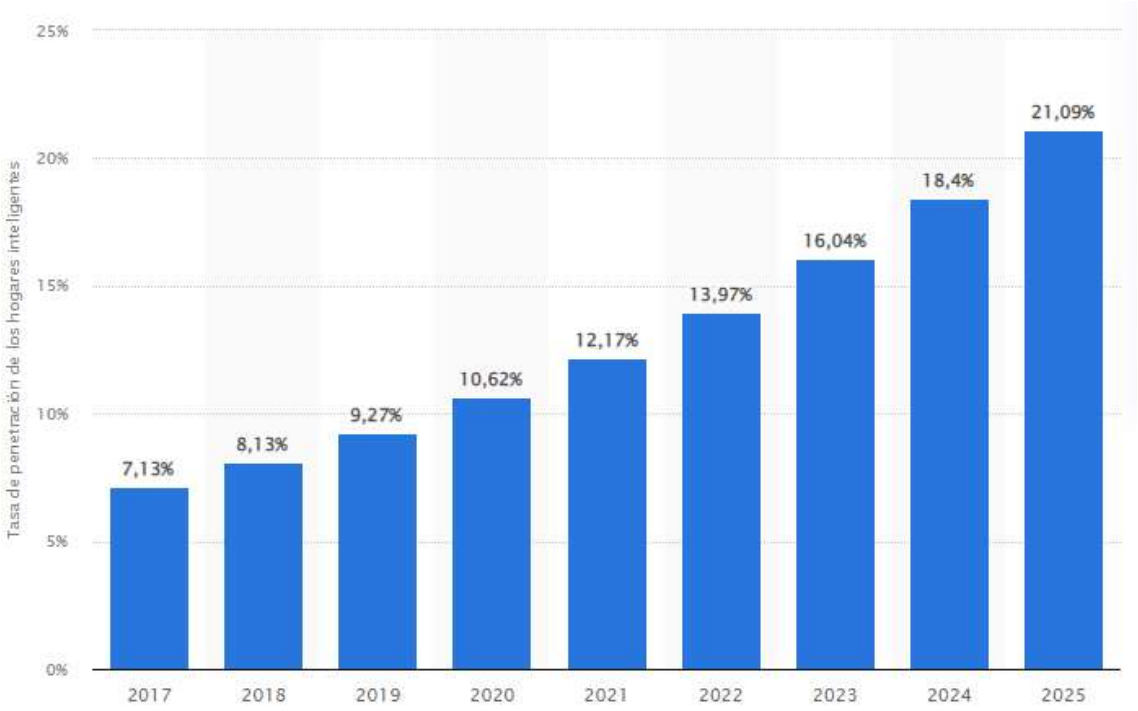

FIGURA 1: PREVISIÓN DE VIVIENDAS INTELIGENTES EN EL MUNDO. FUENTE: HTTPS://ES.STATISTA.COM/ESTADISTICAS/1166176/TASA-DE-PENETRACION-DE-LOS-SMART-HOMES-EN-EL-MUNDO/

Además, los datos del estudio realizado en 2018 por ABI Research [1] pronosticaron un aumento en las ventas de dispositivos inteligentes. De 2017 a 2018 aumentaron las en un 55%, sumando un total de 252 millones de unidades vendidas. Para 2023 ese número sigue aumentando, llegando a los 467 millones de unidades vendidas.

Viendo esta tendencia ascendente en el interés de la domótica y los dispositivos inteligentes es de esperar que cada vez haya más personas que introduzcan en sus hogares dispositivos inteligentes y, por lo tanto, necesitarán herramientas para gestionarlos de manera sencilla.

Por esa razón y junto a la poca madurez que hay aún en las aplicaciones existentes se consideró crear una aplicación web donde el usuario pudiese gestionar de manera sencilla los dispositivos que tuviese en su hogar.

1.2. Objetivo

El objetivo del trabajo final de grado (TFG) es el desarrollo de una aplicación web responsiva que permita al usuario visualizar y controlar los dispositivos inteligentes de la vivienda. La aplicación muestra los dispositivos de los que dispone la vivienda y ofrece funcionalidades para manejar estos dispositivos.

En concreto, la aplicación ha seguido una serie de pautas para conseguir el objetivo ya mencionado. Estas son:

- Diseñar una interfaz de fácil uso para cualquier usuario, independientemente de sus capacidades tecnológicas.
- Facilitar la realización de acciones típicas sobre los dispositivos del hogar, tanto individualmente desde un mismo dispositivo como de manera grupal.
- Crear un sistema de organización de los dispositivos simple y efectivo.
- Facilitar la visualización de los componentes y de los datos de la aplicación para que el usuario pueda recibir y entender una mayor cantidad de información haciendo el menor esfuerzo posible.

1.3. Estructura de la memoria

En este apartado desglosaremos y definiremos de manera breve los puntos que trataremos a continuación.

- **Capítulo 2. Estado del arte**

En este punto compararemos nuestra solución con otras páginas web y aplicaciones desarrolladas por las empresas pioneras en el sector, enumerando sus virtudes y defectos y añadiendo los cambios y mejoras que se incorporaremos respecto a esas aplicaciones.

- **Capítulo 3. Análisis del problema**

El análisis del problema trata del planteamiento de los pasos por los que ha de pasar para llegar a la solución final. Teniendo en cuenta los objetivos de la aplicación, se ha de seguir una serie de pautas y pasos para definir la estructura inicial de la aplicación.

En este punto es donde se separan los elementos o módulos que van a componer la aplicación. En nuestro caso la aplicación está dividida en 5 módulos que analizaremos desde distintos ángulos en los apartados correspondientes: el dashboard, los dispositivos, los modos, las reglas y las estancias.

- **Capítulo 4. Diseño de la solución**

Una vez clara la estructura de la aplicación se comienza la implementación de los módulos en las distintas tecnologías seleccionadas.

Concretamente, en este apartado se detallará la utilización de estas tecnologías y se hablará de manera teórica del framework utilizado y de los patrones y arquitectura que soporta. Se explicarán sus ventajas respecto a las herramientas tradicionales, sus funcionalidades y un poco de la estructura interna.

Por otro lado también se hablará de la base de datos elegida, su configuración y ventajas respecto al resto de opciones.

- **Capítulo 5. Desarrollo de la solución**

En el apartado del desarrollo se profundizará brevemente en los componentes más importantes del patrón MVT tanto de manera teórica como práctica.

Se mostrarán las pantallas más importantes que componen la aplicación y se explicarán desde los distintos puntos de vista del patrón MVT: el modelo, la vista y la template.

- **Capítulo 6. Conclusiones**

En este capítulo se mostrará tanto las facilidades como los problemas e inconvenientes que han surgido durante todas las fases de desarrollo que ha tenido la página web.

- **Capítulo 7. Relación con los estudios cursados**

Finalmente, este capítulo tratará de explicar y relacionar la materia dada en las asignaturas del grado cursadas con el proyecto que se ha desarrollado, y se valorará desde una perspectiva más personal si se considera que han sido realmente útiles en el desarrollo de la aplicación.

2. Estado del arte

Desde hace unos años tanto en España como en prácticamente el resto del mundo se ha visto un auge en la popularidad de los dispositivos domóticos. Cada vez más casas cuentan como mínimo con un dispositivo “inteligente”, que el usuario puede configurar y manejar para que haga diferentes tareas o se apague cuando se le programe.

Las empresas han respondido a esta creciente popularidad e interés en los dispositivos inteligentes, de manera que en los últimos años han ido desarrollando software para la gestión de estos, cada vez con más funcionalidades y facilidades para el usuario.

Dos de las aplicaciones más populares son Google Home y HomeKit, las cuales se describen a continuación.

2.1. Google Home

Google Home[1] es la propuesta de Google para configurar, gestionar y controlar los dispositivos inteligentes de casa. Está desarrollado para su funcionamiento en dispositivos móviles con sistema operativo iOS o Android, y se comporta como un “hub” o centro, desde donde se podrán realizar acciones y gestionar los dispositivos vinculados.

La funcionalidad de Google Home se divide en diferentes aspectos:

- **Dispositivos:** Desde esta sección el usuario puede consultar el estado de los dispositivos, acceder a los parámetros de cada uno y controlar y ajustarlos a placer.
- **Automatizaciones:** Desde esta sección el usuario puede personalizar los dispositivos para que, cuando se cumpla el requisito que el usuario haya puesto, se despliegue de manera automática la acción asignada.
- **Actividad:** Esta funcionalidad ayuda al usuario a estar al tanto de lo que ocurre en todo momento en casa. Dispositivos como cámaras dejarán guardadas las imágenes en cada momento y el usuario podrá consultar el estado de estas, al igual que el registro de imágenes que hayan grabado.
- **Configuración:** El usuario puede crear y gestionar los dispositivos, servicios y miembros o perfiles, cada uno con sus preferencias y ajustes.

2.2. HomeKit

HomeKit[2] es la aplicación desarrollada por Apple que reúne nuestros accesorios o dispositivos domóticos, y nos permite realizar acciones sobre ellos y configurarlos.

[1] https://home.google.com/welcome/
[2] https://www.apple.com/es/home-app/

La funcionalidad básica es similar a la de Google Home: agregamos y configuramos un dispositivo, y la aplicación nos permite gestionarlo y realizar acciones sobre él. También nos permite configurar automatizaciones sobre los dispositivos, para que, dada una condición, se desencadene la acción correspondiente. Como ya hemos hablado de esto no vamos a indagar más en la funcionalidad de HomeKit, aunque sí que vamos a destacar un punto importante: Siri.

Una de las ventajas que nos ofrece Apple es la integración de la aplicación en nuestro dispositivo Apple, lo que nos permite aprovecharnos del asistente de voz que llevan los dispositivos móviles para una mayor comodidad al indicar las acciones o órdenes a realizar.

Otro de los puntos a tener en cuenta es que es una aplicación exclusiva para los dispositivos de la marca Apple, tanto sobre mesa como móviles. Esto nos permite controlar desde el MacBook o desde el iPhone o iPad nuestros dispositivos inteligentes, pero puede ser un problema si no poseemos uno de ellos. Además cuenta con un número limitado de dispositivos compatibles con la aplicación, lo que también limita su uso a un menor grupo de personas.

2.3. Propuesta

Nuestra propuesta se ha basado en seleccionar las características más destacables del resto de aplicaciones del mercado, entre otras las que acabamos de ver en el apartado anterior, y a su vez mejorar algunos de los aspectos que el bajo nuestro punto de vista pueden tener algunos problemas.

Partiendo de esa base se van a listar tanto las características nuevas como las que se han implementado basándose en las aplicaciones existentes.

- **Gestión de dispositivos:** El usuario puede añadir, configurar y eliminar dispositivos a voluntad. De entre una lista de los dispositivos, el usuario puede ver con facilidad las características de estos dispositivos y seleccionar el que quiera para editarlo.
- **Automatización de los dispositivos:** Al igual que algunas aplicaciones existentes pueden hacer, el usuario puede crear una lógica que aplique a un dispositivo dependiendo de que se cumpla una condición.
- **Modificación masiva de dispositivos:** Uno de los problemas que encontramos en alguno de los softwares existentes es la dificultad a la hora de configurar más de un dispositivo a la vez. Muchas veces el usuario debe gestionarlos uno a uno, mientras que en nuestro caso se permite modificar las características de distintos dispositivos a la vez.
- **Accesos directos para los elementos más utilizados:** Para que el usuario pueda ver o modificar elementos de la manera más rápida y simple existe un lugar donde se concentran las acciones rápidas que el usuario puede realizar. De un simple vistazo puede ejecutar las acciones más comunes sobre los

elementos de la aplicación, ver y modificar el estado de los elementos más utilizados, etc.

3. Análisis del problema

A la hora de desarrollar un software, uno de los primeros pasos a realizar es el análisis del problema. En este paso, se debe identificar los requisitos de la aplicación y plantear una solución acorde a estos requisitos.

3.1. Identificación de requisitos

La gestión de los dispositivos de un hogar digital conlleva los siguientes requisitos funcionales:

- Añadir o eliminar dispositivos domóticos.
- Activar o desactivar dispositivos domóticos.
- Establecer parámetros de configuración de los dispositivos domóticos.
- Crear lógicas que controlen y modifiquen el estado de los dispositivos si se cumplen las condiciones necesarias.

Como hemos visto en la propuesta, el usuario debe poder realizar estas acciones de la manera más rápida y sencilla posible. Así que, con el objetivo de simplificar el uso de estos dispositivos, se incluyen los siguientes requisitos en nuestra aplicación:

- Los dispositivos se han de poder configurar tanto de manera individual como de manera grupal, simplificando así la tarea de modificación de varios registros,
- Se ha de poder reconocer fácilmente cada uno de los dispositivos domóticos. Es necesario teniendo en cuenta que puede haber muchos dispositivos en una casa inteligente.
- El usuario ha de poder tener una idea general del estado de los dispositivos de la casa en cualquier momento.

Todos estos requisitos han de desarrollarse proporcionando al usuario una experiencia lo más sencilla y amigable posible.

La realización de las tareas en la aplicación debe estar planteada de tal manera en la que el usuario se sienta lo más cómodo posible; no debe sentir en ningún momento que las acciones que está realizando son tediosas o complicadas.

Así que una vez analizadas las necesidades, se puede tener una idea de las funcionalidades que va a cubrir la aplicación, y se puede comenzar a crear un modelo de dominio que relacione las distintas partes que compondrán la web y la interacción entre ellas.

3.2. Modelo de dominio

Un modelo de dominio [2] se define como una representación conceptual y estructurada de un problema, utilizado para comprender y analizar los elementos y los conceptos clave que lo componen.

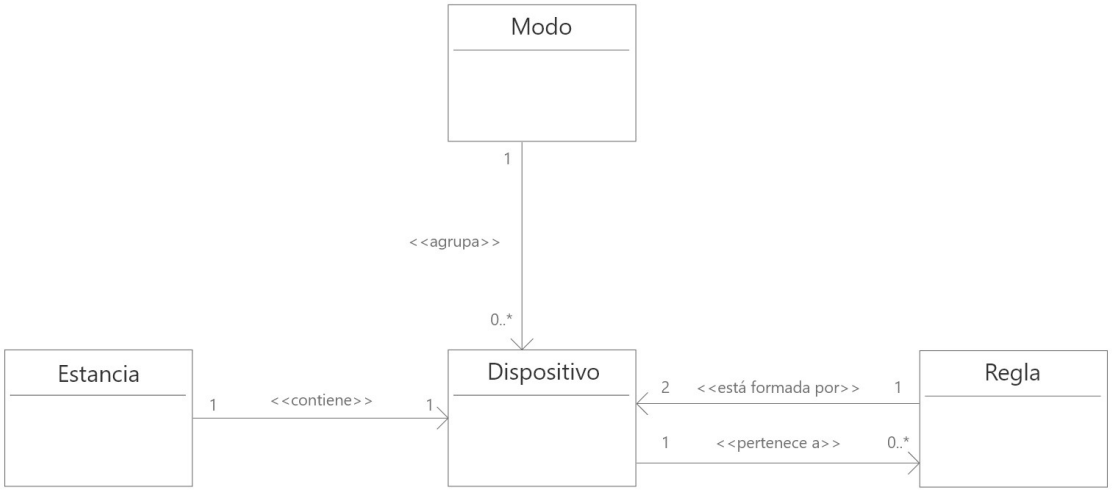

FIGURA 2: MODELO DE DOMINIO DE LA APLICACIÓN DESARROLLADA

En la Figura 2 se puede ver el modelo de dominio de la aplicación. En él podemos identificar los 4 elementos que, interactuando entre sí, componen la funcionalidad de la aplicación y que vamos a explicar a continuación.

3.2.1. Dispositivos

Un dispositivo es un aparato electrónico que forma parte del hogar inteligente. Este aparato automatiza o controla alguna función del hogar, con el objetivo de mejorar la comodidad, la eficiencia energética, la seguridad o la gestión de recursos entre muchos otros ejemplos.

Los dispositivos siempre se encuentran en habitaciones o estancias. Además, los dispositivos pueden formar parte de los modos o las reglas, modelos que se definirán a continuación.

3.2.2. Modos

Los modos representan la funcionalidad de poder actualizar en cualquier momento los valores de los dispositivos a los que ha definido el usuario.

Funcionan como accesos directos que el usuario configura. Para una selección de dispositivos el usuario predetermina un valor a sus atributos y, cuando el modo es activado, esos atributos cambian el valor que tengan en ese momento por el valor predeterminado.

Además, cada modo puede aplicar esta lógica a tantos dispositivos que haya creados en la aplicación como el usuario prefiera, independientemente de su tipo o características.

3.2.3. Reglas

Las reglas forman la lógica que se aplicará a un dispositivo concreto siempre que se cumpla una condición específica.

Una regla se compone de dos partes: la condición de la regla y el resultado que se ejecutará cuando esa condición sea cierta.

Por un lado, la condición viene dada por el valor de un atributo que tiene un dispositivo. Por ejemplo, si un dispositivo tiene el atributo brillo con un valor superior al 50% la condición será cierta, y por lo tanto se ejecutará el resultado.

El resultado representa la lógica que se ejecutará una vez se cumpla la condición de la regla. Una vez se active la regla actualizará el valor actual del atributo por el que haya especificado el usuario.

Se puede entender la funcionalidad de las reglas como algo similar al de los modos. Mientras que los modos se activan por el usuario sobre un conjunto de dispositivos de manera manual, la lógica de las reglas se disparará automáticamente en el momento en el que la condición se evalúe como cierta.

3.2.4. Estancias

Una estancia representa el lugar físico de la casa inteligente donde están situado uno o más dispositivos y tiene la función de agrupar físicamente los dispositivos de la misma manera en la que están situados físicamente en la casa.

Cada dispositivo, como hemos visto, debe estar relacionado con una estancia, y cada una de estas estancias puede tener todos los dispositivos que se necesite.

3.3. Solución propuesta

Teniendo en cuenta los requisitos que se han planteado se va a construir una aplicación web que permita satisfacer cada uno de estos requisitos de forma muy sencilla.

La aplicación se dividirá en módulos, concretamente uno por cada concepto que hemos visto en el modelo de dominio. Aunque ya hemos visto de qué trata cada módulo, vamos a hacer un breve resumen de algunas de las acciones que el usuario podrá realizar sobre estos módulos.

- **Dispositivos**: Este módulo permitirá al usuario gestionar los dispositivos de manera sencilla. Desde aquí se podrá añadir, configurar y eliminar un dispositivo.

- **Modos**: El módulo de los modos permitirá al usuario crear, configurar y eliminar los modos. Estos modos tendrán asociados un grupo de dispositivos, que el usuario podrá editar en cualquier momento. Finalmente, el usuario podrá desde este módulo activar los modos que tenga creados.
- **Reglas**: El usuario podrá desde este módulo crear, modificar o eliminar reglas. Será capaz de determinar tanto los parámetros que constituyen la condición de la regla como los parámetros que constituyen el resultado de la misma.
- **Estancias**: El usuario podrá desde este módulo añadir, editar o eliminar estancias de la aplicación. Cada dispositivo será vinculado a la estancia que el usuario seleccione.

Además de estos módulos la aplicación contará también con un dashboard o cuadro de mandos que funcionará a su vez como pantalla principal.

- **Dashboard**: Este módulo permitirá al usuario tener un conjunto global del estado de la aplicación en un solo vistazo. Mediante la utilización de gráficas y otros recursos que permitan una sencilla visualización de los datos mostrados el usuario podrá, en un solo lugar y de un vistazo, tener acceso a los datos más importantes de la aplicación, como un listado de los dispositivos más utilizados y el estado de los mismos o un acceso directo para activar modos.

4. Diseño de la solución

Este capítulo presenta los cimientos sobre los que está construida la aplicación. Hasta ahora hemos visto las necesidades y requisitos de la aplicación, así que ahora veremos cómo se le ha dado solución al problema planteado.

4.1. Tecnologías utilizadas

En primer lugar, vamos a mostrar las herramientas que se han necesitado para el desarrollo de la aplicación, mostrando sus características y las razones por la que han sido escogidas.

4.1.1. Django[3]

Django es un framework de alto nivel escrito en Python y utilizado para desarrollar aplicaciones web rápidas y seguras.

Fue desarrollado por un equipo de programadores web para gestionar páginas web orientadas a noticias de la World Company de Lawrance, Kansas. Fue lanzado en 2005 como software de código abierto y desde entonces ha ganado popularidad debido a su enfoque en la eficiencia y su gran facilidad de uso.

La meta de Django es facilitar la creación de sitios web complejos. Pone énfasis en la reutilización, la conectividad y extensibilidad de componentes y el desarrollo rápido.

Algunas de las características principales de Django son:

- **Patrón de diseño MVT**: Django sigue el patrón de diseño Modelo-Vista-Template (MVT), que facilita la separación de la lógica de negocio, la presentación y la gestión y manejo de datos.
- **ORM incorporado**: Django proporciona una mapeador objeto-relacional (ORM) incorporado que permite interactuar con la base de datos utilizando código Python en lugar de SQL directamente. Se caracteriza por simplificar el acceso y la manipulación de los datos.
- **Sistema de URLs**: Django cuenta con un sistema de enrutamiento de URLs potente y flexible. Permite mapear las URLs a las vistas correspondientes, lo que facilita la gestión de solicitudes HTTP.
- **Sistema de plantillas**: Django incluye un sistema de plantillas que permite separar la presentación de los datos por pantalla de la lógica del código en Python. Esto facilita la creación de interfaces de usuario dinámicas y reutilizables.
- **Administrador de Django**: Django proporciona un administrador de interfaz de usuario automático y completo para administrar los modelos de la base

[3] https://www.djangoproject.com/

de datos. Esto permite la creación, edición y eliminación de registros de la base de datos de la manera más sencilla posible.

Además cuenta con una serie de ventajas respecto a otras estructuras más tradicionales a la hora de crear una aplicación web.

- **Alta productividad**: Django proporciona un gran abanico de componentes y bibliotecas integradas que aceleran el desarrollo web. Además permite a los desarrolladores crear aplicaciones web de la manera más sencilla y rápida posible dado su enfoque basado en la simplicidad y la automatización.
- **Escalabilidad**: Django es escalable y puede manejar grandes volúmenes de tráfico. Tiene un sistema de caché eficiente y opciones de escalado horizontal que facilitan el manejo de aplicaciones web de alto rendimiento.
- **Seguridad**: Django incorpora mecanismos de seguridad sólidos para proteger las aplicaciones web de ataques comunes, como inyecciones SQL y ataques de falsificación entre sitios.

El patrón de diseño Modelo-vista-template mencionado anteriormente constituye el grueso de la manera de funcionar de Django y por ende, la manera en la que se desarrollan las aplicaciones. Por esta razón entraremos más en detalle en patrón modelo-vista-template en el siguiente apartado.

4.1.1.1. Modelo-Vista-Template (MVT)

Un patrón MVT [3] es un patrón arquitectural de software que separa los datos de la aplicación, la interfaz del usuario y la lógica de control en tres componentes distintos:

- El **modelo** que contiene una representación de los datos que maneja el sistema, la lógica y los mecanismos de persistencia.
- La **vista** o la interfaz de usuario, que compone la información que se envía al cliente y los mecanismos de interacción con éste.
- La **template**, que decide como es mostrada la información por el navegador.

Todos los módulos que se han mostrado en el apartado del análisis siguen este mismo patrón de diseño. Podemos ver en la Figura 3 una representación de los elementos que componen la estructura de Django siguiendo el patrón MVT.

A continuación, se va a definir de manera simple los elementos más importantes y la función que cumplen.

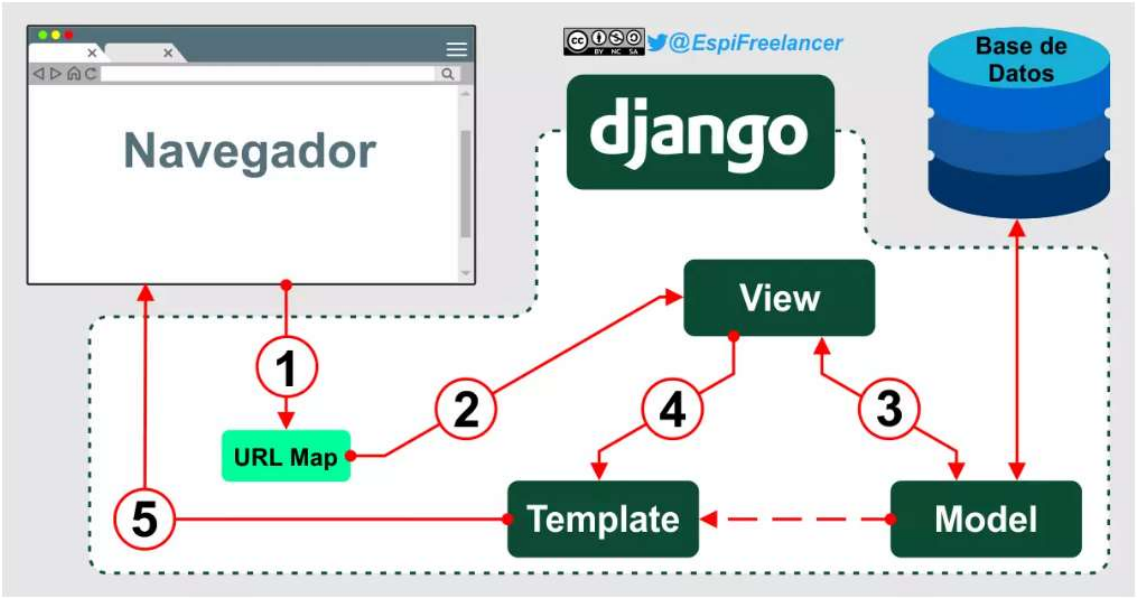

FIGURA 3: COMPONENTES Y ARQUITECTURA INTERNA DE DJANGO. HTTPS://ESPIFREELANCER.COM/MTV-DJANGO.HTML

- **Mapeador URL**: Contiene un vínculo entre la lista de URLs y las vistas asociadas.
- **Vista**: Una vista contiene la lógica que se ejecuta en una pantalla. Se encarga de obtener los datos correspondientes para posteriormente ser mostrados por pantalla a través de la template.
- **Modelo**: Representa los objetos y sus atributos que hay creados en el sistema. Desde la vista y mediante consultas a la base de datos se recuperan esos datos para ser manejados y utilizados.
- **Base de datos**: Guarda todos los registros de los modelos de la aplicación de manera persistente.
- **Template**: Gestiona y muestra por pantalla los datos recibidos desde la vista. Es lo que el usuario ve por pantalla.

De todos estos elementos que hemos visto los más importantes para desarrollar la aplicación son los que aparecen en el nombre del patrón MVT: El modelo, la vista y la template.

4.1.2. Base de datos

Como ya hemos adelantado la base de datos se encarga de ofrecer persistencia a los datos contenidos en la aplicación. Por cada modelo o clase que se haya creado, la base de datos guardará los registros y los atributos con sus respectivos valores. Así, desde cualquier vista, se podrá acceder mediante peticiones a la base de datos a cualquier registro y a cualquier atributo.

En nuestro caso se decidió utilizar la base de datos SQLite [4]. Aunque Django también da soporte a otras bases de datos [5], como MySQL o PostgreSQL se optó por esta

solución porque era la más óptima dadas las necesidades de la aplicación. Algunas de estas razones son:

- Más **práctica y accesible** que el resto de opciones, así que es la indicada para aplicaciones más sencillas y sin demasiado consumo de datos, como es el caso de nuestra aplicación.
- No tiene tanta **capacidad de almacenamiento**, ya que admite bases de datos de entre 250kb a 1GB como máximo. Al desarrollar una aplicación con pocos modelos y no demasiado complejos la cantidad de información a almacenar no requería utilizar una base de datos de gran capacidad.
- **No requiere instalación**, así que era la opción más sencilla de utilizar.

Teniendo en cuenta que no sería una aplicación muy compleja, no se necesitaría demasiado tráfico de datos ni consultas extremadamente rápidas se decidió utilizar SQLite, la opción más sencilla a todos los niveles pero que, a su vez, proporcionaba todo lo necesario teniendo en cuenta las necesidades de la aplicación.

En la Figura 4 se puede ver cómo en el archivo "settings.py" de Django que gestiona las conexiones se ha utilizado SQLite para la base de datos.

```
# Database
# https://docs.djangoproject.com/en/4.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

FIGURA 4: IMPLEMENTACIÓN DE LA BASE DE DATOS SQLITE3 EN EL ARCHIVO DE CONFIGURACIÓN DEL PROYECTO EN DJANGO

4.2. Arquitectura

La arquitectura de la aplicación sigue el patrón Modelo-Vista-Template (MVT) sobre la que ya hemos hablado. En el caso del desarrollo de nuestra aplicación, el patrón Modelo-Vista-Template presenta una serie de facilidades y ventajas que lo hace más interesante respecto a utilizar otras estructuras más tradicionales.

En un primer momento se planteó estructurar la aplicación en un archivo que utilizase HTML y CSS para la gestión de las templates, y JavaScript para ejecutar toda la lógica que necesitase. Esta estructura de la aplicación se desechó rápidamente dado los inconvenientes que conllevaba. Algunos de los más destacables son:

- **No permitía la persistencia**, ya que los datos se enviaban a través de archivos con formato JSON de manera estática. Si se quería crear, modificar o eliminar datos había que instalar librerías externas y cambiar un poco el planteamiento y la estructura de los datos.
- Estaba **mal gestionada la herencia de las templates**, ya que la tecnología utilizada no permitía de manera sencilla la reutilización de código a nivel de HTML.
- **Dificultad a la hora de soportar errores** o excepciones, puesto que Python no ofrece en algunos casos una manera sencilla de manejar los errores, como puede ser a la hora de añadir registros a la base de datos y comprobar que los datos introducidos son correctos, por ejemplo.

Viendo estos problemas y dadas las ventajas que ofrecen los frameworks en general se decidió trabajar con Django y el patrón Modelo-Vista-Template.

4.3. Modelo-Vista-Template en Django

La manera en la que Django gestiona los modelos y las vistas no tiene demasiado misterio ya que se comporta de manera similar a cómo lo hacen otros frameworks.

Como se puede ver en la Figura 5 Django utiliza el archivo “views.py” para la gestión de las vistas y las funciones que son necesarias para el funcionamiento del programa.

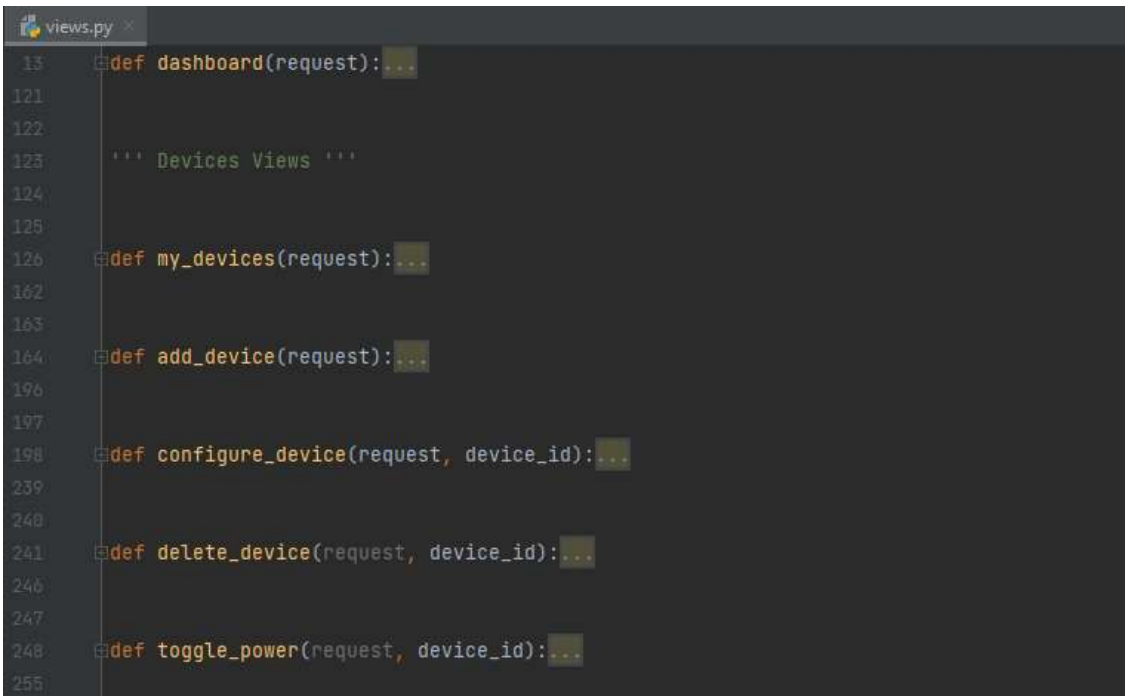

FIGURA 5: VISTAZO GENERAL AL ARCHIVO DE VISTAS EN DJANGO

Por otro lado para la creación y gestión de los modelos Django utiliza el archivo "models.py", como se puede ver en la Figura 6.

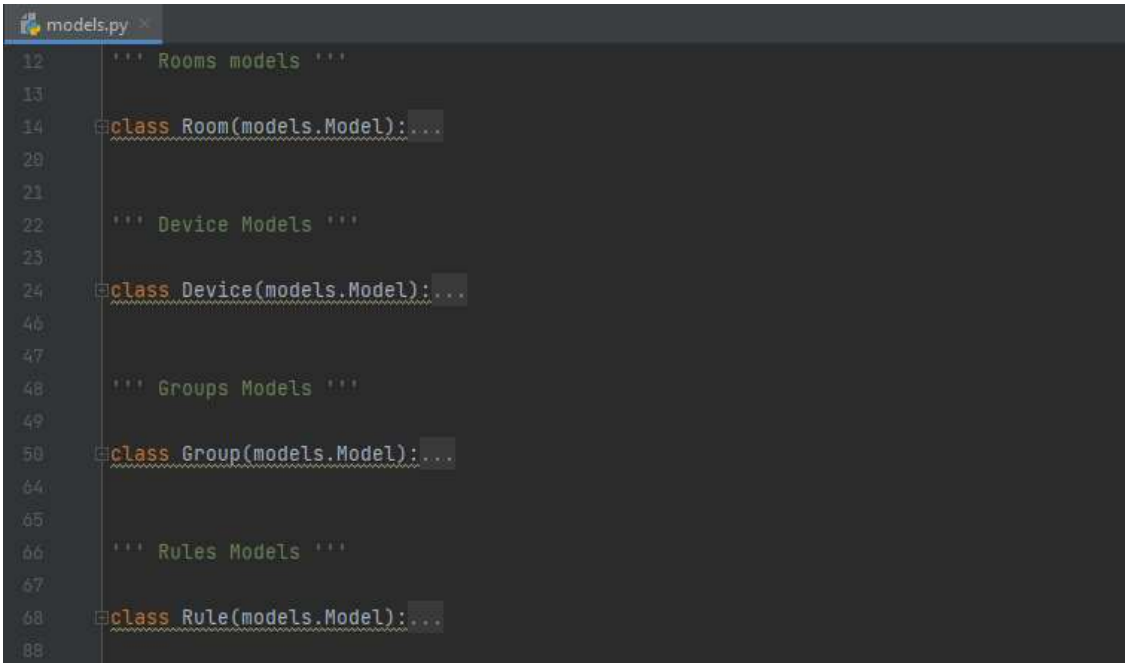

FIGURA 6: VISTAZO GENERAL AL ARCHIVO DE MODELOS EN DJANGO

En cambio para las templates Django incorpora lo que llama "Django Template Language", un lenguaje integrado basado en HTML que proporciona distintas funcionalidades.

4.3.1. Django Template Language

El lenguaje de templates de Django o Django Template Language permite crear y renderizar páginas HTML de manera dinámica. Las templates permiten combinar la lógica y los datos recibidos por las vistas para mostrar información por pantalla.

Las templates de Django contienen etiquetas o "tags", unos marcadores especiales que permiten insertar variables y realizar bucles o condiciones, como podemos ver en la Figura 7.

FIGURA 7: USO DE ETIQUETAS Y CONDICIONES EN UNA TEMPLATE DE DJANGO

Además de incluir este tipo de lógica permite la herencia entre distintas templates.

FIGURA 8: USO DE BLOQUES EN UNA TEMPLATE DE DJANGO

La herencia de templates en Django permite reutilizar y extender la estructura de una de las templates y ser utilizada en cualquier otra template, modificando incluso partes de su código de manera sencilla.

Como podemos ver en la Figura 8 desde una template se puede crear distintos bloques que contendrán parte del código HTML. Otra template diferente podrá extender esa primera template (Figura 9) e incluso añadir o cambiar parte del código modificando esos bloques de código.

FIGURA 9: CÓDIGO EN EL ARCHIVO DE TEMPLATES DE DJANGO PARA EXTENDER LA FUNCIONALIDAD DE UNA TEMPLATE EXTERNA

4.4. Diseño de las interfaces

El diseño de las interfaces que componen la aplicación busca la mayor facilidad y comodidad para el usuario a la hora de realizar acciones.

Como ya se ha comentado en anterioridad, el diseño de la aplicación busca ser lo más amigable para el usuario posible. Para ello se han tenido en cuenta muchos de los principios de usabilidad a la hora de diseñar las interfaces y se ha intentado proporcionar al usuario una experiencia lo más sencilla y agradable posible.

4.4.1. Principios de usabilidad

Jakob Nielsen describió una serie de principios [6] que establecerían unas bases para mejorar la usabilidad de los productos digitales. Estos son:

1. **Visibilidad del estado del sistema:** Los usuarios deben poder percibir el estado actual del sistema, ya sea a través de indicadores visuales, mensajes o retroalimentación en tiempo real.
2. **Coincidencia entre el sistema y el mundo real:** El sistema debe usar un lenguaje y conceptos familiares para los usuarios, basados en su conocimiento y experiencia del mundo real.
3. **Control y libertad del usuario:** Los usuarios deben tener la capacidad de deshacer acciones no deseadas o salir de situaciones o pantallas sin dificultad.
4. **Consistencia y estándares:** El sistema debe seguir convenciones y estándares reconocidos para que los usuarios puedan utilizar su conocimiento previo y aplicarlo a nuevos contextos.
5. **Prevención de errores:** El diseño debe enfocarse en evitar que los usuarios cometan errores. Esto puede incluir medidas como por ejemplo confirmaciones antes de realizar acciones irreversibles.
6. **Reconocer antes que recordar:** El usuario no debe tener que recordar información de una parte del sistema a otra. En su lugar, la información y las opciones deben estar claramente presentes y visibles cuando sean necesarias.
7. **Flexibilidad y eficiencia de uso:** El sistema debe permitir a los usuarios personalizar su experiencia y proporcionar atajos o accesos directos.
8. **Diseño estético y minimalista:** El diseño debe ser limpio, simple y estéticamente agradable, evitando la inclusión de elementos innecesarios o distracciones visuales.
9. **Ayuda y documentación:** Cuando sea necesario se debe proporcionar documentación y ayuda contextual para guiar a los usuarios en la comprensión y uso del sistema.
10. **Recuperación ante errores:** El sistema debe ser capaz de detectar errores introducidos por el usuario y debe proporcionar soluciones y alternativas para solucionar el problema. Por ejemplo, cuando un usuario introduce datos incorrectos en un formulario, el sistema debe de proporcionar unas guías y soluciones para que el usuario sea capaz de introducir de manera correcta esos datos.

Como hemos adelantado estos principios de usabilidad se han tenido en cuenta a la hora de diseñar las interfaces que componen la web. Es por ello que en él próximo capítulo, en el que se mostrará el desarrollo de las pantallas que componen la página web, se relacionará el diseño de las interfaces con algunos de los principios de usabilidad más relevantes que se acaban de definir.

4.4.2. Prototipos con Adobe XD

Para el diseño de las interfaces se ha utilizado Adobe XD[4], un programa de diseño de interfaces de usuario, centrada en la creación de prototipos interactivos para aplicaciones y sitios web.

A la hora de diseñar la interfaz web, aunque se puede desarrollar directamente desde el framework es una buena práctica generar antes unos bocetos o prototipos más manejables y menos costosos de modificar. Así, utilizando estos prototipos, se pueden realizar cambios y modificar la solución de manera más rápida y sencilla.

En la Figura 10 se puede ver un boceto realizado en las primeras etapas del desarrollo de la web donde se muestra la disposición de una pantalla de dispositivos.

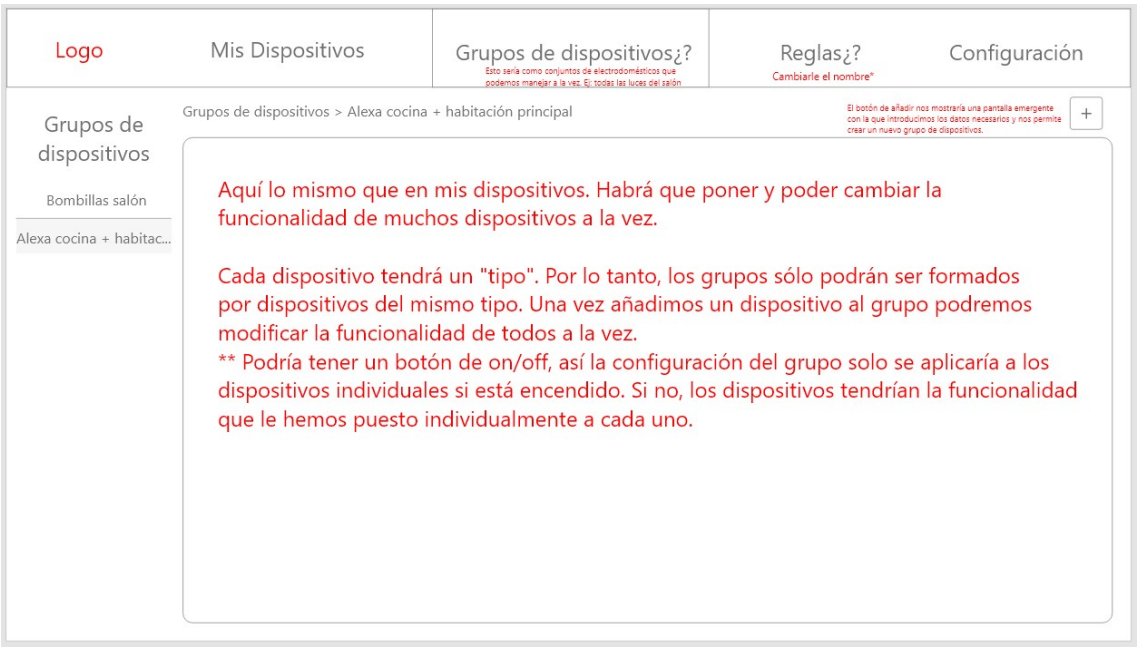

FIGURA 10: PRIMER BOCETO EN ADOBE XD DE LOS GRUPOS DE DISPOSITIVOS

Como veremos en el próximo capítulo con ejemplos del resultado final de algunas pantallas de la aplicación, algunas de las ideas de estos primeros bocetos se conservan en la solución final, mientras que otras se han ido cambiando para ofrecer una mejor interfaz para el usuario.

La Figura 11 también representa un boceto de otra parte de la aplicación, en este caso del dashboard. Se puede apreciar que es un boceto mucho más detallado que el de la Figura 10, con los elementos, sus características y su disposición claras.

[4] https://helpx.adobe.com/es/xd/get-started.html

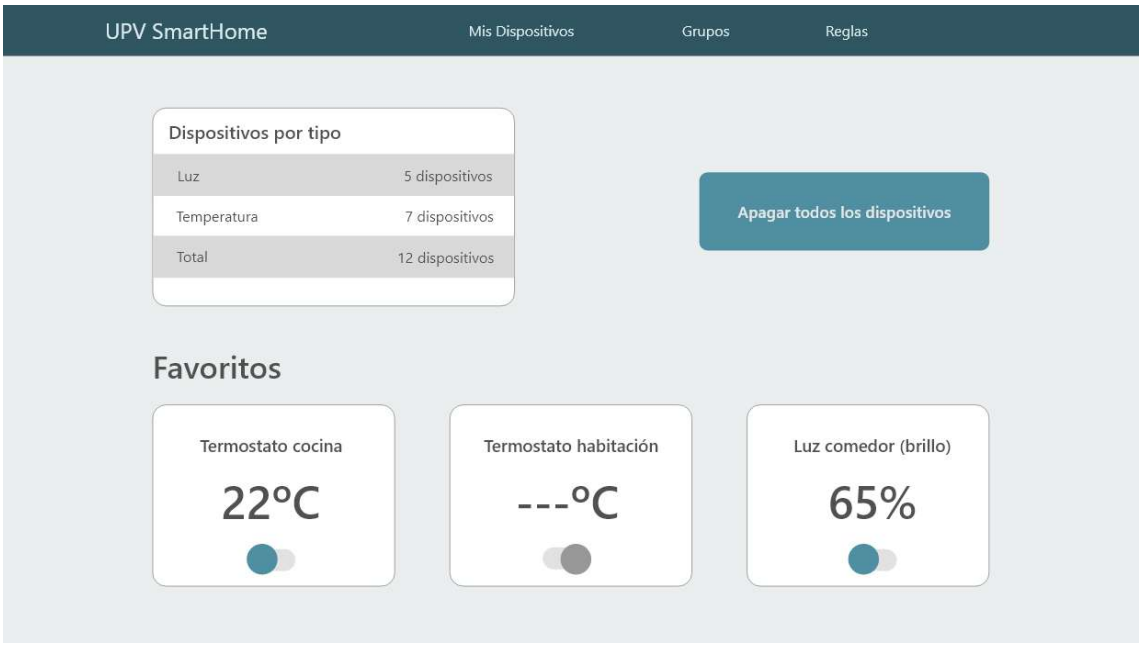

FIGURA 11: BOCETO INICIAL EN ADOBE XD DE LA INTERFAZ DEL DASHBOARD

Al igual que con el otro boceto y, pese a estar mucho más detallado, veremos que el resultado final del dashboard dista bastante de lo mostrado en el boceto de la Figura 11.

Esto se debe a que, en ocasiones, las necesidades cambian mientras se va desarrollando la aplicación. Por eso, el planteamiento de algunas pantallas puede cambiar, y tomar un camino totalmente distinto al que se pudo plantear en el análisis de necesidades.

5. Desarrollo de la solución

En este apartado se va a hacer un repaso por las pantallas más relevantes que componen la aplicación, y se mostrará cómo están construidas siguiendo el patrón Modelo-vista-template sobre el que trabaja Django.

5.1. Capa Modelo

En esta capa tenemos los componentes (llamados Modelos en Django) que implementan la lógica y la manera en la que interactúan los distintos objetos que forman la aplicación. Los modelos creados son guardados en una base de datos y son utilizados por las vistas, que recuperan los objetos necesarios mediante consultas a la base de datos.

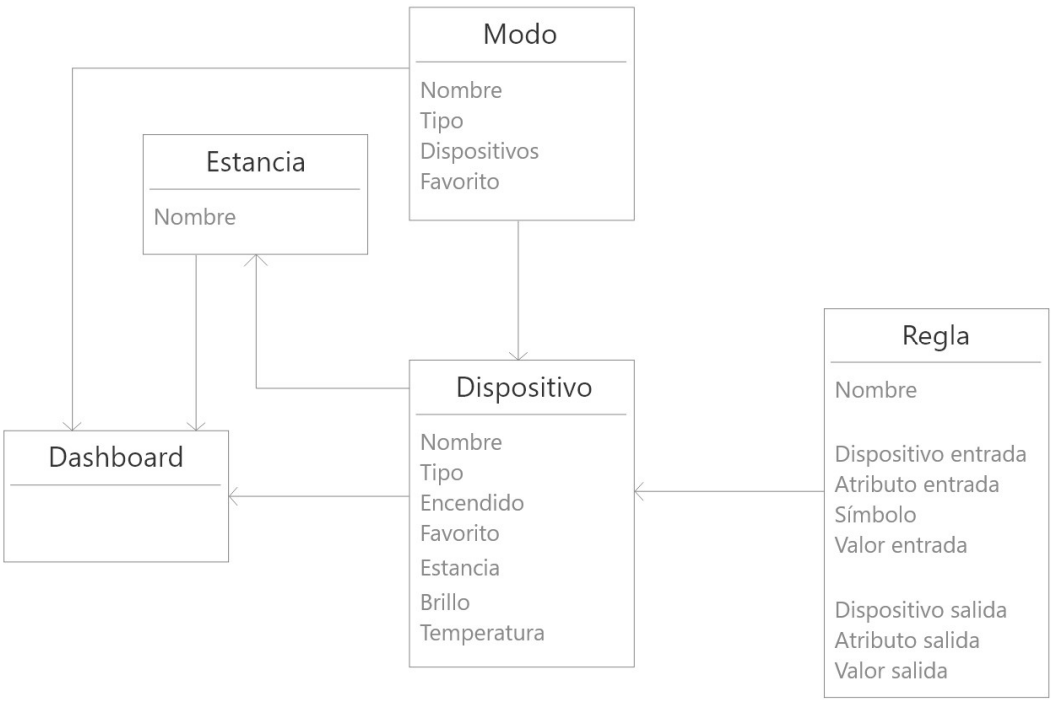

FIGURA 12: DIAGRAMA DE CLASES FINAL DE LA ESTRUCTURA DE LA APLICACIÓN

En la Figura 12 se puede ver el diseño de las clases o modelos que, con sus atributos y dependencias forman la aplicación.

En esta sección vamos a ver en profundidad cada uno de estos modelos.

5.1.1. Dispositivos

Como podemos ver en la Figura 13, el modelo “Device” (“Dispositivo”) cuenta con una serie de atributos o parámetros para su funcionamiento.

```python
class Device(models.Model):

    """..."""

    name = models.CharField(verbose_name='Nombre', max_length=35)
    type = models.CharField('Tipo', max_length=30, choices=GROUPS, default=1)
    power_on = models.BooleanField('Encendido', default=False, blank=True, null=True)
    favourite = models.BooleanField(verbose_name='Favorito', default=False, blank=True, null=True)
    room = models.ForeignKey(Room, null=True, verbose_name='Estancia', on_delete=models.SET_NULL)

    # Brillo
    brightness = models.IntegerField('Brillo', default=0, validators=[MaxValueValidator(100), MinValueValidator(0)], null=True)

    # Temperatura
    temperature = models.IntegerField('Temperatura', default=0, validators=[MaxValueValidator(70), MinValueValidator(-30)], null=True)
```

FIGURA 13: MODELO DE LOS DISPOSITIVOS EN DJANGO

Vamos a repasar el tipo y la funcionalidad que ofrece cada uno de estos atributos:

- **Nombre:** Es un atributo con el que el usuario nombra al dispositivo, es un texto plano que puede rellenar el usuario y no tiene restricción en el tipo de caracteres.
- **Tipo:** Este atributo se encarga de identificar al dispositivo por un tipo específico. De entre un listado cerrado de tipos, el valor que tenga este atributo indicará el tipo al cual pertenece el dispositivo.
- **Encendido:** Este atributo de tipo booleano representa si el dispositivo está encendido o apagado.
- **Favorito:** Es un atributo que solamente tiene uso en el dashboard. Un dispositivo podrá marcarse como favorito, y de ser así saldrá en el apartado de dispositivos favoritos en la pantalla principal.
- **Estancia:** Este atributo representa una clave foránea hacia el modelo “Room” (“Estancia”). Cada dispositivo estará asociado a una estancia, y como veremos en el apartado correspondiente, esta estancia representará la habitación en la que se encuentra el dispositivo.
- **Brillo y Temperatura:** Dependiendo del tipo de dispositivo uno de estos dos atributos estará vacío. En el caso de ser del tipo “Luz”, el dispositivo guardará el valor del brillo en el atributo “brightness”, mientras que si es del tipo “Temperatura” el atributo “temperatura” contendrá el valor que marca el termostato o el dispositivo en cuestión.

5.1.2. Modos

El modelo de los modos también contiene distintos atributos para funcionar, como podemos ver en la Figura 14.

```python
class Group(models.Model):

    name = models.CharField(verbose_name='Nombre', max_length=30)
    favourite = models.BooleanField(verbose_name='Favorito', default=False, blank=True, null=True)

    devices = models.ManyToManyField(Device, null=True)

    # Brillo
    brightness = models.IntegerField('Brillo (%)', default=0, validators=[MaxValueValidator(100), MinValueValidator(0)],
                                     null=True, blank=True)

    # Temperatura
    temperature = models.IntegerField('Temperatura (ºC)', default=0,
                                      validators=[MaxValueValidator(70), MinValueValidator(-30)], null=True, blank=True)
```

FIGURA 14: MODELO DE LOS MODOS EN DJANGO

Al igual que en el modelo de los dispositivos tiene un atributo “name” que contiene el nombre del grupo que el usuario ha creado. También contiene un atributo del tipo booleano (“favourite”) que representa si está marcado como favorito o no. Como la funcionalidad de estos atributos es muy similar al que ya hemos visto en el modelo de los dispositivos, no se van a definir en profundidad.

Sin embargo, a diferencia de en el modelo de los dispositivos ahora contamos con un atributo del tipo “ManyToManyField”, que hace referencia a otro modelo que ya tenemos creado.

Concretamente y, como se puede apreciar en la Figura 14, este atributo hace referencia al modelo “Device” (Dispositivo), y permite que un grupo tenga muchos dispositivos asociados y que un mismo dispositivo esté presente en diferentes grupos. Esa es la diferencia con el campo “ForeignKey” que hemos visto anteriormente, el cual solo permitía que un dispositivo se asociara a una única habitación.

Por otro lado también contamos con dos variables “brightness” (brillo) y “temperatura” (temperatura). Estos dos atributos representan el valor al que el modo actualizará sus atributos cuando se active.

5.1.3. Reglas

En cuanto a las reglas, su estructura interna requiere crear bastantes atributos, muchos de ellos relacionando objetos de otros modelos.

```python
class Rule(models.Model):

    SYMBOLS = [
        ("0", '='),
        ("1", '>'),
        ("2", '<')
    ]

    ATTRIBUTES = ["Brillo", "Temperatura"]

    name = models.CharField(max_length=30, default='')

    from_device = models.ForeignKey(Device, related_name='from_device', on_delete=models.CASCADE, null=True)
    from_attribute = models.CharField(max_length=30, default='')
    symbol = models.CharField('Simbolo', max_length=1, choices=SYMBOLS)
    from_value = models.CharField(max_length=30, default='')

    to_device = models.ForeignKey(Device, related_name='to_device', on_delete=models.CASCADE, null=True)
    to_attribute = models.CharField(max_length=30, default='')
    to_value = models.CharField(max_length=30, default='')
```

FIGURA 15: MODELO DE REGLAS EN DJANGO

Este modelo, como vemos en la Figura 15, es más complejo que lo anteriores así que lo desglosaremos en tres partes:

- **Atributos generales**

En este caso sólo requerimos un único campo “nombre” para identificar con una cadena de texto a la regla. Al igual que en los modelos vistos hasta ahora, cumple la misma función identificativa.

- **Atributos de la condición**

Aquí pertenecen los atributos que constituyen la parte de la condición de la regla. Ya que las reglas requieren para activarse que se genere una condición a partir de un atributo de un dispositivo, necesitaremos los siguientes atributos:

  - **from_device:** proporciona una referencia al dispositivo seleccionado en la regla.
  - **from_attribute:** es el atributo del dispositivo que selecciona el usuario y que se representará con una cadena de texto.
  - **symbol:** dependiendo de la condición podrá tomar el valor de “mayor que”, “menor que” o “igual que”.
  - **from_value:** representa el valor que tendrá el atributo del dispositivo para que se ejecute la regla.

- **Atributos del resultado**

Son similares a los atributos de la condición. En este caso no hace falta seleccionar un símbolo ya que el resultado se representa siempre con el signo igual (=), así que los atributos que necesitemos para generar el resultado de la regla serán:

- **to_device:** proporciona una referencia al dispositivo que ejecutará el cambio en su atributo una vez se active la regla.
- **to_attribute:** es el atributo sobre el cual se ejecutará el cambio de valor una vez se active la regla.
- **to_value:** representa el valor resultante que se le asignará al atributo del dispositivo una vez se realice la regla.

5.1.4. Estancias

Por su parte el modelo estancias es el más simple de todos. Este modelo solamente requiere un atributo que identifique a cada una de las estancias.

En cambio podemos ver que su utilidad reside en que, como hemos visto en uno de los apartados anteriores, los dispositivos tendrán una y sólo una estancia asociada, pero esta vinculación no se hace desde el modelo "Estancia", sino con un atributo del tipo "ForeignKey" desde el modelo "Dispositivo".

```
class Room(models.Model):

    name = models.CharField(verbose_name='Nombre', max_length=30, default='')

    def __str__(self):
        return self.name
```

FIGURA 16: MODELO DE ESTANCIAS EN DJANGO

Como se puede apreciar en la Figura 16, en este modelo hemos generado una función "__str__(self)". Al estar relacionado con los dispositivos mediante una clave foránea, si no se especificase en esta función que devolviese el nombre de la estancia, cuando intentásemos acceder a las diferentes estancias desde los dispositivos veríamos por pantalla el id de la estancia, en vez del nombre, ya que es como por defecto está configurado.

5.2. Capas Vista y Template

Siguiendo de nuevo el patrón "Modelo-vista-template" hasta ahora hemos analizado la función que cumple el modelo y cómo está gestionado en nuestra aplicación, así que en este apartado vamos a profundizar en otro de los componentes de este patrón MVT: la capa de las vistas y la capa de las templates.

Ya hemos visto cuando se ha explicado el patrón Modelo-Vista-Template que existe una conexión entre la vista y la template.

En concreto las vistas recogen los datos y la información que posteriormente la template se encarga de mostrar por pantalla mediante HTML, CSS y JavaScript. Teniendo en cuenta el vínculo que hay entre estos dos componentes vamos a hacer un repaso por las pantallas de la aplicación explicando cómo están construidas a tanto a nivel de vista y como de template.

5.2.1. Navegador

El navegador es un elemento común a todas las vistas de la aplicación. Está situado en la parte superior de la pantalla y se ocupa de servir como acceso directo para cambiar entre módulos.

Capa Template

A nivel de la template, podemos ver en la Figura 17 que el navegador se compone en dos bloques.

En la parte izquierda se muestra el nombre de la aplicación (o el logo en el caso de tenerlo), que sirve como acceso directo para que el usuario vuelva a la pantalla principal de la aplicación (en nuestro caso al dashboard).

En la parte derecha aparece el listado de módulos que componen la aplicación, cada uno con su respectivo acceso directo al módulo que corresponde.

Cabe apuntar que el nombre del módulo en el que se encuentra el usuario cambiará de color para que el usuario pueda situarse fácilmente en la aplicación en todo momento.

FIGURA 17: INTERFAZ DEL NAVEGADOR

Como se ha hablado en un apartado anterior las pantallas que componen la aplicación están diseñadas siguiendo los principios de usabilidad descritos por Jakob Nielsen. Uno de los principios más importantes que se aplica sobre el navegador es el de la **visibilidad del sistema**.

El navegador se sitúa en la parte superior de todas las pantallas que componen la web, por lo tanto cumple con este principio ya que indica permanentemente el lugar en el que se encuentra el usuario.

Otro del os principios que cumple es el de **control y libertad del usuario**. Al aparecer el navegador en cualquier pantalla el usuario puede en todo momento cambiar del módulo o volver a la pantalla inicial.

Capa Vista

En este caso el navegador no recibe ninguna información por parte de la vista para su funcionamiento, ya que lo que contiene es información estática que se escribe directamente desde la template.

5.2.2. Breadcrumbs

Los breadcrumbs [7] o migas de pan están presentes en la mayoría de vistas de la aplicación, y se pueden definir como una referencia a la pantalla en la que está situado actualmente el usuario dentro de la jerarquía de la web.

Concretamente cumplen dos objetivos importantes:

- **Mejoran la navegación del usuario**, ya que actúan como accesos directos entre las páginas de los módulos.
- **Marcan una jerarquía y una ruta de navegación**, haciendo que el usuario sepa cual es el nivel y posicionamiento de la pantalla en la que se encuentra respecto a la jerarquía del módulo.

Capa Template

Como se puede ver en la Figura 18 los breadcrumbs están situados en la parte superior del contenido de la pantalla y están formados por los accesos directos a las páginas que conforman la jerarquía del módulo.

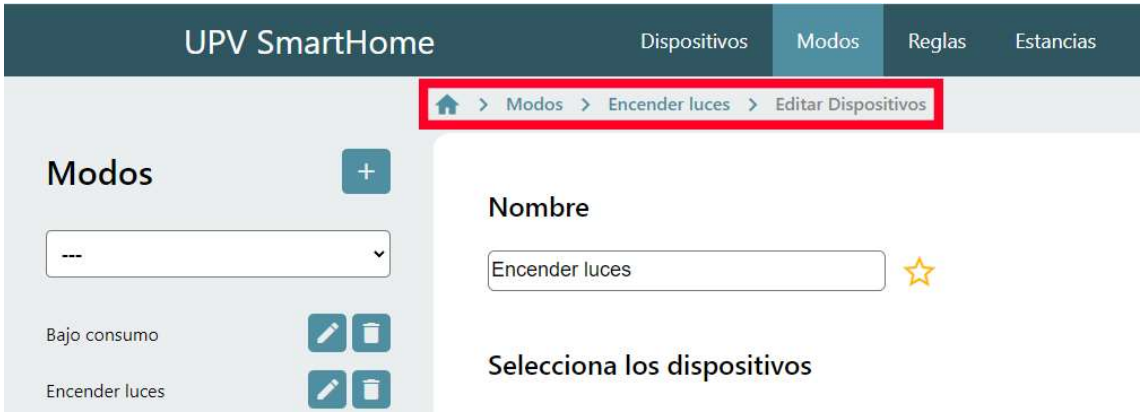

FIGURA 18: BREADCRUMBS O MIGAS DE PAN

Se puede apreciar que en los breadcumbs el enlace que está más situado a la derecha tiene un color gris, dando a entender que es la pantalla en la que se encuentra actualmente el usuario y sólo puede navegar hacia las pantallas anteriores.

Por otro lado los dos primeros enlaces tienen un color azul, y sirven cada uno como acceso directo a las pantallas que se encuentran en la jerarquía del módulo por debajo de la pantalla actual.

En cuanto a algunos de los principios de usabilidad más relevantes presentes en los breadcrumbs se puede destacar, al igual que en el navegador, la **visibilidad del estado del sistema** y el **control y libertad de usuario**.

Dado que cumple con un papel similar al del navegador es lógico que se apliquen prácticamente los mismos principios en los dos, ya que los dos casos sirven para mostrar al usuario el lugar en el que se encuentra y sirven también como una "vía de escape" a la que el usuario puede recurrir si necesita cambiar de pantalla.

Capa Vista

La Figura 19 representa los datos que la vista le envía a la template. Como podemos ver resaltado se envía un diccionario (llamado "breadcrumbs") que contiene los datos de los breadcrumbs que la template se encargará de gestionar y mostrar por pantalla.

```
context = {'form': form,
           'groups': groups_list,
           'groups_dict': {},
           'devices_dict': devices_dict,
           'devices_selected_dict': devices_selected_dict,
           'devices_unselected_dict': devices_unselected_dict,
           'breadcrumbs': {group.name: group.id, 'Editar Dispositivos': str(group.id) + '/devices'},
           'types_groups_dict': types_groups_dict,
           }

return render(request, 'UPVSmartHome/add_group.html', context=context)
```

FIGURA 19: VISTA EN DJANGO QUE ENVÍA LOS DATOS A UNA TEMPLATE

La template será recibe estos datos e itera sobre cada una de las claves del diccionario. Dependiendo del valor que tenga en cada una de esas iteraciones creará los elementos en HTML correspondientes, como se aprecia en la Figura 20, dando forma en su conjunto a los breadcrumbs.

```
<div class="top">
    <a class="material-icons" href="/">home</a>
    <span class="material-icons" style="color: #508fa1">chevron_right</span>
    <a href="/groups">Modos</a>
    {% for name, id in breadcrumbs.items %}
        <span class="material-icons" style="color: #508fa1">
            chevron_right
        </span>
        <a href="/groups/{{id}}">{{name}}</a>
    {% endfor %}
</div>
```

FIGURA 20: TEMPLATE EN DJANGO QUE ITERA SOBRE ELEMENTOS RECIBIDOS DESDE UNA VISTA

5.2.3. Seleccionar elemento

La pantalla de selección de un elemento muestra al usuario el listado de los elementos del modelo que tiene creados en la aplicación y le proporciona herramientas para gestionarlos de manera rápida o crear nuevos.

Capa Template

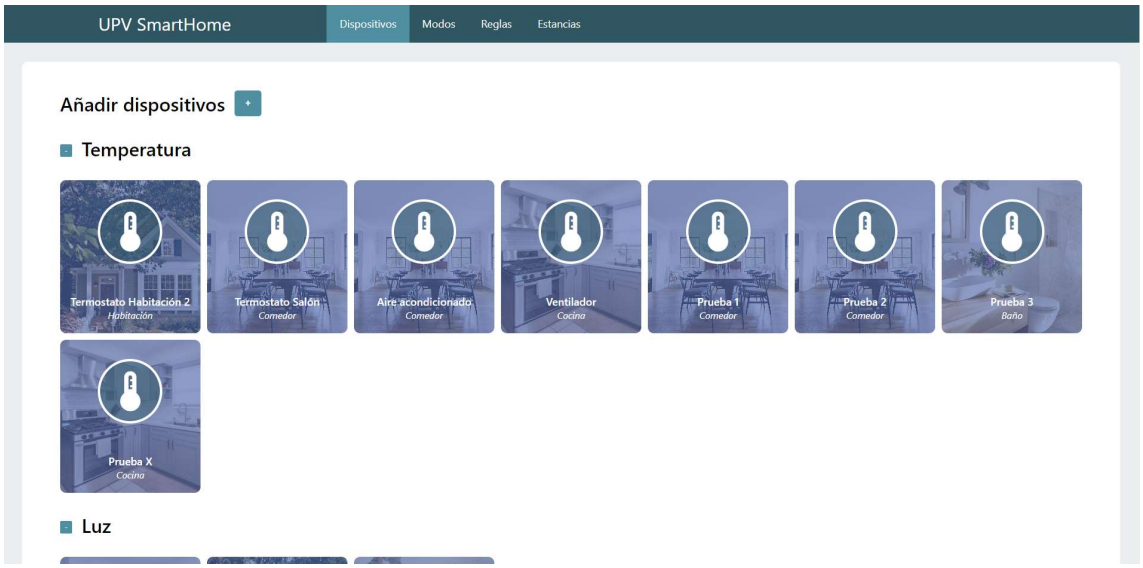

FIGURA 21: PANTALLA DE SELECCIÓN DE DISPOSITIVOS

Como podemos ver en la Figura 21 vamos a poner como ejemplo la pantalla de selección de los dispositivos, aunque comparte estructura con el resto de pantallas de selección en los distintos módulos.

A nivel de la template en la parte superior del contenido de la pantalla hay un botón para añadir dispositivos. Justo debajo se listan los dispositivos que el usuario tiene creados, separados por tipo. Además, estas listas se pueden contraer o expandir pinchando en el botón que hay a la izquierda de los títulos de los tipos de dispositivo.

Esta funcionalidad es útil sobre todo cuando la aplicación cuenta con numerosos dispositivos, ya que aunque estén filtrados según el tipo, el usuario puede ocultar los tipos que no le interesan para tener aún una mayor facilidad a la hora de buscarlos.

Estos dispositivos que hay listados se muestran en forma de tarjetas. Cada una de las tarjetas cuenta con un icono en la parte superior que representa el tipo de dispositivo, el nombre con el que está guardado ese dispositivo y el tipo de estancia en el que está guardado.

Este tipo de estancia está representado tanto de manera escrita (debajo del nombre) como con la imagen que hay en la tarjeta. De esta manera el usuario puede distinguir de manera sencilla y rápida la estancia en la que se encuentra el dispositivo, para una mayor facilidad a la hora de buscarlos.

FIGURA 22: VISTAZO DE UNA TARJETA QUE CONTIENE LOS DISPOSITIVOS EN LA PANTALLA DE SELECCIÓN DE DISPOSITIVOS

Finalmente estas tarjetas cuentan con las funciones de editar y eliminar el dispositivo que contienen. Aunque en un principio no se encuentran visibles, es en el momento en el que el usuario pasa el ratón por encima de la tarjeta cuando aparecen los dos botones que realizan estas funciones, como podemos ver en la Figura 22.

Sobre algunos de los principios de usabilidad presentes en esta pantalla podemos destacar **reconocer antes que recordar**. La forma en la que están construidas las pantallas de selección de elementos es muy similar. Además cuentan con elementos que el usuario puede ver en otras pantallas, como son los botones de editar y eliminar un elemento presentes en las configuraciones de los elementos.

También podemos destacar el principio **coincidencia entre el sistema y el mundo real**, ya que se utilizan iconos fácilmente reconocibles por el usuario. Cuando por ejemplo un dispositivo es del tipo “Temperatura” sale un termostato como icono en la tarjeta. A su vez, dependiendo de la estancia en la que se encuentre el dispositivo aparecerá en el fondo de la tarjeta una imagen de la estancia en cuestión.

Capa Vista

Esta pantalla requiere para funcionar enviar desde la vista el listado de dispositivos, separados ya por tipo.

En primer lugar, desde la misma vista, se hace una petición a la base de datos para recueprar todos los registros del tipo “Dispositivo” que haya guardados. Una vez recuperados se filtran, se separan por tipo y se le envían a la template.

La template será entonces la encargada de crear cada una de estas tarjetas con los datos y los enlaces correspondientes por cada elemento en el listado de dispositivos.

5.2.4. Añadir o configurar elementos

Todos los modelos que hemos creado para el funcionamiento de la aplicación permiten crear, modificar o eliminar registros de la base de datos.

Tanto la pantalla de creación de un nuevo elemento como la pantalla de modificación de un elemento existente se comportan prácticamente igual, tanto a nivel de template como a nivel de vista, así que vamos a explicar cómo funcionan y las diferencias que tienen.

Capa Template

Como hemos adelantado, la pantalla de creación y la pantalla de edición de un registro comparten muchas similitudes. Al nivel de template la funcionalidad es igual, con la única diferencia de que en el caso de la edición de registros los campos que se ven en el formulario de la pantalla están rellenos con los datos que tiene el objeto guardado en la base de datos.

Teniendo en cuenta esta simple diferencia, vamos a repasar las partes que componen estas pantallas tomando como ejemplo la template de creación de un dispositivo.

En la Figura 23 podemos ver la pantalla de creación de un elemento, en este caso del modelo “Dispositivo”. Se puede dividir la pantalla en dos bloques: el bloque izquierdo y el bloque derecho.

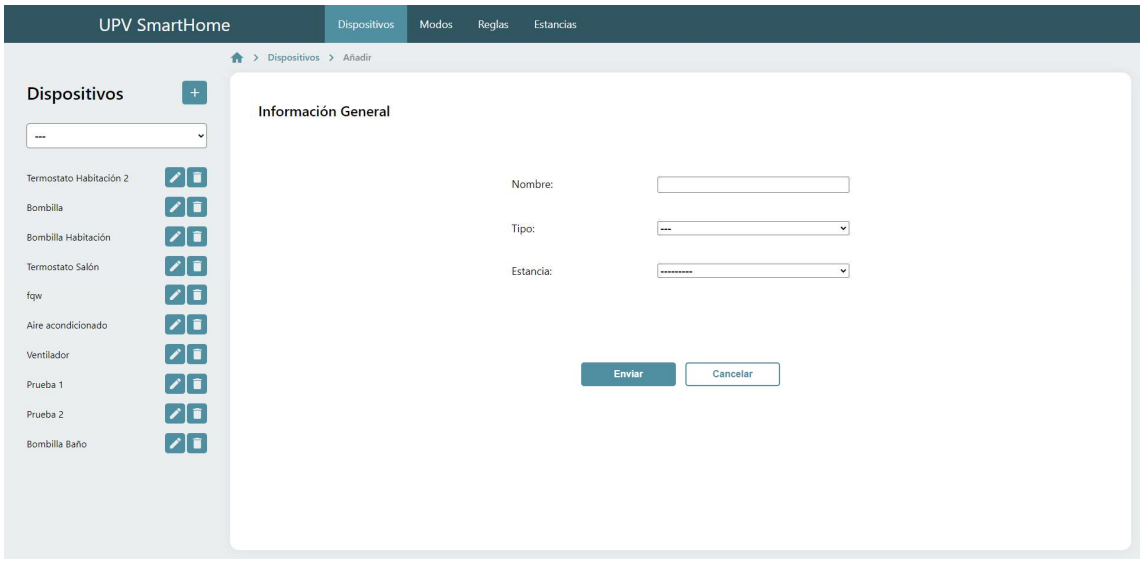

FIGURA 23: PANTALLA DE CREACIÓN DE UN DISPOSITIVO

El bloque izquierdo contiene la lista de elementos que pertenecen al módulo en el que está situado el usuario. Estos elementos pueden ser filtrados mediante el selector que hay encima de la lista (En el caso de los dispositivos el desplegable permite filtrar al usuario los dispositivos por tipo). A su vez cada elemento en la lista contiene dos botones que, como ya hemos visto de manera similar en otros apartados, permiten tanto editar como eliminar un elemento concreto.

El bloque derecho por su parte contiene los datos y los campos que tiene que rellenar el usuario para la creación del objeto. Una vez se pinche sobre el botón de enviar, y si todos los campos tienen valores correctos se creará un nuevo registro en la base de datos.

En el caso de introducir datos erróneos o no introducir un valor en uno de los campos, el formulario no permitirá la creación del objeto y mostrará por pantalla un mensaje pidiendo al usuario que rellene la información correctamente, mostrando ayudas sobre qué es lo ha introducido incorrectamente en el formulario y cómo puede solucionarlo.

Estos mensajes de error forman parte de uno de los principios de usabilidad más importantes de los que se pueden apreciar en esta pantalla: la **recuperación ante errores**.

Capa Vista

Lo que respecta a la parte de las vistas, la template creará el listado de dispositivos que se muestra en la parte izquierda de la pantalla mediante un diccionario clave-valor que

recibe desde la vista, que contendrá toda la información necesaria para crear el listado de dispositivos.

Por otro lado, la parte del formulario situada a la derecha de la pantalla funciona de manera diferente.

Los formularios para crear o modificar un elemento en Django se gestionan mediante un archivo de formularios específico (“forms.py), que contiene las clases formulario que se van a utilizar en la aplicación.

```python
class AddDeviceForm(forms.ModelForm):

    def __init__(self, *args, **kwargs):

        super(AddDeviceForm, self).__init__(*args, **kwargs)

        self.initial['type'] = '0'

    def clean_name(self):
        name = self.cleaned_data['name']
        return name

    class Meta:
        model = Device
        fields = ['name', 'type', 'room']
```

FIGURA 24: FORMULARIO DE CREACIÓN DE DISPOSITIVOS

En la Figura 24 vemos cómo está configurado el formulario de añadir un dispositivo, que está compuesto por distintas partes:

- El código que se ejecuta cuando se carga el formulario (“__init__()”): Esta parte del código compone las acciones que se ejecutarán sobre los campos del formulario una vez se cargue por pantalla. Cosas como deshabilitar un campo o cambiarle el valor a otro a la hora de cargar la página son las que permite esta función.
- Las funciones que proporcionan las restricciones sobre los atributos (“clean_name()”): Estas funciones se crean para comprobar que los valores que se les ha asignado a los atributos a la hora de guardar un formulario son correctas. En el caso de serlo el formulario guardará el objeto en la base de datos, pero en el caso de tener un campo con un valor incorrecto se mostrará un error por pantalla y se pedirá al usuario que introduzca bien los datos.
- El modelo al que hace referencia un formulario (“class Meta”): Establece el modelo sobre el que realizará las acciones el formulario, en este caso el modelo

de los dispositivos. Además se ha de especificar los campos exactos que se van a mostrar por pantalla para rellenar. Como se puede ver en la figura 24 el campo “fields” hace que se muestren en el formulario tres de los atributos que tiene el modelo, por lo tanto el objeto que se creará cuando se guarde el formulario asignará los valores que haya introducido a estos tres atributos mientras que dejará el resto de atributos que tiene el modelo y no se muestran en el formulario en blanco (asumiendo que ninguno de ellos es obligatorio).

Teniendo en cuenta las partes que componen el formulario de añadir un dispositivo, desde la vista se envía a la template este mismo formulario para que lo imprima por pantalla.

5.2.5. Dashboard

Como ya hemos hablado en el análisis del problema el dashboard [8] sirve para que, de un vistazo rápido, el usuario obtenga la mayor cantidad de información relevante posible y de la manera más sencilla para entenderla.

Capa Template

Siguiendo esa premisa se ha dividido el dashboard en tres bloques diferentes, cada uno con un propósito distinto y mostrando datos o acciones diferentes.

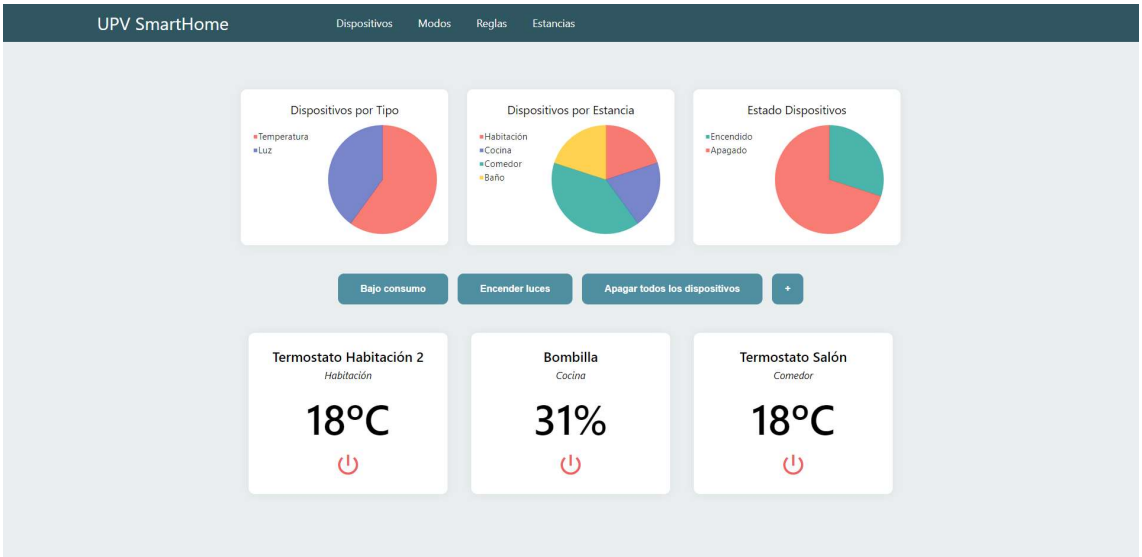

FIGURA 25: INTERFAZ DE LA PANTALLA DEL DASHBOARD

En la Figura 25 se pueden apreciar los tres bloques mencionados: el bloque superior que muestra los datos globales de la aplicación en forma de tabla o de gráfica, el bloque central que permite al usuario activar y crear modos de manera rápida y sencilla y el

bloque inferior, que permite al usuario visualizar y controlar los dispositivos que tiene marcados como favoritos de manera simple.

Vamos a repasar una a uno el cómo están construidos cada uno de estos bloques:

- **Bloque superior**: En este bloque se ha optado por la inclusión de los datos en forma de gráficas, ya que son la manera más visual y fácilmente entendible de mostrar los datos en nuestra aplicación.
- **Bloque central**: Por otro lado en el bloque central contamos con los accesos directos para ejecutar los modos que hemos creado. Una vez el usuario pincha en uno de los modos se actualizarán al instante los valores de los dispositivos asociados. Asimismo, contamos además con un botón (“+”) que sirve como acceso directo para añadir un nuevo modo.
- **Bloque inferior**: Finalmente debajo del todo contamos con el bloque que muestra los dispositivos que tenemos marcados como favoritos. Desde esas tarjetas podemos ver el estado de los atributos de nuestros dispositivos favoritos y apagarlos o encenderlos de manera sencilla y rápida.

En el caso del dashboard hay algunos principios de usabilidad dignos de mención. En primer lugar el **diseño estético y minimalista**. Aunque toda la aplicación intenta seguir este principio, el dashboard compone la página principal de la aplicación, por lo que será una de las pantallas más visitadas y utilizadas por el usuario.

Además, como ya hemos hablado en la funcionalidad que debe ofrecer esta pantalla, el usuario ha de recibir la información de la manera más clara y visual posible, por lo que utilizando herramientas como gráficas o diseñando los elementos que componen el dashboard de la manera más atractiva y simple posible se consigue una pantalla limpia y estética.

Por otro lado otro de los principios de usabilidad más importantes en esta pantalla es la **flexibilidad y eficiencia de uso**. En este caso la pantalla cuenta con muchos atajos para realizar acciones, como puede ser activar o crear un modo o encender y apagar los dispositivos.

Capa Vista

A nivel de vista el dashboard recibe datos de la misma forma que hemos visto que lo hacen las anteriores pantallas. Tanto para mostrar la lista de los dispositivos favoritos como para mostrar los accesos directos para activar los modos, se le envía a la template desde la vista un diccionario con los dispositivos y los modos, y es la propia template la que itera sobre las claves del diccionario y genera el HTML teniendo en cuenta los valores de cada iteración.

En cambio la implementación de las gráficas se ha hecho de manera diferente, ya que se ha tenido que recurrir a la utilización de la librería externa “Pygal”. Esta librería permite la creación de manera sencilla de gráficas de todo tipo.

Para su utilización se ha tenido que instalar primero la librería en el framework. Una vez instalado tecleando en la consola el comando “import pygal” es posible acceder a sus funcionalidades.

Como podemos ver en la Figura 26 se ha creado un objeto “Pie” (es el tipo de gráfica que queremos mostrar, en nuestro caso una gráfica en forma circular). Ese objeto representa el objeto gráfica que posteriormente se va a mostrar por pantalla.

```
import pygal

custom_style = Style(
    legend_font_size=35,
    tooltip_font_size=35,
    title_font_size=45,
    font_family='system-ui',
    background='white'
)

devices_chart = pygal.Pie(title='Dispositivos', style=custom_style, spacing=50)

for key, value in swap_dict.items():
    if key[0] != 'Total':
        devices_chart.add(key[0], value)

devices_chart_object = devices_chart.render_data_uri()
```

FIGURA 26: VISTA DEL DASHBOARD DONDE SON GENERADAS LAS GRÁFICAS

Una vez con el objeto de la tabla creado hay que rellenarla con los valores que debe mostrar. Para ello, se itera sobre la lista o el diccionario que contiene los objetos a mostrar y mediante la función “devices_chart.add()” la tabla va añadiendo los valores con los que se rellenará.

```
<div class="charts-module">
    <embed type="image/svg+xml" src="{{ devices_chart|safe }}" width="350"/>
    <embed type="image/svg+xml" src="{{ rooms_chart|safe }}" width="350"/>
    <embed type="image/svg+xml" src="{{ devices_power_chart|safe }}" width="350"/>
</div>
```

FIGURA 27: IMPLEMENTACIÓN DE LAS GRÁFICAS EN LA TEMPLATE DEL DASHBOARD

Finalmente la vista envía el objeto de la tabla que hemos creado a la template, que se encarga de mostrarlo por pantalla como se ve en la Figura 27. Lo hace utilizando la etiqueta HTML “embed”, que se encarga de incrustar en la pantalla el elemento externo que se le pase, en nuestro caso los archivos de las tablas.

5.2.6. Crear y configurar reglas

La pantalla de creación y la de configuración de las reglas comparte la misma plantilla. La diferencia entre las dos es que la pantalla de configurar reglas tendrá asociado el id de la instancia de la regla a modificar. Esto hará que cuando se cargue la pantalla se rellenen los campos con los datos que tiene guardada en la base de datos mientras que cuando se crea una regla nueva los campos aún están por rellenar.

Como ya sabemos las reglas componen la lógica que será aplicada a unos dispositivos una vez se cumpla una condición.

Capa Template

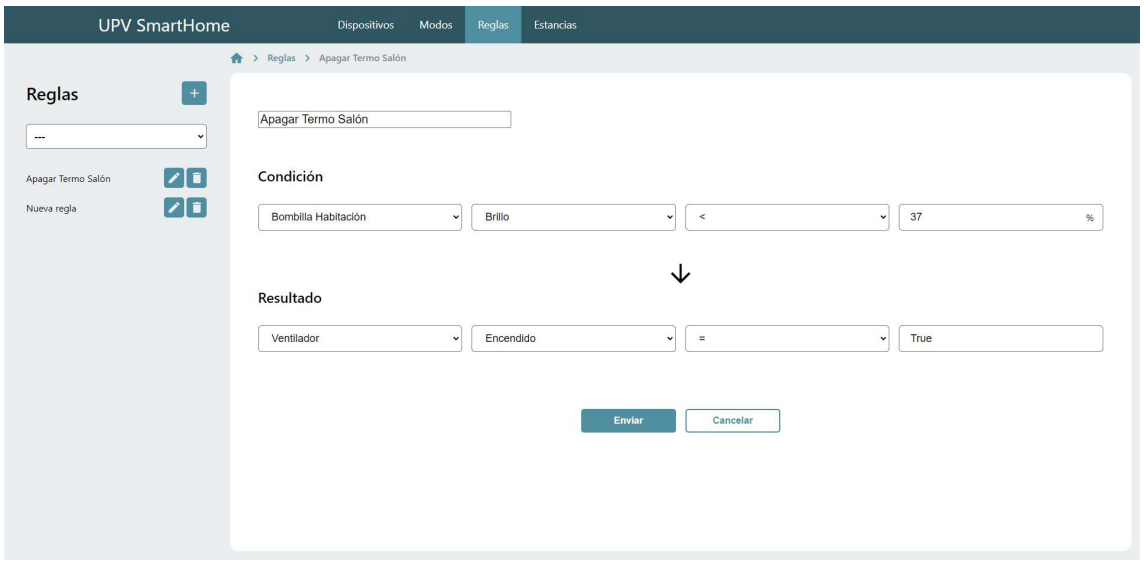

FIGURA 28: PANTALLA DE MODIFICACIÓN DE UNA REGLA

En la Figura 28 vemos la pantalla de configuración de las reglas, que como se puede apreciar es un poco más compleja que las pantallas de configuración o creación del resto de modelos, que se componen solamente por un formulario simple.

Esta pantalla se compone, como los bloques que ya hemos mencionado, de un listado de reglas a la parte izquierda de la pantalla y de un contenedor dinámico a la derecha.

El listado de la izquierda contiene la totalidad de las reglas que el usuario tiene en la aplicación, y le permite añadir, editar o eliminar estas mismas reglas.

Por otro lado el contenedor de la derecha contiene un formulario, solo que ha sido modificado para que se más intuitivo para el usuario. El formulario está compuesto por un campo que representa el nombre de la regla, arriba del todo, y la lógica sobre la que funcionará la regla separada en dos bloques: el bloque superior que compone la lógica de la condición de la regla y el bloque inferior que compone la lógica del resultado que se aplicará una vez se active.

En cuanto al bloque superior o bloque de la condición el usuario puede seleccionar uno de los dispositivos que tenga añadidos en la aplicación y acceder a uno de sus parámetros. En el caso de la Figura 28 se accede al dispositivo llamado “Bombilla Habitación” y concretamente a su atributo “Brillo”.

Una vez seleccionado el atributo que compondrá la condición de la regla el usuario tendrá que indicar el valor que deberá tener para que se cumpla la esta condición y por lo tanto se active y se ejecute la lógica que va asociada al resultado. En el caso de la Figura 28 el atributo “Brillo” que se ha indicado previamente activará la regla en el caso de tener un valor menor al 37%.

Por otro lado la sintaxis para el bloque inferior o bloque del resultado es la misma que en la condición. El usuario seleccionará un dispositivo y un atributo sobre el cual se aplicará el resultado de la regla. En la Figura 28, el dispositivo y el atributo que cambiará será el encendido / apagado del dispositivo “Ventilador”.

Como ya se ha visto con la condición, una vez el usuario ha seleccionado un dispositivo y el atributo le dará el valor al que se actualizará una vez se ejecute la regla. En el caso de la Figura 28 el atributo “Encendido” del dispositivo “Ventilador” será verdadero, es decir, que encenderá el dispositivo.

En referente a los principios de usabilidad utilizados en esta pantalla se podría destacar la **prevención de errores**, dado que en este formulario el usuario puede escribir valores que el sistema no pueda interpretar. Para ello y, como hemos visto en alguno de los formularios, cuando el sistema interpreta que el valor introducido por el usuario no es correcto, muestra por pantalla un aviso para que el usuario cambie ese valor, aportando información y ayudas de cómo puede solucionarlo, como se puede ver en la Figura 29.

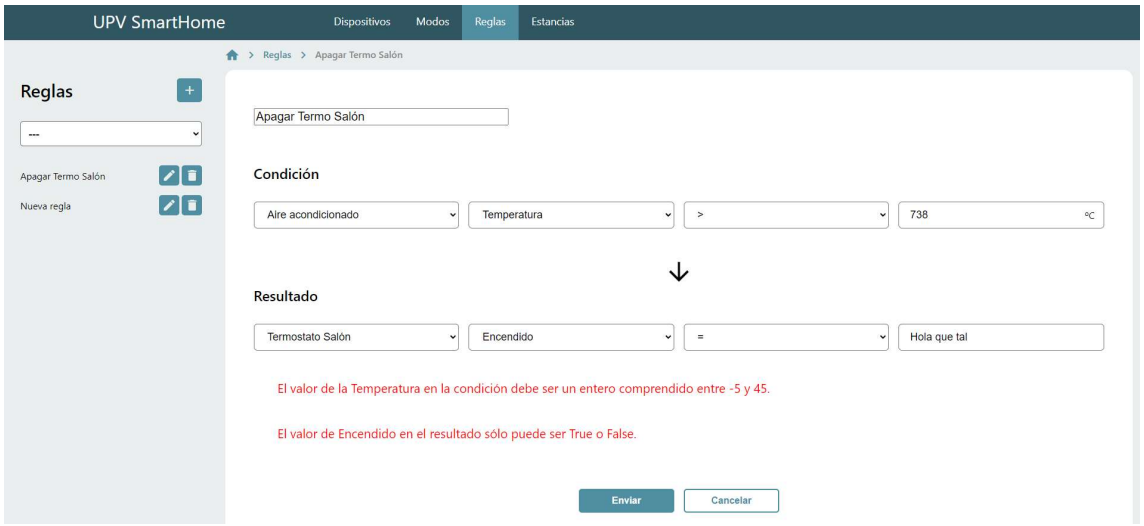

FIGURA 29: PANTALLA DE MODIFICACIÓN DE UNA REGLA CON INTRODUCCIÓN DE DATOS INCORRECTOS

Por otro lado otro de los principios de usabilidad que se podrían destacar de esta pantalla es el **diseño estético y minimalista**, ya que al contener las reglas una funcionalidad relativamente compleja la decisión final sugería dejar la pantalla lo más simple y entendible posible.

Capa Vista

Los datos que la vista envía a la template son similares a los que hemos visto en pantallas anteriores. Para mostrar la lista de reglas que hay en la parte izquierda de la pantalla, desde la vista se recuperarán de la base de datos todas las reglas y los atributos necesarios. Entonces la template será la encargada de formar el listado utilizando elementos HTML.

Por otro lado la parte derecha funciona con un formulario. Como se puede ver en la Figura 30 el formulario está relacionado con el modelo de las reglas y muestra los campos por pantalla. La parte interesante de este formulario es la cantidad de lógica y restricciones que han de aplicarse.

```
class AddRuleForm(forms.ModelForm):

    def __init__(self, *args, **kwargs):...

    def clean_name(self):...

    def clean_from_attribute(self):...

    def clean_from_value(self):...

    def clean_to_value(self):...

    def save(self, commit=True):...

    class Meta:

        model = Rule
        fields = ['name', 'from_device', 'from_attribute', 'symbol', 'from_value',
                  'to_device', 'to_attribute', 'to_value']
```

FIGURA 30: FORMULARIO DE CREACIÓN DE REGLAS

Hasta ahora se no se ha necesitado unas restricciones muy complejas para validar los valores que tienen los atributos de un formulario.

En cambio y, como se puede ver en la Figura 31, las reglas tienen unas restricciones que no permiten que el usuario guarde la regla si rellena los campos con valores incorrectos.

Cosas como salirse del rango que puede tener un valor como el porcentaje de brillo de una bombilla o restringir que una variable “Encendido” sólo pueda tener el valor “Verdadero” o “Falso” son ejemplos de restricciones que, cuando el usuario guarde la regla con un valor incorrecto, harán mostrar por pantalla el error asociado y no permitirán que se modifique o se cree la regla.

```python
def clean_from_value(self):
    from_value = self.cleaned_data['from_value']
    from_attribute = self.cleaned_data['from_attribute']

    if from_attribute == 'Encendido':
        if from_value == 'True' or from_value == 'False':
            pass
        else:
            raise ValidationError('El valor de Encendido en la condición sólo puede ser True o False.')

    if from_attribute == 'Brillo':
        try:
            from_value = int(from_value)
        except:
            pass
        if not isinstance(from_value, int) or from_value < 0 or from_value > 100:
            raise ValidationError(
                'El valor del Brillo en la condición debe ser un entero comprendido entre 0 y 100.')
    if from_attribute == 'Temperatura':...

    return from_value
```

FIGURA 31: FUNCIÓN "CLEAN_FROM_VALUE" EN EL FORMULARIO DE AÑADIR UNA REGLA

6. Conclusiones

El desarrollo de la aplicación no ha generado muchos problemas. Al haber hecho previamente algunos proyectos con el framework y los lenguajes de programación tenía una base sobre la que comenzar a trabajar.

Pese a nunca haber hecho un proyecto web completamente de cero los conocimientos con el framework y el patrón MVT hicieron que el desarrollo fuese relativamente rápido. Todos los conceptos de vistas, templates, modelos y como interactúan unos con otros eran cosas que ya tenía un poco de idea de cómo funcionaban, así que no tuve que "perder" el tiempo aprendiendo sus bases.

Por otro lado y, aunque el desarrollo no presentó muchos inconvenientes, sí que surgieron algunos problemas que hicieron cambiar y realizar algunos ajustes a mitad del desarrollo.

Aunque realmente se ha conseguido implementar prácticamente toda la funcionalidad que se identificó en el análisis de requisitos, es cierto que algunos módulos han tenido que sufrir cambios o recortes en cuanto a funcionalidad, algunos de los cuales vamos a nombrar a continuación:

- **Pantallas modales**

El diseño inicial de la pantalla de creación de objetos en la aplicación, ya sea crear un dispositivo, una regla o un modo, entre otros, planteaba que cuando el usuario fuera a pinchar sobre el botón de crear un nuevo objeto le apareciese una pantalla por encima de la actual que le permitiese crear el objeto.

Esta idea se desechó dada la complejidad que tenía el juntar los formularios, la validación de campos y la creación de los objetos con las pantallas modales, así que se decidió optar por la solución actual. Ahora, cuando el usuario pincha sobre el botón de crear un nuevo registro es redirigido a una pantalla diferente para la creación del objeto, en vez de aparecer en una pantalla modal por encima de la actual.

- **Sistema de usuarios y configuración**

En un principio se planteó la idea de generar un sistema de creación y gestión de usuarios. En la aplicación habría un apartado de perfiles donde diferentes personas podrían crear un usuario. Aunque todos los usuarios estarían vinculados a la misma vivienda (así que compartiría con el resto de usuarios los registros creados, modificados y eliminados) cada uno tendría una configuración y unas preferencias distintas. Por ejemplo uno podría tener unos dispositivos y unos modos favoritos diferentes al resto, o podría seleccionar las unidades (como puede ser elegir la temperatura en grados Celsius o Kelvin) que se mostrasen en la aplicación, entre otras funcionalidades.

Aunque era un concepto interesante se terminó desechando tanto por la complejidad que podría llevar como por la función final que tendría. Se valoró que no había tantos parámetros que los usuarios querrían cambiar como para crear un sistema tan complejo de perfiles.

- **Mejorar el sistema de reglas**

Aunque la creación y configuración de reglas está hecha de la manera más intuitiva posible para el usuario hay mejoras que podrían facilitar la experiencia.

Cambiar de manera dinámica el tipo de los inputs dependiendo del tipo del atributo (Ya sea una cadena de texto, numérico, verdadero o falso, etc.) es una cosa que no se ha podido hacer. Al ser una estructura de formulario compleja de por sí hacer cambios sobre ella implicaba mucha complejidad.

Otra cosa que no se ha podido implementar en las reglas es poder añadir más de una condición y resultado. Ahora mismo cada regla se activa cuando se cumple una sola condición y ejecuta un solo resultado, aunque se pensó a mitad del desarrollo poder añadir más condiciones y más resultados.

- **Actualizar distintos atributos en los modos**

En cuanto a los modos, ahora mismo se puede seleccionar un grupo de dispositivos de entre todos los que hay creados. Cuando se activa el modo se cambian tanto el atributo “Brillo” como el atributo “Temperatura”, dependiendo del tipo del dispositivo.

Una de las ideas al principio era poder cambiar además otros atributos como podría ser el “Encendido / apagado” o la estancia de los dispositivos, por ejemplo. Finalmente no se consideró necesario poder modificar el resto de atributos, ya que en muchos de ellos no tenía sentido cambiarlos de manea grupal, como el nombre de los dispositivos o la estancia en la que estaban.

- **Creación de más tipos de dispositivos**

La idea en un principio era tener más tipos de dispositivos. Algunos dispositivos populares como los altavoces o interruptores inteligentes se propusieron para implementarse en un principio, pero al crear una parte de la estructura de la aplicación de manera estática para los tipos iniciales era costoso cambiar y añadir más dispositivos una vez la aplicación había tomado forma.

7. Relación con los estudios cursados

La informática es un campo muy extenso que contiene muchas variantes. Con ramas como la arquitectura de hardware, computación o el desarrollo de software, la informática cuenta con distintos ámbitos que, de una manera u otra se ha enseñado en el grado.

En este apartado se valorará a nivel personal la importancia que han tenido las enseñanzas del grado en el desarrollo del Trabajo de Fin de Grado.

Aunque en la carrera se han enseñado en mayor o menor profundidad muchos lenguajes de programación Python no ha sido uno de ellos. Realmente no supone ningún problema ya que lo que sí que se ha hecho ha sido enseñar las bases de la programación, que son comunes para prácticamente todos los lenguajes de alto nivel.

Por esa razón aprender Python por mi cuenta no fue ningún problema, ya que las bases que ya tenía sirvieron mucho para agilizar el proceso de aprendizaje.

Por otro lado en cuanto a los lenguajes de programación utilizados para el desarrollo de la funcionalidad de la web sí que se enseñaron explícitamente en la carrera. JavaScript es el ejemplo más claro, teniendo asignaturas como por ejemplo TSR en las que se profundiza mucho en este lenguaje.

En cuanto al diseño de la aplicación en ámbitos como el análisis de necesidades, la usabilidad, el diseño visual o la estructura de los elementos que componen la pantalla hay que remarcar una asignatura que ha tenido suma importancia a la hora de enseñar el proceso y los pasos para crear una aplicación: DCU.

Aunque las tecnologías que se utilizan en DCU no son exactamente las mismas que se han utilizado en el TFG sí que ha sentado las bases para construir desde el principio la aplicación.

Se ha enseñado a hacer un análisis de necesidades, utilizando distintas herramientas para ello. También ha enseñado la utilización de prototipos como bocetos o mockups para generar los primeros diseños de la aplicación, ha mostrado los principios de usabilidad para generar una interfaz que sea lo más sencilla y útil para el usuario y ha enseñado también los lenguajes utilizados para desarrollar estas interfaces: CSS y HTML.

En cuanto a la eficiencia de la aplicación a nivel de código el grado también ha enseñado en diferentes asignaturas mediante la utilización de algoritmos o estructuras de datos

En cuanto al manejo de datos de la aplicación a nivel de programación ha habido asignaturas en el grado que han proporcionado herramientas para aumentar la eficiencia del código. Tanto los algoritmos de búsqueda y de ordenación de elementos como la

utilización de estructuras de datos eficientes son ejemplos de formas de generar un código lo menos pesado y lo más rápido posible.

Sobre la utilización de las bases de datos y del funcionamiento de los protocolos HTTP utilizados para acceder a la página web el framework proporciona muchas ventajas y facilidades.

Aunque a la hora de generar una petición a la base de datos o de acceder a las URLs de la web utilizando Django no hay que saber las bases de su funcionamiento, sí que es cierto que ha sido importante conocer cómo funcionan exactamente las bases de datos y los protocolos de conexión HTTP enseñados en el grado, ya que permitía una mayor comprensión y versatilidad a la hora de escribir el código.

En general considero que los estudios del grado han ayudado mucho a la hora de hacer el Trabajo de Fin de Grado. Es cierto que algunas tecnologías más específicas no se han si quiera nombrado en las asignaturas cursadas, pero sí se han dado las bases para desarrollar cualquier tipo de software o aplicación.

8. Bibliografía

[1] ABIResearch (Oct. de 2018), Smart Home Growth Disrupting Home Service Industries. Disponible en:

https://www.abiresearch.com/press/smart-home-growth-disrupting-home-service-industries/

[2] Wikipedia (Ene. de 2020), Modelo de dominio. Disponible en:

https://es.wikipedia.org/wiki/Modelo_de_dominio

[3] EspiFreelancer (Ago. de 2019) Qué es el patrón MTV (Model Template View). Disponible en:

https://espifreelancer.com/mtv-django.html

[4] HostGator México (Abr. de 2022) SQLite: qué es, cómo funciona y cuál es la diferencia con MySQL. Disponible en:

https://www.hostgator.mx/blog/sqlite-que-es-y-diferencias-con-mysql/

[5] Django (Abr. de 2023) Bases de datos. Disponible en:

https://docs.djangoproject.com/en/4.2/ref/databases/

[6] Beatriz Allas Miguelsanz (Ago. de 2017) Los 10 principios de usabilidad de Jacob Nielsen: be user friendly. Disponible en:

https://profile.es/blog/los-10-principios-de-usabilidad-web-de-jakob-nielsen/

[7] Wikipedia (Jul. de 2022) Miga de pan (informática). Disponible en:

https://es.wikipedia.org/wiki/Miga_de_pan_(inform%C3%A1tica)

[8] Wikipedia (Dic. de 2020) Dashboard (software). Disponible en:

https://es.wikipedia.org/wiki/Dashboard_(software)

9. Anexos

9.1. Objetivos de desarrollo sostenible (ODS)

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Procede**
ODS 1. **Fin de la pobreza.**			X	
ODS 2. **Hambre cero.**			X	
ODS 3. **Salud y bienestar.**	X			
ODS 4. **Educación de calidad.**			X	
ODS 5. **Igualdad de género.**			X	
ODS 6. **Agua limpia y saneamiento.**			X	
ODS 7. **Energía asequible y no contaminante.**			X	
ODS 8. **Trabajo decente y crecimiento económico.**			X	
ODS 9. **Industria, innovación e infraestructuras.**			X	
ODS 10. **Reducción de las desigualdades**.			X	
ODS 11. **Ciudades y comunidades sostenibles**.		X		
ODS 12. **Producción y consumo responsables.**	X			
ODS 13. **Acción por el clima.**			X	
ODS 14. **Vida submarina.**			X	
ODS 15. **Vida de ecosistemas terrestres.**			X	
ODS 16. **Paz, justicia e instituciones sólidas.**			X	
ODS 17. **Alianzas para lograr objetivos.**			X	

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados:

Los Objetivos de Desarrollo Sostenible (ODS) son un conjunto de 17 objetivos establecidos por las Naciones Unidas. Son parte de la Agenda 2030 y tienen como objetivo abordar los desafíos globales y promover el desarrollo sostenible en todo el mundo.

El Trabajo de Fin de Grado cumple en mayor o en menor medida algunos de estos objetivos, así que a continuación se mostrará la relación que tiene el TFG con los Objetivos de Desarrollo Sostenible:

- **Salud y bienestar**

La aplicación facilita la gestión y el manejo de los distintos dispositivos inteligentes que el usuario tenga en casa.

Mediante accesos directos y la configuración sencilla y grupal de dispositivos la aplicación permite al usuario poder centrarse en las cosas que de verdad le importan, en vez de preocuparse por tareas tediosas como puede ser apagar los ventiladores, bombillas o calefactores que se puede haber dejado encendidos por toda la casa.

Tal vez estas funcionalidades no afecten de manera directa a la salud del usuario, pero sí que le aportan una tranquilidad y una comodidad directamente relacionadas con el bienestar en el hogar, y por ende, mejoran su salud y calidad de vida.

- **Producción y consumo responsables**

La aplicación desarrollada también está directamente relacionada con el objetivo del consumo responsable.

Al proporcionarle facilidades al usuario a la hora de encender, apagar o modificar los valores de un dispositivo, problemas como que algún usuario no baje la temperatura del termostato por vaguedad o pereza, o simplemente que el usuario haya salido de casa y se haya dejado encendida la luz de casa deberían ser solucionados con la utilización de nuestra aplicación.

Por otro lado la aplicación también permite la automatización de los dispositivos y sus parámetros. El usuario puede automatizar el encendido o apagado de un dispositivo dependiendo de una condición.

Esta funcionalidad también promueve el consumo responsable de energía, ya que permite al usuario tener una configuración y una lógica en sus dispositivos que aumenta la eficiencia en el tiempo de uso útil y disminuye la posibilidad de tener dispositivos encendidos y no utilizándose.

- **Ciudades y comunidades sostenibles**

Todas las facilidades y ventajas que permite la aplicación en un hogar también pueden aplicar a comunidades y sociedades más grandes. Ya sea una comunidad de vecinos, un bloque de apartamentos o un pueblo, las ventajas que aporta la aplicación pueden escalarse para funcionar en estos casos.

Sí que es cierto que la aplicación no está pensada para ser utilizada en un ámbito tan grande como puede ser una ciudad, pero sí que se podría escalar para llegar a funcionar correctamente en estos casos.

Poder automatizar el apagado o encendido de farolas de una ciudad, o encender o apagar la calefacción de todas las habitaciones de un hotel, son ejemplos de funcionalidades que la aplicación puede llegar a permitir y afecta en la sostenibilidad de esas sociedades.