COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

# SISTEMA DE GAMIFICACIÓN DEL CONSUMO ENERGÉTICO

Autor: Mercedes Burillo Palomino

Director: D. Álvaro Sánchez Miralles

Junio, 2022, Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Sistema de gamificación del consumo energético

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2021/22 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Mercedes Burillo Palomino Fecha: 17/ 06/ 22

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: D. Álvaro Sánchez Miralles Fecha: 17/ 06/ 22

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

# SISTEMA DE GAMIFICACIÓN DEL CONSUMO ENERGÉTICO

Autor: Mercedes Burillo Palomino

Director: D. Álvaro Sánchez Miralles

Junio, 2022, Madrid

# Agradecimientos

Quiero agradecer la colaboración desde la empresa Stemy Energy, que ha proporcionado datos para el desarrollo y ayuda para la instalación y explicación de las tecnologías utilizadas durante el proyecto.

# SISTEMA DE GAMIFICACIÓN DEL CONSUMO ENERGÉTICO

**Autor: Burillo Palomino, Mercedes.**
Director: Sánchez Miralles, Álvaro.
Entidad Colaboradora: Stemy Energy y proyecto ReDREAM

## RESUMEN DEL PROYECTO

El proyecto consiste en el diseño de un sistema de gamificación que promueva la participación de las personas que no están involucradas en el sector energético en la flexibilidad energética. Para ello se desarrolla una web, un servidor y una base de datos, que interactúan entre sí, para garantizar unas recompensas específicas a cada participante.

**Palabras clave**: Gamificación, web, servidor, base de datos.

### 1. Introducción

Como participación con el proyecto ReDREAM, surge la propuesta de realizar un sistema de gamificación del consumo energético. El proyecto ReDREAM tiene como objetivo producir un cambio del sistema eléctrico actual para situar al consumidor en el centro de un nuevo ecosistema en el que la demanda de energía se ajuste a la generación de esta en cada momento. [1]

Es un proyecto muy complejo, en el que se necesita la colaboración de un gran número de personas y, por lo tanto, se plantea la idea de desarrollar un sistema de gamificación, que utilice la mecánica de los juegos para dar a conocer este novedoso proyecto. [2]

Además, ReDREAM está muy comprometido con la introducción de energías renovables y la disminución del $CO_2$ que se emite en la actualidad. Estos datos se tendrán en cuenta a la hora de desarrollar el sistema de gamificación. [1]

### 2. Definición del proyecto

Para el desarrollo del proyecto, en primer lugar, es necesario tener claras las claves para realizar un buen sistema de gamificación que son la utilización de la interactividad para atraer la atención de los usuarios y la utilización de las recompensas que ayudan a promover y mantener la motivación de los jugadores. [2]

Por ello, se plantean algunos objetivos principales entre los que destacan el aprendizaje de los lenguajes de programación utilizados (JavaScript, HTML y CSS), el diseño del sistema de gamificación, el desarrollo de la página web y su conexión con el servidor y base de datos. Finalmente, se desarrolla una mejora de la estética de la web y del diseño del código desarrollado.

### 3. Descripción del sistema

El sistema está formado por tres bloques principales como se puede ver en la Ilustración 1: *Front-End* (web), servidor (en el que se ejecuta también el algoritmo) y base de datos (en la que se guarda la información del usuario y los resultados del algoritmo). Funciona de forma que en la web se realiza un formulario a los usuarios y en función de los datos, el algoritmo calcula una serie de recompensas.

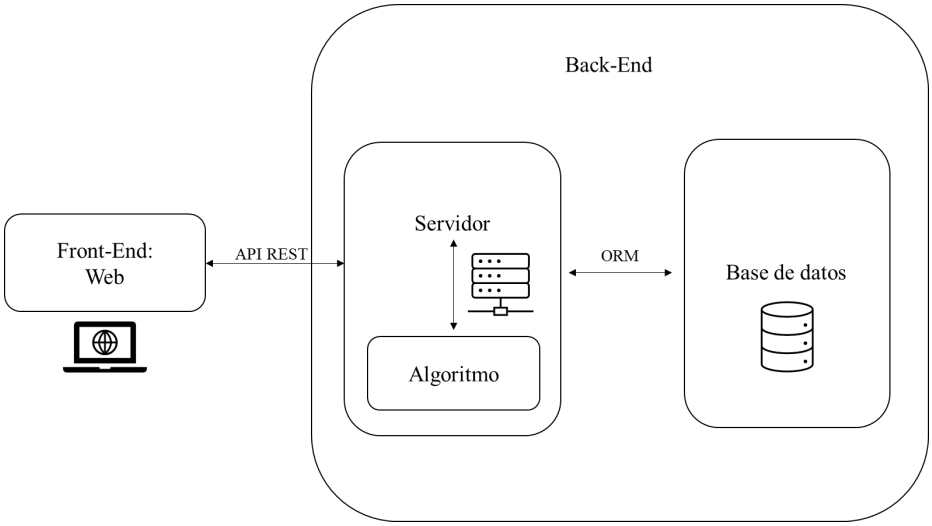


*Ilustración 1: Arquitectura del proyecto*

Para el desarrollo de la web se ha utilizado React, que es una biblioteca de JavaScript declarativa, eficiente y flexible para crear interfaces de Usuario con la que se consigue un alto grado de interactividad en la web. [3]

El servidor se crea con Node.js y el *framework* Express y se utiliza como enlace de la web con la base de datos. La documentación de la *API Rest* y la utilización de Redux RTK hacen posible la utilización de funciones del servidor en el desarrollo web.

En el servidor, se desarrolla el algoritmo, en el que se interpola en función de unos datos para lograr unas recompensas únicas para cada usuario. Estas recompensas son el ahorro de dinero en euros, el ahorro de energía en kWh y el ahorro en $CO_2$ emitido en kilogramos.

Además, el servidor y la web utilizan TypeScript, que añade sintaxis adicional para admitir tipos a JavaScript y permite detectar errores en el paso previo a la compilación. La utilización de esta herramienta facilita enormemente el desarrollo de código y evita numerosos errores. [4]

Finalmente, la base de datos MySQL se crea con la ayuda los contenedores de Docker y se ejecuta en una máquina virtual de Linux. Para el diseño y comunicación de la base de datos con el servidor, se utiliza Prisma. Prisma es un ORM, que permite tener en cuenta un control de versiones y es una capa de abstracción que aporta flexibilidad al sistema para su adaptación a otros motores de bases de datos. [5]

## 4. Resultados

Como resultados, se obtiene un sistema de gamificación interactivo, ya que hay elementos que reaccionan a las actividades del usuario, que se comunica con una base de datos a través de un servidor en el que se guarda la información de los clientes y los resultados del algoritmo con las recompensas de cada uno.

Además, el sistema creado es muy flexible a cambios ya que las partes del proyecto se encuentran claramente diferenciadas lo que facilita, introducir nuevos cambios si fuera necesario.

En la Ilustración 2, se puede ver una imagen de una de las partes de la web.

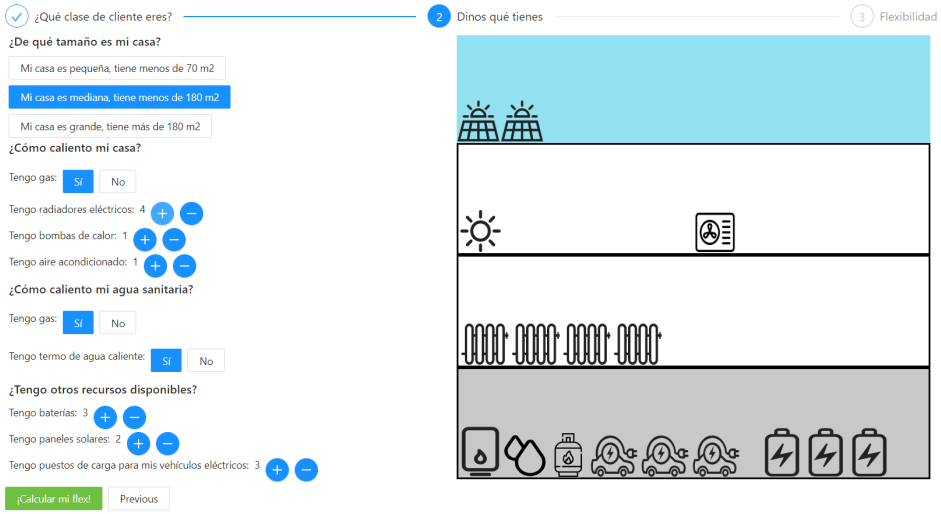


*Ilustración 2: Web interactiva*

## 5. Conclusiones

En conclusión, a partir de una buena planificación del proyecto, se ha utilizado una combinación de numerosas tecnologías para producir un sistema de gamificación por el cual las personas se interesen por su participación en la flexibilidad energética. Todos los recursos utilizados están perfectamente conectados para ofrecer una experiencia al usuario, en la que solo es consciente de la existencia de una web.

## Referencias


[1] «ReDREAM,» [En línea]. Available: https://redream-energy-network.eu/. [Último acceso: 26 05 2022].

[2] «Gamify,» [En línea]. Available: https://www.gamify.com/what-is-gamification . [Último acceso: 30 05 2022].

[3] «Fullstack React,» [En línea]. Available: https://www.newline.co/fullstack-react/30-days-of-react/day-1/ . [Último acceso: 27 05 2022].

[4] «TypeScript,» [En línea]. Available: https://www.typescriptlang.org/ . [Último acceso: 26 05 2022].

[5] «Prisma,» [En línea]. Available: https://www.prisma.io/react-server-components . [Último acceso: 26 05 2022].

# ENERGY CONSUMPTION GAMIFICATION SYSTEM

**Author: Burillo Palomino, Mercedes.**
Supervisor: Sánchez Miralles, Álvaro.
Collaborating Entity: ReDREAM Project and Stemy Energy

## ABSTRACT

The project is about the design of a gamification system that promotes the participation of people who are not involved in the energy sector in energy flexibility. For this purpose, a website, a server, and a database are developed, which interact with each other to guarantee specific rewards to each participant.

**Keywords**: Gamification, web, database, server.

### 1. Introduction

As part of the ReDREAM project, the proposal to create a gamification system for energy consumption arose. The ReDREAM project aims to produce a change in the current electricity system to place the consumer at the center of a new ecosystem in which energy demand is adjusted to the generation of energy at each moment. [1]

It is an extraordinarily complex project, in which the collaboration of a large number of people is needed, and therefore, the idea of developing a gamification system, which uses the mechanics of games to publicize this novel project, is raised. [2]

In addition, ReDREAM is very committed to the introduction of renewable energies and the decrease of CO2 emitted at present. These data will be taken into account when developing the gamification system. [1]

### 2. Project definition

For the development of the project, it is first necessary to understand the keys to making a good gamification system, which is the use of interactivity to attract the attention of users and the use of rewards that help to promote and maintain the motivation of the players. [2]

Therefore, some main objectives are raised among which stand out the learning of the programming languages used (JavaScript, HTML and, CSS), the design of the gamification system, the development of the web page and, its connection with the server and database. Finally, an improvement of the aesthetics of the web and the design of the developed code is developed.

### 3. System description

The system consists of three main blocks as shown in Figure 1: Front-End (web), server (where the algorithm is also executed), and database (where user information and algorithm results are stored). The system works in such a way that users are asked to fill

in a form on the web and depending on the data, the algorithm calculates a series of rewards.

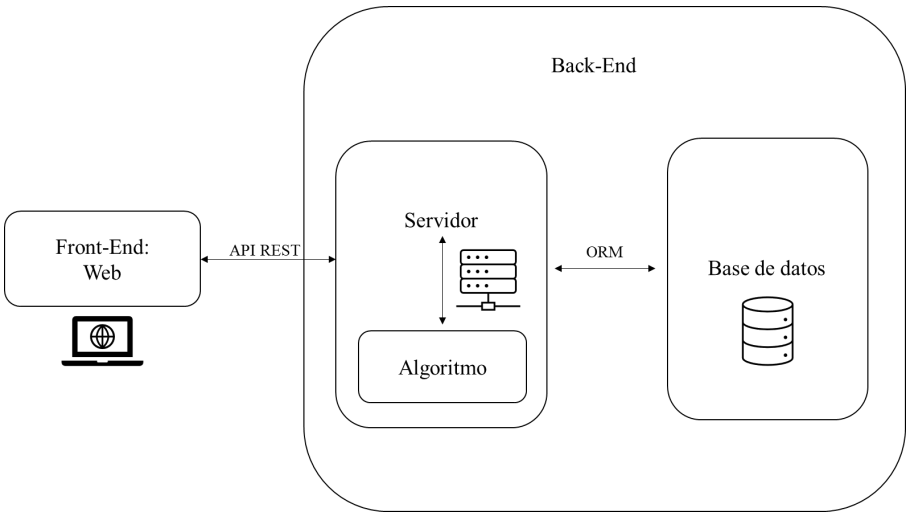


*Figure 1: Web architecture*

For the development of the web, React has been used, which is a declarative, efficient, and flexible JavaScript library to create user interfaces with which a high degree of interactivity is achieved on the web. [3]

The server is created with Node.js and uses the Express framework and is used as a link from the web to the database. The documentation of the Rest API and the use of Redux RTK make it possible to use server functions in web development.

On the server, the algorithm is developed, where it is interpolated based on data to achieve unique rewards for each user. These rewards are money saved in euros, energy saved in kWh, and CO2 emitted in kilograms.

In addition, the server, and the web use TypeScript, which adds additional syntax to support JavaScript types and allows errors to be detected a step before the compilation. Using this tool greatly facilitates code development and avoids numerous errors. [4]

Finally, the MySQL database is created with the help of Docker containers and runs on a Linux virtual machine. For the design and communication of the database with the server, Prisma is used. Prisma is an ORM, which allows taking into account a version control and is an abstraction layer that brings flexibility to the system for adaptation to other database engines. [5]

## 4. Results

As a result, an interactive gamification system is obtained, since there are elements that react to the user's activities, which communicates with a database through a server that stores customer information and the results of the algorithm with the rewards of each one.

In addition, the system created is very flexible to changes since the parts of the project are differentiated, which facilitates the introduction of recent changes if necessary.

In Figure 2, you can see an image of one of the parts of the web.

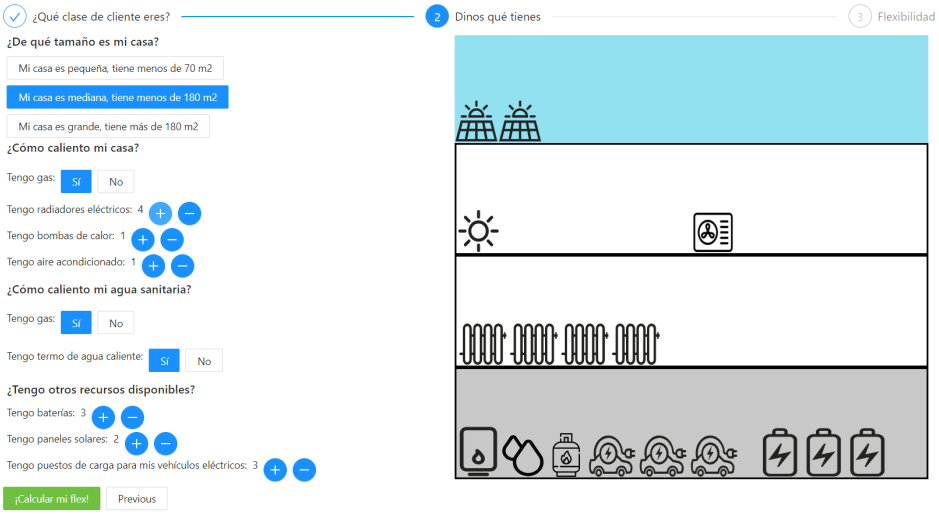


*Figure 2: Interactive web*

## 5. Conclusions

In conclusion, based on good project planning, a combination of numerous technologies has been used to produce a gamification system whereby people are interested in their participation in energy flexibility. All the resources used are perfectly connected to offer an experience to the user, in which he is only aware of the existence of a website.

## 6. References

[1] «ReDREAM,» [Online]. Available: https://redream-energy-network.eu/. [Last access: 26 05 2022].

[2] «Gamify,» [Online]. Available: https://www.gamify.com/what-is-gamification . [Last access: 30 05 2022].

[3] «Fullstack React,» [Online]. Available: https://www.newline.co/fullstack-react/30-days-of-react/day-1/ . [Last access: 27 05 2022].

[4] «TypeScript,» [Online]. Available: https://www.typescriptlang.org/ . [Last access: 26 05 2022].

[5] «Prisma,» [Online]. Available: https://www.prisma.io/react-server-components . [Last access: 26 05 2022].

# *Índice de la memoria*

# *Índice de figuras*

# *Índice de tablas*

# *Índice de códigos*

# Capítulo 1. INTRODUCCIÓN

Este proyecto, sistema de gamificación del consumo energético, surge como colaboración con un proyecto mucho más grande denominado ReDREAM, cuyo objetivo es cambiar el sistema eléctrico actual, para situar al consumidor en el centro de un nuevo ecosistema. Se pretende hacer partícipe al consumidor de lo que ocurre en la red eléctrica.[1]

Sin embargo, la mayoría de las personas no saben cómo funciona la red eléctrica y eso da lugar a este proyecto. Gamificación es la aplicación de elementos de diseño y principios de juegos en contextos que no son de juegos. Son un conjunto de actividades y procesos para resolver problemas o para educar usando elementos de juegos. Algunos de estos elementos pueden ser la obtención de puntos, recompensas u objetivos. De esta forma, se mantiene la motivación y el interés del jugador.[2]

Además, el sistema eléctrico actual, se basa en que la generación y la demanda deben ser iguales en todo momento y es la generación, la que se adapta a la demanda existente en cada momento. Esta idea se hace cada vez más complicada con la introducción de nuevas fuentes de energías renovables, que dependen de elementos de la naturaleza para producir energía.

Por ello, desde ReDREAM, se quiere conseguir que sean los consumidores los que se adapten en cada momento a la energía que se genera en cada momento. Entonces, el concepto de flexibilidad energética consiste en la adaptación de los consumos de energía ante unas condiciones.

Finalmente, desde el proyecto ReDREAM, también se trata de reducir el $CO_2$ emitido, dando gran importancia a las energías renovables. En el sistema, el usuario podrá darse cuenta también, que, si colabora con energías renovables, podrá llegar a unos mejores resultados.

En resumen, el proyecto, utiliza el concepto de la gamificación, para que las personas, vean de una forma interactiva, el concepto de flexibilidad energética. Además, serán capaces de valorar y normalizar el uso de energías renovables, para ayudar a reducir la contaminación

que se produce actualmente. El proyecto ReDREAM, necesita la participación de muchos consumidores para poder llegar a unos buenos resultados. Por ello, se espera que este proyecto ayude a darle visibilidad para llegar al máximo número de personas posible y de esta forma ayudar al proyecto a cumplir sus objetivos.

# Capítulo 2. ESTADO DE LA CUESTIÓN

En la actualidad, existe un problema social crítico[3] relacionado con el consumo de energía eléctrica. Se puede decir que existe una desatención en el consumo doméstico principalmente. Por ello, se con este proyecto se pretende realizar un sistema de gamificación por el cual se consiga concienciar a las personas en la utilización de los diferentes recursos energéticos.

El uso de un sistema de gamificación se puede justificar por algunos estudios encontrados [3] aseguran que el uso de la gamificación permite obtener una experiencia de usuario positiva en el comportamiento, desarrollo cognitivo, conocimiento, aprendizaje y experiencia de usuario.

La mayoría de estos estudios [3], permiten decir que los sistemas de gamificación son efectivos para influir a los usuarios con respecto al consumo de energía en el ámbito doméstico. Por ello se puede decir que es un buen método para conseguir una motivación de la sociedad a seguir aprendiendo sobre la energía y aplicar ese conocimiento hacia un consumo más responsable.

En la investigación realizada, no se han encontrado datos de estudios o proyectos que coincidan exactamente con el trabajo que se realiza. Esto puede deberse a que hasta la actualidad el concepto de flexibilidad energética no era tan importante. Sin embargo, sí se han encontrado estudios que utilizan la gamificación para concienciar sobre el uso y el ahorro de energía eléctrica.

Según algunos artículos, [4] se expresa que una de las dificultades a la hora de desarrollar un sistema de gamificación es la falta de conocimiento en las nuevas tecnologías. Por esta razón, es importante tener en cuenta que el juego debe ser muy fácil e intuitivo y contar con estrategias de diseño que lo hagan lo más llamativo posible.

En relación con el uso de la energía en función de un ahorro en $CO_2$ y con una fijación de los precios de la energía, se ha encontrado una conferencia en la que se analizan los aspectos de la gamificación necesarios para lograr los objetivos buscados.[5]

Para empezar, se sabe que el comportamiento está determinado en gran medida por la motivación, la cual implica procesos psicológicos que son responsables de iniciar y continuar comportamientos dirigidos a objetivos. La motivación se ve además afectada por experiencias pasadas y los deseos personales. Además, las motivaciones pueden dividirse en dos categorías diferentes: motivación intrínseca y motivación extrínseca. Las motivaciones intrínsecas son aquellas en las que la actividad que se desarrolla es la propia recompensa, es decir, la actividad es gratificante en sí misma. Sin embargo, las motivaciones extrínsecas se basan en alcanzar una recompensa o un objetivo por el desarrollo de la actividad.

En el caso del proyecto que se presenta, una motivación extrínseca podría ser el ahorro de dinero que obtienen los usuarios al introducirse en el mundo de la flexibilidad. Esta es una recompensa que todos los usuarios entienden y puede provocar la participación de diferentes personas. Además, una motivación intrínseca sería la información del ahorro $CO_2$ emitido, que puede provocar que la idea de ayudar al medio ambiente sea una recompensa suficiente en el desarrollo de la actividad.

La diferenciación de las categorías de la motivación es importante, ya que existen resultados distintos de cada una de ellas. Se sabe que con una buena motivación intrínseca se obtienen resultados mejores a largo plazo. Por ello, se debe lograr un convencimiento de los usuarios en el desarrollo de la actividad.

Además, conseguir un cambio en el comportamiento de las personas no es sencillo. Según [6], se deben dar cinco etapas para conseguirlo. La primera de ellas es la precontemplación, en la que se asume que el residencial no tiene intención de hacer ningún cambio en su comportamiento, por lo que lo primero que debe hacer el cliente es darse cuenta de las consecuencias que tiene un determinado comportamiento.

Cuando se comienza a dar cuenta de las consecuencias es cuando considera realizar un cambio, y es ahí donde comienza la siguiente etapa contemplación. El cliente necesita buscar ideas, hechos y consejos. En la siguiente etapa, la preparación, conlleva el aprendizaje necesario para realizar los cambios que se quieren implantar.

Las etapas finales se encuentran muy relacionadas: acción y mantenimiento. Actuar de la forma que se había preparado anteriormente y mantenimiento, ya que si esta acción no se desarrolla a lo largo del tiempo no tiene tanta importancia. De ahí la importancia de la motivación intrínseca ya comentada anteriormente.

Además, para lograr una buena gamificación se pueden utilizar elementos que ayuden al usuario que utiliza el juego. Por ejemplo, puede ser buena la utilización de gráficos e imágenes que garanticen una visualización sencilla del comportamiento energético.

La idea de generar una conexión social es interesante según algunos estudios. Las personas son seres sociales y por ello la competición o colaboración con otros usuarios pueden ser importantes para lograr una mejora en la motivación y por lo tanto en los resultados del juego.

Para terminar, la creación de una interfaz de usuario para lograr una mayor atención del jugador y la creación de un cuadro informativo de progresos pueden ser ideas que hagan aumentar las ganas de participación de las personas.

Además, está bien dejar claro que las acciones que se realizan conllevan unas consecuencias inmediatas. De esta forma se puede conseguir un mayor impacto en la actitud de las personas a la hora de cambiar sus modos de acción. [7]

Para continuar, se han encontrado otras buenas prácticas para lograr una buena gamificación. Para empezar, existe un artículo [8], en el que se estudia la flexibilidad de los consumos de energía. En el buscan la disposición de los usuarios a adaptarse a los picos de demanda de las fuentes renovables de energía. El procedimiento que utilizan para el estudio es un ranking.

Se instalan diferentes sensores y llegan a la conclusión de que la variable de presencialidad en el domicilio es una variable muy clave a la hora de estudiar los hábitos de consumo de los usuarios. Por ello, para mantener de forma justa las recompensas utilizan esa variable a la hora de asignar los valores a cada usuario. Se comprobó que se consiguió un cambio de un 5% con los incentivos de ludificación.

Otros proyectos, se basan en influenciar a las personas para conseguir una disminución del consumo energético [9]. Hay un proyecto creado en Europa que se denomina enCOMPASS que se basa en la utilización de sensores y el análisis de las actividades de los participantes en el uso de la energía eléctrica para poder aplicar una gamificación más personalizada a cada uno de ellos.

Sus ideas vienen del trabajo SmartH2O, en el que se conciencia a la sociedad sobre el empleo del agua. En él, los usuarios reciben consejos y elementos de aprendizaje. EnCOMPASS, va más allá, porque con los datos obtenidos de los sensores y de los usuarios, crea sugerencias únicas. Realizar una parte informativa en el juego puede ser interesante para atraer la atención de los jugadores.

En enCOMPASS también utilizan la idea de clasificación a modo de ranking de los usuarios participantes, al igual que en otros casos ya mencionados como [8].

Además de la utilización de recompensas virtuales, utilizan un juego de mesa, Funergy, que está vinculado con el consumo de energía y puede verse como otro método de concienciación. Para llegar a los resultados de las diferentes recompensas, se utiliza un sistema de mapeo en el que se recogen los movimientos de los participantes y se cambian por puntos en base a unas reglas. Para el proyecto que se desarrolla, también será necesaria la introducción de un algoritmo que calcule recompensas personalizadas en función de los diferentes datos de usuario.

Por otro lado, existen más estudios que han utilizado los sistemas novedosos de gamificación en relación con el ahorro energético [10]. El proyecto ChArGED utiliza dispositivos de

internet de las cosas, IoT, para estudiar las variaciones en el comportamiento en el ahorro de energía en los edificios públicos.

Los trabajadores de una empresa de un edificio no van a tener tanto compromiso a un ahorro de energía como el que podrían tener en sus domicilios ya que no son ellos los que corren con los gastos de energía. Sin embargo, se han comprometido a hacer cambios si su comodidad diaria no se ve afectada.

En este caso, la característica de juegos que se utiliza es la creación de diferentes equipos entre los empleados, para que ningún individuo quede expuesto ante el resto del grupo. Finalmente, se va puntuando favorablemente la actitud de los diferentes equipos en función de las acciones que llevan a cabo. Los equipos ganadores, consiguen recompensas que se pueden considerar extrínsecas, como boletos de lotería o cupones para cambiarlos por café. En este caso, como las recompensas parecen funcionar para encontrar una participación de los empleados hacia un ahorro energético.

Por otro lado, en [5], permite a los usuarios el registro a través de la red social Facebook, lo que ayuda a la visualización de la aplicación. Además, utilizan el método de ranking para exponer las posiciones de unos respecto a otros y de esta forma fomentar la competencia entre ellos. En este caso, el método de clasificación que utilizan es la cantidad de carbono emitido.

Este tema de competición entre los usuarios no es el empleado por todos los sistemas que se han encontrado. Por ejemplo, en [4], creen que un método más personalizado para cada usuario será más efectivo.

| Ref. | Flexibilidad | Temática | Método de recompensa | Tipo de gamificación | Método de ranking | Tipo de medio | Objeto de gamificación | Uso de estadísticas | Conexión social |
|---|---|---|---|---|---|---|---|---|---|
| [3] | No | Consumo energía doméstica | Puntos, niveles, objetivos | Colectiva | Contra otros | Estudio en diferentes medios | Estudio uso doméstico de la energía | | |
| [4] | Sí | Compromiso de los consumidores con la red eléctrica | | Colectiva | | Web | Conseguir el compromiso de los consumidores | | Sí |
| [5] | No | Modificación en el comportamiento de consumo residencial | Intrínseca | Colectiva | Contra otros | Aplicación móvil | Modificar el comportamiento en el uso de la energía | | Sí |
| [6] | No | Estudio de la gamificación para consumo residencial de energía | Extrínseca | Colectiva | | Estudio de diferentes medios | Consumo residencial de energía | Sí | Sí |
| [7] | No | Utilización de gamificación en el sector eléctrico | Objetivos, reglas, retos | Colectiva | Contra uno mismo | Información sobre diferentes medios | Provocar un cambio en la actitud de los consumidores | Sí | |
| [8] | Sí | Estudio de la flexibilidad de la demanda | | Colectiva | Contra otros | | Activar la flexibilidad de la demanda | | |
| [9] | No | Ahorro energético | Extrínseca | Colectiva | Contra otros | Web | Provocar ahorro energético | | |
| [10] | No | Conservación de la energía en edificios públicos | Extrínseca / Intrínseca | Colectiva | En equipo contra otros | | Aplicación | | Sí |

*Tabla 1: Comparación de otros proyectos*

En conclusión, como se puede ver en Tabla 1, se muestra de forma resumida las características de los diferentes sistemas comentados anteriormente. En primer lugar, en el proyecto se intentará utilizar la idea de la personalización del juego. Es decir, se plantearán recompensas individuales para cada persona y no se utilizarán métodos de clasificación entre participantes.

Además, el objetivo será bastante novedoso, ya que consiste en la introducción de nuevos participantes en la flexibilidad energética, que es un tema que no se trata en profundidad en ninguna referencia de la tabla.

Finalmente, se emplearán también gráficos y recompensas, de forma que el usuario se vea implicado. De esta forma todo quedará mucho más visual y llamativo. También se descarta la idea de realizar una conexión social, ya que el proyecto que se desarrolla no tiene como objetivo una competición, sino servir como forma de informar de un concepto complejo de una forma más sencilla. Además, se intentará remarcar el ahorro de $CO_2$ que se puede obtener formando parte del proyecto para buscar la motivación intrínseca de los participantes.

# Capítulo 3. DEFINICIÓN DEL TRABAJO

## *3.1 JUSTIFICACIÓN*

Teniendo en cuenta la información analizada en el capítulo anterior, se puede decir que actualmente no existe ningún sistema que utilice la gamificación como medio para intentar incluir a las personas en el mundo de la flexibilidad energética. Por ello, la novedad del sistema que se presenta lo hace más llamativo.

Además, como se explicará en los próximos capítulos, el desarrollo de una web interactiva no es un trabajo sencillo. Para conseguir que todo funcione correctamente, es necesaria la instalación de numerosos programas que interactúan entre sí, llegando a un resultado final.

De hecho, aprovechando la información recogida durante el desarrollo del proyecto, se ponen en práctica ideas que hacen que el resultado final mejore considerablemente. Entre estas ideas destaca la personalización de los resultados que se obtienen al final de la web, la interactividad que proporcionan los diferentes botones y el hecho de que los datos de los usuarios se guarden automáticamente en una base de datos.

El proyecto también tiene un compromiso con el medio ambiente, porque su función es meter a las personas en el mundo de la energía y de los ahorros que se pueden conseguir si se consume responsablemente. De hecho, también ayuda a dar visibilidad a fuentes de energía renovables como los paneles solares, ayudando a visualizar otros tipos de energías limpias que pueden favorecer considerablemente el ahorro de $CO_2$ emitido.

Para terminar, es un proyecto novedoso y que acerca a las personas a entender una nueva forma de utilizar los recursos energéticos disponibles, de una forma didáctica y sencilla, al alcance de todos.

## 3.2 OBJETIVOS

Los objetivos que se alcanzan durante el desarrollo del proyecto son:

- Aprendizaje de los lenguajes de programación utilizados para el desarrollo de la web: principalmente JavaScript, HTML y CSS aplicados a React.
- Diseño del sistema de gamificación.
- Primera versión de una web interactiva sin conexión a otros sistemas.
- Segunda versión de una web interactiva con control de estado de las variables de usuarios.
- Conexión de la web con los sistemas y la base de datos.
- Mejorar el diseño para lograr una buena experiencia de usuario.

## 3.3 PLANIFICACIÓN

Para el desarrollo del proyecto, se ha seguido la siguiente planificación:

| Tareas | 10-ene | 17-ene | 24-ene | 31-ene | 07-feb | 14-feb | 21-feb | 28-feb | 07-mar | 14-mar |
|---|---|---|---|---|---|---|---|---|---|---|
| Investigación redream | x | | | | | | | | | |
| Investigación estado de la cuestión | | x | x | | | | | | | |
| Aprendizaje recursos | | | | x | x | | | | | |
| Primera versión web | | | | | | x | x | x | | |
| Segunda versión web funcional | | | | | | | | | x | x |
| | 21-mar | 28-mar | 04-abr | 11-abr | 18-abr | 25-abr | 16-may | 23-may | 30-may | 06-jun |
| Segunda versión web funcional | x | | | | | | | | | |
| Conexión con otros sistemas | x | x | x | | | | | | | |
| Diseño de algoritmo | | | | x | | | | | | |
| Mejora de estructura de código | | | | | x | x | | | | |
| Mejora del diseño web | | | | | | | x | x | | |
| Memoria del proyecto | | | | | | | x | x | x | x |

*Tabla 2: Planificación del proyecto*

Como se puede ver en la Tabla 2, el proyecto se empezó a principios de enero. En primer lugar, se realizó una investigación sobre el proyecto del que se va a formar parte, ya que es necesario para saber cómo enfocar de la mejor manera el trabajo. Además, se continuó la investigación de otros proyectos relacionados para situar claramente los objetivos a los que se quiere llegar.

A continuación, debida a la falta de conocimiento en los lenguajes de programación necesarios, se iniciaron algunos cursos de JavaScript y React y se consultaron numerosos vídeos tutoriales para la familiarización con el entorno.

Una vez aprendidos, comenzó el desarrollo de una primera versión de una web, relacionada con el tema en cuestión, pero que no era lo que realmente se buscaba. De todas formas, el tiempo empleado en esta primera versión fue útil para aprender funcionalidades del entorno de React.

Posteriormente, se comenzó la segunda versión de la web, que después de muchas mejoras llegó a ser la definitiva. Esta versión en primer lugar se hizo funcional, comprobando que todos los datos se gestionaban correctamente en el entorno de la web y posteriormente se conectó con otros entornos, para que los datos se guardaran en la base de datos.

A continuación, se realizó un algoritmo, en el *back-end* para calcular las recompensas de cada usuario y finalmente, una vez todo funcionaba se comenzó a depurar y mejorar la estructura del código, reduciendo su contenido.

Finalmente, se mejoró el diseño de la web con la ayuda de los componentes de Ant Design y por último se pasó a la redacción de la memoria.

# Capítulo 4. OBJETIVOS Y METAS DE DESARROLLO SOSTENIBLE

Los objetivos y metas de desarrollo sostenible son un conjunto de objetivos globales para erradicar la pobreza, proteger el planeta y asegurar la prosperidad para todos como parte de una nueva agenda de desarrollo sostenible.[11]

En el caso de este proyecto, se puede decir que se alinea con los siguientes ODS:

- Energía asequible y no contaminante: como parte del proyecto ReDREAM que pretende utilizar energías renovables y hacer partícipes a los consumidores de los precios.
- Industria innovación e infraestructuras: se pretende optimizar la energía.
- Ciudades y comunidades sostenibles: que dependen de la energía limpia que ellos mismo producen.
- Producción y consumo responsables: producción de energía de manera limpia y búsqueda de un consumo eficiente de todos los usuarios.
- Acción por el clima: se pretende una reducción significativa en el CO2 emitido, al tratar de suprimir los combustibles fósiles por fuentes de energía renovables.
- Vida submarina: con la disminución de la contaminación se tiene un impacto para frenar el cambio climático y por lo tanto ayudar a preservar la vida submarina.
- Vida de ecosistemas terrestres: al igual que en el caso de la vida submarina, se tiene un impacto en el cambio climático de forma que se ayude a la vida terrestre.
- Alianzas para lograr objetivos: el proyecto ReDREAM cuenta con el apoyo de países europeos para trabajar juntos por el desarrollo de una transición energética.[11]

# Capítulo 5. ARQUITECTURA DEL PROYECTO

El proyecto desarrollado es un sistema de gamificación interactivo para el aprendizaje de la flexibilidad energética. La arquitectura utilizada se puede ver en la Figura 1:

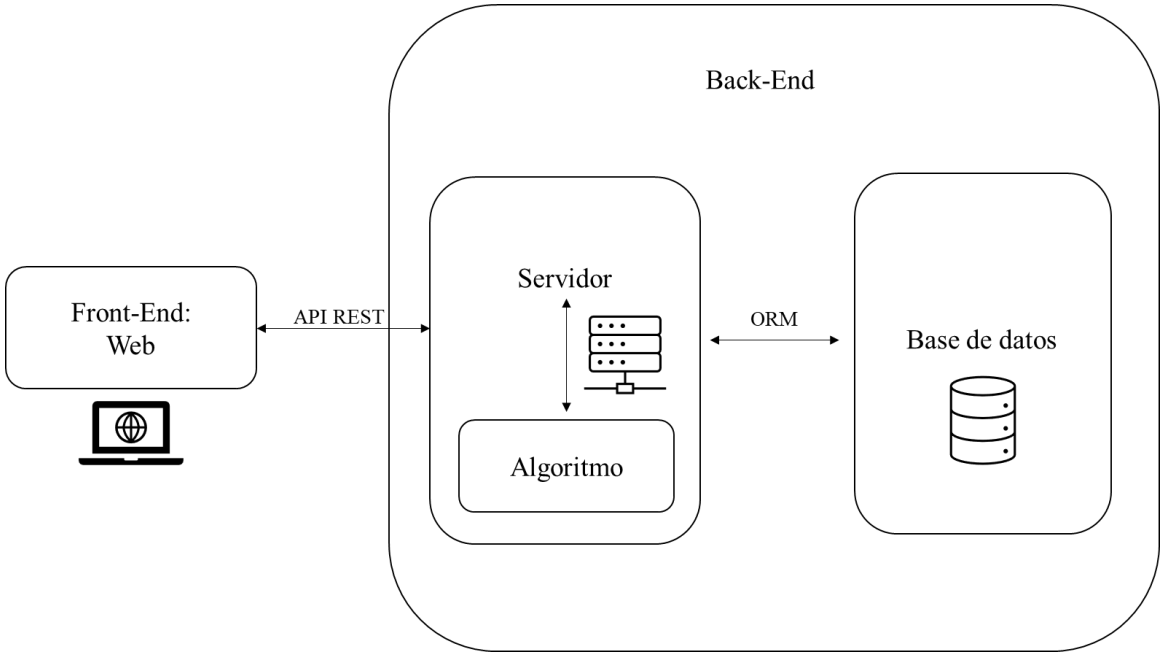


*Figura 1: Arquitectura del proyecto desarrollado*

Como se puede ver en la Figura 1, el proyecto que se desarrolla está formado por tres bloques principales, que interactúan entre ellos, para compartir la información:

- *Front-End*: es la parte del proyecto que ve el cliente. Consiste en el desarrollo de una web empleando React, que garantiza la posibilidad de introducir un alto grado de interacción con el usuario gracias al empleo de JavaScript y TypeScript.
- Servidor: se desarrolla con Node.js y Express y se utiliza como enlace con la base de datos. Además, en el servidor, se desarrolla el algoritmo, que será el encargado de obtener unas recompensas específicas para cada usuario.

- Base de datos: se crea una base de datos MySQL, en la que se almacenará la información proporcionada por cada usuario, así como los resultados obtenidos del algoritmo.

# Capítulo 6. PRIMEROS PASOS EN LA WEB

Para empezar, antes de llegar al proyecto definitivo realizado, se realizó una primera versión de una web interactiva. En ella, se intentaba explicar la flexibilidad, partiendo de la base de explicar el equilibrio entre la demanda y la generación en el sistema energético. Finalmente se decidió que ese no era el camino indicado para el proyecto, pero el proceso fue muy útil a la hora de la familiarización con el entorno de desarrollo de aplicaciones y sus características y funcionalidades.

## *6.1 HISTORIA DEL DESARROLLO WEB*

El inicio del desarrollo web se remonta a 1990 cunado el físico Tim Berners-Lee, trabajador del CERN creó la primera versión de HTML en la que el texto dominaba el espacio de la pantalla. Cabe recordar que en 1990, la pantalla era negra con únicamente texto en verde. Este físico creo el lenguaje HTML, el protocolo HTTP y el sistema de localización de objetos en la web URL.[12]

En 1991, publicó la primera versión de la web pero únicamente los miembros del CERN tenían acceso a esta web. A continuación, en 1992 con la aparición de los navegadores se comenzó a utilizar la organización mediante tablas y en 1994 se conformó el Worl Wide Web Consortium con el fin de desarrollar estándares y recomendaciones web. Finalmente, a partir de 1995, Flash y JavaScript, permitieron la utilización de animaciones con efectos visuales, para ayudar a resolver limitaciones de HTML.[12]

A finales de los años noventa, se empezó a trabajar con CSS y PHP para la creación de páginas web dinámicas y de carga rápida. CSS significa hojas de estilo en cascada, y es un lenguaje de programación que se utiliza para dar formato y apariencia a las páginas web y las interfaces de usuario.[13] PHP es un lenguaje de programación que se utiliza en el desarrollo de aplicaciones y creación de sitios web en el que se puede incrustar lenguaje HTML si se siguen unas normas establecidas esta tecnología, además, permitía la interacción

del usuario con bases de datos.[14] A partir de este momento se dejaron de desarrollar las web a base de tablas, permitiendo lograr el mismo resultado con menos código, lo cual facilita su mantenimiento. En 2003 comenzó la Web 2.0 con las nuevas funcionalidades de CSS3 y una nueva información basada en el usuario. [12]

En la actualidad, el desarrollo web se realiza utilizando JavaScript y generalmente para conseguir una optimización de los recursos de la aplicación se suelen utilizar *frameworks* como React, Vue, Meteor…[15]

## *6.2 JAVASCRIPT*

En 1995 Brendan Eich, un programador que trabajaba en Netscape, inventó un lenguaje de programación que permitiera la interacción del usuario con las páginas web. En un primer momento, este lenguaje se denominó LiveScript, aunque finalmente, se decidió cambiar el nombre a JavaScript por un tema de marketing. Este lenguaje fue un éxito desde el principio.

JavaScript es un lenguaje de programación que permite realizar funciones complejas en el desarrollo de aplicaciones web. Se utiliza para crear contenido dinámico y que la web no sea estática y muestre únicamente información como haría con CSS y HTML. Se puede utilizar para animar imágenes, controlar multimedia…[16].

El lenguaje JavaScript ha evolucionado mucho a lo largo del tiempo, desde su estandarización por la cual surgió la versión de JavaScript 1.1. Se inició una primera propuesta para ECMA en el que se desarrollan la sintaxis, tipos, sentencias, palabras claves y reservadas, operadores y objetos sobre la cual se pueden construir implementaciones. La versión JavaScript 1.3 fue la primera en tener una implementación completa en ECMAScript.

En ECMAScript 3, de 1998, las mejoras incluyeron el soporte de expresiones regulares, nuevas sentencias de control, definición de errores más precisa… En 2004 aparece ECMAScript 4, en el que se introduce el tipado de variables y el concepto de clases e interfaces de estilo. A continuación, en ECMAScript 5, se introducen getters y setters, arrays,

rediseño de atributos y propiedades, acceso a información del prototipo, manipular las propiedades de un objeto, creación de objetos de forma dinámica, cambios a las funciones soporte de JSON… Finalmente, en ECMAScript 6, se introducen módulos, generators, proxys, desestructuring assigments, Rest y default arguments, ámbito a nivel de bloque…[17]

Hay dos formas de tipado en los lenguajes de programación, tipado fuerte o débil. Esto distingue a los lenguajes en función de si permiten o no violaciones de los tipos de datos una vez declarados.

Los lenguajes de tipo fuerte, no permiten comparar u operar con tipos de datos distintos sin realizar una conversión previa. Entre estos lenguajes se encuentra Java o Python. Sin embargo, los de tipado débil, realizan una conversión interna para permitir un tratamiento ambiguo de las variables. En este caso, JavaScript es de tipado débil, lo que puede suponer un peligro si no se tiene el suficiente cuidado.[18]

### *6.2.1 FUNCIONES DE JAVASCRIPT*

Algunas ideas interesantes de JavaScript son las funciones lambda que es una función anónima corta que toma uno o más parámetros y tiene una sola expresión. Esencialmente, permiten que las funciones se proporcionen como parámetros para otras funciones. Debido a que las funciones se ven como objetos en JavaScript, se pueden pasar y devolver desde otras funciones para crear funciones lambda.[19]

Un ejemplo de estas funciones lambda sería:

```
// Función normal para el cálculo de suma de dos potencias
function (p1, p2){
 let p3=50
 return p1 + p2+p3;
}

// Función flecha para el cálculo de suma de dos potencias
(p1, p2) => {
   let p3=50
   return p1+ p2 +p3
```

```
};
```

*Código 1: Ejemplo de funciones flecha*

Los parámetros que llegan a la función se indican entre paréntesis y después de la flecha, se indica lo que tiene que realizar y devolver la función. Este tipo de funciones son muy útiles para la definición de funciones que no se utilizarán más adelante.

```
  const updateGeneration = () => {
        if (delta.generationOrDemand === 'generation') {
            if (delta.upDown === 'up') {
                setCurrentGeneration((prev) => prev + delta.randomDelta)
            } else {
                setCurrentGeneration((prev) => prev - delta.randomDelta)
            }
        }
    }
```

*Código 2: Función flecha en la primera versión de la web*

En el Código 2, se puede ver un ejemplo de la utilización de las funciones flecha en el código de la primera versión desarrollada.

Dado que también se trabaja mucho con variables vectoriales llamadas *Arrays*[20], hay unos métodos o funciones de JavaScript que son muy útiles para tratar este tipo de variables. Algunas de estas propiedades son *map* y *filter*.

```
var powerDevices = [23, 50, 100, 150];
var doublePowerDevices = powerDevices.map((x)=> x*2);
//el resultado de doublePowerDevices será =[46, 100, 200, 300]
```

*Código 3: Ejemplo .map JavaScript*

En el Código 3, se puede ver que lo que hace la función *map*, que tiene como argumento una función, crea un nuevo vector con los resultados de la llamada a la función indicada aplicados a cada uno de los elementos. [21]

El método *filter*, se usa de forma similar. En primer lugar, el método *filter*, al igual que *map* tiene como argumento una función cuyo argumento será el elemento de la iteración. Este método itera todos los argumentos del vector y aplica la función a cada uno de los elementos. La función argumento devuelve un booleano indicando si el elemento formará parte del nuevo vector o no. Estas propiedades y muchas otras se encuentran en la documentación [21]

```
<tr>
    {aparatos.map((aparato) => {
      return (
        <td key={aparato.id}>
             {aparato.potencia} {aparato.unidades}
        </td>
      )
     })}
 </tr>
```

*Código 4: Ejemplo de uso de map en la primera versión*

Como se puede ver en el Código 4, se ha utilizado la función *map* para iterar todos los elementos del objeto aparatos y transformarlos en un componente de React, <td>.

## *6.3 REACT*

Como se ha comentado anteriormente, en la actualidad se utilizan *frameworks* para el desarrollo de webs. En este caso, se ha decidido utilizar React, que es una biblioteca de JavaScript declarativa, eficiente y flexible para crear interfaces de usuario. Permite formar códigos más complejos a partir de secciones de código más pequeños llamados componentes. [22]

React se ejecuta en un DOM virtual, no en el documento del navegador directamente, es decir, resuelve los cambios en un DOM creado y ejecutado completamente en la memoria. Una vez se actualiza el DOM virtual, React indica los cambios que hay que hacer en el navegador real.[22]

El uso de React en lugar de otros sistemas de desarrollo web, viene justificado por el alto grado de interactividad que se consigue ya que por ejemplo una aplicación desarrollada únicamente con HTML y CSS es estática y no reacciona a las entradas del usuario. El código que se utiliza para la programación en React se basa en JavaScript, con el que se consigue que la aplicación vaya cambiando.

Existen otras tecnologías similares a React como VueJs, sin embargo, se ha elegido la utilización de React ya que ha sido la forma más popular para el desarrollo web. Esto se puede ver en la Figura 2

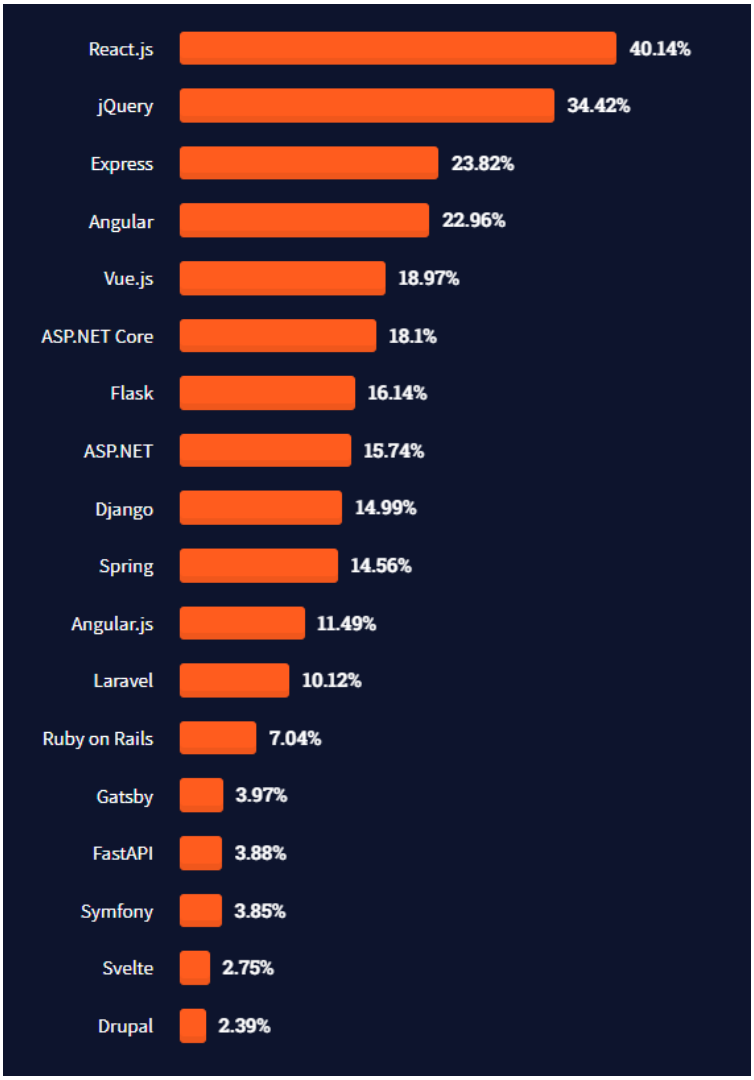


*Figura 2: Web frameworks más utilizados 2022*[23], [24]

### *6.3.1 RENDERIZADO EN REACT*

Los elementos de React, a diferencia de los elementos del DOM de los navegadores, son objetos planos, y su creación es de bajo costo. React DOM se encarga de la actualización del DOM para igualar a los elementos de React. React DOM compara el elemento y sus hijos con el elemento anterior, y solo aplica las actualizaciones del DOM que son necesarias para que el DOM esté en el estado deseado, es decir, sólo renderiza cuando hay un cambio en el estado. Si el componente padre cambia, se renderizan todos los hijos, sin embargo, si son los

hijos los que cambian, solo se renderizan los componentes cuyos estados se han visto alterados.[25]

## *6.4 ESTRUCTURA DE APLICACIONES REACT*

Generalmente, para desarrollar este tipo de aplicaciones en React, se van creando componentes independientes, de forma que los bloques más pequeños están contenidos en otros de mayor tamaño. [22] Esta estrategia se ha seguido en esta primera versión de la web desarrollada:

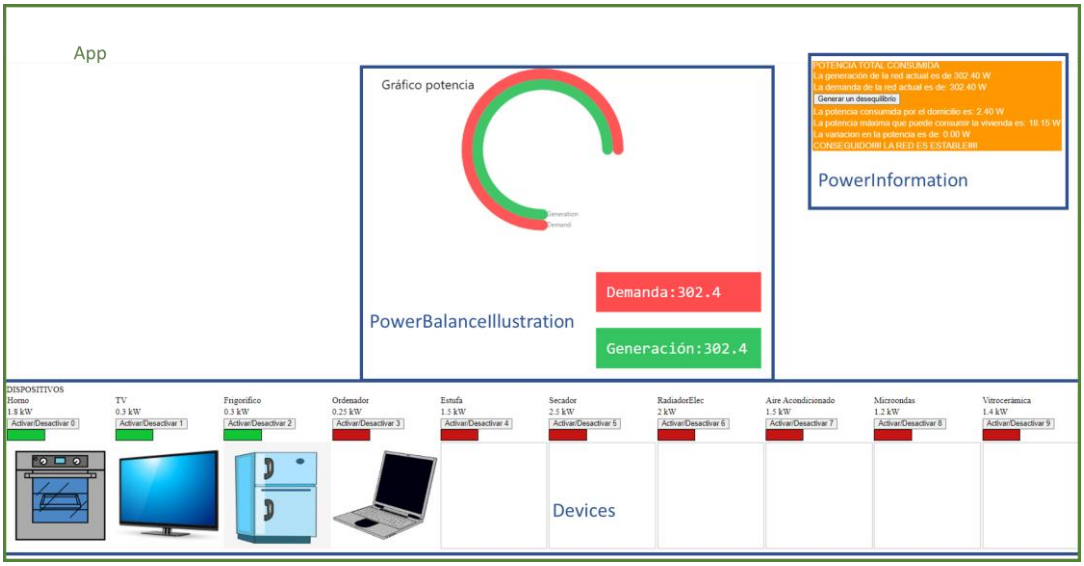


*Figura 3: Bloques de la primera versión web*

Como se puede ver en la Figura 3, la aplicación tiene un componente base denominado App y este está formado por otros tres boques o componentes hijos de App. La generación de componentes se explica en la documentación de React [26].

Hay un detalle muy importante para tener en cuenta y son los estados los que tienen acceso los diferentes componentes que forman parte del proyecto. Estos componentes funcionan de forma que el componente padre, en este caso el App únicamente tiene acceso los estados definidos en App. En cambio, los bloques azules, tienen acceso a sus variables propias y a las que se decidan mandar desde el bloque superior en el momento en el que se les llama.

Tiene importancia en estos casos el concepto de *lifting up the state* o levantamiento de estado, por el cual se plantea el ejemplo de un componente padre en el que se incluyen dos componentes hijos con sus correspondientes estados. El componente padre no puede ver los valores de los estados locales de los hijos, y estos hijos tampoco pueden ver el estado del otro hijo. De esta forma no es posible hacer que los datos de los hijos estén sincronizados, por lo que la solución es definir los estados que antes estaban en los hijos en el padre, de forma que desde ahí se pueda dar acceso a los hijos a los estados que se necesite.[27]

Existe también una forma de crear estados globales que se expondrá más adelante.

```
const App = () => {
    const [devices, setDevices] = useState(aparatos)
    const [currentDemand, setCurrentDemand] = useState(initialPower + initialNetDemand)
    const [currentGeneration, setCurrentGeneration] = useState(initialNetDemand +
initialPower)
    const [delta, setDelta] = useState(deltaAux)
    const [statusMessage, setStatusMessage] = useState('')

    return (
        <div>
            <PowerBalanceIllustration currentDemand={currentDemand}
currentGeneration={currentGeneration} />
            <Devices
                devices={devices}
                setDevices={setDevices}
                currentDemand={currentDemand}
                setCurrentDemand={setCurrentDemand}    />

            <PowerInformation
                devices={devices}
                currentDemand={currentDemand}
                setCurrentDemand={setCurrentDemand}
                currentGeneration={currentGeneration}
                setCurrentGeneration={setCurrentGeneration}
                delta={delta}
                setDelta={setDelta}
                statusMessage={statusMessage}
                setStatusMessage={setStatusMessage}
                maxPower={maxPower}
            />
        </div>
    )
}
```

*Código 5: Componente App en la primera versión*

Como se puede ver en el Código 5, en el momento en el que se llama a los componentes, después del *return*, se puede ver cómo se decide a qué variables definidas en App, pueden acceder estos componentes.

## 6.5 *HOOKS DE REACT*

Además de la organización del código en React, durante el desarrollo de esta web, también se aprendió el funcionamiento de algunos *Hooks* propios de React. Los *Hooks* son funciones que permiten enlazar el estado de React y el ciclo de vida desde componentes de función. [28]

Hay dos *Hooks* que se han usado principalmente para el control de las variables en esta aplicación. El primero de ellos es el *hook* de estado cuya documentación se encuentra en [28]. Estas variables de estado permiten guardar y actualizar el valor de diferentes variables, de forma que estos valores puedan cambiar si así se decide como reacción por ejemplo a algún botón en la web. A continuación, se añade un código sencillo para explicar el funcionamiento de esta propiedad.

```
import React, { useState } from 'react';

function PowerKWh() {
  const [kWh, setKWh] = useState(0);

  return (
    <div>
      <p>Consumes {kWh} kWh</p>
      <button onClick={() => setKWh(kWh + 1)}>
        Click me
      </button>
    </div>
  );}
```

*Código 6: Ejemplo sencillo de hook de estado*

Como se puede ver en el Código 6, lo primero que hay que hacer es importar el Hook desde la librería de *react*. A continuación, dentro de la función que se está desarrollando, se llama a la función *useState*. El valor del paréntesis indica el valor inicial que va a tener la variable, en este caso, es cero por defecto. La función *useState* devuelve un array con dos elementos, donde, el primero de ellos será el valor actual de la variable y el segundo, la función a la que hay que llamar para actualizar este valor. Cada vez que se cambia un estado, se vuelve a renderizar un componente de React, como se comentó anteriormente.

Para que el valor se actualice correctamente en React, hay que llamar a la función *setKWh* como se hace en el código del botón. En el código del botón se está diciendo que cada vez que se haga clic en él, se sumará uno al valor anterior de la variable y no es suficiente con decir que `kWh=kWh+1`, se tiene que hacer por medio de la función *setKWh*.

```
export const Devices = ({ devices, setDevices, currentDemand, setCurrentDemand }) => {
    const toggleDevice = (deviceIndex) => {
        const devicesCopy = [...devices]
        devicesCopy[deviceIndex].estado = !devices[deviceIndex].estado
        const { estado } = devicesCopy[deviceIndex]
        devicesCopy[deviceIndex].styles.color = estado ? green : red
        updateHouseDemand(deviceIndex)
        setDevices(devicesCopy)
        //calculateHouseConsumption()
    }
    const updateHouseDemand = (index) => {
        let currentDemandCopy = currentDemand
        if (devices[index].estado) {
            currentDemandCopy += devices[index].potencia
            setCurrentDemand(currentDemandCopy)
        } else {
            currentDemandCopy -= devices[index].potencia
            setCurrentDemand(currentDemandCopy)
        }
    }

    return (
        <>
            <table>
                <thead>DISPOSITIVOS</thead>
                <tbody>
                    <tr>
                        {aparatos.map((aparato) => {
                            return <td key={aparato.id}>{aparato.name}</td>
                        })}
                    </tr>
                    <tr>
                        {aparatos.map((aparato) => {
                            return (
                                <td key={aparato.id}>
                                    {aparato.potencia} {aparato.unidades}
                                </td>
                            )
                        })}
                    </tr>
                    <tr>
                        {devices.map((device, index) => (
                            <td>
                                <Button
                                    click={() => toggleDevice(index)}
                                    name="Activar/Desactivar"
                                    num={devices[index].id} />
                            </td>
                        ))}
                    </tr>
```

```
                    <tr>
                        {aparatos.map((aparato) => {
                            return (
                                <td key={aparato.id}>
                                    <img src={aparato.imagen}
                                </td>
                            )
                        })}
                    </tr>
                </tbody>
            </table>
        </>
    )
}
```

*Código 7: useState en la primera versión web*

Como se puede ver en el Código 7, una vez se entiende el funcionamiento de los estados se pueden hacer casos más complejos. En este caso, en el elemento *Devices*, se utilizan dos variables que se han definido en *App*. La variable *devices*, en este caso no es un número, si no un vector de objetos, lo que hace su manejo más complicado.

Para actualizar correctamente este tipo de variables, la forma más sencilla es realizar una copia del vector de objetos y ahí cambiar el valor que se desee. Esto es lo que se hace en la función *toggleDevice* del Código 7. Primero se crea una copia, ya que en JavaScript no se puede alterar ese valor directamente y por lo tanto se altera la copia que sí puede ser modificada. Luego se actualiza la característica estado, de un elemento determinado del vector. Finalmente, se llama a la función setDevices cuyo parámetro será el nuevo objeto de copia actualizado.

A continuación, hay que destacar el *Hook* de efecto, cuya documentación se encuentra en [28]. Al igual que en el caso anterior, primero se parte de un ejemplo sencillo para que quede claro el funcionamiento de esta propiedad.

```
import React, { useState, useEffect } from 'react';

function PowerKWh() {
  const [kWh, setKWh] = useState(0);

  useEffect(() => {
    document.title = `La potencia actual es ${count} `;
  },[kWh]);

  return (
```

```
    <div>
      <p>Consumes {kWh} kWh</p>
      <button onClick={() => setKWh(kWh + 1)}>
        Click me
      </button>
    </div>
  );}
```

*Código 8: Ejemplo sencillo de hook de efecto*

Al igual que en el caso anterior, lo primero que hay que hacer es importar *useEffect* desde la librería de *react*. El funcionamiento es muy sencillo, cada vez que la variable *kWh* cambie, se ejecutará la función indicada en el interior de *useEffect*. En este caso, cada vez que el valor de la potencia cambie, el título de la aplicación cambiará y se adaptará al nuevo valor.

```
    useEffect(() => {
        computeStatusMessage()
    }, [currentDemand, devices, currentGeneration])

    useEffect(() => {
        updateDemand()
    }, [delta])
    useEffect(() => {
        updateGeneration()
    }, [delta])
```

*Código 9: Uso de hook efecto en la aplicación*

Como se puede ver en el Código 9, se ha utilizado este *hook* para que se ejecuten determinadas funciones, en función de las variables que se actualizan.

Finalmente, se decidió que la web realizada no era lo que se estaba buscando para el sistema de gamificación, por lo que se comenzó a realizar un sistema más complejo, basado en una web que interactúa con un servidor, que se comunica con una base de datos.

# Capítulo 7. INSTALACIÓN SERVIDOR Y BASE DE DATOS

## *7.1 INSTALACIÓN SERVIDOR*

El servidor se desarrolla con Node.js y Express. Node.js es un entorno de ejecución del motor JavaScript V8, el núcleo de Google Chrome, fuera del navegador, lo que lo hace que sea muy eficaz. Además, proporciona un conjunto de primitivas de E/S que evita que el código JavaScript se bloquee. Esto le permite manejar miles de conexiones simultaneas con un solo servidor. [29]

Node.js también introduce la posibilidad de programar tanto el servidor como el navegador con JavaScript lo que facilita el desarrollo de los trabajos, sin tener que aprender un lenguaje completamente diferente.[29]

Además, se utiliza el *framework* Express que proporciona un conjunto sólido de funciones para aplicaciones web y móviles [30]. Proporciona también, una de las formas más sencillas de crear un servidor y permite hacer llamadas a la API Rest de forma rápida y sencilla.

API es un acrónimo de *Application Programming Interfaces* o en español, interfaz de programación de aplicaciones. Consiste en un grupo de protocolos y definiciones que se utiliza para desarrollar e integrar el software de las aplicaciones, permitiendo la comunicación entre dos aplicaciones cumpliendo un conjunto de reglas. [31] La comunicación con el servidor se realiza por medio de una API Rest.

Una API Rest va más allá de una API ya que debe cumplir que las comunicaciones con la API se realizan a través de HTTP. Es decir, debe tener una arquitectura cliente-servidor compuesta de clientes (en este caso, la aplicación web), servidores y recursos (son las rutas que se indican en el servidor), con la gestión de solicitudes a través de HTTP [32]. Además

las respuestas proporcionadas se dan en formato JSON. Se explicará con más detalle en el Capítulo 9.

Además, Express, permite hacer una validación de los parámetros que se mandan, lo que es muy importante a la hora de detectar errores de programación.[29]

La introducción de Express se hace de la mano con TypeScript, que agrega sintaxis adicional para admitir tipos a JavaScript y permite detectar errores en el paso anterior a la compilación.[33] TypeScript ha sido una herramienta fundamental en el desarrollo del proyecto y se extenderá su explicación en el capítulo 9.5

Una vez instalados los paquetes de Express y TypeScript, se crea un archivo index.ts, donde se indicarán todas las respuestas a las diferentes rutas en el servidor.

```
import express from "express";
const app = express();
app.use(express.json());
const port = 5000;
```

*Código 10: Inicio del código index.ts en el servidor*

Como se puede ver en el Código 10, el programa está escuchando en el puerto 5000 del ordenador y permite realizar varias operaciones y las rutas que se necesitan para que se ejecuten dichas operaciones.

Se utiliza para comunicar o interactuar con otro software y cumplir determinadas funciones. Todo ello depende de los permisos que concede el desarrollador de la API a terceros.[31]

## *7.2 INSTALACIÓN DE LA BASE DE DATOS*

Existen dos tipos de bases de datos, las bases de datos SQL o relacionales y las bases de datos no relacionales o NoSQL. Como su propio nombre indica, en las bases de datos SQL los datos se organizan en tablas cuyos datos están relacionados. Sin embargo, en las NoSQL, los datos son pares clave-valor, basadas en documentos, bases de datos gráficas o almacenes de columnas anchas. Las bases de datos SQL, por tanto son una mejor opción para transacciones de varias filas o sistemas con una estructura relacional. En este caso se elige

una base de datos relacional, ya que se ajusta mejor al manejo de los datos. Además, actualmente existe un mayor soporte para bases de datos SQL que NoSQL, lo que facilita su utilización.[34]

La base de datos que se ha decidido utilizar para el desarrollo del proyecto es del tipo MySQL, ya que es gratuita y de código abierto, además de ser rápida y tener varias capas de seguridad. Es un sistema de administración de bases de datos relacionales desarrollado por Oracle que se basa en un lenguaje de consulta estructurado, SQL. Una base de datos SQL es una colección estructurada de información que se organiza en función de un modelo relacional. Se crean tablas con filas y columnas y se crean diferentes relaciones entre ellas.

Su naturaleza de código abierto, su estabilidad y su rico conjunto de funciones han hecho que se convierta en uno de los motores de bases de datos más popular en la actualidad.[35]

Para la instalación de la base de datos se utiliza un Subsistema de Windows para Linux que es una función opcional de Windows, que permite que los programas de Linux se ejecuten de forma nativa en Windows, gracias a la capa de compatibilidad Ubuntu. Permite a los usuarios de Windows acceder a programas centrales de Linux, incluyendo herramientas GNU como find, awk, sed y grep, que pueden ubicar, buscar y modificar el contenido de los archivos. Sin embargo, no todos los programas de Linux son compatibles con el subsistema de Windows para Linux.[36]

Una máquina virtual consiste en ejecutar un ordenador simulado dentro del ordenador. Consume muchos recursos, ya que la máquina virtual simula que tiene un procesador y una tarjeta gráfica virtuales, es decir, todo el hardware del ordenador y sobre esos recursos virtuales instala el sistema operativo. Sin embargo, con el subsistema instalado no se puede tener una experiencia completa del escritorio de Linux, simplemente se permite la ejecución de algunos programas de Linux.

Además, también fue necesaria la instalación de Docker, que es una plataforma de software para crear aplicaciones basadas en contenedores, que son entornos de ejecución más

pequeños y ligeros que hacen un uso compartido del sistema operativo pero que por lo demás se ejecutan de forma independiente uno de otro. [37]

Permite a los desarrolladores empaquetar aplicaciones en contenedores de componentes que simplifican la entrega de aplicaciones distribuidas y se han vuelto cada vez más populares a medida que las organizaciones cambian al desarrollo nativo de la nube. Docker facilita, simplifica y asegura la creación, implementación y administración de los contenedores, es decir ayuda a instalar y configurar programas en un corto periodo de tiempo. [38]

Una vez instalados la máquina virtual y Docker, se crea el archivo de configuración de Docker, denominado docker-compose.yml, para la creación de las variables de entorno y las configuraciones del contenedor.

```
version: "3.3"
services:
  db:
    image: mysql:5.7
    restart: always
    environment:
      MYSQL_DATABASE: "db"
      # So you do not have to use root, but you can if you like
      MYSQL_USER: "mercedes"
      # You can use whatever password you like
      MYSQL_PASSWORD: "********"
      # Password for root access
      MYSQL_ROOT_PASSWORD: "**********"
    ports:
      # <Port exposed> : < MySQL Port running inside container>
      - "3306:3306"
    expose:
      # Opens port 3306 on the container
      - "3306"
      # Where our data will be persisted
    volumes:
      - my-db:/var/lib/mysql

volumes:
  my-db:
```

*Código 11: docker-compose.yml*

En el Código 11, se define en primer lugar la versión de Docker que se está utilizando y a continuación, en los servicios se definen los contenedores. En este caso se utiliza la imagen mysql 5.7 para levantar una base de datos MySQL local de forma rápida y evitando la necesidad de instalar el software a mano.

El puerto 3306 es el habitual para las bases de datos. Finalmente, los *volumes* permiten la persistencia de los datos en el sistema de archivos de la máquina "host", incluso si se para el contenedor de MySQL. Se puede obtener más información en la documentación de Docker. [39]

Es muy importante cambiar las contraseñas que vienen por defecto a la hora de crear la base de datos, ya que si no se realiza este cambio, hay personas que pueden adueñarse de ella.

Una vez instalada la imagen de MySQL, se instaló DBeaver, herramienta gráfica de gestión de bases de datos gratuita y de código abierto para desarrolladores y administradores de bases de datos. Se utiliza tanto para la creación como administración de bases de datos. Además, funciona con una gran parte de motores de bases de datos como MySQL, PostgreSQL, MariaDB…[40]

Algunas de sus características son, en primer lugar la interfaz gráfica de usuario, que permite ver las bases de datos con sus objetos, la edición de datos en el interior de las tablas, análisis de datos con diagramas visuales, editor de SQL y temas o apariencias específicas si el usuario considera necesario cambiar los que vienen por defecto.[40]

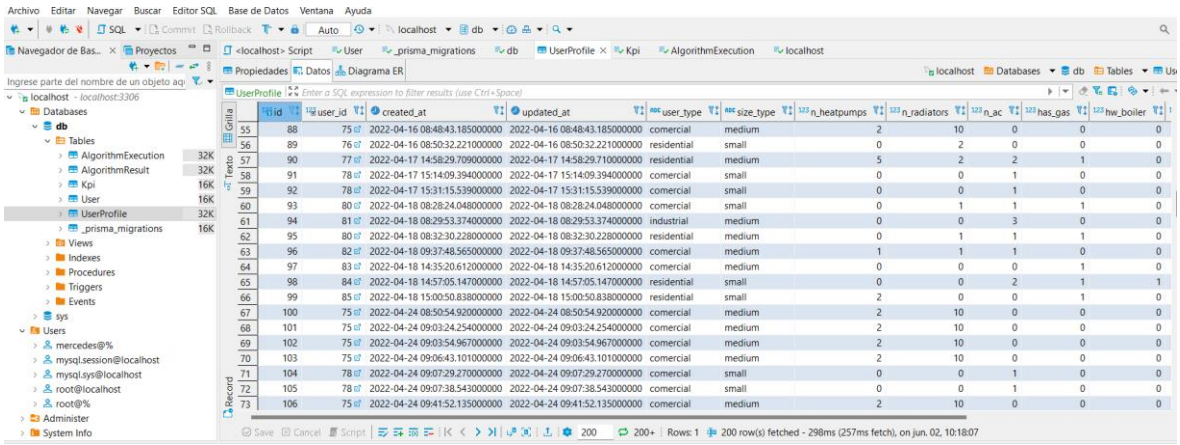

*Figura 4: DBeaver*

Como se puede ver en la Figura 4, DBeaver permite acceder a los datos que se ban guardando. Esto es muy útil porque permite comprobar que los datos se están guardando correctamente. Además, también se pueden editar los datos si fuera necesario.

# Capítulo 8. DEFINICIÓN DE LA BASE DE DATOS

Definir y crear una base de datos a mano no suele ser muy recomendable. Puede conllevar varios inconvenientes. Uno de ellos es el tema de la gestión de cambios, por ello es una buena práctica la utilización de un ORM que permita tener un control de las diferentes versiones de la base de datos, este concepto, se denomina migración. Además, un ORM es una capa de abstracción sobre las bases de datos que haya por debajo, por lo que una consulta que se realice para MySQL, vale de igual forma para otros motores de bases de datos como postgres.

## *8.1 PRISMA*

En este caso, para la creación de la base de datos se ha decidido utilizar Prisma, que es un ORM de código abierto que simplifica drásticamente el modelado de datos, las migraciones y el acceso a datos para bases de datos SQL en Node.js y TypeScript. Generalmente, se utiliza en aplicaciones de *back-end* para enviar consultas a una base de datos. La utilización de este ORM permite a la API Rest que se ha desarrollado en Express ejecutar las consultas a la base de datos. [41]

## *8.2 DEFINICIÓN DE LAS TABLAS DE BASE DE DATOS EN PRISMA*

Una vez instalado y conectado prisma con la base de datos creada por medio de Docker, se definen las variables y relaciones de las tablas de la base de datos en el archivo *schema.prisma*.

Para el diseño de las tablas que se quieren utilizar de una forma más visual se ha utilizado la web [42]:

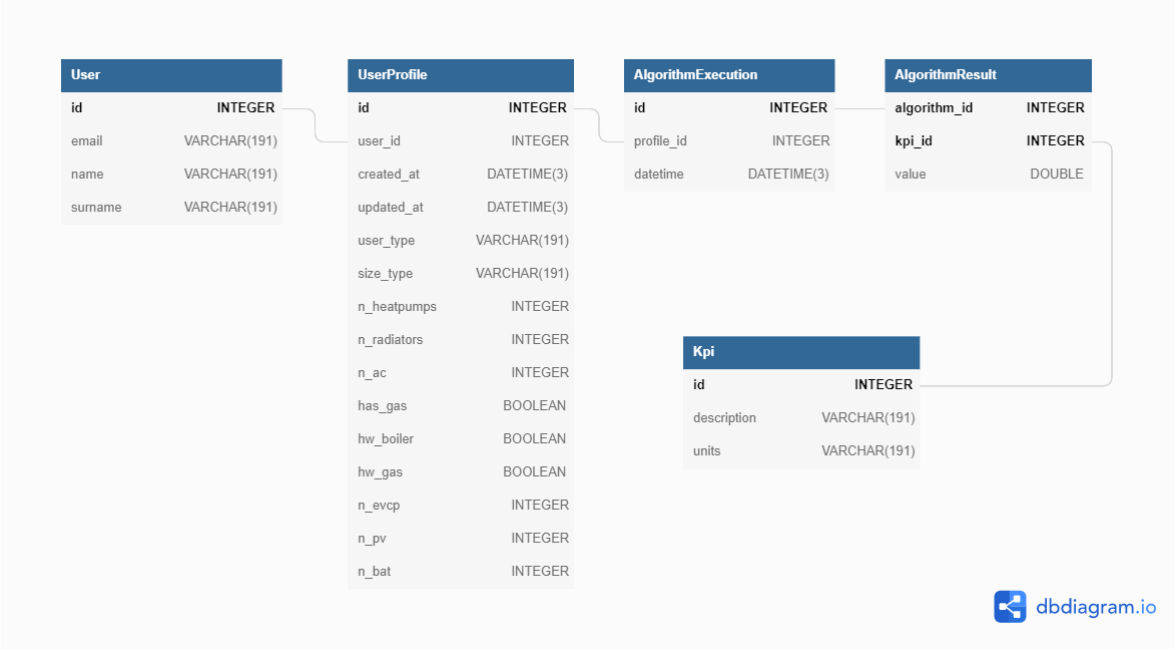


*Figura 5: Tablas de la base de datos* [42]

En la Figura 5, se pueden ver las diferentes tablas que configuran la base de datos y sus relaciones. En primer lugar, se encuentra la tabla *User*, cuyos parámetros son el id, email, *name* y *surname* (nombre y apellido). El id se define como un número entero y es la característica única que identifica al usuario. Además, este id se irá autoincrementando a medida que se van añadiendo usuarios en la base de datos. Los demás parámetros se definen como cadenas de caracteres. Estos parámetros son los que definen al usuario que se registra en la web.

A continuación, se encuentra la tabla *UserProfile* o perfiles de usuario, donde se recogen los resultados del formulario realizado en la web, cuyos parámetros son:

- Id: es un entero que define el perfil de usuario.
- user_id: es un entero que hace referencia al id del usuario comentado anteriormente, es decir, cada usuario, puede tener varios perfiles de usuario. Por ello, con esta característica, se hace referencia a qué usuario pertenece este perfil.

- created_at: es una característica de tipo fecha, que indica el momento en el que se ha creado el perfil. Esta característica está definida de forma que se rellena automáticamente por defecto con la fecha del momento actual.
- updated_at: es una característica de tipo fecha, que indica el momento en el que se ha actualizado el perfil. Esta característica está definida de forma que se rellena automáticamente por defecto con la fecha del momento actual.
- user_type: es una cadena de caracteres que indica el tipo de usuario que es, ya que se consideran tres tipos posibles (residencial, comercial, industrial).
- size_type: es una cadena de caracteres que indica el tamaño de la vivienda del usuario (pequeña, mediana, grande).
- n_heatpumps: es un entero que indica el número de bombas de calor del usuario.
- n_radiators: es un entero que indica el número de radiadores del usuario.
- n_ac: es un entero que indica el número de aires acondicionados del usuario.
- has_gas: es un booleano que indica si el usuario tiene gas o no.
- hw_boiler: es un booleano que indica si el usuario tiene un termo de agua caliente o no.
- hw_gas: es un booleano que indica si el usuario calienta el agua sanitaria con gas o no.
- n_evcp: es un entero que indica el número de puestos de recarga para vehículos eléctricos que tiene el usuario en su vivienda.
- n_pv: es un entero que indica el número de paneles solares que tiene el usuario en su vivienda.
- n_bat: es un entero que indica el número de baterías que tiene el usuario en su vivienda.

Además, se define la tabla *AlgorithmExecution* que contiene un id proprio (de tipo entero), el id del perfil del que se realiza la ejecución (de tipo entero) y la fecha con el momento en el que se ha realizado la ejecución (de tipo fecha).

La tabla KPI contiene un id identificativo (de tipo entero), una descripción (de tipo cadena de caracteres), que indica la recompensa que es y las unidades en las que se guarda esta recompensa (de tipo cadena de caracteres). Finalmente, la tabla *AlgorithmResult*, contiene el valor del resultado de una ejecución del algoritmo, de tipo número decimal, el id de la ejecución del algoritmo de tipo entero y el id de la KPI de tipo entero.

Este diseño realizado con tablas hay que pasarlo a código para definirlo en *schema.prisma*. La documentación para escribir este tipo de código se puede encontrar en [43]

```
model User {
  id       Int           @id @default(autoincrement())
  email    String
  name     String
  surname  String
  profiles UserProfile[]
}
```

*Código 12: Tabla User*

En el Código 12, se define la tabla de los usuarios con las características ya mencionadas. Es puede ver cómo a la derecha de los nombres se indica el tipo de variable que son. Además, todas las tablas deben tener un identificador único. En este caso se define la relación a la derecha del id y se define que se autoincremente por defecto.

```
model UserProfile {
  id                  Int                 @id @default(autoincrement())
  user_id             Int
  created_at          DateTime            @default(now())
  updated_at          DateTime            @default(now())
  user_type           String?
  size_type           String?
  n_heatpumps         Int?
  n_radiators         Int?
  n_ac                Int?
  has_gas             Boolean?
  hw_boiler           Boolean?
  hw_gas              Boolean?
  n_evcp              Int?
  n_pv                Int?
  n_bat               Int?
  user                User                @relation(fields: [user_id], references: [id])
  algorithm_execution AlgorithmExecution?
}
```

*Código 13: Tabla UserProfile*

En el Código 13, también se define in id para identificar cada perfil de usuario. Además, la fecha de creación y modificación de la tabla se ha definido para que se defina por defecto con el momento actual en el que se crea o modifique el perfil. Otro aspecto interesante, es la relación que se ha creado con la tabla *User*, por la cual un usuario puede tener asociados varios perfiles, lo que se indica en la tabla *User* con corchetes de vector, si por ejemplo realiza el formulario varias veces. Sin embargo, un perfil solo puede estar asociado a un usuario, por ello no se indican los corchetes de vector.

Finalmente, otra característica importante de esta tabla es el signo de interrogación que se encuentra en las diferentes características de usuario. Esto indica que, para crear un perfil de usuario, no es necesario definir estas características, es decir, son opcionales. Sin embargo, el id, el id del usuario al que se asocia el perfil y las fechas sí son parámetros obligatorios para la creación del perfil.

```
model AlgorithmExecution {
  id          Int                @id @default(autoincrement()) // auto-increment
  profile_id  Int                @unique
  datetime    DateTime           @default(now())
  profile     UserProfile        @relation(fields: [profile_id], references: [id])
  results     AlgorithmResult[]
}
```

*Código 14: Tabla AlgorithmExecution*

En este Código 14, se definen numerosas relaciones. En primer lugar, al igual que en las dos tablas anteriores, en él se define un id. A continuación, se puede ver cómo se indica con la función *unique* que sólo puede existir una ejecución de usuario para cada perfil. Esto se debe a que no tiene sentido que, a partir de unos mismos datos, se puedan obtener soluciones diferentes de un mismo algoritmo. Finalmente, se puede ver que esta tabla también se relaciona con el resultado del algoritmo.

```
model AlgorithmResult {
  algorithm_id       Int
  kpi_id             Int
  value              Float
  algorithmExecution AlgorithmExecution @relation(fields: [algorithm_id], references: [id])
  kpi                Kpi                @relation(fields: [kpi_id], references: [id])
```

```
  @@id([algorithm_id, kpi_id])
}
```

*Código 15: Tabla AlgorithmResult*

En el Código 15, se puede ver que es la única tabla en la que no se define un id. Sin embargo, siempre es necesaria tener una relación que haga de elemento identificativo de los elementos de la tabla. En este caso esta relación puede verse en la última línea, en la que se indica que sólo existirá un resultado de algoritmo para una ejecución y una *Kpi* determinadas, es decir, para una ejecución de algoritmo, solo existirá un resultado asociado a una *Kpi* determinada. El resultado del algoritmo se define como *float*, porque puede ser decimal.

```
model Kpi {
  id                Int                @id @default(autoincrement())
  description       String
  units             String
  algorithmResult AlgorithmResult[]
}
```

*Código 16: Tabla Kpi*

En el Código 16, se puede ver la definición de la última tabla que contiene un id y está asociado al resultado del algoritmo como ya se ha comentado anteriormente.

## 8.3 PRISMA Y SU ADAPTACIÓN A OTRAS BASES DE DATOS

Al principio del archivo schema.prisma, se define la base de datos en la que se quieren construir las tablas mencionadas.

```
datasource db {
  provider = "mysql"
  url      = env("DATABASE_URL")}
```

*Código 17: Definición de motor de base de datos en Prisma*

Como se puede ver en el Código 17, se define el *provider*, con el motor de base de datos elegido, es decir, en este caso, MySQL. Sin embargo, el diseño de la base de datos no se ve afectado de ninguna manera. La *url*, la obtiene de un archivo privado llamado *.env*, en el que se encuentran el usuario y la contraseña de la base de datos.

Para crear la base de datos por primera vez, se ejecuta en la terminal el comando:

```
npx prisma migrate dev –name init_first_migration
```

*Código 18: Comando para realizar la primera migración de prisma*

Con el comando Código 18, se crea un archivo llamado *migration.sql*, que es el archivo que se ejecuta en la base de datos para crear el modelo. Este archivo se puede encontrar en el Anexo I.I: archivo migration.sql.

Como se ha comentado, en este caso se ha decidido definir que la base de datos que se estaba creando, era de MySQL, pero si se quisiera cambiar a otro motor de base de datos, con indicarlo en *provider* y adaptar el archivo *.env*, se ejecuta de nuevo la migración y se crea el archivo específico para esa base de datos, evitando definir todas las tablas de nuevo.

Esta es una de las grandes ventajas de la utilización de un ORM, es como una capa de abstracción, que facilita la programación y además aporta la flexibilidad para adaptarse a otros lenguajes.

## *8.4 PRISMA CLIENT*

Una vez creada la base de datos, se genera el cliente de Prisma. El cliente de Prisma generará todos los ficheros TypeScript relacionados con la interacción con la base de datos a partir del modelo de base de datos definido en *schema.prisma*, que proporcionarán autocompletado a la hora de construir consultas. Para ello se ejecuta el comando que se puede leer en Código 19:

```
npx prisma generate
```

*Código 19: Comando para la instalación de prisma client*

La utilización de Prisma Client facilita enormemente al desarrollador la elaboración de consultas a la base de datos.

# Capítulo 9. DEFINICIÓN DEL SERVIDOR

El servidor creado, como ya se ha comentado, se basa en el *framework* Express.js e interactúa con la base de datos a través del cliente de prisma, generado en el punto anterior. Es necesario importar prisma en el archivo del servidor para que funcione correctamente, como puede verse en el Código 20.

```
import { PrismaClient } from "@prisma/client";
const prisma = new PrismaClient();
```

*Código 20: Introducción de prisma en el servidor*

## 9.1 CÓDIGOS DE RESPUESTA HTTP

Los códigos de respuesta HTTP indican si se ha realizado correctamente una petición al servidor. Existen numerosos códigos que se pueden encontrar en [44]. Para el desarrollo de este servidor se han utilizado únicamente cuatro de estos códigos, que son los más habituales.

```
app.get("/users",async (req, res) => {
    const users = await prisma.user.findMany();
    if(¡users) return res.status(400).send('users not found');
    res.status(200).send(users);
  }
);
```

*Código 21: Ejemplo de petición de usuarios con respuesta 200*

Como se puede ver en el Código 21, se realiza una petición de datos de tipo *get* a la ruta */usuarios*, si se encuentra esa ruta, se devuelve un vector con todos los usuarios con la respuesta 200 que indica que la petición es correcta, y si no, se devuelve un mensaje de error con la respuesta 400.

Las respuestas que se han utilizado durante el desarrollo del servidor son:

- 200 indica que la solicitud ha sido un éxito.
- 201 indica que la solicitud ha sido un éxito y se ha creado el recurso correctamente.

- 400 indica que el servidor no ha podido interpretar la solicitud, pudiendo tratarse de un error de sintaxis.
- 404 indica que no se ha encontrado la petición.

Es una buena práctica introducir un mensaje junto con estas respuestas para poder identificar más fácilmente los errores en el código.

## 9.2 PRISMA Y EL SERVIDOR

A la hora de escribir las peticiones del servidor hay que tener en cuenta algunos detalles. En el siguiente código (Código 22), por ejemplo, se realiza la petición de usuarios a la base de datos.

```
app.get("/users",async (req, res) => {
    const users = await prisma.user.findMany();
    res.status(200).send(users);
  }
);
```

*Código 22: Petición de vector de usuarios*

Como puede verse en el Código 22, se utiliza *prisma client* como enlace con la base de datos. Prisma permite realizar numerosas peticiones en la base de datos, en este caso se ha elegido *findMany*, que busca todos los usuarios que cumplan una determinada condición. Como no se impone ninguna condición, se devolverán todos los usuarios en forma de vector.

Un detalle muy importante es el tema de la sincronía. Actualmente, se está trabajando con un servidor que se ejecuta en el ordenador y una base de datos, que en este caso también se ejecuta en el ordenador, pero que lo podría estar haciendo en otro sitio distinto como por ejemplo en la nube. El hecho de pedir los datos no es instantáneo si no que tarda un tiempo, por ello es necesario utilizar el término *await*. Esto hace que el código se quede parado en ese punto hasta que la base de datos le responda, sin bloquear la ejecución de comandos en paralelo. De esta forma si se realiza un *await* que tarda unos 20 segundos, se pueden seguir haciendo peticiones al mismo servidor y se va a obtener una respuesta. Además, para que el comando funcione correctamente, la función debe definirse como asíncrona, que quiere decir que tarda lo que necesite en responder.

Para probar que todos los parámetros llegan correctamente al servidor se utiliza Postman, que es una herramienta de desarrollo, que ayuda a crear, probar y modificar una API. Tiene la capacidad de realizar varios tipos de solicitudes HTTP y guardar entornos para su desarrollo posterior.[45] Se utiliza para comprobar que las consultas al servidor se ejecutan correctamente y devuelven los parámetros esperados.

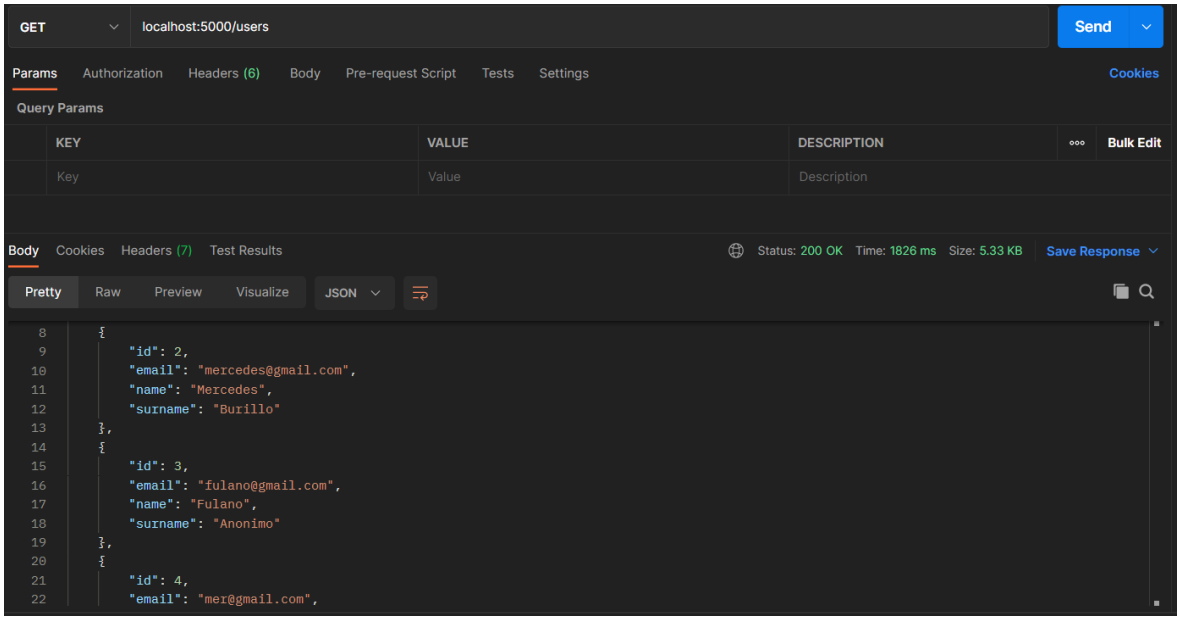


*Figura 6: Petición en Postman*

En la Figura 6, se puede ver que el estado de la respuesta es un 200, lo que indica que la petición se ha realizado correctamente y se puede ver cómo llegan todos los usuarios de la base de datos en forma de vector. La utilización de Postman ha sido muy útil para ir probando que todas las peticiones que se creaban funcionaran correctamente.

## 9.3 PETICIONES DEL SERVIDOR

Para el desarrollo del proyecto, se han tenido que ir definiendo numerosas peticiones HTTP, con diferentes respuestas.

### *9.3.1 PETICIÓN GET*

La primera de ellas es una petición de tipo *get*, es decir, para obtener información de la base de datos.

```
app.get(
  "/users",
  query("email").isEmail().withMessage("is not an email"),
  async (req, res) => {
    const { email } = req.query;
    const users = await prisma.user.findMany({
      where: {
        email,
      },
    });
    res.status(200).send(users);
  }
);
```

*Código 23: Petición get users*

La función que se ve en el Código 23, permite buscar los usuarios que se encuentran en la base de datos. Además, se ha añadido la posibilidad de buscar un usuario concreto, introduciendo un email.

### *9.3.2 PETICIÓN POST*

Si lo que se quiere hacer es introducir información nueva en la base de datos, la función que hay que utilizar es la petición *post*.

```
app.post(
  "/users",
  async (req, res) => {
    const user = await prisma.user.create({
      data: req.body,
    });
    res.status(201).send(user);
  }
);
```

*Código 24: Petición post users*

Como se puede ver en el Código 24, se realiza la función para la creación de nuevos usuarios en la base de datos. Si existe algún problema se devuelve un mensaje de error.

### *9.3.3 PETICIÓN PATCH*

También existe una forma de modificar los datos que hay en la base de datos, es decir, no siempre que se quieran escribir datos es necesario hacer un post. Este es el caso de la función *patch*.

```
app.patch(
  "/usersProfiles/:profileId",async (req, res) => {
    console.log(req.body);
    //   // Finds the validation errors in this request and wraps them in an object with
handy functions
    const { profileId } = req.params;
    const nMatchingProfiles = await prisma.userProfile.count({
      where: { id: parseInt(profileId) },
    });

    if (nMatchingProfiles < 1) return res.status(404).send();
    const updatedProfile = await prisma.userProfile.update({
      where: { id: parseInt(profileId) },
      data: req.body,
    });

    res.status(201).send(updatedProfile);
  }
);
```

*Código 25: Petición patch userProfile*

En el Código 25, se expresa la petición *patch*, para la modificación de los datos de un perfil de usuario dado un determinado id.

## *9.4 EXPRESS VALIDATOR*

A la hora de introducir los datos en la base de datos hay que tener mucho cuidado y suponer que el usuario puede introducir datos que no son correctos, como introducir un número en lugar de un nombre. Para ello, se utiliza la herramienta *express-validator*, que es un middleware de Express[46].

A la hora de realizar todas las peticiones es conveniente realizar unas validaciones de que los parámetros que llegan sean correctos.

```
app.post(
  "/users",
  body("id").not().exists().withMessage("can not pass the id"),
  body("email").isEmail().withMessage("is not an email"),
```

```
  body("name").isString().withMessage("is not a string"),
  body("surname").isString().withMessage("is not a string"),
  async (req, res) => {
    // Finds the validation errors in this request and wraps them in an object with handy
functions
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }
    //const { email, name, surname } = req.body;
    //req.body;
    const user = await prisma.user.create({
      data: req.body,
    });
    res.status(201).send(user);
  }
);
```

*Código 26: Validación de los parámetros que llegan en el post*

En el Código 26, se utiliza la función *body* de la librería *express-validator*, para realizar algunas comprobaciones. En primer lugar, se indica que no se puede pasar el parámetro id, ya que este se configura automáticamente de forma incremental. A continuación, se pide que el email tenga la sintaxis de un email y que el nombre y apellido sean cadenas de caracteres.

Además, es muy buena práctica la introducción de un mensaje adicional, en el que se explique brevemente el error que se está cometiendo, para facilitar la detección de errores.

Si el resultado de la validación indica que no hay errores se ejecuta el resto del código, pero si se encuentran errores se devuelve una respuesta 400 con los mensajes de error que se han producido.

```
app.patch(
  "/usersProfiles/:profileId",
  body("id").not().exists().withMessage("can not pass the id"),
  body("user_id").optional().isInt().withMessage("is not an int"),
  body("user_type").optional().isString().withMessage("is not a string"),
  body("size_type").optional().isString().withMessage("is not a string"),
  body("n_heatpumps").optional().isInt().withMessage("is not an int"),
  body("n_radiators").optional().isInt().withMessage("is not an int"),
  body("n_ac").optional().isInt().withMessage("is not an int"),
  body("has_gas").optional().isBoolean().withMessage("is not a boolean"),
  body("hw_boiler").optional().isBoolean().withMessage("is not a boolean"),
  body("hw_gas").optional().isBoolean().withMessage("is not a boolean"),
  body("n_evcp").optional().isInt().withMessage("is not an int"),
  body("n_pv").optional().isInt().withMessage("is not an int"),
  body("n_bat").optional().isInt().withMessage("is not an int"),
  async (req, res) => {
    console.log(req.body);
```

```
    //   // Finds the validation errors in this request and wraps them in an object with
handy functions

    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }

    const { profileId } = req.params;

    const nMatchingProfiles = await prisma.userProfile.count({
      where: { id: parseInt(profileId) },
    });


    if (nMatchingProfiles < 1) return res.status(404).send();

    const updatedProfile = await prisma.userProfile.update({
      where: { id: parseInt(profileId) },
      data: req.body,
    });

    res.status(201).send(updatedProfile);
  }
);
```

*Código 27: Validación de parámetros en patch*

En el caso de Código 27, como lo que se hace es modificar algunos datos, hay que indicar en la validación que los parámetros introducidos son opcionales, ya que, si no se indicara, y no llegaran esos parámetros, se entendería como un error.

## 9.5 TYPESCRIPT

TypeScript es un lenguaje de programación desarrollado y mantenido por Microsoft y es un superconjunto sintáctico de JavaScript. Agrega escritura estática opcional a JS. TypeScript se diseñó para ayudar a los desarrolladores a crear aplicaciones grandes en JavaScript. Todo el software JavaScript existente también es software TypeScript válido.[47]

Este lenguaje de programación está ganando mucha popularidad ya que tiene muchos beneficios. Entre ellos, destacan los tipos, lo que facilita el control de los parámetros y variables de funciones. Esto facilita la adaptación de programadores que vienen de otros lenguajes de programación como Java o C++..[47]

Además, permite la detección de errores en el paso anterior a la compilación, lo que ahorra mucho tiempo a la hora de programar en este lenguaje.

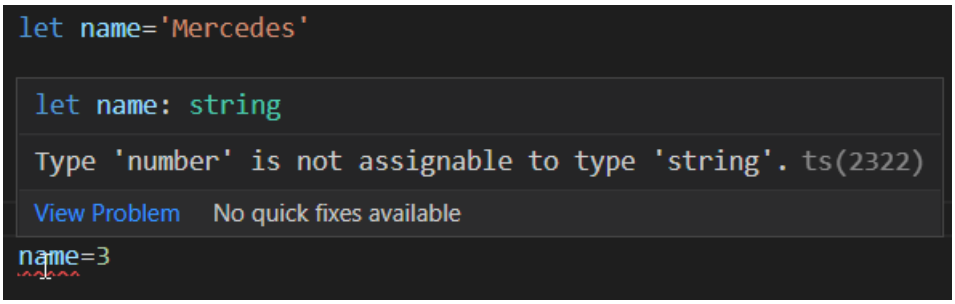


*Figura 7: Tipo de variable TypeScript*

En la Figura 7, se puede ver como TypeScript tiene en cuenta el tipo de variable que se define. Por lo tanto, cuando un nombre se define como una cadena de caracteres, se produce un error si se intenta asignar un número.

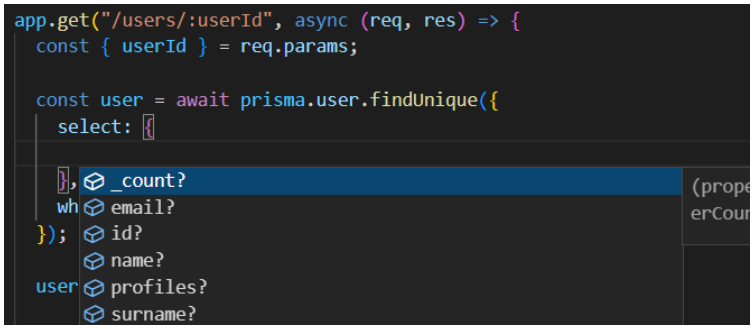


*Figura 8: Ejemplo 1 de TypeScript*

A la hora de hacer una petición puede ser que únicamente interesen unas propiedades de la tabla *user*. Por ello, como se puede ver en la Figura 8, se utiliza *select* y es el programa el que sugiere los parámetros de las tablas que existen.

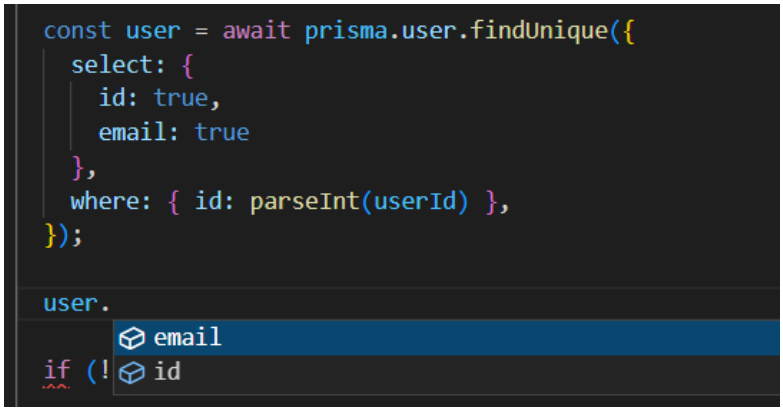


*Figura 9: Ejemplo 2 de TypeScript*

Como se puede ver en la Figura 9, si se hace que *user*, tenga únicamente los parámetros id y email. Si a continuación, se quiere usar esa variable, gracias a la combinación de Prisma y TypeScript, sólo se pueden usar las propiedades id y email. Si se escribiera *name* a continuación del punto, se obtendría un error, porque no es una propiedad del usuario.

Es decir, TypeScript es una herramienta muy potente porque es capaz de interpretar lo que se está haciendo y saber qué campos son los que se necesitan dependiendo de la consulta. Gracias a esta herramienta, se han evitado numerosos errores de programación.

## 9.6 ALGORITMO

### 9.6.1 PETICIONES DE ALGORITMO

Para la ejecución del algoritmo es importante la petición *post* de la ejecución del algoritmo, en la que se va a llamar al algoritmo que trabaja en el servidor. Como se puede ver en el Código 28, esta petición es más compleja que en los casos anteriores.

```
app.post(
  "/algorithmsExecutions",
  body("profile_id").isInt().withMessage("is not a int"),
  body("id").not().exists().withMessage("cannot pass the id"),
  async (req, res) => {
    const { profile_id } = req.body;

    const errors = validationResult(req);

    if (!errors.isEmpty()) {
```

```
      return res.status(400).json({ errors: errors.array() });
    }

    const nExecutionsWithThisProfile = await prisma.algorithmExecution.count({
      where: { profile_id },
    });

    if (nExecutionsWithThisProfile !== 0)
      return res
        .status(400)
        .send(
          `An algorithm execution with profile id: ${profile_id} already exists`
        );

    const newAlgorithmExecution = await prisma.algorithmExecution.create({
      data: { ...req.body, datetime: new Date() },
    });

    // 1. llamo a base de datos con await y pido el UserProfileId que me han pasado
    const userProfile = await prisma.userProfile.findUnique({
      where: { id: parseInt(profile_id) },
    });

    const algorithmResults = findClosestMatch(
      newAlgorithmExecution.id,
      userProfile
    );
    // 3. inserto algorithm results
    const newAlgorithmResult = await prisma.algorithmResult.createMany({
      data: algorithmResults,
    });
    const algorithmWithResults = await prisma.algorithmExecution.findFirst({
      where: { id: newAlgorithmExecution.id },
    });

    // algorithmWithResults.results

    res.status(201).send(algorithmResults);
  }
);
```

*Código 28: Petición post de ejecución de algoritmo*

En el Código 28, se puede ver, en primer lugar la validación de los parámetros que llegan con Express-validator. A continuación, si no hay ningún error, se comprueba que no exista ya una ejecución de algoritmo para ese perfil de usuario, ya que solo puede existir una ejecución para cada perfil. Si ya existe, se proporciona un mensaje de error.

Si todo ha funcionado correctamente, se crea una entrada nueva en la base de datos para el id del perfil de usuario proporcionado y se buscan los datos de ese perfil en la base de datos. Con los datos del perfil y el id de la ejecución del algoritmo, se llama a la función en la que se ejecutará el algoritmo, del que se obtendrán los resultados.

Finalmente, los resultados obtenidos del algoritmo se introducen también en la base de datos.

La ventaja de realizar el algoritmo de una forma independiente es la flexibilidad que aporta. En primer lugar, a la hora de ir desarrollando el proyecto, fue bueno, porque se pudieron establecer unos datos por defecto, que no eran los reales pero que permitían comprobar que todo funcionara correctamente. Además, si más adelante se decidiera cambiar el algoritmo o las recompensas que se proporcionan, se podría hacer sin muchas dificultades.

### *9.6.2 FUNCIONAMIENTO DEL ALGORITMO*

El sistema de gamificación que se diseña es personalizado para cada usuario. Por ello, se ha determinado que las recompensas que recibe el jugador, en función de los dispositivos de la vivienda, se ajusten a cada persona en concreto. Las recompensas que se obtienen del algoritmo son el ahorro en euros, el ahorro en energía en kWh y el ahorro en kilogramos de $CO_2$, que se obtendría al formar parte del proyecto y ayudar a la flexibilidad durante un año completo.

Para realizarlo, se empieza con cinco modelos que se proporcionan desde Stemy de cinco usuarios de los que se sabe las recompensas que obtendrían para los dispositivos de los que disponen. En primer lugar, una vez recibidos los datos del nuevo usuario, se hace una clasificación del usuario en uno de los cinco modelos.

```
//Funcion para clasificar los datos en uno de los modelos
function classificationModel(userProfile: UserProfile): number {
  let select = 1;
  let ac = 0;
  let heatpumps = 0;
  let radiators = 0;
  let evcp = 0;
  let hwBoiler = 0;
  if (userProfile.n_ac > 0) {
    ac = 1;
  }
  if (userProfile.n_heatpumps > 0) {
    heatpumps = 1;
  }

  if (userProfile.n_radiators > 0) {
    radiators = 1;
  }
  if (userProfile.n_evcp > 0) {
    evcp = 1;
```

```
  }
  if (userProfile.hw_boiler) {
    hwBoiler = 1;
  }
  if (userProfile.has_gas || userProfile.hw_gas) {
    select = 1;
    if (ac > 0) {
      select = 2;
    }
    if (evcp + radiators + heatpumps + hwBoiler + ac > 0) {
      select = 3;
      if (userProfile.n_pv + userProfile.n_bat > 0) {
        select = 5;
      }
    } else if (userProfile.n_pv + userProfile.n_bat > 0) {
      select = 4;
    }
    if (
      userProfile.n_bat !== 0 ||
      userProfile.n_pv !== 0 ||
      userProfile.n_evcp !== 0
    ) {
      select = 4;
    }
  } else {
    select = 3;
    if (userProfile.n_pv !== 0 || userProfile.n_bat !== 0) {
      select = 5;
    }
  }

  return select;
}
```

*Código 29: Función de clasificación del algoritmo*

Como se puede ver en Código 29, en función de los dispositivos que tiene el usuario se clasifica en un modelo o en otro y se devuelve el resultado.

Una vez se ha clasificado el perfil, se pasa a hacer una ponderación de las recompensas en función del tamaño de la vivienda y del número de dispositivos que tiene el usuario, ya que no es lo mismo una casa pequeña con dos radiadores que una de mayor tamaño con veinte, por ello, el número de dispositivos es una variable para tener en cuenta.

Debido a que la cantidad de energía y el ahorro en euros, no es proporcional exactamente, se han realizado dos funciones de ponderación de los resultados diferentes, para obtener resultados más cercanos a la realidad.

```
function eurosCalculate(userProfile: UserProfile, select: number): number {
  let value = 0;
```

```
  let factor = 1;
  if (userProfile.size_type === "medium") {
    factor += 0.05;
  } else if (userProfile.size_type === "big") {
    factor += 0.1;
  }
  let ac = 0;
  let heatpumps = 0;
  let radiators = 0;
  let evcp = 0;
  let hwBoiler = 0;
  if (userProfile.n_ac > 0) {
    ac = 1;
  }
  if (userProfile.n_heatpumps > 0) {
    heatpumps = 1;
  }

  if (userProfile.n_radiators > 0) {
    radiators = 1;
  }
  if (userProfile.n_evcp > 0) {
    evcp = 1;
  }
  if (userProfile.hw_boiler) {
    hwBoiler = 1;
  }
  //si se tiene más de un aire acondicionado se aumenta el factor
  if (select === 2) {
    if (userProfile.n_ac > 1) factor += 0.04 * userProfile.n_ac;
  }

  if (select === 3) {
    factor += (radiators + evcp + ac + heatpumps + hwBoiler - 4) * 0.05;
    factor += 0.01 * (userProfile.n_radiators - 3);
    factor += 0.01 * (userProfile.n_evcp - 1);
    factor += 0.01 * (userProfile.n_heatpumps - 1);
    factor += 0.01 * (userProfile.n_ac - 1);
  }
  if (select === 4) {
    factor += 0.12 * (userProfile.n_pv + userProfile.n_bat - 3);
  }
  if (select === 5) {
    factor += 0.12 * (userProfile.n_pv + userProfile.n_bat - 11);
    console.log(factor);
    factor += 0.01 * (userProfile.n_radiators - 3);
    factor += 0.01 * (userProfile.n_evcp - 1);
    factor += 0.01 * (userProfile.n_heatpumps - 1);
    factor += 0.01 * (userProfile.n_ac - 1);
  }
  for (let i = 0; i < CaseData.length; i++) {
    if (select === CaseData[i].select) {
      value = CaseData[i].valueEuros * factor;
      if (value < 0) value = 0;
    }
  }

  return value;
}
```

*Código 30: Función para calcular el ahorro en euros en el algoritmo*

En el Código 30, se puede ver que partiendo de un factor 1, para el caso de que el usuario tenga los mismos dispositivos que el caso base, este factor se va cambiando en función del número de aparatos específicos del perfil que ha llegado. Además, al final, si el resultado del valor obtenido es menor de cero euros, este pasa a ser cero, ya que no tiene sentido un resultado negativo en este contexto. La mínima recompensa que puede obtener un usuario es de cero euros.

```
function kwhCalculate(userProfile: UserProfile, select: number): number {
  let value = 0;
  let factor = 1;
  if (userProfile.size_type === "medium") {
    factor += 0.05;
  } else if (userProfile.size_type === "big") {
    factor += 0.1;
  }
  let ac = 0;
  let heatpumps = 0;
  let radiators = 0;
  let evcp = 0;
  let hwBoiler = 0;
  if (userProfile.n_ac > 0) {
    ac = 1;
  }
  if (userProfile.n_heatpumps > 0) {
    heatpumps = 1;
  }

  if (userProfile.n_radiators > 0) {
    radiators = 1;
  }
  if (userProfile.n_evcp > 0) {
    evcp = 1;
  }
  if (userProfile.hw_boiler) {
    hwBoiler = 1;
  }
  //si se tiene mas de un aire acondicionado se aumenta el factor
  if (select === 2) {
    if (userProfile.n_ac > 1) factor += 0.04 * userProfile.n_ac;
  }

  if (select === 3) {
    factor += (radiators + evcp + ac + heatpumps + hwBoiler - 4) * 0.05;
    factor += 0.011 * (userProfile.n_radiators - 3);
    factor += 0.011 * (userProfile.n_evcp - 1);
    factor += 0.011 * (userProfile.n_heatpumps - 1);
    factor += 0.011 * (userProfile.n_ac - 1);
  }
  if (select === 4) {
    factor += 0.121 * (userProfile.n_pv + userProfile.n_bat - 3);
  }
```

```
  if (select === 5) {
    factor += 0.12 * (userProfile.n_pv + userProfile.n_bat - 11);
    console.log(factor);
    factor += 0.009 * (userProfile.n_radiators - 3);
    factor += 0.009 * (userProfile.n_evcp - 1);
    factor += 0.009 * (userProfile.n_heatpumps - 1);
    factor += 0.009 * (userProfile.n_ac - 1);
  }
  for (let i = 0; i < CaseData.length; i++) {
    if (select === CaseData[i].select) {
      value = CaseData[i].valueKwh * factor;
      if (value < 0) value = 0;
    }
  }
  //EL GAS NO AFECTA AL AHORRO EN EUROS
  return value;
}
```

*Código 31: Función para calcular el ahorro en kWh en el algoritmo*

Como se puede ver en el Código 31, la estructura es muy similar a la realizada en el Código 30, sin embargo, los valores del factor se han ajustado para llegar a un mayor acercamiento a la realidad.

De esto se deduce que el código es muy flexible y permite adaptarse a nuevos cambios, como la introducción de nuevos dispositivos o el ajuste de algunos parámetros concretos.

El resultado de la KPI de ahorro de $CO_2$, es directamente proporcional al ahorro de kWh. A través de la web [48], se ha establecido que el ahorro de 1 kWh implica el ahorro de 140 g de $CO_2$ en España. Sin embargo, este valor realmente va cambiando en cada momento en la realidad. Por ello, el cálculo de la última KPI es mucho más sencillo y no se necesita una función específica para llegar al resultado.

Una función interesante que se ha utilizado para el desarrollo del algoritmo es “enum”, como se puede ver a continuación en el Código 32. Las recompensas o KPI estaban se definen con un id en la base de datos. Este id es un número, por lo que puede ser complicado, si se trabaja con un gran número de variables, recordar qué id, corresponde a cada KPI. Por ello, esta función permite asociar a estos números un nombre, que permite hacer referencia a cada una de ellas de una forma mucho más intuitiva.

```
enum Kpi {
  AnnualKwh = 1,//kwh saved in a year
  CO2KgSaved, //it will be the kg of co2 saved
```

```
  EurosSaved, //euroSaved será el 3
}
```

*Código 32: Utilización de "enum" para las recompensas*

## *9.7 OTRAS PETICIONES*

A la hora de desarrollar el servidor se han utilizado otras peticiones, que no aportan información nueva sobre los desarrollos del código, pero que se pueden encontrar en el Anexo I.II: archivo index.ts (servidor)

# Capítulo 10. CONEXIÓN SERVIDOR-WEB

El siguiente paso del proyecto es hacer que la web se comunique con el servidor para poder acceder a la información de la base de datos.

## *10.1 DOCUMENTACIÓN DE LA API*

Para ello, lo primero que se hizo, fue documentar la API, es decir, se realiza un registro de las peticiones que se pueden hacer al servidor y de las respuestas que se van a obtener.

La API se ha documentado en Apicurio Studio, que es un entorno de diseño creada principalmente para admitir el desarrollo de API REST que prioriza el diseño con estándar OpenApi [49]. La documentación de una API incluye las instrucciones sobre cómo usar e integrar de manera efectiva las funcionalidades de una API.

Además, genera muchas ventajas como la posibilidad de incluir herramientas de generación de código para crear funciones en React que puedan comunicarse con la API, conociendo las respuestas y los argumentos. Además, la documentación favorece la adaptación a otros lenguajes de programación, por lo que hace que el trabajo desarrollado se pueda hacer más flexible en un futuro.[50]

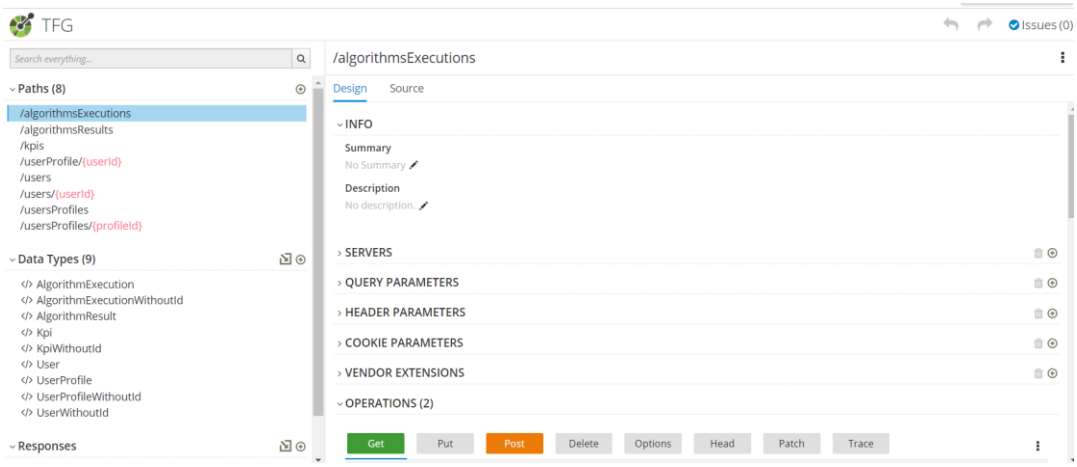


*Figura 10: Apicurio*

El funcionamiento de Apicurio es sencillo. Consiste en indicar las diferentes rutas que tiene la API con los tipos de argumentos y respuestas que se generan. En la Figura 10, se pueden ver las diferentes rutas que se han documentado y los tipos de argumentos o respuestas utilizados.

Algo interesante a la hora de crear los tipos es la posibilidad de introducir un ejemplo, para que automáticamente, se deduzcan los parámetros y los tipos de variables que son, como puede verse en la Figura 11.

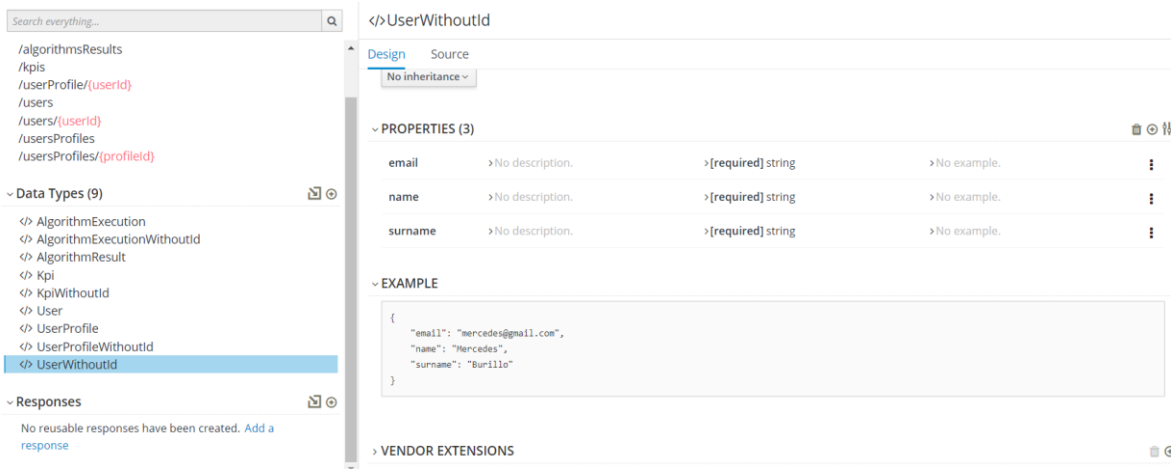


*Figura 11: Ejemplos para tipos en Apicurio*

## *10.2 INSTALACIÓN DE REDUX RTX*

Para enlazar la API con React, se utiliza Redux RTK. Redux es un contenedor de estado diseñado para escribir aplicaciones con JavaScript. Se utiliza como herramienta para la gestión de estados en React pero puede ser utilizado en otras bibliotecas de JavaScript. Con Redux, el estado de la web se mantiene almacenado y cada componente puede acceder al estado que necesite de este almacén.[40][51]

RTK Query es una biblioteca de Redux cuyo objetivo principal es generar clientes de APIs que permitan obtener y almacenar datos en caché para su aplicación web y utiliza opciones de configuración avanzadas para manejar sus necesidades de una forma más flexible. La memoria caché hace que la web sea más rápida ya que se pueden evitar consultas, si estas se han realizado anteriormente con los mismos parámetros de búsqueda.[52] De esta forma se consigue que la base de datos y el servidor tengan menos carga computacional y que la web del usuario se ejecute más fluida.

## *10.3 ENLAZAR API Y REDUX RTK*

Uno de los archivos de configuración de Redux y la API es el que se puede ver en Código 33:

```
import { ConfigFile } from "@rtk-query/codegen-openapi";
const tfgConfig: ConfigFile = {
  schemaFile: "./tfg-api-documentation.yml",
  apiFile: "../redux/apis/emptyTfgApi.ts",
  apiImport: "emptyTfgApi",
  outputFile: "../redux/apis/tfgApi.ts",
  exportName: "tfgApi",
  hooks: true,
};
export default tfgConfig;
```

*Código 33: Archivo de configuración openapi-tfg-config.ts*

En este Código 33, se puede ver cómo se indica el archivo del que se tiene que obtener la información de la API, tfg-api-documentation.yml, que es el código que se extrae directamente desde Apicurio.

```
npm run apidoc
```

*Código 34: Comando para la generación de archivo tfgApi*

Ejecutando el Código 34, se genera el archivo tfgApi, basado en el código proporcionado por la documentación de la Api en Apicurio. En este nuevo archivo se registran todas las funciones, respuestas y argumentos que se necesitan para que todo funcione correctamente. Estos archivos se pueden encontrar en el ANEXO II: Documentación Api y Redux

Otra ventaja de la documentación de la API y el uso de Redux RTK es que tienen soporte para TypeScript, lo que ayuda a la generación de código de una manera mucho más intuitiva con la sugerencia de parámetros que son necesarios a la hora de realizar las consultas.

## *10.4 UTILIZACIÓN DE LOS HOOKS EN REACT*

Como ya se ha comentado, con la documentación de la API y Redux, se consiguen generar hooks que se pueden utilizar en el desarrollo de la web de React. Estas funciones se pueden ver en el Código 35 a continuación:

```
export const {
    useGetUsersQuery,
    useCreateUserMutation,
    useGetUserQuery,
    useGetUsersProfilesQuery,
    useCreateUserProfileMutation,
    useChangeUsersProfilesMutation,
    useGetUserProfileQuery,
    useGetAlgorithmExecutionQuery,
    useCreateAlgorithmExecutionMutation,
    useGetAlgorithmsResultsQuery,
    useCreateAlgorithmsResultsMutation,
    useGetKpisQuery,
    useCreateKpiMutation
} = injectedRtkApi
```

*Código 35: Funciones generadas por Redux RTK*

Un ejemplo de utilización de estas funciones es el que se utiliza para identificar si un usuario ya ha sido registrado o no en la web. Para ello, se necesita hacer una consulta a la base de datos, buscando ese usuario.

```
const { data: matchingUsers, isFetching } = useGetUsersQuery(
        { email: user?.email },
        { skip: !user }}
```

*Código 36: Uso de useGetUsersQuery*

Como se puede ver en el Código 36, se busca si el usuario registrado existe ya en la base de datos. Para comprobar si el usuario existe se elige el email como característica de búsqueda, ya que un email es único para una persona, es decir, no puede haber más de una persona con un mismo email. Además, la propiedad *skip* permite controlar en qué momento se lanza la consulta, por lo tanto, esta petición se hace en el momento en el que *user*, deja de ser indefinido, es decir, en el momento en el que *user* pasa a contener la información de un usuario.

La posibilidad de poder elegir el momento en el que se realizan las consultas es muy útil, ya que no se puede buscar un email en la base de datos si el usuario que se está registrando no lo ha introducido todavía.

En la variable *matchingUsers* se guardan todos los usuarios que cumplan la condición del email, pero la función *useGetUsersQuery*, proporciona más información sobre la consulta, como la variable *isFetching*, que indica que la consulta está en proceso de realización.

No solo se puede pedir información a la API Rest, sino que también se puede escribir información en ella.

```
const [createUser, createUserResult] = useCreateUserMutation()

createUser({ userWithoutId: { email, name, surname } })
```

*Código 37: Consulta para creación de información en la base de datos*

La creación de una entrada en la base de datos, a través de la API Rest no se hace de manera inmediata. En la primera línea de Código 37, se devuelve la función que se utiliza para la creación de la entrada y los resultados de esta ejecución que permiten hacer un seguimiento de si la consulta se ha realizado correctamente y de los datos que se han introducido. En la segunda línea se utiliza la función para introducir un nuevo usuario en la web. Los parámetros que necesita la función son sugeridos, gracias a una correcta documentación de la API y el uso de TypeScript.

# Capítulo 11. FUNCIONAMIENTO DE LA WEB

Antes de explicar cómo se estructura el código que forma la web, se van a explicar unas variables globales que se utilizan a lo largo de todo el código.

## *11.1 ESTADO GLOBAL CON REDUX*

A la hora de desarrollar el código, fue necesario el uso de unas variables globales, que pudieran ser accesibles a todos los componentes de React. Esto es para evitar el *prop drilling* de React, por el cual se tienen que ir pasando los estados desde el padre a todos los hijos que lo quieren utilizar. De hecho, puede ser que un estado se necesite desde un hijo que esté anidado cinco veces desde el padre. Este problema se soluciona con la creación de un estado global.

La creación de estos estados globales puede hacerse con la API de contexto propia de React. Esta es la forma de crear estados globales sin la necesidad de instalar Redux. Sin embargo se ha decidido realizar con Redux, ya que era el sistema que se estaba utilizando para el caché de la API REST en comunicación con el servidor, es decir, ya estaba instalado, por lo que no supone un mayor esfuerzo y de hecho ayuda a que todo esté creado desde el mismo sistema.

Estos estados globales serán comunes a toda la aplicación y pueden ser leídos y modificados por todos los componentes de la aplicación.

Para la creación de los estados globales, primero se crean dos interfaces, que indican el tipo que va a tener la variable.

```
export interface UserState {
    email: string
    name: string
    surname: string
    id?: number
}
export interface UserProfileState {
```

```
    id?: number
    user_id?: number
    user_type?: string
    size_type?: string
    has_gas?: boolean
    hw_gas?: boolean
    hw_boiler?: boolean
    n_ac?: number
    n_radiators?: number
    n_heatpumps?: number
    n_pv?: number
    n_bat?: number
    n_evcp?: number}
```

*Código 38: Definición de interfaces para el estado global*

En el Código 38, se puede ver que se crean dos interfaces. Una para los datos que puede contener la tablade usuario, en el que se decide que el valor del id no tiene por qué ser obligatorio. La otra interfaz contendrá los datos del perfil del usuario y en ella se indica que ninguna de las propiedades debe ser obligatoria. Esto se decide escribiendo el signo de interrogación y se realiza de esta forma, ya que según el cliente vaya contestando a las preguntas, se irán guardando en el perfil, pero no se guardarán todas de golpe.

```
export interface GlobalState {
    user: UserState | null
    userProfile: UserProfileState | null
}

const initialState: GlobalState = {
    user: null,
    userProfile: null
}
```

*Código 39: Definición de estados globales y estado inicial*

A continuación, en el Código 39, se definen dos estados globales. Uno de ellos *user*, para guardar los datos del usuario y otro *userProfile*, para guardar los datos del formulario. Para ambos se indica que pueden ser de tipo *UserState*, *UserProfileState* respectivamente o *null*. Esto se debe a que en un inicio, no se sabe que usuario está entrando en la aplicación y por lo tanto el valor de estas variables debe ser nulo.

```
export const globalStateSlice = createSlice({
    name: 'globalState',
    initialState,
    reducers: {
        setUser: (state, action: PayloadAction<UserState>) => {
            state.user = action.payload
        },
        setUserProfile: (state, action: PayloadAction<UserProfileState>) => {
```

```
            state.userProfile = action.payload
        }
    }
})
```

*Código 40: Modificación del estado global*

Para los estados globales, es también necesario definir unas funciones que serán las encargadas de modificar el valor de los estados. Este tipo de programación se conoce como arquitectura flux, en el que el flujo de información es lineal.

En este caso, en el Código 40, se puede ver que se han definido dos funciones. Una de ellas es la encargada de modificar el estado *user*. Además, en la *PayloadAction* se indica que el parámetro que se manda para cambiarla de estado debe ser de tipo *UserState*. Finalmente, se define también una función para la modificación del estado *userProfile*. En este caso el parámetro debe ser de tipo *UserProfileState*.

## 11.2 ESTRUCTURA DE LA WEB

Para la programación de la web, es necesario tener claro los componentes o bloques que la conforman, siguiendo con la idea que ya se ha comentado en el Capítulo 6. En este caso, esos bloques se han dividido en páginas, es decir, los componentes irán apareciendo en la web de forma individual. Todas estas páginas forman parte de un mismo bloque superior denominado App.

```
const App = () => {
    const user = useSelector((state: RootState) => state.globalState.user)

    const userProfile = useSelector(
        (state: RootState) => state.globalState.user
    )

    return !user || !user.id ? (
        <LoginPage />
    ) : !userProfile || userProfile.id ? (
        <NewUserProfile />
    ) : (
        <SurveyPage />
    )
}
```

*Código 41: Código del componente principal App*

Como se puede ver en el Código 41, se utilizan las variables globales definidas en la sección anterior para controlar los componente o páginas que aparecerán por pantalla.

En el momento inicial, en el que el usuario es nulo o no existe un id del usuario, aparece la página principal de inicio, donde el usuario deberá ejecutar su registro.

A continuación, cuando ya existe un usuario, pero no existe un perfil para este usuario, aparece una página intermedia de espera. Esta espera es corta, es lo que se tarda en crear un nuevo perfil para el usuario registrado.

Finalmente, se muestra la página de la encuesta, que como ya se explicará más adelante, también se divide en varias páginas.

## *11.3 LOGINPAGE*

Siguiendo el orden en el que van apareciendo los componentes en pantalla, el primero de ellos es LoginPage.

Primero, se explicará cómo funciona la lógica diseñada para hacer un registro correcto de los usuarios y a continuación, se explicará cómo ve esto el cliente.

```
export const LoginPage = () => {
    const user = useSelector((state: RootState) => state.globalState.user)
    const dispatch = useDispatch()

    const [createUser, createUserResult] = useCreateUserMutation()

    const matchingUsersRes = useGetUsersQuery(
        { email: user?.email },
        { skip: !user }
    )

    // React to user created on DB and save to Redux state when new user is created
    useEffect(() => {
        if (!createUserResult.isSuccess) return
        // This saves the user that the server created WITH THE ID to Redux state
        dispatch(setUser(createUserResult.data))
    }, [createUserResult, dispatch])

    // Save to redux state when user is found or trigger user creation on server

    useEffect(() => {
        if (!user) return
```

```
    const { data: matchingUsers, isFetching } = matchingUsersRes
    if (!matchingUsers || isFetching) return

    const userDoesntExist = matchingUsers.length === 0
    if (userDoesntExist && createUserResult.isUninitialized) {
        const { email, name, surname } = user
        // This creates the user on the server
        createUser({ userWithoutId: { email, name, surname } })
    } else {
        console.log('entra en else')
        // This saves the user WITH THE ID to Redux state
        dispatch(setUser(matchingUsers[0]))
    }
}, [createUser, dispatch, matchingUsersRes, user, createUserResult])
```

*Código 42: Lógica de LoginPage*

En el Código 42, se pueden ver, en primer lugar, la declaración de las variables que se van a utilizar. En este caso, son el estado global de usuario y las funciones de Redux relacionada con la consulta y escritura en la tabla usuario. Además, se utiliza el hook useEffect que como ya se comentó en el capítulo 6.3 Hooks de React, funciona de forma que se ejecuta lo que contiene en el momento en el que se modifica una de las variables que se indican.

Cuando se introduce un usuario, se piden a la base todos los usuarios que coincidan con el email introducido. Si la longitud del vector es cero, es que no se ha encontrado ningún usuario y por lo tanto se crea uno nuevo con los datos introducidos por el cliente. Además, se cargan los datos de este usuario creado en la variable global, para que de esta forma se actualice su id.

Si la longitud del vector no es cero, significa que el usuario ya se había registrado anteriormente y por lo tanto, no hace falta crear uno nuevo, únicamente es necesario seleccionar los datos que ya existían en la base de datos para poder rescatar el id de este usuario.

A continuación, se puede ver el código que se utiliza para recoger los datos del cliente.

```
return (
    <Row justify="center" align="middle" style={{ height: '100vh' }}>
        <Col xs={22} sm={18} md={8} lg={8} xl={8}>
            <div>
                <h1>¡Hola!</h1>
                <p>Primero, necesito saber quién eres:</p>
```

```
<Formik
    initialValues={{ email: '', name: '', surname: '' }}
    onSubmit={async (values) => {
        // This saves the user into Redux state, not the database
        dispatch(setUser(values))
    }}
    validationSchema={Yup.object().shape({
        email: Yup.string().email().required('Required'),
        name: Yup.string().required('Required'),
        surname: Yup.string().required('Required')
    })}
>
    {(props) => {
        const {
            values,
            touched,
            errors,
            isSubmitting,
            handleChange,
            handleBlur,
            handleSubmit
        } = props
        return (
            <form onSubmit={handleSubmit}>
                <label
                    htmlFor="email"
                    style={{ display: 'block' }}
                >
                    Email
                </label>
                <Input
                    id="email"
                    placeholder="Introduce tu email"
                    value={values.email}
                    onChange={handleChange}
                    status={
                        errors.email && touched.email
                            ? 'error'
                            : undefined
                    }
                />
                <label
                    htmlFor="name"
                    style={{ display: 'block' }}
                >
                    Nombre
                </label>
                <Input
                    id="name"
                    placeholder="Introduce tu nombre"
                    value={values.name}
                    onChange={handleChange}
                    status={
                        errors.name && touched.name
                            ? 'error'
                            : undefined
                    }
                />

                <label
```

```
                                    htmlFor="surname"
                                    style={{ display: 'block' }}
                                >
                                    Apellido
                                </label>
                                <Input
                                    id="surname"
                                    placeholder="Introduce tu apellido"
                                    value={values.surname}
                                    onChange={handleChange}
                                    status={
                                        errors.surname && touched.surname
                                            ? 'error'
                                            : undefined
                                    }
                                />

                                <Button
                                    htmlType="submit"
                                    type="primary"
                                    disabled={
                                        isSubmitting ||
                                        matchingUsersRes.isFetching ||
                                        createUserResult.isLoading
                                    }
                                    loading={
                                        isSubmitting ||
                                        matchingUsersRes.isFetching ||
                                        createUserResult.isLoading
                                    }
                                    style={{
                                        margin: '4px'
                                    }}
                                >
                                    Acceder
                                </Button>
                            </form>
                        )
                    }}
                </Formik>
            </div>
        </Col>
    </Row>
  )
}
```

*Código 43: Return del componente LoginPage*

Como se puede ver en el Código 43, se utiliza una herramienta llamada Formik, que se va a utilizar para la gestión de todas las variables de información de usuario a lo largo de los diferentes componentes.

### *11.3.1 FORMIK*

Formik es una librería gratuita de código abierto para ReactJs o React Native y se utiliza para la creación y gestión de formularios. Con ella se pueden controlar los estados de los formularios, así como manejar validaciones y la creación de mensajes de error para que los usuarios sepan lo que está ocurriendo.

Esta biblioteca fue creada por Jared Palmer, para estandarizar los componentes de entrada y el flujo el envío de los formularios. De esta forma se organiza todo el formulario en un mismo lugar y se simplifican las pruebas y la gestión de los datos en el formulario.[42] [53]

El funcionamiento de esta herramienta es el que sigue. En primer lugar, se deben indicar las variables que se quiere que se gestione, con unos valores iniciales. En este caso se eligen las variables de registro email, *name* y *surname*, con valor inicial de una cadena de caracteres vacía. Los valores de se guardarán en la variable *values*.

A continuación, se introducen componentes que permiten la entrada de texto desde el teclado y se indica en cada espacio, a qué valor de *values* se corresponde esa entrada. Una vez el cliente ha introducido sus datos, se pulsa el botón Acceder, que provoca que los valores introducidos se guarden en la variable de estado global usuario.

Es necesario hacer una validación de que los datos introducidos son correctos. Para ello se utiliza Yup, en el que se indican el tipo de variables que pueden ser introducidos y que todos los campos son obligatorios. En este caso, todos los datos introducidos deben ser cadenas de caracteres.

### *11.3.2 ANT DESIGN*

En el Código 43, también se puede ver que se utilizan unos componentes que no son propios de React sino que pertenecen a una librería externa denominada Ant Design. El uso de estos componentes se hace necesario para mejorar la imagen de la web. No aportan una funcionalidad en este caso, pero sí mejoran considerablemente la apariencia de la web.

Ant Design contiene una librería de React con un conjunto de componentes y ejemplos de calidad que ayudan a realizar interfaces de usuario de una manera mucho más sencilla. Además estos componentes ayudan a que la aplicación sea mucho más atractiva para los usuarios. [54] La alternativa a la utilización de componentes de Ant Design era el desarrollo de código CSS para mejorar la estética de cada uno de los elementos utilizados.

En este caso, los componentes que se han usado son *Row*, *Col*, *Input* y *Button*. *Coll* y *Row* se utilizan para ajustar el contenido de la web al centro de la pantalla.

El componente Input, crea el espacio para que el usuario introduzca sus datos. Además, este se ha personalizado de forma que si los datos introducidos no son correctos o falta información, se resalten en rojo los campos que tienen errores.

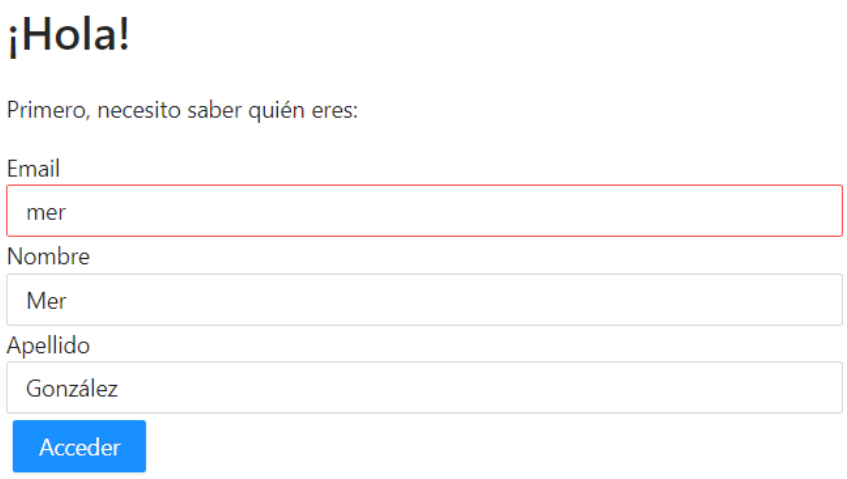


*Figura 12: Página de inicio con prueba de error*

En la Figura 12, se puede ver como se señala en rojo el campo para la introducción del email. Esto se debe a que el dato introducido no tiene forma de email y por lo tanto, no se considera un parámetro válido para el registro. De esta forma, el usuario tiene claro cuál es el campo que tiene que arreglar.

El componente *Button*, se utiliza para crear el botón de acceso. Se elige utilizar este componente porque su apariencia es mucho mejor que la que se proporciona con los componentes de React.

## *11.4 NEWUSERPROFILE*

El siguiente componente que se utiliza para la web es el denominado NewUserProfile. Una vez el usuario se ha registrado hay que comprobar si ya tenía un perfil creado o es un usuario nuevo.

```
const [show, setShow] = useState(false)
 const userProfile = useSelector(
     (state: RootState) => state.globalState.userProfile
 )
 const user = useSelector((state: RootState) => state.globalState.user)
 const dispatch = useDispatch()
 const matchingUsersProfileRes = useGetUsersProfilesQuery(
     {
         userId: user?.id
     },
     { skip: !user }
 )
 const [createUserProfile, createUserProfileResult] =
     useCreateUserProfileMutation()

 const [editUserProfile] = useChangeUsersProfilesMutation()
 // React to user created on DB and save to Redux state when new user is created
 useEffect(() => {
     if (!createUserProfileResult.isSuccess) return
     // This saves the user that the server created WITH THE ID to Redux state
     console.log('Muestro el perfil en use effect despues de crearlo')
     console.log(userProfile)
     //Extract data from the data base
     const {
         has_gas,
         hw_boiler,
         hw_gas,
         n_ac,
         n_bat,
         n_evcp,
         n_heatpumps,
         n_pv,
         n_radiators,
         size_type,
         user_id,
         user_type,
         id
     } = createUserProfileResult.data
     // Save data in global variables
     dispatch(
         setUserProfile({
```

```
                has_gas,
                hw_boiler,
                hw_gas,
                n_ac,
                n_bat,
                n_evcp,
                n_heatpumps,
                n_pv,
                n_radiators,
                size_type,
                user_id,
                user_type,
                id
            })
        )
        setShow(true)
    }, [createUserProfileResult])

    // Save to redux state when user is found or trigger user creation on server

    useEffect(() => {
        if (userProfile) return

        const { data: matchingUsersProfile, isFetching } =
            matchingUsersProfileRes
        if (!matchingUsersProfile || isFetching) return
        const userProfileDoesntExist = matchingUsersProfile.length === 0
        console.log('ha cargado el matching user')
        const emptyProfile = {
            has_gas: false,
            hw_boiler: false,
            hw_gas: false,
            n_ac: 0,
            n_bat: 0,
            n_evcp: 0,
            n_heatpumps: 0,
            n_pv: 0,
            n_radiators: 0,
            size_type: '',

            user_id: user.id,
            user_type: ''
        }

        if (userProfileDoesntExist && createUserProfileResult.isUninitialized) {
            console.log('entro en el if para: no ha encontrado un profile')
            createUserProfile({ userProfileWithoutId: emptyProfile })
        } else {
            const {
                has_gas,
                hw_gas,
                hw_boiler,
                n_radiators,
                n_ac,
                n_pv,
                n_bat,
                n_evcp,
                user_type,
                size_type,
                n_heatpumps,
```

```
            user_id
        } = matchingUsersProfile[matchingUsersProfile.length - 1]

        createUserProfile({
            userProfileWithoutId: {
                has_gas,
                hw_gas,
                hw_boiler,
                n_radiators,
                n_ac,
                n_pv,
                n_bat,
                n_evcp,
                user_type,
                size_type,
                n_heatpumps,
                user_id
            }
        })
    }
}, [matchingUsersProfileRes])
```

*Código 44: NewUserProfile*

En el Código 44, se puede ver la lógica que se ha seguido para la creación de los nuevos perfiles de usuario. El perfil de usuario es el que contiene la información de los dispositivos que tiene el usuario. La idea es que si el usuario ya ha rellenado el perfil con anterioridad, pueda partir de sus datos anteriores para cambiar algunas cosas pero no tenga que empezar desde cero.

En este caso, cuando existe un usuario, se busca si hay algún perfil con el id de este usuario. Los perfiles encontrados, se van guardando en un vector llamado *matchingUsersProfile*. Si el vector se queda vacío, se crea un perfil con los datos vacíos. Además, cuando se ha creado correctamente este perfil, se guardan los datos en el estado global *UserProfile*.

Si se han encontrado perfiles, se hace una desestructuración de los datos encontrados en el último perfil y se crea un nuevo perfil que contiene los datos antiguos. Como el estado global se actualiza en el *useEffect* en el que se controla que se ha creado correctamente el perfil, cuando se crea el perfil adecuadamente, se cargan los datos en el estado global de la misma manera que en el otro caso.

Aunque el usuario ya tuviera un perfil es necesario crear otro porque un id de un perfil únicamente puede tener una ejecución de algoritmo y por lo tanto, lo que se hace es crear un

perfil nuevo con los datos anteriores. De esta forma, sí que se puede obtener una ejecución de algoritmo correctamente.

## *11.5 SURVEYPAGE*

Para la realización del formulario se ha decidido utilizar un componente de Ant Design llamado *Step* [54], que permite hacer un menú progresivo con los diferentes pasos que hay que seguir para llegar al final del formulario.

```
const steps = [
    {
        title: '¿Qué clase de cliente eres?'
    },
    {
        title: 'Dinos qué tienes'
    },
    {
        title: 'Flexibilidad'
    }]
```

*Código 45: Pasos en el formulario*

En el Código 45, se pueden ver los tres pasos que se han seleccionado para que aparezcan por pantalla.

```
const getCurrentStepContent = (currentStep: number): JSX.Element => {
        // ¿Qué clase de cliente eres?
        if (currentStep === 0) {
            // aquí returneo un formik con las preguntas de user type
            return (
                <>
                    <UserTypeForm />
                </>
            )
        } else if (currentStep === 1) {
            return <UserHouseForm />
        } else if (currentStep === 2) {
            return (
                <>
                    <Results
                        createAlgorithmExecution={createAlgorithmExecution}
                        createAlgorithmExecutionResult={
                            createAlgorithmExecutionResult
                        }
                    />
                </>
            )
        }
    }
```

*Código 46: Llamadas a los componentes en cada paso del formulario*

En el Código 46, se decide a qué componentes se llama en función del paso en el que se encuentra en ese momento. Se empieza llamando al componente UserTypeForm, en el que el usuario tiene que elegir de qué tipo es. En el siguiente paso, se muestra el formulario en el que se preguntan los dispositivos que tiene el cliente en la vivienda. Finalmente, se muestran los resultados del algoritmo.

El algoritmo, se manda ejecutar desde este componente y son los resultados de la ejecución del algoritmo los que se estudiarán en el componente Results.

```
return (
        <>
            <div>
                <Row
                    justify="center"
                    align="middle"
                    style={{ height: '100vh' }}
                >
                    <Col xs={20} sm={34} md={50} lg={70} xl={80}>
                        <Steps current={currentStep}>
                            {steps.map((item) => (
                                <Step key={item.title} title={item.title} />
                            ))}
                        </Steps>
                        <div className="steps-content">
                            {getCurrentStepContent(currentStep)}
                        </div>
                        <div className="steps-action">
                            {currentStep < steps.length - 2 && (
                                <Button
                                    type="primary"
                                    onClick={() => goToNextStep()}
                                    style={styles.button}
                                >
                                    Siguiente paso
                                </Button>
                            )}
                            {currentStep === steps.length - 2 && (
                                <Button
                                    type="primary"
                                    style={{
                                        backgroundColor: '#70C040',
                                        borderColor: '#70C040'
                                    }}
                                    onClick={() => {
                                        message.success('Processing complete!')
                                        setCurrentStep(currentStep + 1)
                                        calculateAlgorithmValues()
                                    }}
                                >
                                    ¡Calcular mi flex!
```

```
                                </Button>
                            )}
                            {currentStep > 0 && currentStep < 2 && (
                                <Button
                                    style={{ margin: '0 8px' }}
                                    onClick={() => goToPreviousStep()}
                                >
                                    Previous
                                </Button>
                            )}
                        </div>
                    </Col>
                </Row>
            </div>
        </>
    )
}
```

*Código 47: Utilización de componente Steps*

En función del paso en el que se encuentra, se muestra una información u otra. Esto se puede ver en el Código 47, en el que se van haciendo comprobaciones del paso en el que se encuentran y se hace una operación lógica *and* con la información que se quiere mostrar. De esta forma si la condición es falsa, no se encuentra en ese paso y por lo tanto no se muestra y si sí lo es, aparece por pantalla.

### *11.5.1 USERTYPEFORM*

La web que se ha creado se ha centrado en un público residencial, es decir, lo que se quiere estudiar es los dispositivos típicos de una vivienda. Sin embargo, en un futuro podría expandirse a otros sectores como el comercial o industrial. Por ello, se plantea la pregunta de qué tipo de usuario es.

Para realizar las preguntas, al igual que en el caso del registro de usuario se utiliza Formik[53]. De esta forma, la gestión de los datos se hace mucho más sencilla.

```
return (
        <div>
            <h1></h1>
            <Formik
                initialValues={{
                    user_type: userProfile.user_type,
                    profileId: userProfile.id
                }}
                onSubmit={async (values) => {
                    await new Promise((resolve) => setTimeout(resolve, 500))
                    alert(JSON.stringify(values, null, 2))
                }}
```

```
            validationSchema={Yup.object().shape({
                email: Yup.string().email().required('Required')
            })}
        >
            {(props) => {
                const { values, setFieldValue } = props
                return (
                    <>
                        <FormikListener
                            values={values}
                            callback={handleFormChange}
                        />
                        <form>
                            <h3>¿Qué tipo de usuario eres? </h3>

                            <Radio.Group
                                buttonStyle="solid"
                                value={values.user_type}
                                onChange={(e) =>
                                    setFieldValue(
                                        'user_type',
                                        e.target.value
                                    )
                                }
                            >
                                <Radio.Button
                                    value={'residential'}
                                    name="radio"
                                    style={{
                                        marginTop: '10px',
                                        marginBottom: '10px'
                                    }}
                                >
                                    <Field
                                        type="radio"
                                        name="user_type"
                                        value="residential"
                                    />
                                    Residencial
                                </Radio.Button>
                                <Radio.Button
                                    value={'comercial'}
                                    name="user_type"
                                >
                                    <Field
                                        type="radio"
                                        name="user_type"
                                        value="comercial"
                                    />
                                    Comercial
                                </Radio.Button>
                                <Radio.Button
                                    value={'industrial'}
                                    name="user_type"
                                >
                                    <Field
                                        type="radio"
                                        name="user_type"
                                        value="industrial"
                                    />
```

```
                                        Industrial
                                    </Radio.Button>
                                </Radio.Group>
                            </form>
                        </>
                    )
                }}
            </Formik>
        </div>
    )
```

*Código 48: Uso de Formik en UserTypeForm*

En este caso Código 48, no el usuario no tiene que escribir nada, simplemente elegir entre una de las tres opciones. Este tipo de botones en los que se muestran varias opciones se denominan botones radio. El componente *Radio.Button* se ha seleccionado también de Ant Design [54]. En este caso también se utiliza la validación de los parámetros pero no debería haber ningún problema porque el usuario no tiene la libertad de introducir la información que considere, simplemente puede elegir entre una de las tres opciones marcadas.

Las variables de Formik que se han elegido en este caso son *user_type* y *profileId*. En este caso, además se debe destacar el uso de la herramienta llamada *FormikListener*, por la cual se pueden gestionar los valores en el momento en el que cambian. La explicación de este recurso se hará en el siguiente apartado, UserHouseForm, donde ya que también se usa para la gestión de esos datos.

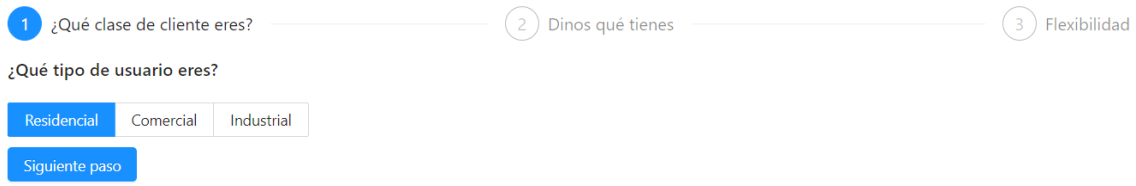


*Figure 13: UserTypeForm*

En la Figure 13, se puede ver cómo queda la formulación de la pregunta para que el usuario seleccione la opción más adecuada. Además, se puede apreciar el menú de los pasos que se comentó en el apartado anterior.

### *11.5.2 USERHOUSEFORM*

El siguiente componente que se utiliza es UserHouseForm, en el que se realizan las preguntas sobre los dispositivos de la vivienda. Al igual que en el caso anterior, se utiliza Formik para la gestión del formulario y junto a él FormikListener. El código de este componente es muy extenso y no aporta una mayor información que en el apartado anterior, por lo que no se va a introducir a continuación. La estructura es muy similar a la del Código 48, pero incluye nuevas preguntas y variables. Este código se podrá encontrar en Anexo III.I: Código UserHouseForm.

```
export const FormikListener = ({ values, callback }) => {
    const [updateProfile] = useChangeUsersProfilesMutation()
    const [updateProfileData, setUpdateProfileData] = useState({})
    useEffect(() => {
        callback(values, updateProfile, setUpdateProfileData)
    }, [callback, values])
    return <></>
}
//const [changeUserProfile, changeUserProfileResult] =
//useChangeUsersProfilesMutation();
//me cargo el servidor con esta funcion: TENGO QUE PROBAR SI QUITANDOLA FUNCIONA
CORRECTAMENTE
const convertString = (word) => {
    switch (word) {
        case 'true':
            return true

        case 'false'::
            return false
        default:
            return Boolean(word)
    }
}
export const handleFormChange = (v, updateProfile, setUpdateProfileData) => {
    console.log(v)
    const has_gas = convertString(v.has_gas)
    const hw_boiler = convertString(v.hw_boiler)
    const hw_gas = convertString(v.hw_gas)
    setUpdateProfileData({
        has_gas: has_gas,
        hw_boiler: hw_boiler,
        hw_gas: hw_gas,
        n_ac: v.n_ac,
        n_bat: v.n_bat,
        n_evcp: v.n_evcp,
        n_heatpumps: v.n_heatpumps,
        n_pv: v.n_pv,
        n_radiators: v.n_radiators,
        size_type: v.size_type,
        user_type: v.user_type
    })
```

```
    if (v.profileId !== 0) {
        updateProfile({
            profileId: v.profileId,
            userProfileWithoutId: {
                has_gas: has_gas,
                hw_boiler: hw_boiler,
                hw_gas: hw_gas,
                n_ac: v.n_ac,
                n_bat: v.n_bat,
                n_evcp: v.n_evcp,
                n_heatpumps: v.n_heatpumps,
                n_pv: v.n_pv,
                n_radiators: v.n_radiators,
                size_type: v.size_type,
                user_type: v.user_type
            }
        })
    }
}
```

*Código 49: FormikListener*

En el desarrollo del sistema, se decidió que era fundamental que los datos de los usuarios se fueran guardando en la base de datos a medida que se va rellenando el formulario. Esto es muy importante, porque si por ejemplo el usuario pierde conexión a internet tiene la posibilidad de restaurar la información que ya había completado.

Por ello, se para poder realizar estas consultas a la base de datos se utiliza FormikListener. En el Código 49, se puede ver que se define un estado para guardar los datos del perfil de usuario y también se crea una función de consulta a la base de datos que permite modificar los datos de un perfil. Está función viene de la documentación de la API REST y Redux RTK Query.

Cada vez que el valor de la variable de Formik, *values* cambia, se llama a estas funciones para que guarden los datos correspondientes. Es importante indicar el id del perfil del que se quieren cambiar los datos, que es también una variable que se encuentra en *values*.

Un detalle importante, es que los *values* de Formik tienen que ser cadenas de caracteres, o números, pero no booleanos. Por ello es necesaria la utilización de la función *ConvertString* para transformar las cadenas de caracteres a booleanos antes de introducirlos en la base de datos.

### *11.5.3 INTERACTIVIDAD EN USERHOUSEFORM*

Al desarrollar esta web se pretende que sea lo más interactiva posible para que el usuario encuentre amena la contestación del formulario. Por ello se utiliza una vivienda unifamiliar en la que irán apareciendo los iconos de los diferentes dispositivos que tiene el usuario.

Para la realización de la vivienda se utiliza CSS, que significa hojas de estilo en cascada, y es un lenguaje de programación que se utiliza para dar formato y apariencia a las páginas web y las interfaces de usuario. Generalmente se utiliza junto con HTML y JavaScript para mejorar la apariencia de las aplicaciones que se desarrollan.[13]

Con la utilización de CSS, se puede repartir el espacio que ocupan los diferentes elementos de la web en la pantalla.

De esta forma, el espacio de divide principalmente en dos partes, la primera de ellas es el formulario y la segunda, la vivienda.

```
.formAndHouse {
    display: flex;
    flex-direction: row;
    margin: 5px;
}
.house {
    display: flex;
    flex-direction: column;
    flex: 7;
    margin: 5px;
    max-width: 650px;
    max-height: 650px;
}
.form {
    flex: 3;
    display: flex;
    flex-direction: row;
}
```

*Código 50: Distribución de espacio con CSS*

En el Código 50, se puede ver que en la clase *formAndHouse*, se establece como *flex*, que ayuda a la distribución automática del espacio. Los elementos flexibles se pueden estirar hasta ocupar todo el espacio disponible, en proporción con su factor de crecimiento, evitando el desbordamiento [55].

Se determina un padre contenedor y unos hijos que se encuentran dentro de este contenedor para distribuirse en el espacio del padre. Las propiedades de *flex* se pueden encontrar en [56]. En la Figura 14, se puede ver el diagrama de cómo funcionan los contenedores de información.


*Figura 14: Flex information* [56]

Finalmente, se determina que los componentes que formen parte de ella se distribuyan en una fila. Además, en este caso se determina, que el formulario ocupe un 30% del espacio total y la vivienda el resto.

```
.pv {
    display: flex;
    flex-direction: row;
    flex: 1;
    align-items: flex-start;
    flex-wrap: wrap-reverse;
    background-color: rgb(145, 225, 241);}
.insideHouse {
    display: flex;
    flex-direction: column;
    flex: 4;
    align-items: flex-end;
    flex-wrap: wrap-reverse;
    background-color: white;
    border: 3px;
    border-color: black;
    border-style: solid;}
.garage {
    display: flex;
    flex-direction: row;
    flex: 1;
    align-items: flex-end;
    background-color: rgb(200, 200, 200);
    border: 3px;
    border-color: black;
    border-style: solid;
}
```

*Código 51: Distribución vivienda*

La distribución interior de la vivienda se hace en primer lugar en función de plantas. Se establece una planta superior en la que se expondrán los paneles solares, una planta central de la vivienda y por debajo un garaje. Esta distribución puede verse en el Código 51. Además, en este caso, también se establecen los colores de fondo.

```
.roomIndG {
    display: flex;
    flex-direction: row;
    flex: 1;
    align-items: flex-end;
    flex-wrap: wrap-reverse;
    background-color: rgb(200, 200, 200);
}
.roomG {
    display: flex;
    flex-direction: row;
    flex: 4;
    align-items: flex-start;
    flex-wrap: wrap-reverse;
    background-color: rgb(200, 200, 200);
}
```

*Código 52: Distribución de las plantas de la vivienda*

En el Código 52, se puede ver el siguiente nivel de distribución de las plantas que conforman la vivienda. Las habitaciones, en este caso, las habitaciones del garaje. Existen dos tipos de elementos: los que se puede tener uno o ninguno y los que se pueden tener varios. Por ello, se establece que el espacio que ocupan los elementos individuales sea menor que los elementos en los que puede haber más de un dispositivo.

```
.radiatorIcon {
    height: 60px;
    width: 70px;
}
```

*Código 53: Ajuste tamaño iconos*

Finalmente, se ajusta el tamaño de los iconos que forman parte de la vivienda. En el Código 53, se puede ver el ejemplo del ajuste del tamaño de un radiador. La mayoría de los iconos de los dispositivos fueron proporcionados por Stemy, sin embargo hubo dos que no se encontraban disponibles, la bomba de calor y el icono del gas. [57], [58] El código completo de estilo se puede encontrar en Anexo III.II: Código index.css.

```
                        <div className="house">
                            <div className="pv">
                                {[...Array(values.n_pv).keys()].map((i) => (
```

```
                    <img src={pvIcon} className="pvIcon" />
                ))}
            </div>

            <div className="floor">
                <div className="room">
                    {' '}
                    {[...Array(values.n_ac).keys()].map(
                        (i) => (
                            <img
                                src={acIcon}
                                className="acIcon"
                            />
                        )
                    )}
                </div>
                <div className="room">
                    {[
                        ...Array(values.n_heatpumps).keys()
                    ].map((i) => (
                        <img
                            src={heatpumpIcon}
                            className="heatpumpIcon"
                        />
                    ))}
                </div>
            </div>
            <div className="floor">
                <div className="room">
                    {' '}
                    {[
                        ...Array(values.n_radiators).keys()
                    ].map((i) => (
                        <img
                            src={radiatorIcon}
                            className="radiatorIcon"
                        />
                    ))}
                </div>
            </div>

            <div className="garage">
                <div className="roomIndG">
                    {' '}
                    {convertString(values.hw_boiler) && (
                        <img
                            src={hwBoilerIcon}
                            className="hwBoilerIcon"
                        />
                    )}
                </div>

                <div className="roomIndG">
                    {convertString(values.hw_gas) && (
                        <img
                            src={hwGasIcon}
                            className="hwGasIcon"
                        />
                    )}
                </div>
```

```
                    <div className="roomIndG">
                        {convertString(values.has_gas) && (
                            <img
                                src={gasIcon}
                                className="gasIcon"
                            />
                        )}
                    </div>
                    <div className="roomG">
                        {[...Array(values.n_evcp).keys()].map(
                            (i) => (
                                <img
                                    src={evcpIcon}
                                    className="evcpIcon"
                                />
                            )
                        )}
                    </div>
                    <div className="roomG">
                        {[...Array(values.n_bat).keys()].map(
                            (i) => (
                                <img
                                    src={batIcon}
                                    className="batIcon"
                                />
                            )
                        )}
                    </div>
                </div>
            </div>
```

*Código 54: UserHouseForm interactividad*

En el Código 54, se puede ver cómo se van definiendo las clases para los diferentes componentes. Además, se consigue que aparezcan los diferentes dispositivos haciendo un mapeado del vector de valores de Formik y representando tantas imágenes como elementos haya en ese vector.

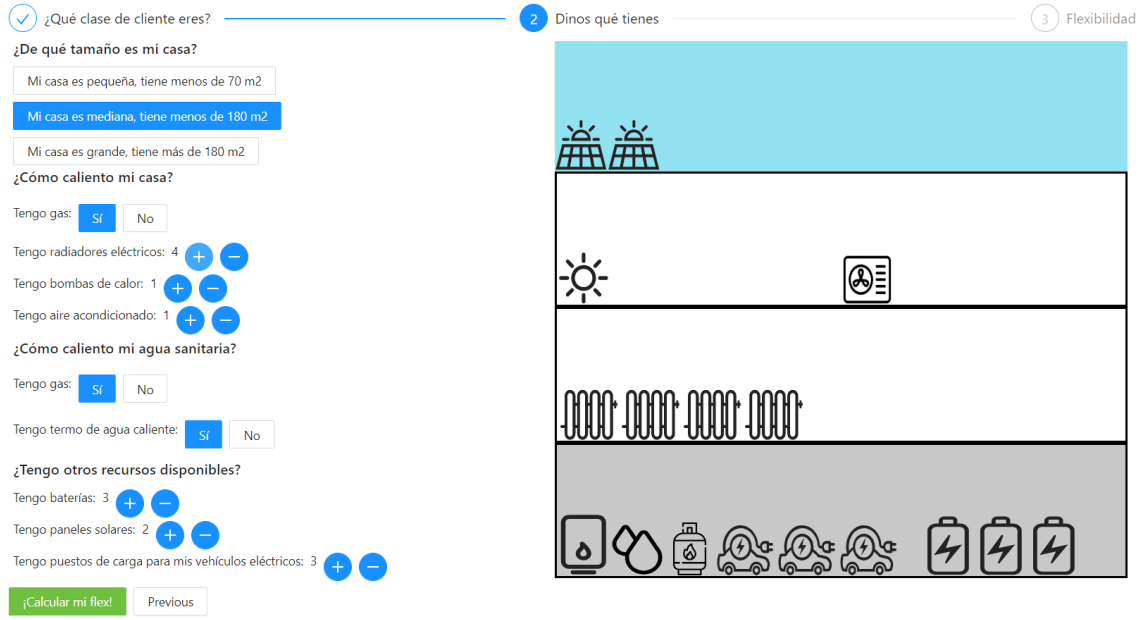


*Figura 15: Web UserHouseForm*

En la Figura 15, se puede ver la distribución del espacio comentada anteriormente, en la que el formulario queda a la izquierda de la pantalla y la vivienda con los diferentes elementos queda a la derecha. Además, se representan los números de elementos que va eligiendo el usuario en la vivienda. La parte superior, en azul, pretende representar el cielo, ya que los paneles solares se encontrarán al aire libre. La parte central, en blanco, representa la vivienda y finalmente, la inferior gris, representa el garaje.

### *11.5.4 RESULTS*

Una vez el usuario ha completado el formulario con los diferentes dispositivos que tiene en su vivienda puede comprobar los resultados de los ahorros que obtendría si se participara en el proyecto de flexibilidad.

```
export const Results = ({
    createAlgorithmExecution,
    createAlgorithmExecutionResult
}) => {
    const userProfile = useSelector(
        (state: RootState) => state.globalState.userProfile
    )

    useEffect(() => {
        if (createAlgorithmExecutionResult.isSuccess) {
```

```
        }
        if (createAlgorithmExecutionResult.isError) {
            alert('Ha ocurrido un error en la ejecucion del algoritmo')
        }
    }, [createAlgorithmExecutionResult])

    //      algorithmExecutionWithoutId: { profile_id: lastProfile.id }
    return (
        <div>
            {createAlgorithmExecutionResult.isLoading ? (
                <p>Loading...</p>
            ) : (
                <></>
            )}

            {createAlgorithmExecutionResult.isSuccess ? (
                <>
                    <h2>
                        Para los datos proporcionados se han obtenido los
                        siguientes resultados
                    </h2>
                    <h3>
                        Se ha obtenido un ahorro de:{' '}
                        {createAlgorithmExecutionResult.data.map((data, i) =>
                            data.kpi_id === 3 ? (
                                <Space size={20}>
                                    <p key={i}>{data.value.toFixed(2)} euros</p>

                                    <Progress
                                        type="circle"
                                        status="normal"
                                        percent={(data.value * 100) / 3000}
                                        width={100}
                                        showInfo={true}
                                        format={() => (
                                            <EuroCircleOutlined
                                                style={{
                                                    fontSize: '45px',
                                                    color: '#1890FF'
                                                }}
                                            />
                                        )}
                                    />
                                </Space>
                            ) : (
                                <></>
                            )
                        )}
                    </h3>
                    <h3 style={{ marginTop: '100px' }}>
                        Se ha obtenido un ahorro de:{' '}
                        {createAlgorithmExecutionResult.data.map((data, i) =>
                            data.kpi_id === 1 ? (
                                <Space size={20}>
                                    <p key={i}>{data.value.toFixed(2)} KWh</p>

                                    <Progress
                                        type="circle"
                                        percent={(data.value * 100) / 2600}
                                        status="normal"
```

```
                                        width={100}
                                        showInfo={true}
                                        strokeColor={'#FC408D'}
                                        format={() => (
                                            <ThunderboltFilled
                                                style={{
                                                    fontSize: '45px',
                                                    color: '#FC408D'
                                                }}
                                            />
                                        )}
                                    />
                                </Space>
                            ) : (
                                <></>
                            )
                        )}
                    </h3>
                    <h3 style={{ marginTop: '100px' }}>
                        Con la disminución de:{' '}
                        {createAlgorithmExecutionResult.data.map((data, i) =>
                            data.kpi_id === 2 ? (
                                <Space size={20}>
                                    <p key={i}>
                                        {data.value.toFixed(2)} kg de CO2
                                    </p>

                                    <Progress
                                        type="circle"
                                        percent={((data.value / 25) * 100) / 15}
                                        status="normal"
                                        width={100}
                                        showInfo={true}
                                        strokeColor={'#3AF320'}
                                        format={() => (
                                            <TiTree
                                                style={{
                                                    fontSize: '45px',
                                                    color: '#3AF320'
                                                }}
                                            />
                                        )}
                                    />
                                </Space>
                            ) : (
                                <></>
                            )
                        )}
                    </h3>
                </>
            ) : (
                <></>
            )}
        </div>
    )
}
```

*Código 55: Results*

En el Código 55, se puede ver que llegan los parámetros de la ejecución del algoritmo. Esto se debe, a que esta ejecución se genera desde SurveyPage, pero en función de los resultados de esta ejecución, se controlará esta página de la web.

Si la ejecución no ha terminado, en la pantalla aparecerá un mensaje de *Loading…* (Cargando …). Una vez finaliza esta ejecución, se muestran los resultados de esta. Se van buscando los resultados que devuelve el algoritmo para cada KPI o recompensa.

Para que la forma en la que aparecen los datos sea más amena, se utilizan elementos de Ant Design, como barras de progreso e iconos, de diferentes colores.[54]

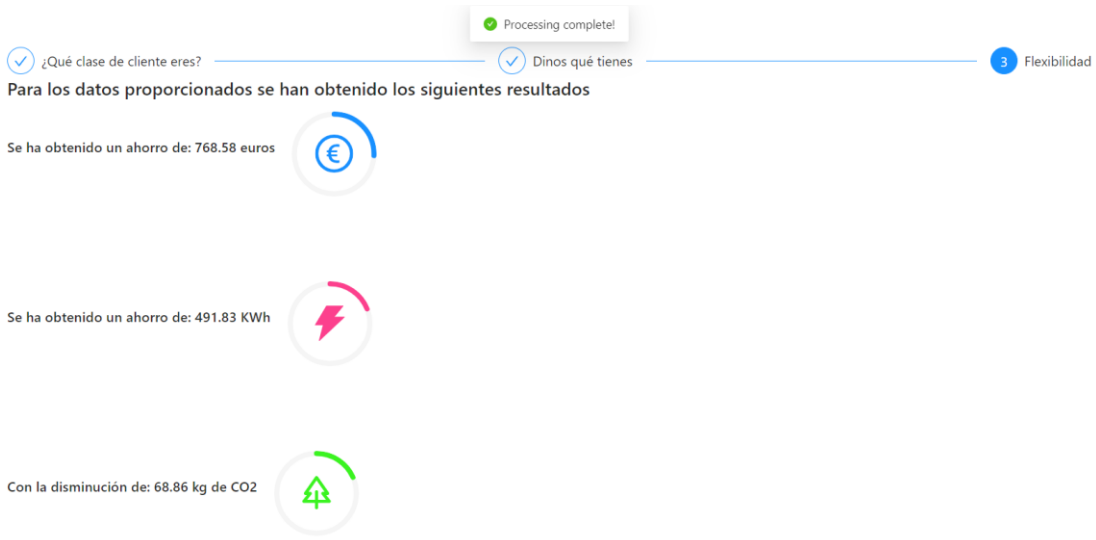


*Figura 16: Results en la web*

En la Figura 16, se expresan los resultados que se obtienen por parte del algoritmo. En primer lugar, se enseña el ahorro en euros, a continuación el ahorro en energía y finalmente la disminución de CO2 que se produce en el ambiente por la participación en el proyecto de flexibilidad. Las barras de progresos pretender señalar cómo de flexible se puede llegar a ser en función de los elementos de los que se dispone. Por ejemplo, una persona que dispone muchos paneles solares y baterías puede notar más el ahorro que una persona que no dispone de estos elementos.

# Capítulo 12. CONTROL DE VERSIONES

Como se ha podido ver a lo largo de la exposición del proyecto, se han desarrollado muchas líneas de código y componentes diferentes. Por ello, suele ser una buena práctica, llevar un control de versiones del proyecto que se está realizando, ya que si por ejemplo se cambia el código de una función y luego no funciona como se esperaba, se pueda volver atrás.

Además, actualmente también es muy interesante el hecho de poder guardar todo el trabajo en la nube, de esta forma se consigue que si ocurre cualquier problema con el ordenador de trabajo se puedan restaurar todos los datos en otro dispositivo.

En este caso, para el control de versiones se ha utilizado GitLab, que es una empresa de software privado que proporciona un servidor central que gestiona los repositorios Git y se utiliza para la simplificación de tareas de administración de muchas empresas. [59]

Git es un sistema de control de versiones que se utiliza para detectar los cambios que se van produciendo en los archivos de los proyectos. Esto ayuda a tener un registro de versiones anteriores del proyecto. De esta forma, si a lo largo del desarrollo del programa se produce algún error, se pueden rescatar versiones anteriores. Además, una de las grandes ventajas que ofrece es la posibilidad de realizar un trabajo junto con otros usuarios.[59]

Junto a Gitlab, se ha utilizado también Sourcetree, que es un cliente de escritorio de interfaz gráfica de usuario gratuito que facilita la forma en la que se interactúa con los repositorios Git. Con esta interfaz, el control de las versiones se realiza con forma de árbol, que es una forma mucho más sencilla y visual que los comandos Git.[59]

Gracias a Sourcetree, el manejo de las versiones que se iban produciendo era más intuitivo. Los dos proyectos de los que se ha llevado un control de versiones son el servidor y la web, que contienen toda la información relevante.

# Capítulo 13. ANÁLISIS DE RESULTADOS

Los resultados que se han obtenido del proyecto dependen en gran medida de la correcta utilización de las diferentes tecnologías que se han ido utilizando a lo largo de la realización del proyecto. El diseño de la arquitectura del proyecto ha sido una clave muy importante para llegar a los resultados obtenidos.

En primer lugar, utilización de la base de datos, ha permitido obtener un registro de los diferentes usuarios, con todos los datos y resultados de algoritmos para cada uno de ellos. De esta forma, se ha podido utilizar esta información a lo largo del código de la web, como por ejemplo, a la hora de comprobar si un usuario ya estaba registrado.

Además, algo muy positivo que llega de la mano de la base de datos es la posibilidad de ir guardando la información de los usuarios a medida que se va rellenando el formulario, de esta forma si el usuario pierde conexión de internet puede retomar su formulario dónde lo había dejado.

El servidor, es otra pieza clave del proyecto, en el que se van definiendo las diferentes respuestas que se pueden obtener de la base de datos y del algoritmo ante las diferentes llamadas de la web. Este algoritmo, ha sido el encargado de obtener las diferentes recompensas de los usuarios. Estas recompensas eran una de las partes claves del proyecto ya que se sabe que la obtención de recompensas aumenta la motivación de los usuarios.

Finalmente, la utilización de React para el desarrollo de la web ha permitido dar interactividad a las páginas. De esta forma, se consigue una participación más activa por parte del usuario y una mayor implicación a la hora de llegar al final del juego. Además, la utilización de componentes Ant Design, ha permitido llegar a un resultado mucho más atractivo y de una forma mucho más eficiente.

# Capítulo 14. CONCLUSIONES Y TRABAJOS FUTUROS

El objetivo principal de este proyecto es la realización de un sistema de gamificación por el cual los usuarios ajenos al funcionamiento de la energía eléctrica se interesen por la participación en un proyecto para la flexibilidad energética. Partiendo de esta idea se han utilizado una serie de tecnologías: React, bases de datos MySQL, Servidor… para desarrollar un sistema que logre este objetivo. Este objetivo se ha cumplido con la realización de una web interactiva conectada con una base de datos a través de un servidor.

Para que dicho objetivo principal se cumplirá, se fueron cumpliendo todos los objetivos que se plantearon al inicio de la memoria. En primer lugar, para el desarrollo del proyecto fue necesario el aprendizaje de los lenguajes de programación utilizados. Entre ellos JavaScript, HTML y CSS aplicados a React. Una vez aprendidas estas tecnologías, fue posible hacer un diseño del sistema de gamificación al que se quería llegar, estudiando las diferentes tecnologías disponibles para el desarrollo del proyecto.

A continuación, se realizó una primera web que ayudó al aprendizaje de las características de React y posteriormente, se comenzó el desarrollo de la página web final. A ella, se añadieron las tecnologías de servidor y base de datos. Finalmente, una vez desarrollada una web funcional, se realizó una depuración del código para obtener un resultado más limpio y una mejora de la visualización de los componentes de la web.

En conclusión, gracias a una planificación del proyecto y una buena gestión de los recursos tecnológicos utilizados, se ha podido lograr un sistema de gamificación que promueva la participación de las personas en el proyecto de flexibilidad energética. Todos los recursos trabajan juntos para lograr un resultado final único.

Además, la estructura que se ha utilizado para hacer el proyecto es muy flexible y está abierta a cambios en el futuro. Por ejemplo, la utilización de un ORM para el diseño de la base de datos hace que el código pueda adaptarse a otros motores de bases de datos. Lo mismo ocurre con la documentación de la API. Estas ideas que se han ido utilizando son de gran importancia ya que facilitan posibles mejoras en un futuro.

## *14.1 TRABAJOS FUTUROS*

Aunque los objetivos iniciales del proyecto se han cumplido, el trabajo realizado está abierto a nuevos campos o mejoras.

En primer lugar, en la web, se ha desarrollado el sistema del formulario para un usuario de tipo residencial. Es decir, en la web se puede ver que las preguntas que se plantean y el diseño de la vivienda están enfocados a una persona que habita en una vivienda. Sin embargo, la idea es que la flexibilidad llegue a todo tipo de usuarios de la red y no únicamente residenciales. Por ello, una mejora del proyecto sería la introducción de nuevas preguntas y dispositivos, así como la adaptación de la parte interactiva de la red, para usuarios del ámbito comercial o industrial.

A continuación, otra mejora del sistema podría venir de la mano del diseño de la web. Aunque se hayan utilizado componentes de Ant Design para hacer que la web sea más atractiva, todavía se puede mejorar el diseño y las respuestas de la web.

En cuanto al algoritmo desarrollado, también se pueden realizar algunas mejoras que hagan que sus resultados se ajusten más a la realidad o se podrían plantear la introducción de nuevas recompensas que llamen más la atención de los usuarios.

En conclusión, se podrían plantear diferentes mejoras al proyecto en algunos de sus campos. De hecho, como se ha comentado anteriormente, estos cambios serían sencillos de introducir dada la flexibilidad conseguida con las diferentes tecnologías utilizadas.

# Capítulo 15. BIBLIOGRAFÍA

[1] "ReDREAM." https://redream-energy-network.eu/the-project/#about (accessed May 30, 2022).

[2] "Gamify." https://www.gamify.com/what-is-gamification (accessed May 30, 2022).

[3] D. Johnson, E. Horton, R. Mulcahy, and M. Foth, "Gamification and serious games within the domain of domestic energy consumption: A systematic review," *Renewable and Sustainable Energy Reviews*, vol. 73, pp. 249–264, 2017, doi: https://doi.org/10.1016/j.rser.2017.01.134.

[4] A. Giersiepen, C. Koronen, A. Mäkivierikko, A. Nilsson, and H. Shahrokni, "D1.4 Consumers engagement strategies," Jun. 2018.

[5] A. Kashani and Y. Ozturk, "Residential energy consumer behavior modification via gamification," in *2017 IEEE 6th International Conference on Renewable Energy Research and Applications (ICRERA)*, Dec. 2017, pp. 1221–1225. doi: 10.1109/ICRERA.2017.8191247.

[6] T. AlSkaif, I. Lampropoulos, M. van den Broek, and W. van Sark, "Gamification-based framework for engagement of residential customers in energy applications," *Energy Research & Social Science*, vol. 44, pp. 187–195, 2018, doi: https://doi.org/10.1016/j.erss.2018.04.043.

[7] Delta-EE, *Delta-EE Webinar: Gamification – how can it intensify energy customer engagement?*, (Nov. 04, 2020). Accessed: Jan. 23, 2022. [Online Video]. Available: https://www.youtube.com/watch?v=J61Y_szKMow&ab_channel=Delta-EE

[8] E. Doerre, D. Nestle, and J. von Appen, "Enabling flexible electricity demand through event based incentives and gamification," in *International ETG-Congress 2019; ETG Symposium*, 2019, pp. 1–5.

[9] P. Fraternali *et al.*, "A Socio-Technical System Based on Gamification Towards Energy Savings," in *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2018, pp. 59–64. doi: 10.1109/PERCOMW.2018.8480405.

[10] T. G. Papaioannou *et al.*, "IoT-enabled gamification for energy conservation in public buildings," in *2017 Global Internet of Things Summit (GIoTS)*, 2017, pp. 1–6. doi: 10.1109/GIOTS.2017.8016269.

[11] "Naciones Unidas." https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/ (accessed May 26, 2022).

[12] "Eniun." https://www.eniun.com/evolucion-diseno-web/ (accessed Jun. 13, 2022).

[13] "Java T Point." https://www.javatpoint.com/what-is-css (accessed May 28, 2022).

[14] "Epitech." https://www.epitech-it.es/que-es-php/ (accessed Jun. 13, 2022).

[15] "Comunicare." https://www.comunicare.es/javascript-lenguaje-para-el-buen-uso-del-desarrollo-web/#%C2%BFQue_es_la_programacion_y_el_desarrollo_web (accessed Jun. 13, 2022).

[16] "Mdn web docs." https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/What_is_JavaScript (accessed May 26, 2022).

[17] "Openwebinars." https://openwebinars.net/blog/que-es-javascript/ (accessed Jun. 13, 2022).

[18] "EtnasSoft." http://www.etnassoft.com/2011/01/27/tipado-blando-en-javascript/ (accessed Jun. 13, 2022).

[19] "Hevo." https://hevodata.com/learn/javascript-lambda-function/#:~:text=In%20JavaScript%2C%20JavaScript%20Lambda%20usually,pass%20behavior%20as%20a%20value (accessed Jun. 13, 2022).

[20] "mdn web docs." https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array (accessed Jun. 13, 2022).

[21] "Developer Mozilla." https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array (accessed May 30, 2022).

[22] "Fullstack React." https://www.newline.co/fullstack-react/30-days-of-react/day-1/ (accessed May 27, 2022).

[23] "StackOverflow." https://insights.stackoverflow.com/survey/2021#most-popular-technologies-webframe (accessed Jun. 13, 2022).

[24] "Monterail." https://www.monterail.com/blog/vue-vs-react (accessed Jun. 13, 2022).

[25] "React." https://es.reactjs.org/docs/rendering-elements.html (accessed Jun. 13, 2022).

[26] "React Components." https://es.reactjs.org/docs/components-and-props.html (accessed May 30, 2022).

[27] "React." https://es.reactjs.org/docs/lifting-state-up.html (accessed Jun. 13, 2022).

[28] "React Hooks." https://es.reactjs.org/docs/hooks-overview.html (accessed May 30, 2022).

[29] "Node js." https://nodejs.dev/learn (accessed May 27, 2022).

[30] "Express." https://expressjs.com/ (accessed May 27, 2022).

[31] "Xataka." https://www.xataka.com/basics/api-que-sirve (accessed May 27, 2022).

[32] "Red Hat." https://www.redhat.com/es/topics/api/what-is-a-rest-api (accessed Jun. 14, 2022).

[33] "Typescript." https://www.typescriptlang.org/ (accessed May 26, 2022).

[34] "Geeks for Geeks." https://www.geeksforgeeks.org/difference-between-sql-and-nosql/ (accessed Jun. 14, 2022).

[35] "Talend." https://www.talend.com/resources/what-is-mysql/ (accessed May 27, 2022).

[36] "Computer Hope." https://www.computerhope.com/jargon/w/wsl.htm (accessed May 27, 2022).

[37] "InfoWorld." https://www.infoworld.com/article/3204171/what-is-docker-the-spark-for-the-container-revolution.html (accessed May 27, 2022).

[38] "IBM." https://www.ibm.com/in-en/cloud/learn/docker (accessed May 27, 2022).

[39] "Docker Docs." https://docs.docker.com/compose/compose-file/ (accessed Jun. 14, 2022).

[40] "Database.Guide." https://database.guide/what-is-dbeaver/ (accessed May 27, 2022).

[41] "Prisma." https://www.prisma.io/react-server-components (accessed May 26, 2022).

[42] "Db diagram." https://dbdiagram.io/d/6294aa5cf040f104c1c0802b (accessed May 30, 2022).

[43] "Prisma." https://www.prisma.io/docs/concepts/components/prisma-schema/data-model (accessed May 31, 2022).

[44] "mdn web docs." https://developer.mozilla.org/es/docs/Web/HTTP/Status (accessed May 31, 2022).

[45] "Geeks for Geeks." https://www.geeksforgeeks.org/introduction-postman-api-development/ (accessed May 27, 2022).

[46] "Github." https://express-validator.github.io/docs/ (accessed May 31, 2022).

[47] "Codete." https://codete.com/blog/type-script-vs-java-script-which-one-to-choose-for-2021-web-applications (accessed Jun. 14, 2022).

[48] "Electricity Map." https://app.electricitymap.org/zone/ES (accessed May 18, 2022).

[49] "Apicurio." https://www.apicur.io/studio/ (accessed May 27, 2022).

[50] "Akana." https://www.akana.com/blog/what-is-api-documentation (accessed May 27, 2022).

[51] "Redux." https://redux-toolkit.js.org/introduction/getting-started (accessed Jun. 01, 2022).

[52] "LogRocket Fronted Monitoring." https://blog.logrocket.com/why-use-redux-reasons-with-clear-examples-d21bffd5835/ (accessed May 27, 2022).

[53] "Formik." https://formik.org/ (accessed May 27, 2022).

[54] "Ant Design." https://ant.design/docs/react/introduce (accessed May 27, 2022).

[55] "mdn web docs." https://developer.mozilla.org/es/docs/Web/CSS/flex (accessed Jun. 06, 2022).

[56] "CSS-Tricks." https://css-tricks.com/snippets/css/a-guide-to-flexbox/ (accessed Jun. 15, 2022).

[57] "Noun project." https://thenounproject.com/icon/heat-pump-3498284/ (accessed Jun. 06, 2022).

[58] "FlatIcon." https://www.flaticon.es/icono-gratis/gas_3520393 (accessed Jun. 06, 2022).

[59] "SimpliLearn." https://www.simplilearn.com/tutorials/git-tutorial/what-is-gitlab (accessed May 27, 2022).

# ANEXO I: ARCHIVOS BASE DE DATOS Y SERVIDOR

## *ANEXO I.I: ARCHIVO MIGRATION.SQL*

```
-- CreateTable
CREATE TABLE `User` (
    `id` INTEGER NOT NULL AUTO_INCREMENT,
    `email` VARCHAR(191) NOT NULL,
    `name` VARCHAR(191) NOT NULL,
    `surname` VARCHAR(191) NOT NULL,

    PRIMARY KEY (`id`)
) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- CreateTable
CREATE TABLE `UserProfile` (
    `id` INTEGER NOT NULL AUTO_INCREMENT,
    `user_id` INTEGER NOT NULL,
    `created_at` DATETIME(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
    `updated_at` DATETIME(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
    `user_type` VARCHAR(191) NULL,
    `size_type` VARCHAR(191) NULL,
    `n_heatpumps` INTEGER NULL,
    `n_radiators` INTEGER NULL,
    `n_ac` INTEGER NULL,
    `has_gas` BOOLEAN NULL,
    `hw_boiler` BOOLEAN NULL,
    `hw_gas` BOOLEAN NULL,
    `n_evcp` INTEGER NULL,
    `n_pv` INTEGER NULL,
    `n_bat` INTEGER NULL,

    PRIMARY KEY (`id`)
) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- CreateTable
CREATE TABLE `AlgorithmExecution` (
    `id` INTEGER NOT NULL AUTO_INCREMENT,
    `profile_id` INTEGER NOT NULL,
    `datetime` DATETIME(3) NOT NULL,

    UNIQUE INDEX `AlgorithmExecution_profile_id_key`(`profile_id`),
    PRIMARY KEY (`id`)
) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- CreateTable
CREATE TABLE `AlgorithmResult` (
    `algorithm_id` INTEGER NOT NULL,
    `kpi_id` INTEGER NOT NULL,
    `value` DOUBLE NOT NULL,

    PRIMARY KEY (`algorithm_id`, `kpi_id`)
) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

```
-- CreateTable
CREATE TABLE `Kpi` (
    `id` INTEGER NOT NULL AUTO_INCREMENT,
    `description` VARCHAR(191) NOT NULL,
    `units` VARCHAR(191) NOT NULL,

    PRIMARY KEY (`id`)
) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- AddForeignKey
ALTER TABLE `UserProfile` ADD CONSTRAINT `UserProfile_user_id_fkey` FOREIGN KEY (`user_id`)
REFERENCES `User`(`id`) ON DELETE RESTRICT ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE `AlgorithmExecution` ADD CONSTRAINT `AlgorithmExecution_profile_id_fkey`
FOREIGN KEY (`profile_id`) REFERENCES `UserProfile`(`id`) ON DELETE RESTRICT ON UPDATE
CASCADE;

-- AddForeignKey
ALTER TABLE `AlgorithmResult` ADD CONSTRAINT `AlgorithmResult_algorithm_id_fkey` FOREIGN
KEY (`algorithm_id`) REFERENCES `AlgorithmExecution`(`id`) ON DELETE RESTRICT ON UPDATE
CASCADE;

-- AddForeignKey
ALTER TABLE `AlgorithmResult` ADD CONSTRAINT `AlgorithmResult_kpi_id_fkey` FOREIGN KEY
(`kpi_id`) REFERENCES `Kpi`(`id`) ON DELETE RESTRICT ON UPDATE CASCADE;
```

*Código 56: Archivo migration.sql*

## ANEXO I.II: ARCHIVO INDEX.TS (SERVIDOR)

```
import express from "express";
import { PrismaClient } from "@prisma/client";

import { body, query, validationResult } from "express-validator";
import { findClosestMatch } from "./algorithm";
const prisma = new PrismaClient();

const app = express();
app.use(express.json());

const port = 5000;

app.get(
  "/users",
  query("email").isEmail().withMessage("is not an email"),
  async (req, res) => {
    const { email } = req.query;
    const users = await prisma.user.findMany({
      where: {
        email,
      },
    });
    res.status(200).send(users);
  }
);
```

```
app.get("/users/:userId", async (req, res) => {
  const { userId } = req.params;

  const user = await prisma.user.findUnique({
    where: { id: parseInt(userId) },
  });

  if (!user) return res.status(404).send(`User ${userId} not found`);

  res.status(200).send(user);
});

app.post(
  "/users",
  body("id").not().exists().withMessage("can not pass the id"),
  body("email").isEmail().withMessage("is not an email"),
  body("name").isString().withMessage("is not a string"),
  body("surname").isString().withMessage("is not a string"),
  async (req, res) => {
    // Finds the validation errors in this request and wraps them in an object with handy
functions
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }
    //const { email, name, surname } = req.body;
    //req.body;
    const user = await prisma.user.create({
      data: req.body,
    });
    res.status(201).send(user);
  }
);


app.get(
  "/usersProfiles",
  query("user_id").isInt().withMessage("is not an Int"),
  async (req, res) => {
    // se llama a base de datos para pedir todos los perfiles

    let { user_id } = req.query;
    user_id = parseInt(user_id);
    const usersProfiles = await prisma.userProfile.findMany({
      where: {
        user_id,
      },
    });

    res.status(200).send(usersProfiles);
  }
);
app.get("/userProfile/:userId", async (req, res) => {
  const { userId } = req.params;

  const userProfile = await prisma.userProfile.findMany({
    where: { user_id: parseInt(userId) },
  });
  //console.log(userProfile.lenght)
```

```
  if (userProfile.length === 0)
    return res.status(404).send(`User ${userId} not found`);

  res.status(200).send(userProfile);
});
// Se crea un perfil para un usuario
app.post(
  "/usersProfiles",
  body("id").not().exists().withMessage("can not pass the id"),
  body("user_id").isInt().withMessage("is not an int"),
  body("user_type").optional().isString().withMessage("is not a string"),
  body("size_type").optional().isString().withMessage("is not a string"),
  body("n_heatpumps").optional().isInt().withMessage("is not an int"),
  body("n_radiators").optional().isInt().withMessage("is not an int"),
  body("n_ac").optional().isInt().withMessage("is not an int"),
  body("has_gas").optional().isBoolean().withMessage("is not a boolean"),
  body("hw_boiler").optional().isBoolean().withMessage("is not a boolean"),
  body("hw_gas").optional().isBoolean().withMessage("is not a boolean"),
  body("n_evcp").optional().isInt().withMessage("is not an int"),
  body("n_pv").optional().isInt().withMessage("is not an int"),
  body("n_bat").optional().isInt().withMessage("is not an int"),

  async (req, res) => {
    console.log(req.body);
    //   // Finds the validation errors in this request and wraps them in an object with
handy functions

    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }

    const newProfile = await prisma.userProfile.create({
      data: req.body,
    });
    res.status(201).send(newProfile);
  }
);

app.patch(
  "/usersProfiles/:profileId",
  body("id").not().exists().withMessage("can not pass the id"),
  body("user_id").optional().isInt().withMessage("is not an int"),
  body("user_type").optional().isString().withMessage("is not a string"),
  body("size_type").optional().isString().withMessage("is not a string"),
  body("n_heatpumps").optional().isInt().withMessage("is not an int"),
  body("n_radiators").optional().isInt().withMessage("is not an int"),
  body("n_ac").optional().isInt().withMessage("is not an int"),
  body("has_gas").optional().isBoolean().withMessage("is not a boolean"),
  body("hw_boiler").optional().isBoolean().withMessage("is not a boolean"),
  body("hw_gas").optional().isBoolean().withMessage("is not a boolean"),
  body("n_evcp").optional().isInt().withMessage("is not an int"),
  body("n_pv").optional().isInt().withMessage("is not an int"),
  body("n_bat").optional().isInt().withMessage("is not an int"),

  async (req, res) => {
    console.log(req.body);
    //   // Finds the validation errors in this request and wraps them in an object with
handy functions
```

```
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }

    const { profileId } = req.params;

    const nMatchingProfiles = await prisma.userProfile.count({
      where: { id: parseInt(profileId) },
    });

    if (nMatchingProfiles < 1) return res.status(404).send();

    const updatedProfile = await prisma.userProfile.update({
      where: { id: parseInt(profileId) },
      data: req.body,
    });

    res.status(201).send(updatedProfile);
  }
);

app.get("/algorithmsExecutions", async (req, res) => {
  // se llama a base de datos para pedir todos las ejecucuines de algoritmos
  const algorithmExecution = await prisma.algorithmExecution.findMany();
  // una vez responda, devuelvo todos algoritmos
  res.status(200).send(algorithmExecution);
});
app.get("/algorithmsExecutions/:userProfileId", async (req, res) => {
  const { userProfileId } = req.params;

  const algorithmExecution = await prisma.algorithmExecution.findMany({
    where: { profile_id: parseInt(userProfileId) },
  });
  //console.log(userProfile.lenght)
  if (algorithmExecution.length === 0)
    return res.status(404).send(`User profile ${userProfileId} not found`);

  res.status(200).send(algorithmExecution);
});
app.post(
  "/algorithmsExecutions",
  body("profile_id").isInt().withMessage("is not a int"),
  body("id").not().exists().withMessage("can not pass the id"),
  async (req, res) => {
    const { profile_id } = req.body;

    const errors = validationResult(req);

    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }

    const nExecutionsWithThisProfile = await prisma.algorithmExecution.count({
      where: { profile_id },
    });

    if (nExecutionsWithThisProfile !== 0)
      return res
        .status(400)
```

```
        .send(
          `An algorithm execution with profile id: ${profile_id} already exists`
        );

    const newAlgorithmExecution = await prisma.algorithmExecution.create({
      data: { ...req.body, datetime: new Date() },
    });

    // 1. llamo a base de datros con await y pido el UserProfileId que me han pasado
    const userProfile = await prisma.userProfile.findUnique({
      where: { id: parseInt(profile_id) },
    });

    const algorithmResults = findClosestMatch(
      newAlgorithmExecution.id,
      userProfile
    );
    // 3. inserto algorithm results
    const newAlgorithmResult = await prisma.algorithmResult.createMany({
      data: algorithmResults,
    });
    const algorithmWithResults = await prisma.algorithmExecution.findFirst({
      where: { id: newAlgorithmExecution.id },
    });

    // algorithmWithResults.results

    res.status(201).send(algorithmResults);
  }
);
app.get("/algorithmsResults", async (req, res) => {
  // se llama a base de datos para pedir todos las ejecucuines de algoritmos
  const algorithmResult = await prisma.algorithmResult.findMany();
  // una vez responda, devuelvo todos algoritmos
  res.status(200).send(algorithmResult);
});
app.post(
  "/algorithmsResults",
  body().isArray(),
  body("*.algorithm_id").isInt().withMessage("is not an Int"),
  body("*.kpi_id").isInt().withMessage("is not an Int"),
  body("*.value").isFloat().withMessage("is not a Float"),

  async (req, res) => {
    // Finds the validation errors in this request and wraps them in an object with handy
functions
    const errors = validationResult(req);
    console.log("Miro errores");
    console.log(errors);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }

    const newAlgorithmResult = await prisma.algorithmResult.createMany({
      data: req.body,
    });

    console.log(req.body);
    res.status(201).send(newAlgorithmResult);
  }
```

```
);

app.get("/kpis", async (req, res) => {
  // se llama a base de datos para pedir todos las ejecucuines de algoritmos
  const kpi = await prisma.kpi.findMany();
  // una vez responda, devuelvo todos algoritmos
  res.status(200).send(kpi);
});
app.post(
  "/kpis",
  body("description").isString().withMessage("is not a string"),
  body("units").isString().withMessage("is not a String"),

  async (req, res) => {
    // Finds the validation errors in this request and wraps them in an object with handy
functions
    const errors = validationResult(req);
    console.log("Miro errores");
    console.log(errors);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }
    // const { algorithm_id, kpi_id, value } = req.body;
    const newKpi = await prisma.kpi.create({ data: req.body });
    console.log(req.body);
    res.status(201).send(newKpi);
  }
);
app.listen(port, () => console.log(`Running on port ${port}`));
```

*Código 57: Archivo index.ts de peticiones del servidor*

# ANEXO II: DOCUMENTACIÓN API Y REDUX

## *ANEXO II.I: CÓDIGO DE DOCUMENTACIÓN EN APICURIO*

```yaml
openapi: 3.0.2
info:
    title: TFG
    version: 1.0.0
paths:
    /users:
        get:
            parameters:
                - name: email
                  description: ''
                  schema:
                      type: string
                  in: query
                  required: false
            responses:
                '200':
                    content:
                        application/json:
                            schema:
                                type: array
                                items:
                                    $ref: '#/components/schemas/User'
                    description: OK
            operationId: getUsers
        post:
            requestBody:
                content:
                    application/json:
                        schema:
                            $ref: '#/components/schemas/UserWithoutId'
                required: true
            responses:
                '201':
                    content:
                        application/json:
                            schema:
                                $ref: '#/components/schemas/User'
                    description: OK
            operationId: createUser
    '/users/{userId}':
        get:
            responses:
                '200':
                    content:
                        application/json:
                            schema:
                                $ref: '#/components/schemas/User'
                    description: OK
                '404':
```

```
                description: NF
        operationId: getUser
    parameters:
        - name: userId
          schema:
              type: integer
          in: path
          required: true
/usersProfiles:
    get:
        parameters:
            - name: user_id
              description: ''
              schema:
                  type: integer
              in: query
              required: false
        responses:
            '200':
                content:
                    application/json:
                        schema:
                            type: array
                            items:
                                $ref: '#/components/schemas/UserProfile'
                description: ok
        operationId: getUsersProfiles
        description: Obtengo todos los perfiles de los usuarios
    post:
        requestBody:
            content:
                application/json:
                    schema:
                        $ref: '#/components/schemas/UserProfileWithoutId'
            required: true
        responses:
            '201':
                content:
                    application/json:
                        schema:
                            $ref: '#/components/schemas/UserProfile'
                description: ok
            '400':
                description: bad
        operationId: createUserProfile
'/usersProfiles/{profileId}':
    patch:
        requestBody:
            content:
                application/json:
                    schema:
                        $ref: '#/components/schemas/UserProfileWithoutId'
            required: true
        responses:
            '201':
                content:
                    application/json:
                        schema:
                            type: array
                            items:
```

```
                                $ref: '#/components/schemas/UserProfile'
                    description: ok
                '400':
                    description: bad
            operationId: changeUsersProfiles
        parameters:
            - name: profileId
              schema:
                  type: integer
              in: path
              required: true
    '/userProfile/{userId}':
        description: Devuelve todos los perfiles del usuario que se le indica
        get:
            responses:
                '200':
                    content:
                        application/json:
                            schema:
                                type: array
                                items:
                                    $ref: '#/components/schemas/UserProfile'
                    description: ok
                '404':
                    description: bad
            operationId: getUserProfile
        parameters:
            - name: userId
              schema:
                  type: integer
              in: path
              required: true
    /algorithmsExecutions:
        get:
            parameters:
                - name: userProfileId
                  description: ''
                  schema:
                      type: integer
                  in: query
            responses:
                '200':
                    content:
                        application/json:
                            schema:
                                type: array
                                items:
                                    $ref: '#/components/schemas/AlgorithmExecution'
                    description: ok
            operationId: getAlgorithmExecution
        post:
            requestBody:
                content:
                    application/json:
                        schema:
                            $ref: '#/components/schemas/AlgorithmExecutionWithoutId'
                required: true
            responses:
                '201':
                    content:
```

```
                    application/json:
                        schema:
                            type: array
                            items:
                                $ref: '#/components/schemas/AlgorithmResult'
                description: ok
            '400':
                description: bad
        operationId: createAlgorithmExecution
/algorithmsResults:
    get:
        responses:
            '200':
                content:
                    application/json:
                        schema:
                            type: array
                            items:
                                $ref: '#/components/schemas/AlgorithmResult'
                description: ok
        operationId: getAlgorithmsResults
    post:
        requestBody:
            content:
                application/json:
                    schema:
                        type: array
                        items:
                            $ref: '#/components/schemas/AlgorithmResult'
            required: true
        responses:
            '201':
                content:
                    application/json:
                        schema:
                            $ref: '#/components/schemas/AlgorithmResult'
                description: ok
            '400':
                description: bad
        operationId: createAlgorithmsResults
/kpis:
    get:
        responses:
            '200':
                content:
                    application/json:
                        schema:
                            type: array
                            items:
                                $ref: '#/components/schemas/Kpi'
                description: ok
        operationId: getKpis
    post:
        requestBody:
            content:
                application/json:
                    schema:
                        $ref: '#/components/schemas/KpiWithoutId'
            required: true
        responses:
```

```
                '201':
                    content:
                        application/json:
                            schema:
                                $ref: '#/components/schemas/Kpi'
                    description: ok
                '400':
                    description: bad
            operationId: createKpi
components:
    schemas:
        UserWithoutId:
            title: Root Type for UserWithoutId
            description: ''
            required:
                - email
                - name
                - surname
            type: object
            properties:
                email:
                    type: string
                name:
                    type: string
                surname:
                    type: string
            example:
                email: mercedes@gmail.com
                name: Mercedes
                surname: Burillo
        User:
            description: ''
            type: object
            allOf:
                - required:
                      - id
                  type: object
                  properties:
                      id:
                          description: ''
                          type: integer
                - $ref: '#/components/schemas/UserWithoutId'
        UserProfileWithoutId:
            title: Root Type for UserProfileWithoutId
            description: 'Perfil de usuario sin el id del perfil, no del usuario'
            type: object
            properties:
                user_id:
                    format: int32
                    type: integer
                user_type:
                    type: string
                size_type:
                    type: string
                n_heatpumps:
                    format: int32
                    type: integer
                n_radiators:
                    format: int32
                    type: integer
```

```
            n_ac:
                format: int32
                type: integer
            has_gas:
                type: boolean
            hw_boiler:
                type: boolean
            hw_gas:
                type: boolean
            n_evcp:
                format: int32
                type: integer
            n_pv:
                format: int32
                type: integer
            n_bat:
                format: int32
                type: integer
        example:
            user_id: 1
            user_type: residential
            size_type: small
            n_heatpumps: 2
            n_radiators: 3
            n_ac: 4
            has_gas: true
            hw_boiler: true
            hw_gas: true
            n_evcp: 5
            n_pv: 6
            n_bat: 7
    UserProfile:
        description: ''
        type: object
        allOf:
            - required:
                - id
              type: object
              properties:
                id:
                    description: ''
                    type: integer
            - $ref: '#/components/schemas/UserProfileWithoutId'
    AlgorithmExecutionWithoutId:
        title: Root Type for AlgorithmExecutionWithoutId
        description: ''
        required:
            - datetime
            - profile_id
        type: object
        properties:
            profile_id:
                format: int32
                type: integer
            datetime:
                format: date-time
                type: string
        example:
            profile_id: 1
            datetime: '2022-04-04T13:25:57.843Z'
```

```
    AlgorithmExecution:
        description: ''
        type: object
        allOf:
            - required:
                - id
              type: object
              properties:
                  id:
                      description: ''
                      type: integer
            - $ref: '#/components/schemas/AlgorithmExecutionWithoutId'
    AlgorithmResult:
        title: Root Type for AlgorithmResult
        description: ''
        type: object
        properties:
            algorithm_id:
                format: int32
                type: integer
            kpi_id:
                format: int32
                type: integer
            value:
                format: int32
                type: integer
        example:
            algorithm_id: 1
            kpi_id: 1
            value: 3
    KpiWithoutId:
        title: Root Type for KpiWithoutId
        description: ''
        required:
            - description
            - units
        type: object
        properties:
            description:
                type: string
            units:
                type: string
        example:
            description: es una recompensa en dinero
            units: euros
    Kpi:
        description: ''
        type: object
        allOf:
            - required:
                - Id
              type: object
              properties:
                  Id:
                      description: ''
                      type: integer
            - $ref: '#/components/schemas/KpiWithoutId'
```

*Código 58: tfg-api-documentation.yml*

# *ANEXO II.II: CÓDIGO TFGAPI*

```
import { emptyTfgApi as api } from './emptyTfgApi'
const injectedRtkApi = api.injectEndpoints({
    endpoints: (build) => ({
        getUsers: build.query<GetUsersApiResponse, GetUsersApiArg>({
            query: (queryArg) => ({
                url: `/users`,
                params: { email: queryArg.email }
            })
        }),
        createUser: build.mutation<CreateUserApiResponse, CreateUserApiArg>({
            query: (queryArg) => ({
                url: `/users`,
                method: 'POST',
                body: queryArg.userWithoutId
            })
        }),
        getUser: build.query<GetUserApiResponse, GetUserApiArg>({
            query: (queryArg) => ({ url: `/users/${queryArg.userId}` })
        }),
        getUsersProfiles: build.query<
            GetUsersProfilesApiResponse,
            GetUsersProfilesApiArg
        >({
            query: (queryArg) => ({
                url: `/usersProfiles`,
                params: { user_id: queryArg.userId }
            })
        }),
        createUserProfile: build.mutation<
            CreateUserProfileApiResponse,
            CreateUserProfileApiArg
        >({
            query: (queryArg) => ({
                url: `/usersProfiles`,
                method: 'POST',
                body: queryArg.userProfileWithoutId
            })
        }),
        changeUsersProfiles: build.mutation<
            ChangeUsersProfilesApiResponse,
            ChangeUsersProfilesApiArg
        >({
            query: (queryArg) => ({
                url: `/usersProfiles/${queryArg.profileId}`,
                method: 'PATCH',
                body: queryArg.userProfileWithoutId
            })
        }),
        getUserProfile: build.query<
            GetUserProfileApiResponse,
            GetUserProfileApiArg
        >({
            query: (queryArg) => ({ url: `/userProfile/${queryArg.userId}` })
        }),
        getAlgorithmExecution: build.query<
            GetAlgorithmExecutionApiResponse,
```

```
            GetAlgorithmExecutionApiArg
        >({
            query: (queryArg) => ({
                url: `/algorithmsExecutions`,
                params: { userProfileId: queryArg.userProfileId }
            })
        }),
        createAlgorithmExecution: build.mutation<
            CreateAlgorithmExecutionApiResponse,
            CreateAlgorithmExecutionApiArg
        >({
            query: (queryArg) => ({
                url: `/algorithmsExecutions`,
                method: 'POST',
                body: queryArg.algorithmExecutionWithoutId
            })
        }),
        getAlgorithmsResults: build.query<
            GetAlgorithmsResultsApiResponse,
            GetAlgorithmsResultsApiArg
        >({
            query: () => ({ url: `/algorithmsResults` })
        }),
        createAlgorithmsResults: build.mutation<
            CreateAlgorithmsResultsApiResponse,
            CreateAlgorithmsResultsApiArg
        >({
            query: (queryArg) => ({
                url: `/algorithmsResults`,
                method: 'POST',
                body: queryArg.body
            })
        }),
        getKpis: build.query<GetKpisApiResponse, GetKpisApiArg>({
            query: () => ({ url: `/kpis` })
        }),
        createKpi: build.mutation<CreateKpiApiResponse, CreateKpiApiArg>({
            query: (queryArg) => ({
                url: `/kpis`,
                method: 'POST',
                body: queryArg.kpiWithoutId
            })
        })
    }),
    overrideExisting: false
})
export { injectedRtkApi as tfgApi }
export type GetUsersApiResponse = /** status 200 OK */ User[]
export type GetUsersApiArg = {
    email?: string
}
export type CreateUserApiResponse = /** status 201 OK */ User
export type CreateUserApiArg = {
    userWithoutId: RootTypeForUserWithoutId
}
export type GetUserApiResponse = /** status 200 OK */ User
export type GetUserApiArg = {
    userId: number
}
export type GetUsersProfilesApiResponse = /** status 200 ok */ UserProfile[]
```

```
export type GetUsersProfilesApiArg = {
    userId?: number
}
export type CreateUserProfileApiResponse = /** status 201 ok */ UserProfile
export type CreateUserProfileApiArg = {
    userProfileWithoutId: RootTypeForUserProfileWithoutId
}
export type ChangeUsersProfilesApiResponse = /** status 201 ok */ UserProfile[]
export type ChangeUsersProfilesApiArg = {
    profileId: number
    userProfileWithoutId: RootTypeForUserProfileWithoutId
}
export type GetUserProfileApiResponse = /** status 200 ok */ UserProfile[]
export type GetUserProfileApiArg = {
    userId: number
}
export type GetAlgorithmExecutionApiResponse =
    /** status 200 ok */ AlgorithmExecution[]
export type GetAlgorithmExecutionApiArg = {
    userProfileId?: number
}
export type CreateAlgorithmExecutionApiResponse =
    /** status 201 ok */ RootTypeForAlgorithmResult[]
export type CreateAlgorithmExecutionApiArg = {
    algorithmExecutionWithoutId: RootTypeForAlgorithmExecutionWithoutId
}
export type GetAlgorithmsResultsApiResponse =
    /** status 200 ok */ RootTypeForAlgorithmResult[]
export type GetAlgorithmsResultsApiArg = void
export type CreateAlgorithmsResultsApiResponse =
    /** status 201 ok */ RootTypeForAlgorithmResult
export type CreateAlgorithmsResultsApiArg = {
    body: RootTypeForAlgorithmResult[]
}
export type GetKpisApiResponse = /** status 200 ok */ Kpi[]
export type GetKpisApiArg = void
export type CreateKpiApiResponse = /** status 201 ok */ Kpi
export type CreateKpiApiArg = {
    kpiWithoutId: RootTypeForKpiWithoutId
}
export type RootTypeForUserWithoutId = {
    email: string
    name: string
    surname: string
}
export type User = {
    id: number
} & RootTypeForUserWithoutId
export type RootTypeForUserProfileWithoutId = {
    user_id?: number
    user_type?: string
    size_type?: string
    n_heatpumps?: number
    n_radiators?: number
    n_ac?: number
    has_gas?: boolean
    hw_boiler?: boolean
    hw_gas?: boolean
    n_evcp?: number
    n_pv?: number
```

```
    n_bat?: number
}
export type UserProfile = {
    id: number
} & RootTypeForUserProfileWithoutId
export type RootTypeForAlgorithmExecutionWithoutId = {
    profile_id: number
    datetime: string
}
export type AlgorithmExecution = {
    id: number
} & RootTypeForAlgorithmExecutionWithoutId
export type RootTypeForAlgorithmResult = {
    algorithm_id?: number
    kpi_id?: number
    value?: number
}
export type RootTypeForKpiWithoutId = {
    description: string
    units: string
}
export type Kpi = {
    Id: number
} & RootTypeForKpiWithoutId
export const {
    useGetUsersQuery,
    useCreateUserMutation,
    useGetUserQuery,
    useGetUsersProfilesQuery,
    useCreateUserProfileMutation,
    useChangeUsersProfilesMutation,
    useGetUserProfileQuery,
    useGetAlgorithmExecutionQuery,
    useCreateAlgorithmExecutionMutation,
    useGetAlgorithmsResultsQuery,
    useCreateAlgorithmsResultsMutation,
    useGetKpisQuery,
    useCreateKpiMutation
} = injectedRtkApi
```

*Código 59: tfgApi.ts*

# ANEXO III: CÓDIGOS DESARROLLO WEB

## *ANEXO III.I: CÓDIGO USERHOUSEFORM*

```
import { Field, Form, Formik, FormikHelpers, FormikValues } from 'formik'

import { useEffect, useState } from 'react'
import radiatorIcon from '../../assets/radiatorIcon.svg'
import hwBoilerIcon from '../../assets/hwBoilerIcon.svg'
import pvIcon from '../../assets/pvIcon.svg'
import acIcon from '../../assets/acIcon.svg'
import evcpIcon from '../../assets/evcpIcon.svg'
import batIcon from '../../assets/batIcon.svg'
import heatpumpIcon from '../../assets/heatpumpIcon.png'
import hwGasIcon from '../../assets/hwGasIcon.svg'
import gasIcon from '../../assets/gasIcon.png'
import { DisplayFormikState } from './helper'
import * as Yup from 'yup'
import { FormikListener, handleFormChange } from './FormikListener'
import { useSelector } from 'react-redux'
import { RootState } from 'redux/store'
import { Button, Col, Radio, Row, Space } from 'antd'
import { MinusOutlined, PlusOutlined } from '@ant-design/icons'

export const UserHouseForm = () => {
    const userProfile = useSelector(
        (state: RootState) => state.globalState.userProfile
    )
    const convertString = (word) => {
        switch (word) {
            case 'yes':
            case 'true':
            case '1':
                return true
            case 'no':
            case 'false':
            case '0':
            case null:
                return false
            default:
                return Boolean(word)
        }
    }

    return (
        <div>
            <Formik
                initialValues={{
                    has_gas: userProfile.has_gas.toString(),
                    n_radiators: userProfile.n_radiators,
                    n_heatpumps: userProfile.n_heatpumps,
                    n_ac: userProfile.n_ac,
```

```
                hw_boiler: userProfile.hw_boiler.toString(),
                hw_gas: userProfile.hw_gas.toString(),
                n_evcp: userProfile.n_evcp,
                n_pv: userProfile.n_pv,
                n_bat: userProfile.n_bat,
                profileId: userProfile.id,
                size_type: userProfile.size_type
            }}
            onSubmit={async (values) => {
                await new Promise((resolve) => setTimeout(resolve, 500))
                alert(JSON.stringify(values, null, 2))
            }}
            validationSchema={Yup.object().shape({
                email: Yup.string().email().required('Required')
            })}
        >
            {(props) => {
                const {
                    values,
                    touched,
                    errors,
                    dirty,
                    isSubmitting,
                    handleChange,
                    handleBlur,
                    handleSubmit,
                    handleReset,
                    setFieldValue
                } = props
                return (
                    <div className="formAndHouse">
                        <div className="form">
                            <form onSubmit={handleSubmit}>
                                <h3>¿De qué tamaño es mi casa?</h3>

                                <Radio.Group
                                    buttonStyle="solid"
                                    value={values.size_type}
                                    onChange={(e) =>
                                        setFieldValue(
                                            'size_type',
                                            e.target.value
                                        )
                                    }
                                >
                                    <Space direction="vertical">
                                        <Radio.Button
                                            value={'small'}
                                            name="radio"
                                        >
                                            <Field
                                                type="radio"
                                                name="size_type"
                                                value="small"
                                            />
                                            Mi casa es pequeña, tiene menos
                                            de 70 m2
                                        </Radio.Button>
                                        <Radio.Button
                                            value={'medium'}
```

```
                        name="size_type"
                    >
                        <Field
                            type="radio"
                            name="size_type"
                            value="medium"
                        />
                        Mi casa es mediana, tiene menos
                        de 180 m2
                    </Radio.Button>
                    <Radio.Button
                        value={'big'}
                        name="size_type"
                    >
                        <Field
                            type="radio"
                            name="size_type"
                            value="big"
                        />
                        Mi casa es grande, tiene más de
                        180 m2
                    </Radio.Button>
                </Space>
            </Radio.Group>

            <FormikListener
                values={values}
                callback={handleFormChange}
            />
            <h3>¿Cómo caliento mi casa?</h3>
            <Space>
                <p>Tengo gas: </p>
                <Radio.Group
                    buttonStyle="solid"
                    value={values.has_gas}
                    onChange={(e) =>
                        setFieldValue(
                            'has_gas',
                            e.target.value
                        )
                    }
                >
                    <Space size={8}>
                        <Radio.Button
                            value={'true'}
                            name="radio"
                            style={{
                                marginTop: '10px',
                                marginBottom: '10px'
                            }}
                        >
                            <Field
                                type="radio"
                                name="has_gas"
                                value="true"
                            />
                            Sí
                        </Radio.Button>
                        <Radio.Button
                            value={'false'}
```

```
                                name="has_gas"
                            >
                                <Field
                                    type="radio"
                                    name="has_gas"
                                    value="false"
                                />
                                No
                            </Radio.Button>
                        </Space>
                    </Radio.Group>
                </Space>

                <br></br>

                <label>
                    <Space>
                        <p>Tengo radiadores eléctricos:</p>

                        <Button
                            type="primary"
                            shape="circle"
                            icon={<PlusOutlined />}
                            onClick={() => {
                                setFieldValue(
                                    'n_radiators',
                                    values.n_radiators + 1
                                )
                            }}
                        />
                        <Button
                            type="primary"
                            shape="circle"
                            icon={<MinusOutlined />}
                            onClick={() =>
                                values.n_radiators > 0 &&
                                setFieldValue(
                                    'n_radiators',
                                    values.n_radiators - 1
                                )
                            }
                        />
                        <p>{values.n_radiators}</p>
                    </Space>
                </label>

                <br></br>
                <label>
                    <Space>
                        <p>Tengo bombas de calor:</p>
                        <Button
                            type="primary"
                            shape="circle"
                            icon={<PlusOutlined />}
                            onClick={() => {
                                setFieldValue(
                                    'n_heatpumps',
                                    values.n_heatpumps + 1
                                )
                            }}
```

```
                        />
                        <Button
                            type="primary"
                            shape="circle"
                            icon={<MinusOutlined />}
                            onClick={() =>
                                values.n_heatpumps > 0 &&
                                setFieldValue(
                                    'n_heatpumps',
                                    values.n_heatpumps - 1
                                )
                            }
                        />
                        <p>{values.n_heatpumps}</p>
                    </Space>
                </label>
                <br></br>
                <label>
                    <Space>
                        <p>Tengo aire acondicionado:</p>
                        <Button
                            type="primary"
                            shape="circle"
                            icon={<PlusOutlined />}
                            onClick={() => {
                                setFieldValue(
                                    'n_ac',
                                    values.n_ac + 1
                                )
                            }}
                        />
                        <Button
                            type="primary"
                            shape="circle"
                            icon={<MinusOutlined />}
                            onClick={() =>
                                values.n_ac > 0 &&
                                setFieldValue(
                                    'n_ac',
                                    values.n_ac - 1
                                )
                            }
                        />
                        <p>{values.n_ac}</p>
                    </Space>
                </label>
                <br></br>

                <h3>¿Cómo caliento mi agua sanitaria?</h3>
                <Space>
                    <p>Tengo gas:</p>
                    <Radio.Group
                        buttonStyle="solid"
                        value={values.hw_gas}
                        onChange={(e) =>
                            setFieldValue(
                                'hw_gas',
                                e.target.value
                            )
                        }
```

```
                    >
                        <Space size={8}>
                            <Radio.Button
                                value={'true'}
                                name="radio"
                                style={{
                                    marginTop: '10px',
                                    marginBottom: '10px'
                                }}
                            >
                                <Field
                                    type="radio"
                                    name="hw_gas"
                                    value="true"
                                />
                                Sí
                            </Radio.Button>
                            <Radio.Button
                                value={'false'}
                                name="hw_gas"
                            >
                                <Field
                                    type="radio"
                                    name="hw_gas"
                                    value="false"
                                />
                                No
                            </Radio.Button>
                        </Space>
                    </Radio.Group>
                </Space>
                <br></br>
                <Space>
                    <p>Tengo termo de agua caliente:</p>

                    <Radio.Group
                        buttonStyle="solid"
                        value={values.hw_boiler}
                        onChange={(e) =>
                            setFieldValue(
                                'hw_boiler',
                                e.target.value
                            )
                        }
                    >
                        <Space size={8}>
                            <Radio.Button
                                value={'true'}
                                name="radio"
                                style={{
                                    marginTop: '10px',
                                    marginBottom: '10px'
                                }}
                            >
                                <Field
                                    type="radio"
                                    name="hw_boiler"
                                    value="true"
                                />
                                Sí
```

```
                        </Radio.Button>
                        <Radio.Button
                            value={'false'}
                            name="hw_boiler"
                        >
                            <Field
                                type="radio"
                                name="hw_boiler"
                                value="false"
                            />
                            No
                        </Radio.Button>
                    </Space>
                </Radio.Group>
            </Space>

            <br></br>

            <h3>¿Tengo otros recursos disponibles?</h3>
            <label>
                <Space>
                    <p>Tengo baterías:</p>
                    <Button
                        type="primary"
                        shape="circle"
                        icon={<PlusOutlined />}
                        onClick={() => {
                            setFieldValue(
                                'n_bat',
                                values.n_bat + 1
                            )
                        }}
                    />
                    <Button
                        type="primary"
                        shape="circle"
                        icon={<MinusOutlined />}
                        onClick={() =>
                            values.n_bat > 0 &&
                            setFieldValue(
                                'n_bat',
                                values.n_bat - 1
                            )
                        }
                    />
                    <p>{values.n_bat}</p>
                </Space>
            </label>
            <br></br>
            <label>
                <Space>
                    <p>Tengo paneles solares:</p>
                    <Button
                        type="primary"
                        shape="circle"
                        icon={<PlusOutlined />}
                        onClick={() => {
                            setFieldValue(
                                'n_pv',
                                values.n_pv + 1
```

```
                                        )
                                    }}
                                />
                                <Button
                                    type="primary"
                                    shape="circle"
                                    icon={<MinusOutlined />}
                                    onClick={() =>
                                        values.n_pv > 0 &&
                                        setFieldValue(
                                            'n_pv',
                                            values.n_pv - 1
                                        )
                                    }
                                />
                                <p>{values.n_pv}</p>
                            </Space>
                        </label>
                        <br></br>
                        <label>
                            <Space>
                                <p>
                                    Tengo puestos de carga para mis
                                    vehículos eléctricos:
                                </p>
                                <Button
                                    type="primary"
                                    shape="circle"
                                    icon={<PlusOutlined />}
                                    onClick={() => {
                                        setFieldValue(
                                            'n_evcp',
                                            values.n_evcp + 1
                                        )
                                    }}
                                />
                                <Button
                                    type="primary"
                                    shape="circle"
                                    icon={<MinusOutlined />}
                                    onClick={() =>
                                        values.n_evcp > 0 &&
                                        setFieldValue(
                                            'n_evcp',
                                            values.n_evcp - 1
                                        )
                                    }
                                />
                                <p>{values.n_evcp}</p>
                            </Space>
                        </label>

                    </form>
                </div>
                <div className="house">
                    <div className="pv">
                        {[...Array(values.n_pv).keys()].map((i) => (
                            <img src={pvIcon} className="pvIcon" />
                        ))}
                    </div>
```

```
<div className="floor">
    <div className="room">
        {' '}
        {[...Array(values.n_ac).keys()].map(
            (i) => (
                <img
                    src={acIcon}
                    className="acIcon"
                />
            )
        )}
    </div>
    <div className="room">
        {[
            ...Array(values.n_heatpumps).keys()
        ].map((i) => (
            <img
                src={heatpumpIcon}
                className="heatpumpIcon"
            />
        ))}
    </div>
</div>
<div className="floor">
    <div className="room">
        {' '}
        {[
            ...Array(values.n_radiators).keys()
        ].map((i) => (
            <img
                src={radiatorIcon}
                className="radiatorIcon"
            />
        ))}
    </div>
</div>

<div className="garage">
    <div className="roomIndG">
        {' '}
        {convertString(values.hw_boiler) && (
            <img
                src={hwBoilerIcon}
                className="hwBoilerIcon"
            />
        )}
    </div>

    <div className="roomIndG">
        {convertString(values.hw_gas) && (
            <img
                src={hwGasIcon}
                className="hwGasIcon"
            />
        )}
    </div>

    <div className="roomIndG">
        {convertString(values.has_gas) && (
```

```
                                        <img
                                            src={gasIcon}
                                            className="gasIcon"
                                        />
                                    )}
                                </div>
                                <div className="roomG">
                                    {[...Array(values.n_evcp).keys()].map(
                                        (i) => (
                                            <img
                                                src={evcpIcon}
                                                className="evcpIcon"
                                            />
                                        )
                                    )}
                                </div>
                                <div className="roomG">
                                    {[...Array(values.n_bat).keys()].map(
                                        (i) => (
                                            <img
                                                src={batIcon}
                                                className="batIcon"
                                            />
                                        )
                                    )}
                                </div>
                            </div>
                        </div>
                    </div>
                )
            }}
        </Formik>
    </div>
  )
}
```

*Código 60: UserHouseForm completo*

## *ANEXO III.II: CÓDIGO INDEX.CSS*

```
.formAndHouse {
    display: flex;
    flex-direction: row;
    margin: 5px;
}
.house {
    display: flex;
    flex-direction: column;
    flex: 7;
    margin: 5px;
    max-width: 650px;
    max-height: 650px;
}
.pv {
    display: flex;
    flex-direction: row;
    flex: 1;
```

```
    align-items: flex-start;
    flex-wrap: wrap-reverse;
    background-color: rgb(145, 225, 241);
}
.insideHouse {
    display: flex;
    flex-direction: column;
    flex: 4;
    align-items: flex-end;
    flex-wrap: wrap-reverse;
    background-color: white;
    border: 3px;
    border-color: black;
    border-style: solid;
}
.garage {
    display: flex;
    flex-direction: row;
    flex: 1;
    align-items: flex-end;
    background-color: rgb(200, 200, 200);
    border: 3px;
    border-color: black;
    border-style: solid;
}
.roomIndG {
    display: flex;
    flex-direction: row;
    flex: 1;
    align-items: flex-end;
    flex-wrap: wrap-reverse;
    background-color: rgb(200, 200, 200);
}
.roomG {
    display: flex;
    flex-direction: row;
    flex: 4;
    align-items: flex-start;
    flex-wrap: wrap-reverse;
    background-color: rgb(200, 200, 200);
}
.roomInd {
    display: flex;
    flex-direction: row;
    flex: 1;
    align-items: flex-end;
    flex-wrap: wrap-reverse;
    background-color: white;
}
.room {
    display: flex;
    flex-direction: row;
    flex: 4;
    align-items: flex-start;
    flex-wrap: wrap-reverse;

    background-color: white;
}
.floor {
    flex: 1;
```

```
    display: flex;
    flex-direction: row;

    align-items: flex-end;
    border: 3px;
    border-color: black;
    border-style: solid;
}

.form {
    flex: 3;
    display: flex;
    flex-direction: row;
}
.radiatorIcon {
    height: 60px;
    width: 70px;
}
.acIcon {
    height: 60px;
    width: 60px;
}

.batIcon {
    height: 70px;
    width: 60px;
}

.hwBoilerIcon {
    height: 70px;
    width: 60px;
}
.pvIcon {
    height: 60px;
    width: 60px;
}
.evcpIcon {
    height: 60px;
    width: 70px;
}
.heatpumpIcon {
    height: 60px;
    width: 60px;
}
.hwGasIcon {
    height: 60px;
    width: 60px;
}
.gasIcon {
    height: 60px;
    width: 60px;
}
```

*Código 61: index.css*