Grado en Ingeniería Informática

Trabajo de Final de Grado

Aplicación web para la consulta del consumo de agua

Autor:
Rosa María de Juan Oliva

Supervisor:
Jose Luís Martínez Pérez
Tutor académico:
María de las Mercedes
Fernández Redondo

Fecha de lectura: 16 de septiembre de 2020
Curso académico 2019/2020

Resumen

Este documento contiene la memoria del trabajo de final de grado en el que se expone el desarrollo de una aplicación web para la consulta del consumo de agua de los clientes abonados y la gestión de alarmas en sus contadores con el fin de detectar problemas o fugas. Esta aplicación va dirigida tanto a clientes particulares como a empresas, aunque esta memoria se centra en el desarrollo de la parte del cliente particular.

El *frontend* de la aplicación se ha implementado utilizando el *framework* Angular 8 y el lenguaje de programación *Typescript*. Por el contrario, para el *backend* se ha empleado el *framework Spring* y el lenguaje de programación *Java*.

Palabras clave

Contadores, consumo de agua, aplicación web, Angular, Spring framework.

Keywords

Water meters, water consumption, web application, Angular, Spring framework.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

El proyecto documentado en esta memoria se desarrolló en la empresa IoTsens. Esta empresa se dedica a la venta de soluciones software e IoT a medida para la gestión de datos, ofreciendo también soporte y mantenimiento de las mismas [1]. La empresa IoTsens forma parte de Grupo Gimeno, un conjunto de empresas de varios sectores que tienen como objetivo la innovación, el desarrollo sostenible y la mejora de la calidad de vida de los ciudadanos e industrias [2].

A raíz de una idea surgida durante el Hackathon 2019 de la Universitat Jaume I, la cual consistía en una aplicación que gestionara la recogida de residuos de una ciudad de forma inteligente, se decidió proponer su desarrollo a la empresa IoTsens. Esto fue posible gracias a las profesoras María Asunción Castaño Álvarez y Lledó Museros Cabedo, quienes facilitaron el contacto con Ignacio Llopis, director gerente de IoTSens. Finalmente, se decidió el desarrollo de una aplicación con objetivos similares, pero enfocada a la consulta del consumo de agua. El progreso de este proyecto, desde el inicio de su propuesta hasta parte de su implementación, es el que se desarrolla en esta memoria.

La realización de este proyecto viene dada por la necesidad de proporcionar a clientes particulares una plataforma donde poder visualizar información sobre su consumo de agua en un formato fácil de comprender, con el fin de hacer un uso responsable de la misma. También se quería poder detectar y controlar las posibles fugas o problemas que tuviera un contador lo más rápido posible para poder tomar medidas en consecuencia.

Otros de los motivos por los que se decidió desarrollar este proyecto fue el hecho de disponer de una aplicación parecida llamada *Smart Water*[1], la cual ya estaba implementada y enfocada a los profesionales del sector de la gestión del ciclo del agua, pero no a los clientes particulares o a grandes consumidores.

[1]La información de esta aplicación se encuentra en el enlace: `www.iotsens.com/solucion/smart-water/`

1.2. Objetivos del proyecto

El objetivo principal de este proyecto consiste en la creación de una aplicación que permita al usuario llevar un control sobre sus contadores y su consumo de agua de manera sencilla. Este objetivo, se puede desglosar en los siguientes:

- Permitir que el usuario pueda consultar su consumo de agua.
- Visualizar de manera gráfica y fácilmente comprensible la lectura y consumo de un contador.
- Permitir que el usuario gestione las alarmas de sus contadores y que éstas le informen en caso de algún problema.
- Posibilitar las predicciones de consumo de agua.

Los principales beneficios que conllevaría la existencia de una aplicación de este tipo serían el consumo más responsable de agua, así como una mejor gestión y conocimiento sobre ello por parte de clientes particulares.

1.3. Descripción del proyecto

A continuación, se explica con más detalle los requisitos especificados para la aplicación web desarrollada durante la estancia en prácticas, llamada *Water Clients*.

En primer lugar, el sistema debe permitir la autentificación de los clientes para poder acceder a la aplicación, así como desconectarse de la misma.

En segundo lugar, el sistema debe mostrar los datos de los contadores que tenga el cliente, así como su consumo. Por una parte, esto incluye la visualización de la póliza, código y dirección de un contador. A su vez, el sistema debe mostrar las gráficas de consumo de agua y lecturas en un período de tiempo e intervalos especificados por el cliente. También debe permitir la visualización del consumo diario promedio del contador, el consumo hecho a lo largo del día y la diferencia con respecto al día anterior.

El sistema también debe permitir que el cliente pueda añadir, modificar o desactivar alarmas para uno o varios contadores. Esto incluye la posibilidad de que el cliente pueda especificar el tipo de alarma y un período de tiempo determinado en el que esté activa.

Además, el cliente debe tener la posibilidad de añadir datos adicionales sobre el contexto en el que se encuentra su contador, como es la cantidad de baños que tiene la vivienda, cantidad de personas, si se dispone de lavavajillas o lavadora, etc.

Por otro lado, el sistema debe permitir al administrador gestionar las cuentas de sus clientes, así como ver la información de sus contadores y alarmas.

Finalmente, el cliente debe poder visualizar en un mapa interactivo la ubicación de sus contadores.

Adicionalmente, a modo de mejora, se debe tener en cuenta que el sistema pueda mostrar estadísticas sobre el consumo de agua de un contador a partir de los datos adicionales proporcionados por el cliente. Aunque esto queda fuera del alcance funcional del proyecto.

Cabe mencionar que la parte del sistema relacionada con la administración de usuarios no se incluye en esta memoria, puesto que fue implementada por otro miembro del equipo de desarrollo. Tampoco se incluye la gestión de alarmas, a excepción de las consultas, por la misma razón mencionada anteriormente. En la sección 2.4 se explica en detalle el porqué de este hecho.

1.4. Estructura de la memoria

El contenido de esta memoria está organizado en varios capítulos, comentados a continuación.

El capítulo 1 explica la motivación, los objetivos y de qué trata el proyecto que se ha realizado durante la estancia en prácticas. También se describe brevemente la empresa donde se ha llevado a cabo.

En el capítulo 2 se muestra la planificación que se ha seguido para llevar a cabo el desarrollo del proyecto, así como la estimación de recursos, las tecnologías utilizadas y la metodología adoptada.

El capítulo 3 lleva a cabo la definición de los requisitos del sistema y explica la arquitectura del mismo. A su vez analiza el diseño de la interfaz gráfica de la aplicación.

El capítulo 4 se centra en la implementación de la aplicación llevada a cabo y los resultados obtenidos.

En el capítulo 5 se muestran las pruebas de aceptación diseñadas para validar el sistema y se comentan las pruebas unitarias y de integración realizadas.

El capítulo 6 trata de las reflexiones sobre el proyecto llevado a cabo y la estancia en prácticas.

Capítulo 2

Planificación del proyecto

2.1. Metodología

Debido a la naturaleza de este proyecto, sujeto a la posibilidad de cambios y mejoras en algunos de sus aspectos menos definidos, se optó por seguir la metodología ágil *Kanban*, con algunos matices propios de la metodología *Scrum*. Para poder analizar mejor en qué consistió esta agrupación de ambas metodologías, es necesario definirlas antes.

Por un lado, la metodología ágil *Kanban* tiene como objetivo la gestión eficiente del tiempo y el trabajo, ayudando a limitar la acumulación de las tareas pendientes y generando un flujo de trabajo constante y equilibrado [3]. Para ayudar a visualizar ello, se dispone del tablero *Kanban*, el cual se divide en diversos estados. Las tareas van pasando por cada uno de ellos hasta quedar finalizadas. La cantidad de estados puede variar dependiendo del tipo de proyecto y su complejidad.

Por otra parte, la metodología Scrum se caracteriza por dividir la realización de un proyecto en pequeños ciclos temporales de duración fija llamados iteraciones. Al final de cada una de ellas se realiza una reunión para dar un feedback del resultado. Las tareas a completar se suelen medir según su prioridad y complejidad [11].

Para la realización de este proyecto, se estableció una pila del producto para poder determinar la complejidad de cada funcionalidad a implementar y su prioridad. Además, se planificaron reuniones semanales para el seguimiento del proyecto. A su vez, se dispuso de un tablero *Kanban* donde se podía visualizar el estado de cada tarea en todo momento.

2.2. Tecnología

A la hora de hablar de las tecnologías que se utilizaron en la elaboración del código, de debe distinguir entre dos partes bien diferenciadas: *frontend* y *backend*. La primera hace referencia a la parte web de la aplicación, la capa de presentación, aquella que interactúa con los usuarios

[21]. Por contra, la capa de acceso a datos, el *backend*, es la encargada de interactuar con el servidor, la base de datos u otras aplicaciones con el objetivo de procesar la información proveniente tanto de alguno de los tres anteriores para enviarla al *frontend* y viceversa [24].

En primer lugar, con respecto a la implementación del *frontend*, se utilizó el *framework*[1] *Angular*, concretamente la versión 8, una de las más recientes. Este *framework* incluye el uso del lenguaje de programación *Typescript* y los lenguajes de etiquetas *HTML* y *CSS*, con los cuales se obtuvo una interfaz gráfica cómoda y agradable, a la vez que un código bien estructurado. Además, en relación a la interfaz de usuario, para lograr que ésta fuera *responsive*[2], se utilizó otro *framework* llamado *Bootstrap*.

Con respecto al *backend*, se usó *Spring*, el cual es un *framework* de código abierto para el desarrollo de aplicaciones con *Java* [20], cuyo lenguaje de programación se utilizó conjuntamente con este *framework*. El uso de *Spring* simplificó las configuraciones iniciales, el acceso a la base de datos y las consultas a los microservicios de IoTsens.

También, cabe destacar la parte de las posibles mejoras que se plantearon para la aplicación. Estas mejoras están relacionadas con el análisis predictivo a modo de introducción al *machine learning*, que es la disciplina perteneciente al ámbito de la IA (Inteligencia Artificial) focalizada en el aprendizaje automático [17]. Para poder llevar a cabo su implementación en un futuro, se decidió estudiar el uso de *Python*, así como de *R*. Esto se debió a que ambos lenguajes de programación tienen librerías muy populares para *machine learning*, con un gran soporte técnico y documentación.

En relación con los recursos de *hardware* utilizados durante el desarrollo del proyecto, sólo se requirió de un ordenador de sobremesa para cada miembro del equipo. Aunque durante el estado de alarma debido al COVID-19, se tuvieron que usar los ordenadores personales.

2.3. Herramientas

Para este proyecto se utilizaron una serie de herramientas con la finalidad de ayudar en la correcta gestión, documentación y desarrollo del mismo, así como para mantener una buena comunicación dentro del equipo de trabajo.

2.3.1. Herramientas de gestión y documentación

Inicialmente, se usó la herramienta *Jira Software*, una plataforma web que ayuda en la planificación, desarrollo, gestión y supervisión de proyectos software ágiles [4]. Dispone de un tablero *Kanban* para visualizar el flujo de trabajo y hacer un seguimiento de las tareas, lo que hizo tener una visión general del avance del proyecto en todo momento. La figura 2.1 muestra el tablero *Kanban* utilizado durante la implementación del proyecto.

[1] Es un marco de trabajo cuya estructura o esquema establecido se utiliza para desarrollar *software* [13].

[2] Capacidad para redimensionar el contenido de la aplicación de manera coherente para adaptarse a diferentes dispositivos electrónicos, cuyos tamaños son variados [14].

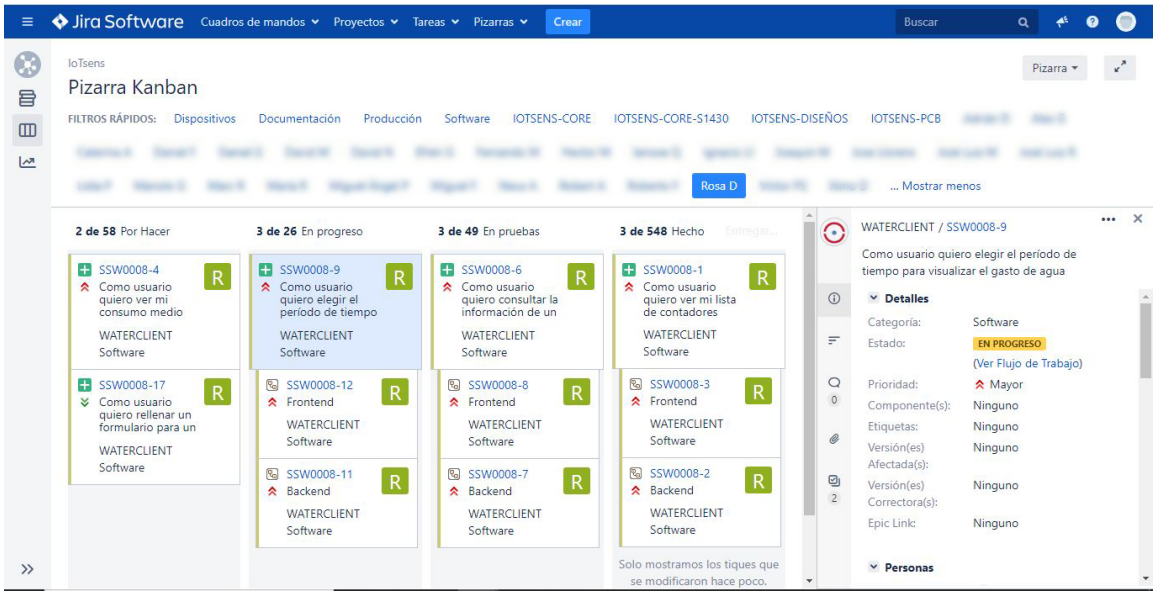

Figura 2.1: Tablero *Kanban*

Puesto que la herramienta *Jira Software* está especializada en el desarrollo de proyectos informáticos, ésta permite la creación de tareas de diferente tipo relacionadas con la implementación de funcionalidades de una aplicación. El tipo de tarea más común es la historia de usuario, la cual hace referencia a los requisitos y funcionalidades de un sistema informático expresadas desde el punto de vista del usuario final que quiere utilizar dicho sistema. También están las épicas, que son historias de usuario de gran tamaño que se descomponen en varias de tamaño menor para poder ser gestionadas individualmente [18]. Por otro lado, se pueden crear tareas que indiquen errores.

Asimismo, es posible detallar las diferentes tareas creadas añadiéndoles descripción, prioridad, puntos de historia[3] y subtareas. Cada una puede ser asignada al miembro del equipo responsable de realizarla y/o al responsable de su supervisión.

Por otra parte, cabe destacar que *Jira* permite también la creación y gestión de esprints, que son propios de la metodología ágil *Scrum*. A pesar de que en este proyecto no se realizaron esprints, es importante tener en cuenta esta posibilidad, ya que es otra forma más de facilitar la combinación de las metodologías ágiles *Scrum* y *Kanban*, de las cuales se utilizaron elementos de ambas para el desarrollo de este proyecto, como se explica en la sección 2.1.

Otra herramienta utilizada fue *Confluence*, con el fin de consultar la documentación de los diferentes microservicios creados por la empresa y utilizados para la obtención de diferentes tipos de mediciones de dispositivos y alarmas. *Confluence* es una wiki[4] utilizada en entornos corporativos para la colaboración en equipo [26]. Esta herramienta fue imprescindible, puesto que se usaron varias de estos microservicios ya documentados en esta plataforma para la implementación de la aplicación *Water Clients*. Cabe añadir que *Confluence* también permite redactar informes sobre la evolución y reuniones que se han realizado a lo largo del desarrollo

[3]Medida subjetiva para estimar la complejidad de una tarea [23].
[4]Plataforma web que se utiliza para la colaboración, documentación y recopilación de información [26].

de un proyecto, para poder tener un seguimiento bastante preciso del mismo.

Finalmente, cabe destacar que, debido a situación derivada a causa del COVID-19, se pasó a trabajar en remoto. Este evento se detalla mejor en la sección 2.4. Es por ello por lo que la herramienta *Microsoft Teams* pasó a tener una gran relevancia a lo largo del desarrollo del proyecto, ya que mantuvo a todos los miembros del equipo en comunicación continua. A través de esta plataforma, fue posible organizar reuniones e informar del estado del proyecto en todo momento tanto al mánager como al resto del equipo.

2.3.2. Herramientas de desarrollo

Puesto que la aplicación web a desarrollar de este proyecto requería del almacenamiento de diferentes tipos de datos, se utilizó para ello *MySQL*, un sistema de gestión de bases de datos relacional de código abierto [9]. Además, con la intención de crear de manera gráfica, fácil y rápida tanto la base de datos necesaria para la plataforma web como los datos de prueba iniciales, se usó la herramienta *phpMyAdmin*.

Por otro lado, una parte crítica en el desarrollo de un proyecto informático es el momento de unir las diferentes partes del código, puesto que a lo largo de la implementación de una aplicación el equipo trabaja paralelamente en las diferentes funcionalidades. La herramienta *GitLab*, una plataforma web basada en *Git*[5] para el control de versiones [15], fue fundamental en esta etapa del proyecto.

Por último, la IDE (Entorno de Desarrollo Integrado, o en inglés *Integrated Development Environment*) utilizada para la implementación fue *IntelliJ IDEA*, el cual facilitó el desarrollo del código de la aplicación.

2.4. Planificación

Primeramente, para la planificación de este proyecto, se creó una pila de producto con el fin de tener una idea más clara sobre la complejidad de cada funcionalidad y establecer un orden de prioridad a la hora de su implementación. La tabla 2.1 muestra la pila de producto final, formada por las HU (Historias de Usuario) establecidas para representar las funcionalidades y requisitos de la aplicación. También se diseñó la base de datos necesaria para el almacenamiento de la información que usaría la aplicación y se planificó tanto la duración total del proyecto como la estimación de las tareas a realizar durante el período de prácticas.

Una vez realizada esta planificación inicial, antes de empezar con la implementación de la aplicación, se estableció un período de formación. Este período tendría una duración entre una y dos semanas para el estudio y aprendizaje de las tecnologías que se iban a usar en este proyecto, explicadas con más detalle en la sección 2.2. Es por ello por lo que se consultaron varios tutoriales sobre el *framework Angular 8*, el lenguaje de programación *Typescript* y el uso de escritura lambda en *Java*. A su vez, se consultaron proyectos realizados anteriormente, lo que

[5]Herramienta que sirve para el control de versiones del código [22].

fue de gran ayuda para la estimación del tiempo y los costes, así como para entender y analizar el estilo de programación utilizado por los miembros del Departamento de *Software*, con el fin de generar un código relativamente homogéneo y fácil de comprender.

Cabe añadir que, durante esta etapa de formación, se estuvo estudiando las posibles mejoras previstas para la aplicación y cómo sería su implementación. Puesto que dichas mejoras están relacionadas con las predicciones de consumo de agua, se optó por la opción de usarlas como pretexto para la introducción al mundo del *machine learning* por parte del equipo de desarrollo *software*. Además, estas mejoras a largo plazo son independientes del producto base diseñado, por lo que no se incluyen en el coste y tiempo estimado del proyecto, documentado en la sección 2.5.

En última instancia, se procedió a la implementación del proyecto empezando por su configuración inicial, prosiguiendo con la creación de la base de datos y continuando con las funcionalidades de la aplicación, de mayor a menor prioridad.

Código	**Historia de usuario**	**Puntos de historia**	**Prioridad**
HU01	Como usuario quiero ver mi lista de contadores para saber sus datos.	3	1
HU02	Como usuario quiero ver todas mis alarmas y los contadores en las que están activas para poder saber si necesito añadir o quitar alguna.	2	9
HU03	Como usuario quiero loguearme en la plataforma para poder consultar la información de mis contadores de agua y mi consumo.	3	2
HU04	Como usuario quiero consultar la información de un contador para saber el consumo y los datos del contador.	8	3
HU05	Como usuario quiero elegir el período de tiempo para visualizar de manera gráfica el gasto de agua que indica un contador.	5	4
HU06	Como usuario quiero modificar mi perfil para cambiar mis datos personales.	8	5
HU07	Como usuario quiero visualizar las alarmas programadas de un contador en un período de tiempo para saber cuáles tengo activas y cuáles no.	3	6

Código	Historia de usuario	Puntos de historia	Prioridad
HU08	Como usuario quiero poder programar una alarma para uno o varios contadores para que me avise cuando suceda el evento que haya especificado.	8	7
HU09	Como usuario quiero modificar una alarma para poder cambiar sus especificaciones.	3	10
HU10	Como usuario quiero poder desactivar una alarma para uno o varios contadores de agua para que ya no me avise del evento que había especificado.	2	8
HU11	Como usuario quiero rellenar un formulario con información adicional de un contador para poder especificar mejor las condiciones que se deberían de tener en cuenta a la hora de mostrar datos estadísticos y predictivos de mi consumo de agua.	5	11
HU12	Como usuario quiero modificar el formulario con información adicional de un contador para indicar que mi situación actual ha cambiado y saber si influye en mi consumo de agua.	5	12
HU13	Como usuario quiero consultar en un mapa la posición de mis contadores para así poder visualizar su ubicación geográfica.	13	13
HU14	Como administrador quiero ver los contadores que gestiono para llevar un control de ellos.	3	15
HU15	Como administrador quiero poder crear perfiles de usuario y habilitarlos o deshabilitarlos para poder gestionar dichos perfiles.	13	14

Tabla 2.1: Pila del producto

2.5. Estimación de recursos y costes del proyecto

Para la estimación de los recursos y costes de este proyecto se utilizó la estimación LDC (líneas de código) y el modelo *COCOMO* (Modelo Constructivo de Costos, o en inglés

COnstructive COst MOdel). Este modelo se basa en estimaciones matemáticas que miden el tamaño del producto final, principalmente a través del conteo de líneas de código [6]. Aunque, para este proyecto se consideró utilizar el modelo *COCOMO* Básico, su versión más sencilla.

Por otra parte, se tuvo en cuenta tanto el coste proporcional del *hardware* como del *software* utilizado durante el período de prácticas, que fue de tres meses. Estos costes están reflejados en la tabla 2.2 y la tabla 2.3. Se debe mencionar que estos costes son a grandes rasgos, es decir, no se tienen en cuenta el mantenimiento de un servidor, dominio, etc.

Activos *hardware*	**Coste total (€)**	**Tiempo de vida útil (años)**	**Coste proporcional (€)**
Ordenador de sobremesa	1133,20	5	55,66
Ratón	9,90	1	2,48
Teclado	29,99	2	3,75
Monitor extra	374,90	5	18,74
		Total:	**80,63**

Tabla 2.2: Costes de los activos *hardware*

Activos *software*	**Coste anual (€)**	**Coste proporcional (€)**
Jira Software & Confluence	168	51
Office 365 Empresa Premium	126	31,15
	Total:	**82,15**

Tabla 2.3: Costes de los activos *software*

2.5.1. Estimación LDC

La estimación LDC calcula las líneas de código que aproximadamente tendrá una aplicación en base a la cantidad de líneas de código estimadas [8]. Para ello, se usa la fórmula 2.1, donde la cantidad de líneas estimadas se obtienen a partir de la cantidad optimista (O), la más probable (MP) y la pesimista (P).

$$Estimada = \frac{O + (4 \times MP) + P}{6} \tag{2.1}$$

Los valores pesimistas, optimistas y probables usados para el cálculo del valor esperado de

cada módulo del sistema, se obtuvieron considerando proyectos de envergadura similar realizados anteriormente en la empresa. A su vez, se tuvo en cuenta los lenguajes de programación usados en proyectos pasados, puesto que la cantidad de líneas puede fluctuar dependiendo del lenguaje de programación utilizado. Para este proyecto en concreto, se acordó la utilización del lenguaje de programación *Typescript* para el *frontend* y de Java para el *backend*. Si bien es cierto que para el *frontend* se usó también el lenguaje de etiquetas *HTML*, éste no está contemplado en la estimación por no ser un lenguaje de programación.

La tabla 2.4 muestra la cantidad de líneas que se estimaron para cada módulo aplicando la fórmula 2.1 y el total obtenido al sumarlas.

Módulo	**Optimista**	**Pesimista**	**Más probable**	**Estimada**
Visualización de los datos de un contador	250	300	270	271,66
Visualización de la gestión de datos adicionales	100	150	110	115
Visualización de la gestión de alarmas	400	500	450	450
Visualización del mapa interactivo	200	300	270	263,33
Visualización de la gestión de perfiles	600	700	650	650
Visualización de la administración de usuarios	650	700	660	665
Tratamiento de excepciones	250	350	280	286,66
Acceso a la base de datos	1500	2000	1600	1650
Acceso al microservicios	180	230	200	201,66
Gestión del usuario	1000	1500	1200	1066,66
			Total:	**5619,97 LDC**

Tabla 2.4: Cantidad de LDC estimadas para cada módulo

2.5.2. Estimación *COCOMO* Básico

Para estimar los costes temporales, económicos y de recursos, se empleó el modelo *COCOMO* Básico, como se menciona al inicio de la sección 2.5.

Con el fin de poder realizar los cálculos necesarios y llegar a la estimación total de recursos, se tuvo en cuenta el resultado obtenido al sumar las líneas estimadas (véase 2.4). Después, se pasaron a KLDC (miles de líneas de código), tal y como muestra la operación 2.2, vista a continuación:

$$KLDC = \frac{5619,97}{1000} = 5,620 \tag{2.2}$$

Como se puede observar en la tabla 2.5, el proyecto es de tipo orgánico, puesto que no supera las 50 KLDC. Esto quiere decir que las variables de la ecuación 2.3 y la ecuación 2.4 se sustituyen por los valores indicados conforme al tipo de proyecto. Se debe tener en cuenta que estos valores y tipos de proyectos son estándares utilizados en el modelo *COCOMO* Básico [8].

Tipo de proyecto	**a**	**b**	**c**	**d**
Orgánico	2,4	1,05	2,5	0,38
Semiacoplado	3	1,12	2,5	0,35
Empotrado	3,6	1,2	2,5	0,32

Tabla 2.5: Valores según el tipo de proyecto del modelo *COCOMO* Básico

En primer lugar, se estimó el esfuerzo persona/mes siguiendo la fórmula 2.3, mostrada a continuación:

$$E = a \times (KLDC)^b = 2,4 \times 6,126 = 14,7\ personas/mes \tag{2.3}$$

Después de haber obtenido el resultado de la ecuación anterior, se aplicó la fórmula 2.4, mostrada a continuación, para averiguar la cantidad de meses necesarios para desarrollar el proyecto.

$$D = c \times (E)^d = 2,5 \times 2,78 = 6,95\ meses \tag{2.4}$$

Una vez obtenidos ambos resultados, se calculó el coste total de personas necesarias (véase ecuación 2.5).

$$costeH = E/D = 14,7/6,95 = 2,12\ personas \tag{2.5}$$

Sin embargo, teniendo en cuenta que el período de estancia de prácticas es de aproximadamente tres meses, si se hubiera querido terminar el proyecto en ese plazo de tiempo, el total de personas necesarias serían 5, tal y como muestra la ecuación 2.6.

$$costeH' = E/D = 14,7/3 = 4,9\ personas \tag{2.6}$$

El coste total del proyecto se calculó en base a los salarios promedios de los trabajadores (véase tabla 2.6), el impuesto de contratación (25 %), el período de tiempo (3 meses) y la suma de los costes indirectos (20 %). Estos últimos engloban los costes comentados y los costes calculados de *software* y *hardware* utilizados durante el desarrollo del proyecto, mencionados al inicio de la sección 2.5.

Trabajador	**Salario anual (€)**	**Salario mensual (€)**
Analista programador	27500	2292
Programador junior	19000	1583
Programador sénior	31500	2625
Tester	26000	2167
Project manager	45000	3750

Tabla 2.6: Salarios promedios de los trabajadores

Como se puede observar en la ecuación 2.7, primeramente se obtuvo el coste de los salarios de tres meses multiplicado por el impuesto de contratación. Acto seguido, se calculó el total de los costes indirectos sumando los salarios de los trabajadores y multiplicándolos por el 20 % correspondiente (véase ecuación 2.8). La suma de los resultados anteriores conforman el presupuesto total del proyecto si se realizara en un período de tres meses disponiendo de cinco trabajadores (véase ecuación 2.9).

$$C = (2292 + 1583 + 2625 + 2167 + 3750) \times 3 \times 1,25 = 46563,75\ € \tag{2.7}$$

$$I = (2292 + 1583 + 2625 + 2167 + 3750) \times 1,20 = 14900,4\ € \tag{2.8}$$

$$PresupuestoTotal = C + I = 46563,75 + 14900,4 = 61464,15\ € \tag{2.9}$$

Finalmente, se debe mencionar que la estimación realizada se aleja de un caso real, puesto que normalmente los equipos están formados por un número menor de personas y el tiempo para realizar el proyecto es mayor. Esto se debe a que el aumento de trabajadores y el decremento

del tiempo de realización de un proyecto no son lineales y puede llegar a ser improductivo tener a un gran número de trabajadores para un proyecto de esta envergadura.

Para el desarrollo de este proyecto, realmente se planificó un mayor período de tiempo y un menor número de trabajadores. En un primer momento, se contó con la supervisión de un *project manager* y el alumno de prácticas, seguido de la incorporación tardía de un programador junior que debido a una lesión estaba de baja. Asimismo, se planificó que el mánager del proyecto pasara a tomar también el rol de programador sénior una vez el alumno hubiera acabado su estancia en prácticas. Las soluciones de las operaciones 2.4 y 2.5, muestran resultados más realistas que se ajustan más a la planificación real del proyecto.

2.5.3. Estimación del coste por el trabajo realizado

Una vez hecha la estimación total del proyecto, se puede concretar la estimación del coste que le supondría a la empresa contratarme por el trabajo realizado durante el período de prácticas.

Si se toma como referencia el salario del programador junior mostrado en la tabla 2.6, cobraría aproximadamente 9,9€la hora. A esta cantidad se le tendría que multiplicar las horas totales (300 horas) y los costes de *hardware* y *software* (véase tablas 2.2 y 2.3). La ecuación 2.10 muestra el cálculo total:

$$Costetrabajador = 9,9 \times 300 + 80,63 + 82,15 = 3132,78 \text{ €} \tag{2.10}$$

La estimación total que supone contratarme durante la estancia de prácticas es de aproximadamente 3132,78 €.

2.6. Gestión de riesgos

Durante la elaboración de un proyecto informático, se pueden producir una serie de riesgos, los cuales implican condiciones inciertas que pueden tener un impacto tanto positivo como negativo sobre el propio proyecto.

Los riesgos pueden ser tanto generales como específicos. Las tablas 2.7 y 2.8 muestran los riesgos más comunes de ambos tipos.

Para el proyecto, se tuvo en cuenta todos estos riesgos y, de hecho, se produjo uno de ellos. Este riesgo fue el R00, la baja de un miembro del equipo de desarrollo por una lesión grave.

Sin embargo, durante el desarrollo el proyecto no se tuvo en cuenta el riesgo de sufrir una pandemia. Esto se debió a una situación inesperada e inusual a nivel mundial, por lo que realmente ni siquiera se tuvo en cuenta en ningún momento como un posible riesgo. A pesar de ello, se tomaron medidas preventivas rápidamente y, a pesar de que se retrasó un poco el desarrollo del proyecto, esto no provocó un gran impacto.

ID	Breve descripción del riesgo
R00	Bajas en el equipo de desarrollo.
R01	Listado de requisitos incompleto o incorrecto.
R02	Estimación de la duración incorrecta.
R03	Presupuesto insuficiente.
R04	Sobrecarga de trabajo.
R05	Falta de experiencia en tareas de desarrollo.
R06	Diseño erróneo.
R07	Dificultades técnicas.
R08	Elección inadecuada de las tecnologías a usar.

Tabla 2.7: Tabla de riesgos genéricos más comunes

ID	Breve descripción del riesgo
R09	Caída del servidor.
R10	Base de datos llena.
R12	Caída del microservicio

Tabla 2.8: Tabla de riesgos específicos más comunes

2.7. Seguimiento del proyecto

La planificación del proyecto establecida se llevó a cabo sin ninguna incidencia durante la dos primeras etapas. Sin embargo, una vez llegada la fase de la implementación de la aplicación web, se decidió realizar algunos cambios con respecto el reparto de las tareas a realizar y sus prioridades. Esto se debió a varios factores, los cuales algunos se tuvieron en cuenta y otros no (véase sección 2.6). A continuación, se explica el seguimiento del proyecto, especialmente durante esta última fase, los problemas que surgieron y los cambios realizados. Se debe añadir que, a pesar de que el proyecto no se dividió en esprints, sí que se pueden diferenciar 4 etapas a lo largo de su desarrollo. Además, se hicieron diversas reuniones de seguimiento cada dos

semanas, y a través de la herramienta Jira se tuvo una clara visión del flujo de trabajo.

2.7.1. Inicio del proyecto y fase de documentación

Como se ha comentado al inicio de la sección 2.7, durante las dos primeras semanas se dedicó la mayor parte del tiempo a la planificación del proyecto, así como al aprendizaje de las tecnologías a usar. Esto incluyó la realización de varios tutoriales y la visualización de proyectos similares anteriormente creados.

A su vez, durante este período de tiempo, se sucedieron una serie de reuniones para ayudar a definir las funcionalidades de la aplicación, el diseño de la base de datos y el reparto de tareas. Todo ello conformó la pila de producto final que muestra la tabla 2.1 de la sección 2.4.

Por otro lado, se tuvo en cuenta la agregación de un traductor para tener la posibilidad de elegir otros idiomas en un futuro. También se decidió legar la implementación de las funcionalidades más complejas, como la administración de los usuarios y las vistas de la aplicación relacionadas, a otro miembro del equipo. Esto se debió a su experiencia y a su conocimiento de la estructura interna sobre la gestión de datos en la empresa. Asimismo, esta decisión se aplicó a aquellas funcionalidades relacionadas con la adición y desactivación de las alarmas de los contadores, puesto que implicaba realizar modificaciones sobre diferentes sensores. Todo se desarrolló con normalidad según lo previsto, no hubo incidencias de ningún tipo ni retrasos, ya que se tuvo en cuenta durante la planificación la baja de un compañero y su incorporación en el proyecto más adelante.

2.7.2. Inicio de la implementación

Durante la fase inicial de la implementación se procedió a la creación de la base de datos inicial del proyecto, añadiéndole algunos datos de prueba. Por otra parte, se hicieron las configuraciones iniciales de la aplicación incluyendo el traductor, la implementación parcial la funcionalidad de listar los contadores y se generó la vista del *Login*.

Lamentablemente, a mitad de la segunda quincena, la inminente expansión global de la pandemia del COVID-19 obligó a que las prácticas pasaran a realizarse de manera telemática. Esto provocó un retraso en la planificación inicial, puesto que hubo ciertas dificultades a la hora de configurar los ordenadores personales y que éstos pudieran acceder remotamente la red de la empresa. La fluidez de comunicación dentro del equipo y la rapidez de reacción frente a dudas y/o problemas también descendió, por lo que hizo que todo el proceso de desarrollo e implementación de la aplicación se ralentizara.

2.7.3. Desarrollo de la aplicación

Durante la siguiente quincena, antes de proseguir con el resto de las funcionalidades y acabar el listado de contadores, se estuvo considerando qué librería gratuita sería la más cómoda de

utilizar para mostrar de manera gráfica el consumo y la lectura de un contador, con la finalidad de que el usuario pudiera consultar estos datos y los entendiera fácilmente.

Una vez tomada la decisión, se prosiguió con la implementación del componente relacionado con los datos del contador y el listado de contadores del usuario, que también mostraba una pequeña gráfica para cada uno.

En este período, también hubo varios cambios relacionados con qué datos mostrar y de qué manera, como fue el consumo promedio, la diferencia, etc. Además, se debatió sobre cómo gestionar las alarmas de los contadores, por lo que hubo cierto retraso a la hora de terminar de implementar las diversas funcionalidades de la aplicación.

El resto del tiempo sucedió sin más incidentes. A excepción de las funcionalidades que previamente se asignaron al otro miembro del equipo (véase subsección 2.7.1), se acabó de implementar todas las historias de usuario mostradas en la pila de producto final, incluyendo la visualización de un mapa con la ubicación de los contadores. A su vez, se trabajó para que la aplicación fuera completamente *responsive*.

2.7.4. Etapa final

Puesto que tan sólo quedaba una semana para terminar la estancia de prácticas, se decidió usar ese tiempo para llevar a cabo la implementación de un mapa en el que poder visualizar la ubicación de los contadores de un usuario.

Capítulo 3

Análisis y diseño del sistema

3.1. Análisis del sistema

Para poder llevar a cabo un proyecto informático es necesario realizar un análisis del sistema, el cual se suele realizar mediante la definición de los requisitos, los casos de uso y/o las historias de usuario. Para llevar a cabo este proceso, es importante tener en cuenta con qué enfoque se intentan representar las características que puede tener el sistema [25]. También depende de la metodología usada y el equipo de trabajo, puesto que en un contexto ágil, la visión de qué puede llegar a hacer el usuario a través de las HU fomenta la colaboración grupal [25]. Por el contrario, los requisitos y los casos de uso se ciñen más a las especificaciones del sistema y se suelen realizar para proyectos que utilizan metodologías predictivas [25].

Con el fin de realizar un correcto análisis del sistema para este proyecto, se optó por la definición de tanto las historias de usuario, mostradas en la pila de producto (véase sección 2.4), como por la definición de los casos de uso. Esto se hizo para definir mejor las características de los usuarios dependiendo de su rol, así como para entender mejor sus interacciones con el sistema.

3.1.1. Casos de uso

La figura 3.1 muestra el diagrama de casos de uso que se diseñó para la aplicación *Water Clients*. En él se distinguen dos actores:

- **Cliente**: puede consultar y añadir información adicional de sus contadores, gestionar alarmas y administrar su perfil.
- **Administrador**: puede generar y administrar los perfiles de usuarios de sus clientes y consultar información de ellos y sus contadores.

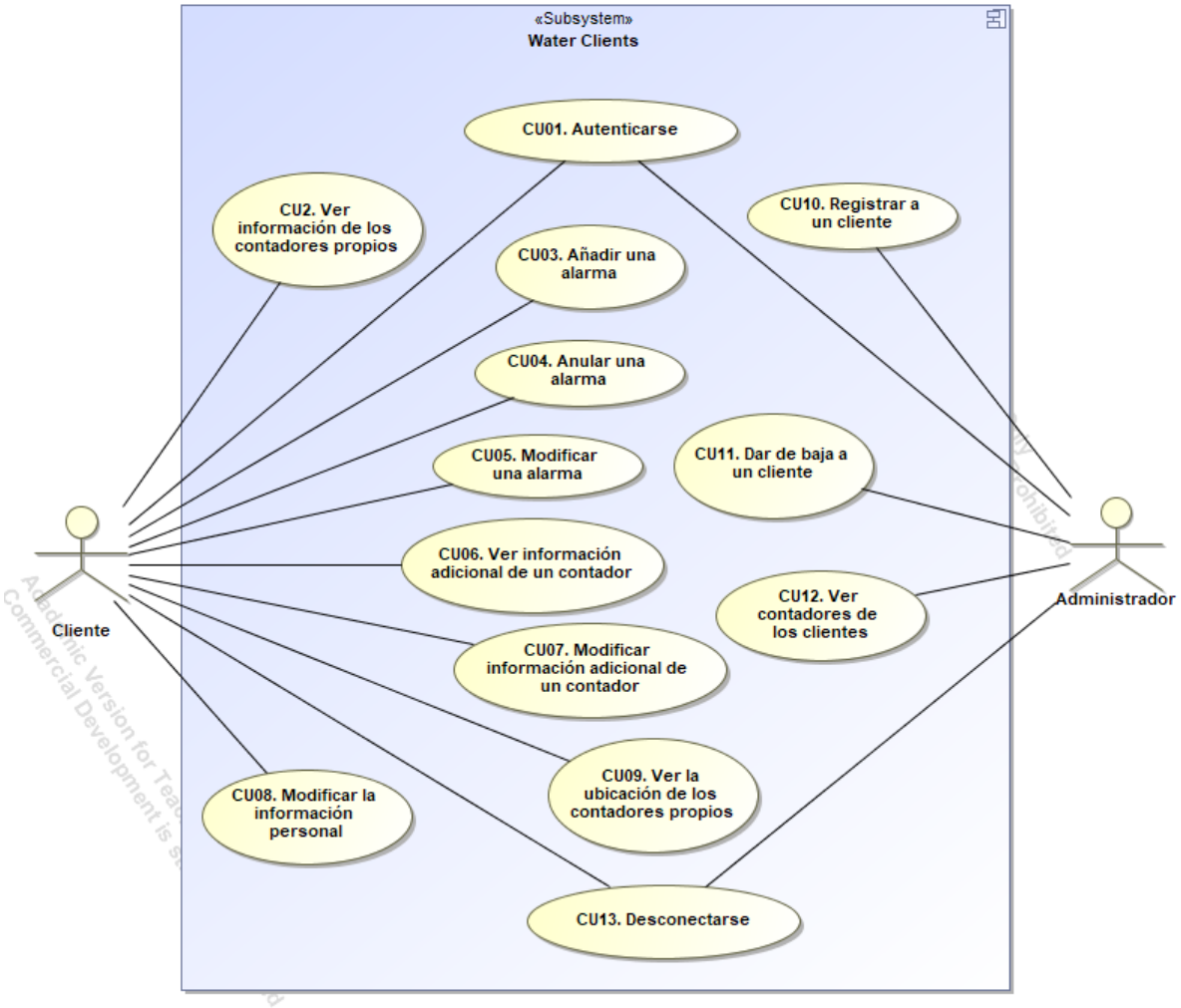

Figura 3.1: Diagrama de casos de uso

A continuación, se muestran desglosados los casos de uso definidos anteriormente, los cuales van desde la tabla 3.1 a la tabla 3.13.

Especificación del caso de uso CU01	
Identificador	CU01
Nombre	Autenticarse
Descripción	El sistema debe permitir que el usuario acceda a la plataforma usando sus credenciales.
Autor	Rosa María de Juan Oliva
Supervisor	Jose Luís Martínez Pérez
Actores	Cliente, Administrador
Precondición	El usuario debe estar registrado en la plataforma.
Secuencia normal	1. El usuario introduce su email 2. El usuario introduce su contraseña 3. El sistema valida los datos y muestra la página principal
Excepciones	▪ El usuario introduce un email erróneo ▪ El usuario introduce una contraseña errónea ▪ El usuario no introduce ningún dato
Prioridad	Alta
Comentarios	Por defecto el sistema tiene como página principal el listado de contadores.

Tabla 3.1: Especificación del caso de uso CU01.

Especificación del caso de uso CU02	
Identificador	CU02
Nombre	Ver información de los contadores propios
Descripción	El sistema debe permitir que el usuario consulte información detallada sobre sus contadores y su consumo.
Autor	Rosa María de Juan Oliva
Supervisor	Jose Luís Martínez Pérez
Actores	Cliente
Precondición	El usuario debe estar autenticado en la plataforma.
Secuencia normal	1. El cliente pincha en el botón "Contadores" 2. El sistema muestra el listado de los contadores del cliente 3. El cliente selecciona un contador 4. El sistema le muestra información detallada del contador seleccionado
Excepciones	No se aplica
Prioridad	Alta
Comentarios	Sin comentarios

Tabla 3.2: Especificación del caso de uso CU02.

Especificación del caso de uso CU03	
Identificador	CU03
Nombre	Añadir una alarma
Descripción	El sistema debe permitir que el usuario cree una alarma para uno o varios de sus contadores.
Autor	Rosa María de Juan Oliva
Supervisor	Jose Luís Martínez Pérez
Actores	Cliente
Precondición	El usuario debe estar autentificado en la plataforma.
Secuencia normal	1. El sistema muestra el listado de alarmas 2. El cliente pincha en el botón "Añadir" 3. El sistema le muestra un diálogo con los campos a completar 4. El cliente introduce la fecha de inicio y fin 5. El cliente selecciona el tipo de alarma 6. El cliente selecciona el o los contadores que tendrán la nueva alarma
Excepciones	▪ El cliente no completa algún campo obligatorio ▪ El cliente introduce un período de tiempo erróneo ▪ El cliente no selecciona ningún contador ▪ El cliente no selecciona el tipo de alarma
Prioridad	Media
Comentarios	Si no se especifica la fecha de finalización, la alarma continuará activa hasta que se desactive o se modifique la dicha fecha.

Tabla 3.3: Especificación del caso de uso CU03.

Especificación del caso de uso CU04	
Identificador	CU04
Nombre	Anular una alarma
Descripción	El sistema debe permitir que el usuario desactive una alarma.
Autor	Rosa María de Juan Oliva
Supervisor	Jose Luís Martínez Pérez
Actores	Cliente
Precondición	El usuario debe estar autenticado en la plataforma y debe haber creado antes la alarma que va a desactivar.
Secuencia normal	1. El sistema muestra el listado de alarmas 2. El cliente pincha en el botón "Modificar" 3. El sistema muestra un diálogo con los campos a modificar 4. El cliente pincha sobre el botón "Desactivar" 5. El sistema pide confirmación al usuario 6. El usuario confirma la desactivación de la alarma
Excepciones	No se aplica
Prioridad	Media
Comentarios	El usuario puede cancelar en cualquier momento la operación si pincha sobre el botón "Atrás".

Tabla 3.4: Especificación del caso de uso CU04.

Especificación del caso de uso CU05	
Identificador	CU05
Nombre	Modificar una alarma
Descripción	El sistema debe permitir que el usuario cambie las especificaciones de una alarma.
Autor	Rosa María de Juan Oliva
Supervisor	Jose Luís Martínez Pérez
Actores	Cliente
Precondición	El usuario debe estar autenticado en la plataforma y debe haber creado antes la alarma que va a modificar.
Secuencia normal	1. El sistema muestra el listado de alarmas 2. El cliente pincha en el botón "Modificar" 3. El sistema muestra un diálogo con los campos a modificar 4. El cliente modifica los datos que desea cambiar 5. El cliente pincha sobre el botón "Guardar" 6. El usuario confirma la desactivación de la alarma
Excepciones	▪ El cliente introduce un período de tiempo inválido ▪ El cliente deselecciona todos los contadores que tienen la alarma
Prioridad	Media
Comentarios	El usuario puede cancelar en cualquier momento la operación si pincha sobre el botón "Atrás" y no se guardarán los cambios realizados.

Tabla 3.5: Especificación del caso de uso CU05.

Especificación del caso de uso CU06	
Identificador	CU06
Nombre	Ver información adicional de un contador
Descripción	El sistema debe permitir que el usuario consulte la información adicional que tiene su contador.
Autor	Rosa María de Juan Oliva
Supervisor	Jose Luís Martínez Pérez
Actores	Cliente
Precondición	El usuario debe estar autenticado en la plataforma.
Secuencia normal	1. El sistema muestra la información perteneciente al contador del cliente 2. El cliente pincha sobre el enlace “Información adicional”
Excepciones	No se aplica
Prioridad	Media
Comentarios	Todos los contadores tendrán información adicional por defecto y no se tomará como información real hasta que ésta se modifique.

Tabla 3.6: Especificación del caso de uso CU06.

Especificación del caso de uso CU07	
Identificador	CU07
Nombre	Modificar información adicional de un contador
Descripción	El sistema debe permitir que el usuario cambie los datos adicionales de su contador.
Autor	Rosa María de Juan Oliva
Supervisor	Jose Luís Martínez Pérez
Actores	Cliente
Precondición	El usuario debe estar autenticado en la plataforma.
Secuencia normal	1. El sistema muestra la información perteneciente al contador del cliente 2. El cliente pincha sobre el enlace "Información adicional" 3. El sistema muestra un formulario con los datos 4. El cliente rellena los campos 5. El cliente pincha sobre el botón "Guardar" 6. El sistema guarda los datos
Excepciones	▪ El cliente introduce una cantidad de personas o baños erróneos cliente deja algún campo en blanco
Prioridad	Media
Comentarios	El usuario puede cancelar en cualquier momento la operación si pincha sobre el botón "Atrás" y no se guardarán los cambios realizados. Si el usuario no ha modificado ningún dato con anterioridad, el sistema mostrará unos datos por defecto.

Tabla 3.7: Especificación del caso de uso CU07.

Especificación del caso de uso CU08	
Identificador	CU08
Nombre	Modificar la información personal
Descripción	El sistema debe permitir que el usuario cambie los datos de su perfil.
Autor	Rosa María de Juan Oliva
Supervisor	Jose Luís Martínez Pérez
Actores	Cliente, Administrador
Precondición	El usuario debe estar registrado y autenticado en la plataforma.
Secuencia normal	1. El usuario pincha sobre el botón "Perfil" y selecciona la opción de "Ver perfil" 2. El sistema muestra los datos personales del usuario 3. El usuario modifica el dato que desea cambiar 4. El usuario pincha sobre el botón "Modificar" situado al lado del campo que ha cambiado 5. El sistema guarda el dato modificado
Excepciones	▪ El usuario introduce una contraseña inválida ▪ El usuario introduce un correo electrónico inválido ▪ El usuario deja en blanco algún campo
Prioridad	Media
Comentarios	Para el cambio de contraseña, el sistema pedirá confirmación del usuario. Si el usuario pincha sobre el botón "Modificar" y el dato no está cambiado, el sistema no realizará ningún cambio.

Tabla 3.8: Especificación del caso de uso CU08.

Especificación del caso de uso CU09	
Identificador	CU09
Nombre	El sistema debe permitir que el usuario consulte la posición de sus contadores en el mapa.
Descripción	El sistema debe permitir que el usuario acceda a la plataforma usando sus credenciales.
Autor	Rosa María de Juan Oliva
Supervisor	Jose Luís Martínez Pérez
Actores	Cliente
Precondición	El usuario debe estar autenticado en la plataforma.
Secuencia normal	1. El cliente pincha sobre el botón “Mapa” 2. El sistema le muestra la ubicación de todos los contadores del cliente
Excepciones	No se aplica
Prioridad	Baja
Comentarios	Sin comentarios

Tabla 3.9: Especificación del caso de uso CU09.

Especificación del caso de uso CU10	
Identificador	CU10
Nombre	Registrar a un cliente
Descripción	El sistema debe permitir que el usuario añada a un cliente nuevo con sus respectivos contadores a la plataforma.
Autor	Rosa María de Juan Oliva
Supervisor	Jose Luís Martínez Pérez
Actores	Administrador
Precondición	El usuario debe estar autenticado en la plataforma.
Secuencia normal	1. El administrador pincha sobre el botón "Gestión" 2. El sistema muestra un listado de clientes 3. El administrador pincha sobre el botón "Añadir nuevo cliente" 4. El sistema muestra un formulario 5. El administrador rellena los campos con los datos del cliente nuevo 6. El sistema pide confirmación al usuario 7. El administrador pincha sobre el botón "Registrar" 8. El sistema registra al nuevo usuario y guarda sus datos
Excepciones	▪ El administrador introduce un correo inválido ▪ El administrador introduce algún dato erróneo ▪ El administrador no completa todos los campos obligatorios
Prioridad	Media
Comentarios	El sistema creará una contraseña por defecto y se enviará al correo que el administrador haya especificado.

Tabla 3.10: Especificación del caso de uso CU10.

Especificación del caso de uso CU11	
Identificador	CU11
Nombre	Dar de baja a un cliente
Descripción	El sistema debe permitir que el usuario desactive la cuenta de un cliente.
Autor	Rosa María de Juan Oliva
Supervisor	Jose Luís Martínez Pérez
Actores	Administrador
Precondición	El usuario debe estar autenticado en la plataforma y debe de haber registrado al cliente anteriormente.
Secuencia normal	1. El administrador pincha sobre el botón "Gestión" 2. El sistema muestra un listado de clientes 3. El administrador pincha sobre el botón "Dar de baja" situado al lado del nombre del cliente 4. El sistema pide confirmación y muestra un campo de texto opcional para rellenarlo con detalles de la baja 5. El administrador confirma la operación 6. El sistema desactiva la cuenta del cliente indicado
Excepciones	No se aplica
Prioridad	Media
Comentarios	El sistema permite cancelar la operación antes al pinchar sobre el botón "Cancelar".

Tabla 3.11: Especificación del caso de uso CU11.

Especificación del caso de uso CU12	
Identificador	CU12
Nombre	Ver contadores de los clientes
Descripción	El sistema debe permitir que el usuario pueda visualizar la información de los contadores de sus clientes.
Autor	Rosa María de Juan Oliva
Supervisor	Jose Luís Martínez Pérez
Actores	Administrador
Precondición	El usuario debe estar autenticado en la plataforma.
Secuencia normal	1. El administrador pincha sobre el botón "Contadores" 2. El sistema muestra un listado de los contadores de sus clientes 3. El administrador selecciona el contador cuya información desea ver 4. El sistema le muestra los datos del contador seleccionado
Excepciones	No se aplica
Prioridad	Baja
Comentarios	Sin comentarios

Tabla 3.12: Especificación del caso de uso CU12.

Especificación del caso de uso CU13	
Identificador	CU13
Nombre	Desconectarse
Descripción	El sistema debe permitir que el usuario pueda cerrar su sesión.
Autor	Rosa María de Juan Oliva
Supervisor	Jose Luís Martínez Pérez
Actores	Administrador, Cliente
Precondición	El usuario debe estar autenticado en la plataforma.
Secuencia normal	1. El usuario pincha sobre el botón "Perfil" y selecciona la opción de "Desconectarse" 2. El sistema cierra la sesión del usuario.
Excepciones	No se aplica
Prioridad	Alta
Comentarios	Sin comentarios

Tabla 3.13: Especificación del caso de uso CU13.

3.1.2. Diagrama de clases

Para tener una idea clara de la estructura del sistema a construir, se diseñó el diagrama de clases en *UML*[1] que se muestra en la figura 3.2. En este diagrama se puede observar los atributos y métodos de los componentes, así como las relaciones entre ellos.

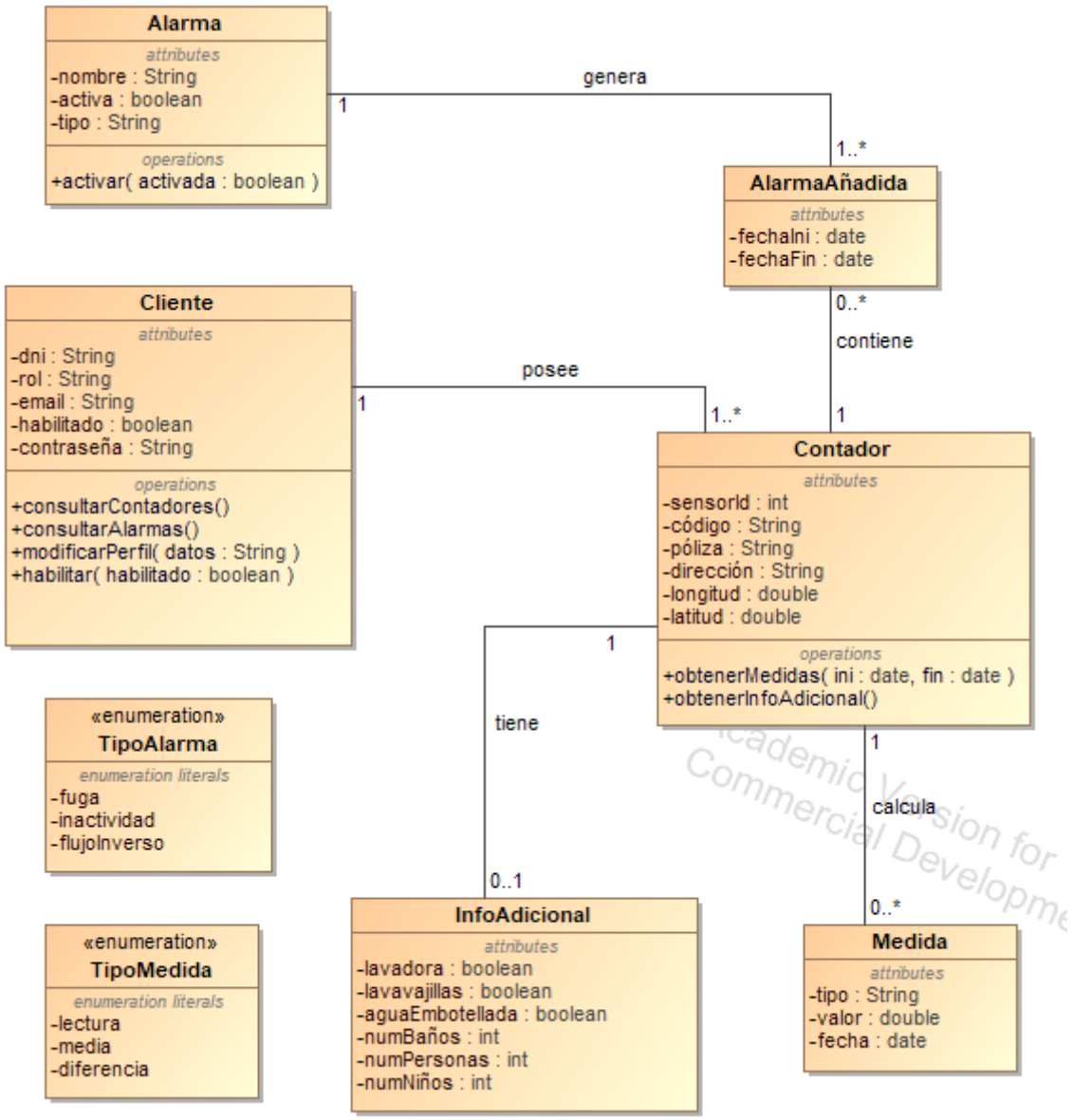

Figura 3.2: Diagrama de clases

[1]Lenguaje Unificado de Modelado o *Unified Modeling Language* en inglés. Este lenguaje de modelado se usa como estándar para definir el diseño de la estructura y procesos de un sistema informático.[19]

3.2. Diseño de la arquitectura del sistema

Las aplicaciones web suelen basarse en el modelo Cliente/Servidor de tres capas, en el cual los clientes hacen peticiones al servidor web para obtener la información solicitada a través del protocolo *HTTP* [5].

Las tres capas de este modelo son: la de presentación (parte del cliente y el servidor), la de proceso (servidor web) y de datos (servidor de datos) [5]. En la figura 3.3 se puede apreciar la arquitectura de la aplicación y la división entre sus capas.

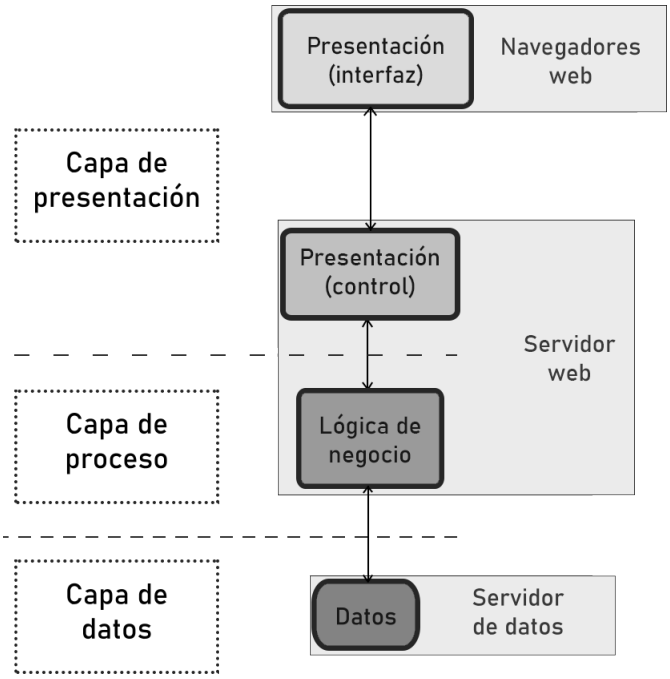

Figura 3.3: Esquema de la arquitectura

3.2.1. Base de datos

Se creó una base de datos con el fin de almacenar la información personal de los usuarios y sus roles, así como los datos básicos de los contadores e información adicional de los mismos (véase figura 3.4). Por el contrario, no se creó ninguna tabla para el guardado de lecturas de los contadores ni alarmas. Esto se debió a que la administración de dichos datos la gestionaban varios microservicios que ya contaban con su propia base de datos.

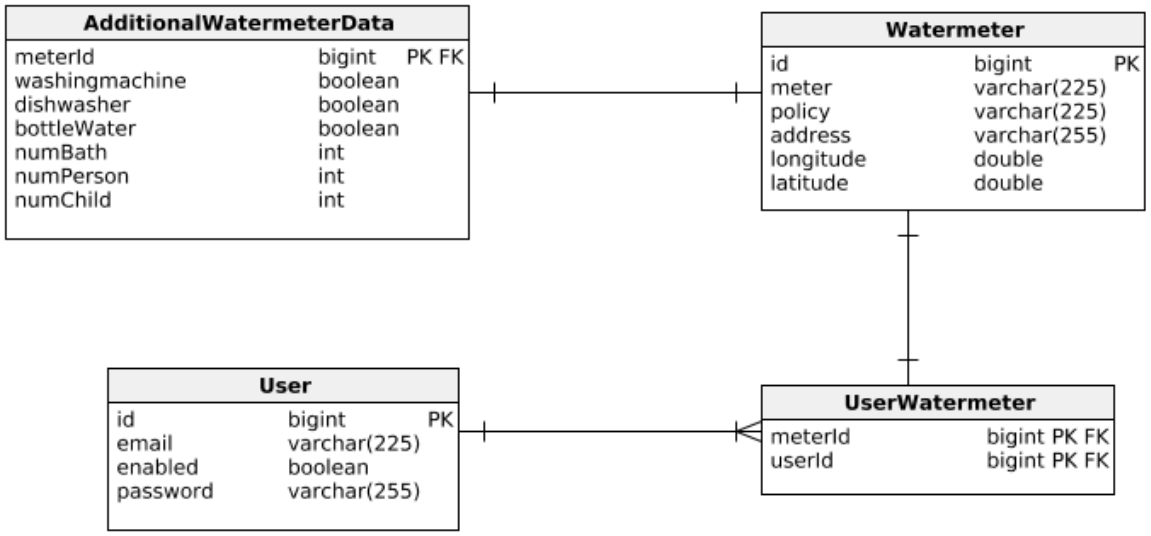

Figura 3.4: Diseño lógico de la base de datos

3.3. Diseño de la interfaz

Para el diseño de la interfaz de usuario, se partió del prototipo desarrollado por el mánager del equipo de desarrollo, el cual a su vez partió del diseño de una aplicación anteriormente implementada llamada *Smart Water*. Esta aplicación está relacionada con la gestión de contadores, pero con la diferencia de estar enfocada únicamente al ámbito profesional, en concreto a las empresas que se dedican al abastecimiento de agua. Esto se explica con más detalle en la sección 1.2.

También se tuvo en cuenta seguir las tres reglas de oro de Theo Mandel [7], las cuales se citan a continuación:

1. Dar el control al usuario.
2. Reducir la carga de memoria del usuario.
3. Mantener la consistencia de la interfaz.

La primera regla hace referencia a la libertad que tiene el usuario para interactuar con los elementos de la interfaz y deshacer o cancelar acciones [7]. La segunda se centra en hacer dicha interfaz lo más estructurada e intuitiva posible para que el usuario tenga que recordar la mínima información [7]. La última regla trata de mantener la armonía visual en toda la aplicación para hacer predecible y cómoda la navegación por ella [7].

3.3.1. Definición de arquetipos y escenarios

Sin embargo, para crear un diseño adecuado, no basta con tener en cuenta las tres reglas de oro anteriormente explicadas. Se debe conocer las características de los usuarios finales que usarán la aplicación para desarrollar un producto que se adecúe a ellos, teniendo en cuenta, su edad, sus costumbres, sus conocimientos, etc. A continuación, se explican dos arquetipos con dos escenarios distintos para cada uno, los cuales se definieron con la intención de tener una idea general de los tipos de usuarios finales que usarían la aplicación.

Como se puede observar en la tabla 3.14, el primer ejemplo propuesto alude a un ciudadano de un municipio. En cambio, el segundo ejemplo mostrado por la tabla 3.15 está relacionado con el personal de una empresa intermediaria o ayuntamiento que administra las cuentas y da soporte a sus clientes. Los roles dentro de la plataforma de ambos arquetipos se han mencionado anteriormente al principio de la sección 3.3.

El primer arquetipo hace referencia a Marta Sánchez Ruiz, una posible cliente de esta aplicación a la que le gustaría consultar su consumo mensual de agua y programar alarmas para sus contadores con la intención de ser avisada en caso de haber alguna fuga o emergencia similar.

Marta Sánchez Ruiz

- Mujer de mediana edad.
- Usuaria frecuente de dispositivos informáticos como teléfonos móviles, *tablets* y ordenadores.
- Accede a internet diariamente para ver periódicos digitales, ver vídeos en *Youtube* o usar *Netflix*.
- Eventualmente hace alguna compra *online*.
- De vez en cuando accede de manera digital a su cuenta bancaria para comprobar gastos y recibos o hacer alguna transferencia.
- Pese a que está acostumbrada a hacer trámites por internet, no tiene una gran fluidez a la hora de llevarlos a cabo y de vez en cuando llega a pedir ayuda a algún familiar.
- Marta está registrada en la plataforma como cliente y ha iniciado sesión.

Escenario 1	Marta desea consultar el consumo de agua de su casa este último mes y compararlo con el gasto promedio de su hogar, en el cual viven su pareja, sus dos hijos y ella.
Qué necesita	Espera que haya botón o pestaña visible y fácil de reconocer para acceder al listado de sus contadores de agua, ya que tiene dos viviendas en propiedad. Además, necesita que la información sobre el consumo del contador que esté consultando se vea de manera clara y comprensible.
Cómo le ayuda la plataforma web	La plataforma web debe contener un listado de los contadores pertenecientes al cliente. La vista del contador consultado será fácil de entender gracias al diseño de la plataforma web, que mostrará la gráfica de consumo junto con un pequeño formulario para especificar el período de tiempo y la frecuencia que se quiere consultar. También dispondrá de datos del consumo del día actual, la diferencia con el día anterior y el promedio. A su vez habrá un pequeño listado con las alarmas que tiene el contador.
Escenario 2	Marta desea activar una alarma para el contador de la casa de campo que tiene, la cual sólo visita en verano, para que le avise en caso de que se produzca una fuga.
Qué necesita	Espera que haya un botón o pestaña visible para poder programar una alarma en el contador en el que desea añadirla. También necesita que el proceso sea fácil y rápido.
Cómo le ayuda la plataforma web	La plataforma web dispondrá de una pestaña en la cual se listen las alarmas de todos los contadores del cliente y un botón fácil de localizar llamado “Añadir”. Además, se podrá añadir una alarma desde la vista que contiene la información de un contador, ya que en ésta también se verán las alarmas que contiene.

Tabla 3.14: Ejemplo de arquetipo de usuario cliente

El segundo arquetipo hace referencia a Sara Martínez García, una posible trabajadora de una empresa intermediaria entre la empresa encargada del suministro de agua de una zona y los clientes finales. Este tipo de empresa suele operar en una ciudad o pueblo concretos.

Sara Martínez García

- Mujer de mediana edad.
- Oficinista.
- Usuaria común de dispositivos informáticos, especialmente del teléfono móvil y el ordenador, puesto que los utiliza durante su jornada laboral.
- Suele hacer trámites a través de internet.
- Accede a la plataforma desde su ordenador de oficina.
- Tiene una gran fluidez con el uso de diferentes programas y manejo del ordenador, ya que es fundamental para realizar su trabajo.
- Sara está registrada en la plataforma como administradora y ha iniciado sesión.

Escenario 1	Sara desea dar de alta a un nuevo cliente en la plataforma.
Qué necesita	Espera poder realizar el alta del cliente de manera segura, con la opción de cancelar la operación en cualquier momento en caso de detectar algún error en los datos del cliente.
Cómo le ayuda la plataforma web	La plataforma web debe tener de una pestaña en la cual se listen los clientes que gestiona la administradora y un botón fácil de localizar llamado "Añadir nuevo cliente". También dispondrá de un formulario con los campos a rellenar el cual necesitará confirmación por parte de la administradora para poder registrar los datos proporcionados.
Escenario 2	Sara desea consultar los datos del consumo promedio de los habitantes de una de las ciudades donde su empresa suministra el agua.
Qué necesita	Espera que haya algún tipo de filtro para poder indicar el listado de los contadores que desea ver. Además, necesita poder ver el promedio de consumo de los contadores filtrados.
Cómo le ayuda la plataforma web	La plataforma web debe tener un listado de contadores con una serie de filtros, uno de ellos para indicar las localidades que se quieren consultar. A su vez, se dispondrá de un campo de búsqueda donde poder escribir el nombre de un cliente para encontrar rápidamente sus contadores. También tendrá un botón con el que poder calcular el promedio de consumo con los datos de los contadores filtrados.

Tabla 3.15: Ejemplo de arquetipo de usuario administrador

3.3.2. Guía de estilos

Los prototipos creados (véase anexo A.1) tienen en cuenta las pautas mencionadas anteriormente. El diseño es sencillo, con gráficas que facilitan la lectura al usuario y con paneles de información muy claros, concisos, sin textos largos y con iconos descriptivos. Sin embargo, dichos prototipos no mostraban los colores y los estilos finales que se usaron en la implementación final (véase sección 4.2), por lo que se diseñó una pequeña guía de estilo que los complementara.

La paleta de colores que se utilizó en la interfaz de usuario, mostrada en la figura 3.5, está compuesta principalmente por diferentes tonalidades de azul y gris. Esto fue debido a la intención de dar una sensación de tranquilidad y seriedad a la aplicación, así como relacionarla con la temática del agua y los contadores. Para llegar a esta conclusión se realizó un estudio previamente. Cabe añadir que la última fila hace referencia a los colores usados para mostrar avisos de errores, como sería la introducción de un campo inválido en un formulario.

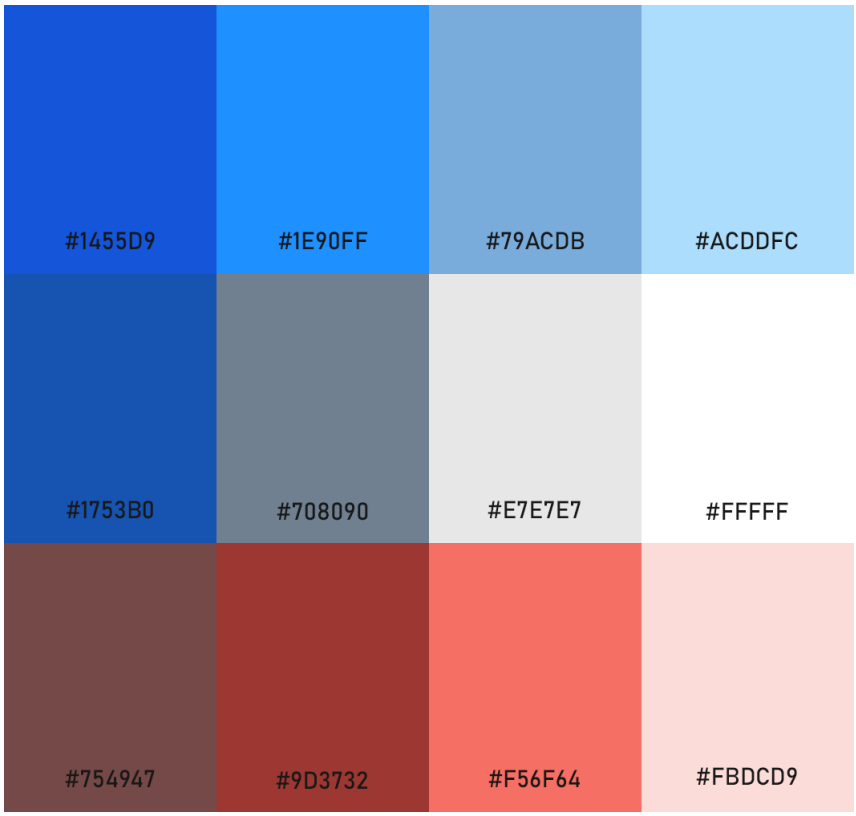

Figura 3.5: Paleta de colores

Con respecto a los textos, para los encabezados y el cuerpo de cada vista, se establecieron unas directrices a seguir con la finalidad de ayudar a mantener la consistencia entre las diferentes vistas de la interfaz gráfica. Además, se optó por usar la fuente estándar *Arial*, la cual es universalmente conocida y cómoda de leer. La tabla 3.16 muestra el estilo y color establecidos para la tipografía según su función.

Título	Usada para indicar el título de cada bloque de información.
Cifra	Usada como texto informativo en los formularios y bloques con datos.
Información	Usada para los datos numéricos pertenecientes a las medidas tomadas del contador.
Dirección postal	Usada para las direcciones postales.

Tabla 3.16: Tipografías utilizadas en la interfaz gráfica

Por último, se diseñó el botón principal de la aplicación, el cual se utilizaría para las búsquedas, el guardado de datos y la autenticación del usuario. La tabla 3.17 que aparece a continuación muestra el diseño del botón en sus tres estados posibles.

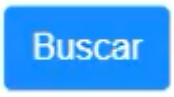	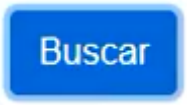	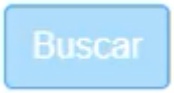
Botón normal	Botón pulsado	Botón deshabilitado

Tabla 3.17: Diseño del botón principal

Capítulo 4

Implementación

4.1. Detalles de implementación

La estructura general de la aplicación se muestra en la figura 4.1. La parte del *frontend* se implementó con el *framework* de Angular, el cual se explica con más detenimiento en la sección 2.2. Además, se usaron varias librerías externas, como *Transloco* para la traducción de los textos mostrados en la interfaz de usuario, *Chart.js* para las gráficas, *Leaflet* para la visualización de los mapas y *Fortawsome-Angular* para los iconos.

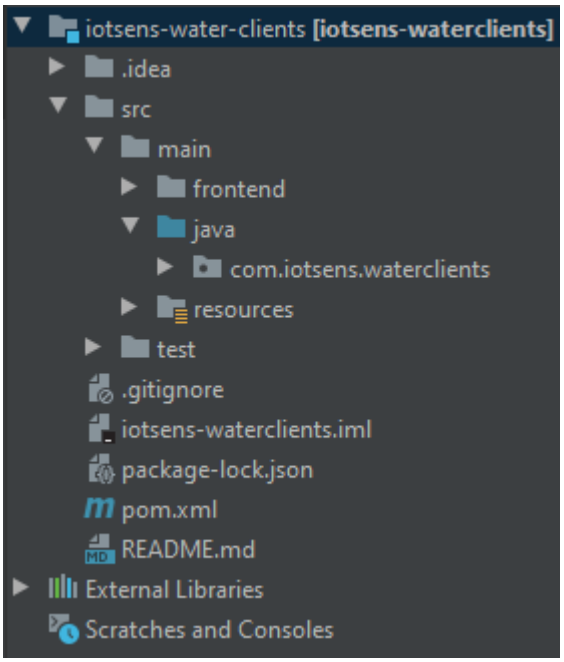

Figura 4.1: Estructura general de la aplicación

Por el contrario, el directorio con el nombre de java contiene la parte del *backend* de la aplicación. En su implementación se utilizó el *framework Spring*, definido con más detalle en la sección 2.2.

Por último, el directorio *resources* contiene las sentencias de creación de la base de datos, así como varias inserciones por defecto con el fin de mantener una consistencia de datos mínima. Esto implica que si, por ejemplo, se borrara manualmente de la base de datos la tabla *AdditionalWatermeterData* (véase figura 4.2), ésta se volvería a crear al ejecutarse la aplicación y añadiría los valores por defecto estipulados para cada contador. En caso de que los valores se fueran actualizando, esta sentencia no se ejecutaría, ya que no sería necesaria.

```
create table AdditionalWatermeterData (

    meterId bigint unsigned not null,
    washingmachine boolean not null,
    dishwasher boolean not null,
    bottleWater boolean not null,
    numBath int unsigned not null,
    numPerson int unsigned not null,
    numChild int unsigned not null,

    primary key (meterId),

    constraint fk_AdditionalData_Watermeter_id foreign key (meterId) references Watermeter(id)
);

insert into AdditionalWatermeterData(meterId, washingmachine, dishwasher, bottleWater,
                                     numBath, numPerson, numChild)

select id, false, false, false, 0, 0, 0
from Watermeter;
```

Figura 4.2: Fichero *V5__AdditionalWatermeterData.sql*

4.1.1. Estructura del *frontend*

La estructura del *frontend* de la aplicación está dividida principalmente en componentes, servicios y modelos. Antes de entrar en profundidad a explicarlos, se deben mencionar otros elementos que también la conforman.

Como se observa en la figura 4.3, hay varios archivos de configuración situados en el directorio raíz del *frontend* que contienen metadatos del proyecto, así como las dependencias que utiliza. Estos archivos van ligados con la carpeta *nodes_modules*, en la cual se encuentran las librerías instaladas. A su vez, se pueden visualizar los ficheros de inicio de la aplicación y el fichero *styles.scss*. En este último se especifican los estilos comunes de la interfaz de usuario, como la tipografía, el color y forma de los botones, etc.

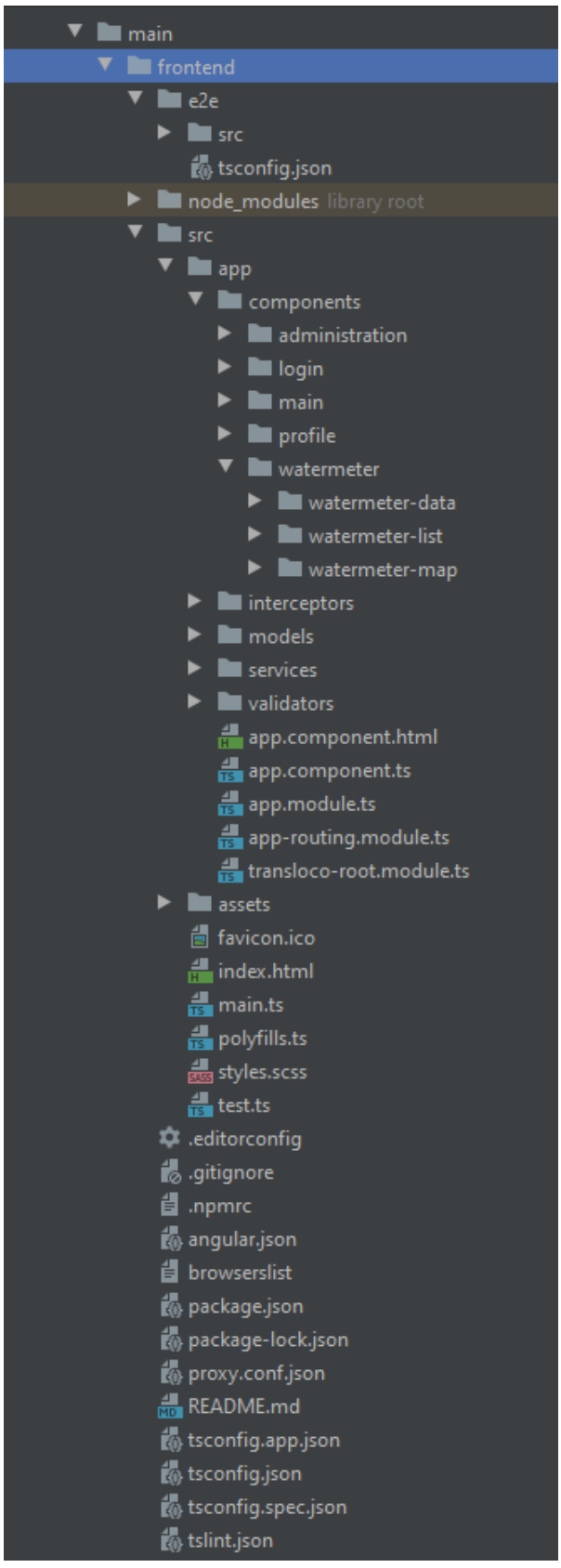

Figura 4.3: Estructura del *frontend* de la aplicación

Otro directorio que destacar es *assets*, en que están almacenados los ficheros que contienen las traducciones de los textos que aparecen en las vistas. Para acceder a ellos se utiliza un servicio de la librería *Transloco*, el cual se puede llamar tanto desde la vista como desde el controlador de un componente. Hay un ejemplo de su uso al principio del código mostrado en la figura 4.4.

```
<ng-container *transloco="let t;">
   <div class="row" *ngIf="processed">
      <div class="col ml-auto justify-content-center border background shadow-sm">
         <div class="row p-3">
            <div class="col-4 p-3">
               {{ t('watermeter.meterId') }}:
               <h5 class="text-secondary-app">{{watermeter.meter}}</h5>
            </div>
            <div class="col p-3">
               <p class="same-line">{{ t('watermeter.policy') }}: </p>
               <h5 class="same-line"><strong>{{watermeter.policy}}</strong></h5>
               <div>
                  <fa-icon [icon]="faHome" class="text-tertiary"></fa-icon>
                  <p class="font-italic text-tertiary same-line p-2">{{watermeter.address}}</p>
               </div>
            </div>
         </div>
      </div>
      <div class="col-2 mr-auto p-3 justify-content-center border background shadow-sm text-align-center">
         <div class="row p-3 text-align-center justify-content-center">
            <div class="text-align-center justify-content-center" *ngIf="consumptionIsProcessed!=undefined">
               <div *ngIf="!consumptionIsProcessed">
                  <fa-icon [icon]="faShower" class="text-primary-app"></fa-icon>
                  <h5 class="p-2 font-italic text-primary-app same-line">{{ t('watermeter.-') }}</h5>
               </div>
               <div *ngIf="consumptionIsProcessed">
                  <fa-icon [icon]="faShower" class="text-primary-app"></fa-icon>
                  <h5 class="p-2 same-line">{{todayConsumption}} m3</h5>
               </div>
               <p>{{ t('watermeter.todayConsumption') }}</p>
            </div>
            <div class="text-align-center justify-content-center" *ngIf="consumptionIsProcessed===undefined">
               <div class="spinner-border text-primary-app" role="status">
                  <span class="sr-only"></span>
               </div>
            </div>
         </div>
      </div>
   </div>
   <app-watermeter-measures [meter]= "meter" (notifyParent)="getTodayConsumption($event)"></app-watermeter-measures>
   <app-watermeter-alarms [meter]= "meter"></app-watermeter-alarms>
   <router-outlet></router-outlet>
</ng-container>
```

Figura 4.4: Código del fichero *watermeter.component.html* (componente *Watermeter*)

Por otra parte, encontramos dentro del directorio *app* los ficheros de enrutado de los componentes y su jerarquía. Esto es importante para poder especificar, por ejemplo, que la barra de navegación debe de estar presente en todas las vistas a excepción del *Login*. Esta carpeta también contiene otros directorios, como *validators*, que almacena los validadores usados para

verificar los datos de los formularios, o *interceptors*, para mostrar las vistas en función de los permisos que tengan los usuarios.

Con respecto a los componentes, cada uno de ellos está formado por dos ficheros. El primero (véase figura 4.4) es un fichero de *html* que contiene únicamente los elementos que se muestran al usuario y su formato, es decir, se trata de una vista completamente pasiva. El segundo fichero (véase 4.5) tiene la extensión *ts* y engloba el código que se encarga de pedir la información necesaria que debe mostrarse por pantalla. Este fichero contiene una clase de *Typescript* con la anotación *@Component*. A diferencia de otros *frameworks* como *React*, *Angular* permite hacer este tipo de separación entre vista y controlador, facilitando así su implementación.

```
// imports

@Component({
    selector: "app-watermeter",
    templateUrl: "./watermeter.component.html",
    styleUrls: [ "../watermeter.component.scss" ]
})

export class WatermeterComponent implements OnInit {

    watermeter: Watermeter;
    meter: string; processed = false;
    todayConsumption: string; // comes from child component
    consumptionIsProcessed: boolean;
    faHome = faHome; faShower = faShower; // icons
    public static BSCONFIG = {containerClass: "datepicker"};

    constructor(private watermeterService: WatermeterService,
                private activatedRoute : ActivatedRoute) {}

    ngOnInit() { this.updateList(); }

    updateList(){ // get watermeter data from db
          this.activatedRoute.params.subscribe( options: params => {
           this.updateMeter(params["meter"]);
           this.watermeterService.getWatermeter(params["id"]).subscribe(res => {
               this.watermeter = res;
               this.processed = true;
           })
       });
    }

    updateMeter(meter: string){ this.meter = meter; }

    // executes when component child send the data
    getTodayConsumption(data) {
        if( data !== "-") {
            this.todayConsumption = data;
            this.consumptionIsProcessed = true;
        }else {
            this.consumptionIsProcessed = false;
        }
    }
}
```

Figura 4.5: Código del fichero *watermeter.component.ts* (componente *Watermeter*)

Asimismo, se debe tener en cuenta que una determinada vista de la interfaz del usuario

puede estar formada por varios componentes. En la aplicación, el componente que muestra una tabla con el listado de alarmas que tiene un contador lo engloba un componente padre que contiene los datos básicos de dicho contador y sus medidas. Al final del código mostrado en la figura 4.4 se puede ver un ejemplo de ello, incluyendo el paso de parámetros y propiedades necesarios para que ambos componentes puedan comunicarse.

Relacionado con esto último, se puede observar el mecanismo de comunicación entre el componente padre e hijos en la figura 4.6, que hace referencia a un fragmento del componente hijo de *Watermeter* y es el que muestra las lecturas del contador. El atributo que está marcado con la anotación *@Input* sirve para obtener los valores de los parámetros pasados por el componente padre. Por el contrario, la anotación *@Output* indica el evento que se ejecutará para avisar al componente padre y enviarle la información necesaria proveniente del hijo. Este tema está comentado en la subsección 4.1.3, el cual explica los problemas que hubo durante la implementación del *frontend*.

```
// ...

@Input() meter: string;
@Output() notifyParent = new EventEmitter<string>();

constructor(private watermeterService: WatermeterService,
            private formBuilder: FormBuilder,
            private translocoService: TranslocoService) {}

ngOnInit() {...}

updateList() {...}

getAverageAndDifference() {...}

calculateAverageAndDifference(measures: Measure[]) {
    let sum: number = 0;
    for (let measure of measures) {
        sum+= parseFloat(measure.value);
    }

    this.average = Math.round( x: (sum/measures.length) * 1000) / 1000;
    this.averageIsProcessed = true;

    let measure: Measure = measures[measures.length-1];
    let lastDateWithMeasures = moment(measure.timestamp).toDate();
    let yesterday = new Date();
    yesterday.setDate(yesterday.getDate() - 2);

    if (lastDateWithMeasures < yesterday) {...}else {...}

    this.passTodayConsumption();
}

passTodayConsumption() {...}

updateLineChart(option: string): boolean {...}

// ...
```

Figura 4.6: Fragmento del componente hijo de *Watermeter* para obtener las lecturas del contador

Otra característica a destacar de los componentes es la implementación de la interfaz *OnInit* (véase figura 4.6), que sirve para indicar qué métodos se quieren ejecutar al renderizar la vista del componente. También se debe considerar cómo realiza la llamada al método de un servicio, ya que realmente se suscribe a él para ser notificado cuando le lleguen los datos y, mientras tanto, se ejecute paralelamente el resto del programa. Esto es así para optimizar su ejecución, ya que si se quiere hacer más de una consulta a la base de datos sería tedioso tener que esperar a que acabaran todas ellas para cargar la vista. En otras palabras, los servicios implementan el patrón *Observer* a través del uso de los *observables* de *Angular*. Además, como se muestra en la figura 4.4, es posible indicar cuándo se debe visualizar determinados elementos de la vista usando **ngIf*, puesto que consulta una variable booleana que cambia una vez se reciben los datos.

Por otro lado, están los servicios, que son los que se comunican directamente con el *backend* de la aplicación y en ellos están definidas las llamadas necesarias para tal propósito. Pueden ser utilizados por diferentes componentes y cada servicio se encarga de la gestión de un determinado tipo de datos, como las medidas de un contador, las alarmas o la información del usuario. La figura 4.7 hace referencia un fragmento de la clase *watermeter.service.ts*, en el cual se puede ver los dos tipos de llamadas: la que realiza para obtener la información adicional de un contador y la que actualiza los datos. Como se puede observar, no es necesario indicar la *URL*[1] del *backend* completa, ésta se puede especificar en los ficheros de configuración.

```
getAlarms(meter: string, isActive: boolean, dateFrom: Date, dateUntil: Date): Observable<Alarm[]> {
    let dateUntilString = moment(dateUntil).format( date: "YYYY-MM-DDTHH:mm");
    let params = new HttpParams()
        .set("dateUntil", dateUntilString);

    if (isActive != null) {
        params = params.append( name: "isActive", String(isActive));
    }
    if (dateFrom != null) {
        let dateFromString = moment(dateFrom).format( date: "YYYY-MM-DDTHH:mm");
        params = params.append( name: "dateFrom", dateFromString);
    }

    return this.http.get<Alarm[]>(`services/watermeters/${meter}/alarms`, {params: params});
}

getAdditionalWatermeterData(id: number): Observable<AdditionalWatermeterData> {
    return this.http.get<AdditionalWatermeterData>(`services/watermeters/${id}/additionalData`);
}

updateAdditionalWatermeterData(additionalData: AdditionalWatermeterData) {
    return this.http.put<void>(`services/watermeters/${additionalData.meterId}/additionalData`, additionalData);
}

getCoordinatesFromWatermeters(): Observable<SensorCoordinate[]> {
    return this.http.get<SensorCoordinate[]>(`services/watermeters/map`);
}
```

Figura 4.7: Fragmento del fichero *watermeter.service.ts*

[1]Localizador Uniforme de Recursos o *Uniform Resource Identifier* en inglés. Hace referencia a la dirección específica de un recurso red con el fin de poder identificarlo [12]

En referencia a los modelos, éstos son clases en *Typscript* que ayudan a dar forma a los datos provenientes del *backend*. La figura 4.8 muestra la clase *Watermeter.ts*, correspondiente a los datos básicos que tiene un contador. Para que la información se almacene correctamente, es necesario que la clase tenga los mismos atributos y se llamen igual que las clases definidas en el modelo del *backend* (véase figura 4.10).

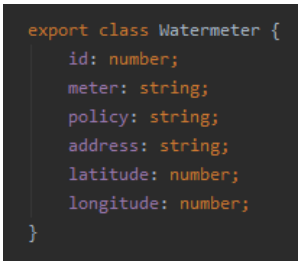

Figura 4.8: Clase *Watermeter.ts*

Finalmente, cabe destacar el uso del patrón MVVM (*model-view-viewmodel*) en la implementación del *frontend*, el cual es una variación del patrón clásico MVC (modelo-vista-controlador, o en inglés *model-view-controller*). Esto es así debido a la sincronización de datos que tiene el *framework Angular*, el cual implica una dependencia entre la vista y el modelo [16].

4.1.2. Estructura del *backend*

La estructura del *backend* de la aplicación se puede contemplar en la figura 4.9, que muestra una serie de directorios entre los que podemos encontrar *services* y *models*, los cuales tienen el mismo nombre que en el *frontend* (véase figura 4.3). Ambos desempeñan funciones similares, aunque en este caso los servicios son más bien los intermediarios que hacen consultas a microservicios externos o usan los métodos de las clases que se encargan de pedir la información a la base de datos. Los servicios también están divididos según sus cometidos, son usados por los controladores y están marcados con la anotación *@Service*.

Por el contrario, los modelos, al igual que en *frontend*, sirven para representar los datos obtenidos de las peticiones de la base de datos y los microservicios.

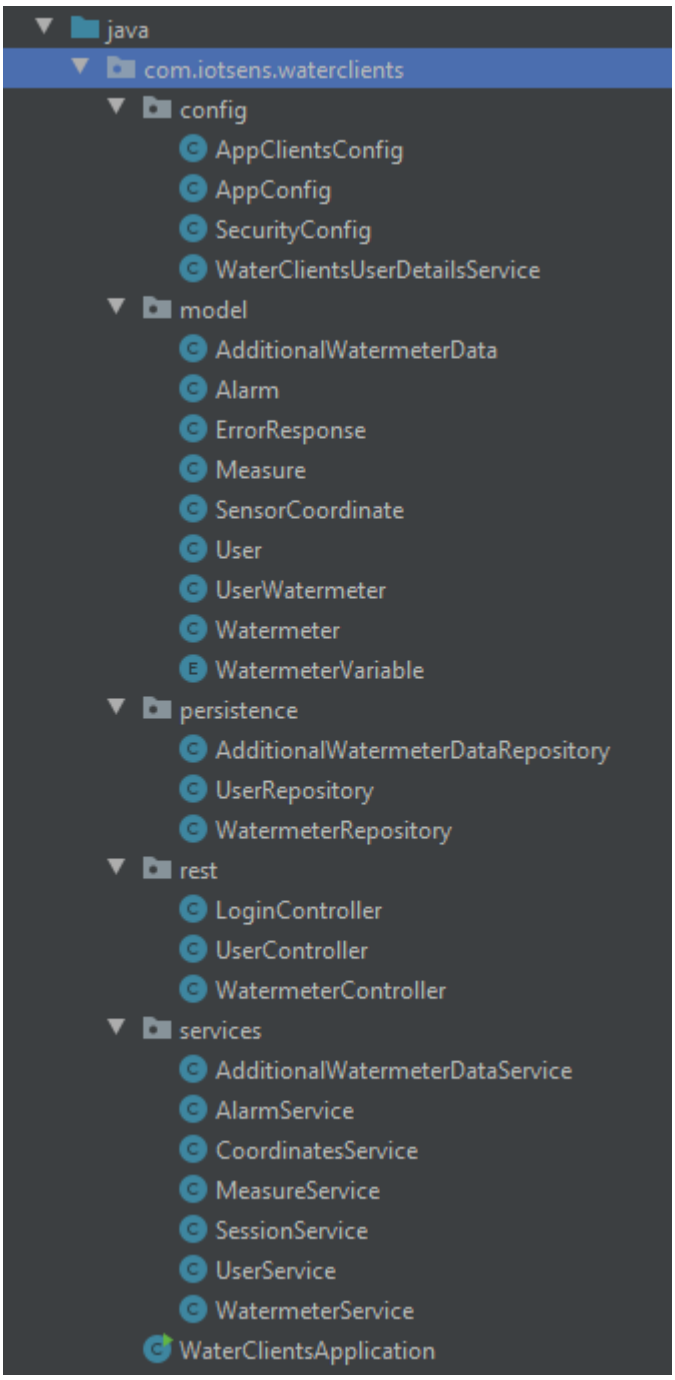

Figura 4.9: Estructura del *backend* de la aplicación

Si estos datos provienen de las consultas de la base de datos, requieren de un método estático que genere un objeto de esa clase (véase figura 4.10). Además, su tratamiento y transformación se realizan en las clases que se encuentran en el directorio *persistence*. Estas clases están marcadas con la anotación *@Repository* y son las popularmente conocidas como DAOs (Objeto de Acceso a Datos, en inglés *Data Acess Object*), ya que implementan este patrón de diseño. Parte de la implementación de una de ellas se puede examinar en la figura 4.11.

```java
public class Watermeter {

    private long id;
    private String meter;
    private String policy;
    private String address;
    private double latitude;
    private double longitude;

    public Watermeter(
            long id, String meter, String policy, String address, double latitude, double longitude) {
        this.id = id;
        this.meter = meter;
        this.policy = policy;
        this.address = address;
        this.latitude = latitude;
        this.longitude = longitude;
    }

    public static Watermeter readWatermeter(
            long id, String meter, String policy, String address, double latitude, double longitude) {
        return new Watermeter(id, meter, policy, address, latitude, longitude);
    }

    public String getMeter() { return meter; }
    public String getPolicy() { return policy; }
    public String getAddress() { return address; }
    public double getLatitude() { return latitude; }
    public void setLatitude(double latitude) { this.latitude = latitude; }
    public double getLongitude() { return longitude; }
    public void setLongitude(double longitude) { this.longitude = longitude; }

    @Override
    public boolean equals(Object o) {...}

    @Override
    public int hashCode() {
        return Objects.hash(id, meter, policy, address, latitude, longitude);
    }

    @Override
    public String toString() {...}
}
```

Figura 4.10: Clase *Watermeter.java*

```java
@Repository
public class AdditionalWatermeterDataRepository {

    private NamedParameterJdbcTemplate namedParameterJdbcTemplate;

    public AdditionalWatermeterDataRepository(NamedParameterJdbcTemplate namedParameterJdbcTemplate) {
        this.namedParameterJdbcTemplate = namedParameterJdbcTemplate;
    }

    public void updateAdditionalDatafromMeter(long meterId, AdditionalWatermeterData additionalData) {
        SqlParameterSource parameters = new MapSqlParameterSource()
                .addValue("meterId", meterId)
                .addValue("washingmachine", (additionalData.isWashingmachine()) ? 1 : 0)
                .addValue("dishwasher", (additionalData.isDishwasher()) ? 1 : 0)
                .addValue("bottleWater", (additionalData.isBottledWater()) ? 1 : 0)
                .addValue("numBath", additionalData.getNumBath())
                .addValue("numPerson", additionalData.getNumPerson())
                .addValue("numChild", additionalData.getNumChild());

        this.namedParameterJdbcTemplate.update(
                "update AdditionalWatermeterData " +
                        "set washingmachine = :washingmachine, " +
                        "dishwasher = :dishwasher, " +
                        "bottleWater = :bottleWater, " +
                        "numBath = :numBath, " +
                        "numPerson = :numPerson, " +
                        "numChild = :numChild " +
                        "where AdditionalWatermeterData.meterId = :meterId", parameters);
    }

    // ...
```

Figura 4.11: Fragmento de la clase *AdditionalWatermeterDataRepository.java*

Por otro lado, en el directorio *rest* se encuentra la implementación los controladores que mapean las peticiones *http* al *backend* de la aplicación. Estas clases están señaladas con la anotación *@RestController* que proporciona el *framework Spring* y contienen los servicios que van a utilizar. A su vez, estas clases utilizan otras anotaciones como *@RequestMapping* para indicar la ruta común o *@GetMapping* para especificar que es una petición de tipo *GET*. La figura 4.12 muestra un fragmento de la clase *WatermerController.java* donde se observa la estructura de estos controladores.

Al describir estos elementos que conforman la estructura del *backend*, es posible percatarse de la presencia del patrón compuesto *MVC*. El *framework Spring* facilita su implementación gracias a sus anotaciones, clases y métodos propios.

Finalmente, se debe mencionar el directorio *config* que, como su nombre indica, contiene las configuraciones globales del *backend* de la aplicación, así como las restricciones de seguridad, el acceso a la base de datos y a los microservicios. La configuración de acceso a estas últimas está representada en la figura 4.13. En ella se puede comprobar cómo se obtienen las credenciales necesarias a través de la anotación *@Value* con el fin de acceder a los microservicios utilizados para la obtención de medidas y alarmas de los contadores. También se puede observar cómo se crea una instancia de esos microservicios con la anotación *@Bean*, que hace uso del patrón *Singleton*.

```java
@RequestMapping("services")
@RestController
public class WatermeterController {

    // ...

    public WatermeterController(···
    }

    @GetMapping("/watermeters/{watermeterId}")
    public Watermeter getWatermeter(@PathVariable long watermeterId) {
        return watermeterService.getWatermeter(watermeterId);
    }

    @GetMapping("/watermeters/{meterId}/measures")
    public List<Measure> getMeasures(···
    }

    @GetMapping("/watermeters/{meterId}/alarms")
    public List<Alarm> getAlarmsFromMeter(
            @PathVariable String meterId,
            @RequestParam(required = false) Boolean isActive,
            @RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) Date dateFrom,
            @RequestParam @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) Date dateUntil)
            throws AuthorizationException {

        return alarmService.getAlarmsFromMeter(meterId, isActive, dateFrom, dateUntil);
    }

    @GetMapping("/watermeters/{watermeterId}/additionalData")
    public AdditionalWatermeterData getAdditionalDataFromMeter(@PathVariable long watermeterId) {
        return this.additionalWatermeterDataService.getAdditionalDataFromMeter(watermeterId);
    }

    @PutMapping("/watermeters/{watermeterId}/additionalData")
    public void updateAdditionalDataFromMeter(
        @PathVariable long watermeterId,
        @RequestBody AdditionalWatermeterData additionalWatermeterData) {
        this.additionalWatermeterDataService.updateAdditionalDatafromMeter(watermeterId, additionalWatermeterData);
    }

    // ...
```

Figura 4.12: Fragmento de la clase *WatermeterController.java*

```java
@Configuration
public class AppClientsConfig {

    // ...

    @Value("${IOTSENS_MICROSERVICE_MEASURES_URL}")
    private String measuresMicroserviceBaseURL;

    @Value("${IOTSENS_API_USER}")
    private String iotsensApiUser;

    @Value("${IOTSENS_API_APP}")
    private String iotsensApiApp;

    @Bean
    public OkHttpClient okHttpClient() {
        return new OkHttpClient.Builder()
                .connectTimeout(30, TimeUnit.SECONDS)
                .writeTimeout(180, TimeUnit.SECONDS)
                .readTimeout(180, TimeUnit.SECONDS)
                .build();
    }

    @Bean
    public MeasuresApiClient measuresApiClient() {
        MeasuresApiClientFactory factory = new MeasuresApiClientFactory();
        return factory.getMeasuresApiClient(measuresMicroserviceBaseURL, okHttpClient());
    }

    @Bean
    public Credentials measuresCredentials() {
        return new Credentials(iotsensApiUser, iotsensApiApp);
    }

    // ...
```

Figura 4.13: Fragmento de la clase *AppiClientsConfig.java*

4.1.3. Desarrollo de la implementación

Entre los problemas que surgieron durante la implementación del *frontend*, el más destacable fue el paso de información entre componentes. Esto se debió a que, en algunos casos, se debía estimar si valía la pena crear una dependencia entre componente padre e hijo o hacer varias peticiones al *backend*. En el caso de los componentes que mostraban las alarmas y las medidas de un contador, se optó por obtener los datos necesarios desde el componente padre, el cual contenía la información básica de dicho contador. Se llegó a esa conclusión por el hecho de estar fuertemente relacionados entre sí, ya que todos ellos formaban parte de una única vista en la interfaz de usuario. También se tomó la decisión de no calcular el consumo promedio total de todos los contadores del cliente al no ser un dato útil que consultar, a diferencia de lo que se mostraba en el prototipo inicial (véase figura A.6 del anexo A.1).

Durante el desarrollo del *backend* se optó por hacer uso de las expresiones lambda para simplificar la transformación de datos. Esto estuvo ligado a la decisión de hacer las llamadas a los microservicios directamente en los propios servicios del *backend*, sin necesidad de utilizar ninguna clase intermediaria encargada de gestionarlas. Esto se debió a una cuestión de simplicidad, ya que sólo se requería llamar a los métodos del microservicio instanciado y construir el resultado obtenido mediante una expresión lambda sencilla. La figura 4.14 representa la implementación de la clase *AlarmService*.

```java
@Service
public class AlarmService {

    private EventsApiClient eventsApiClient;
    private Credentials credentials;

    public AlarmService(EventsApiClient eventsApiClient, Credentials credentials) {
        this.credentials = credentials;
        this.eventsApiClient = eventsApiClient;
    }

    public List<Alarm> getAlarmsFromMeter(
        String meterId, Boolean isActive, Date dateFrom, Date dateUntil)
        throws AuthorizationException {
        EventsRequestBuilder eventsRequest = EventsRequestBuilder
                .anEventsRequest()
                .withSensorId(meterId)
                .withEventType(EventType.ALARM)
                .withUntilDate(dateUntil);

                if(dateFrom != null) {
                    eventsRequest.withFromDate(dateFrom);
                }

                if(isActive != null) {
                    eventsRequest.withIsActive(isActive);
                }

        return eventsApiClient
                .searchEvents(credentials, eventsRequest.build())
                .getCurrentResults()
                .stream()
                .map(this::alarmEventDTOToAlarm)
                .collect(toList());
    }

    private Alarm alarmEventDTOToAlarm(EventDTO e) {
        return new Alarm(e.getSensorId(), e.getTimestamp(),
            e.getName(), e.getActive(), e.getDeactivationTime());
    }
}
```

Figura 4.14: Clase *AlarmService.java*

4.2. Resultados de la implementación

A continuación, se muestran los resultados de la implementación realizada durante la estancia de prácticas. Esto engloba tanto las funcionalidades implementadas de la aplicación

como la apariencia final de cada componente, los cuales están presentados desde la figura 4.15 hasta la figura 4.21. Cabe mencionar que todas las vistas y operaciones mostradas hacen referencia a las del cliente, puesto que las del administrador las desarrolló otro miembro del equipo. En la sección 2.7 se explica con más detalle la planificación y la división de tareas.

4.2.1. Vista del *login*

La figura 4.15 presenta la vista del *login* de la aplicación. Comparado con el resto de las vistas, en ella no se muestra la barra de navegación, ya que necesitas acceder a la aplicación primero.

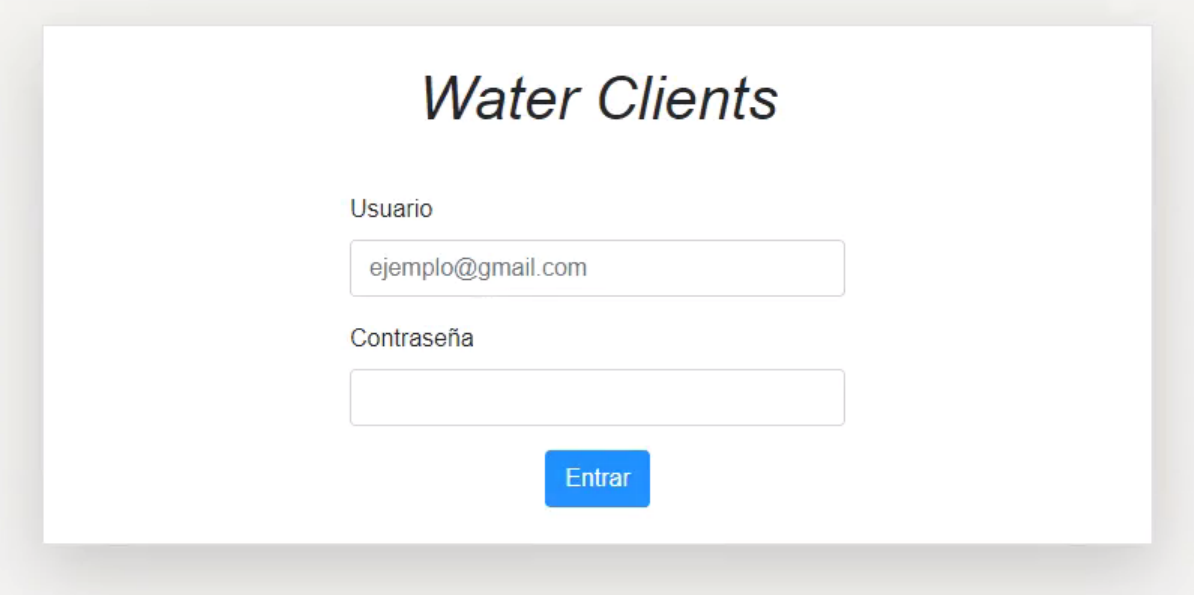

Figura 4.15: Vista del *login*

4.2.2. Vista del listado de contadores

La figura 4.16 muestra un fragmento del listado de contadores que componen la vista. Como se puede apreciar, tanto en esta vista como en las siguientes se observa la barra de navegación. Esto se debe a la jerarquía establecida en los ficheros de configuración del *frontend* para los componentes que conforman las vistas.

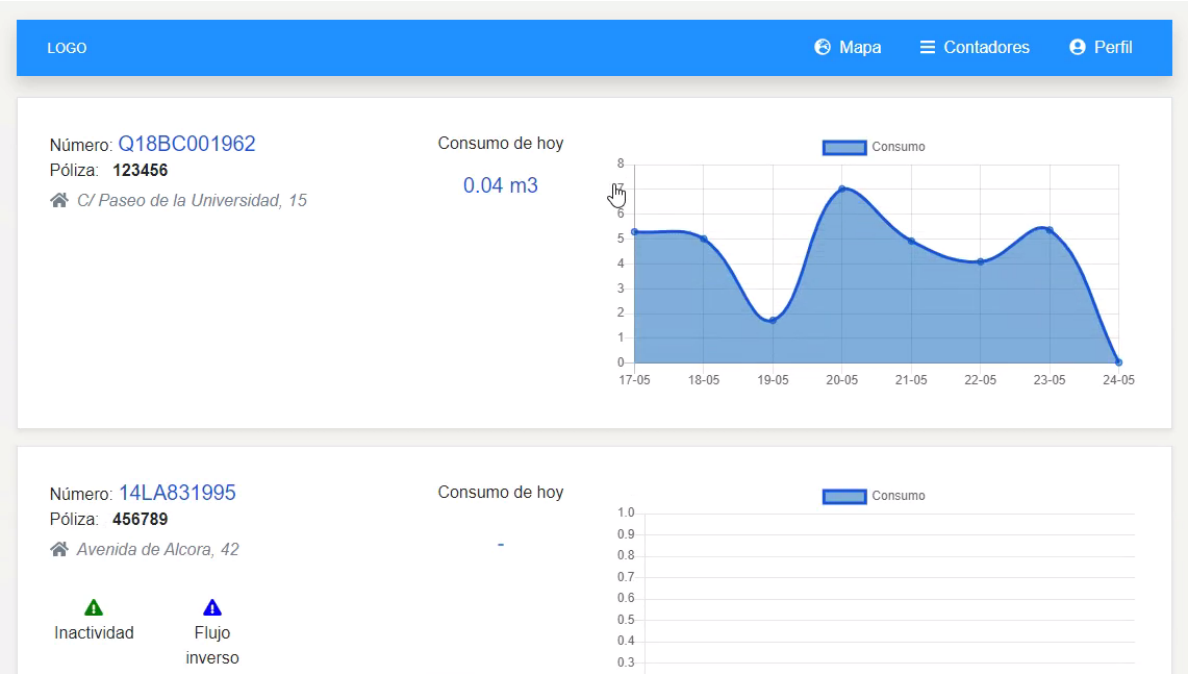

Figura 4.16: Vista del listado de contadores del usuario

4.2.3. Vista del contador

La vista completa del contador lo componen una serie de figuras divididas en los diferentes componentes que la conforman. Las figuras 4.17 , 4.18 y 4.19 muestran de manera ordenada la vista completa.

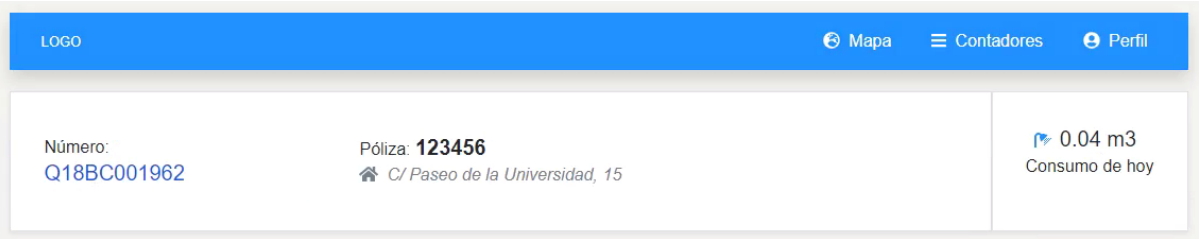

Figura 4.17: Componente con los datos básicos del contador (Vista del contador)

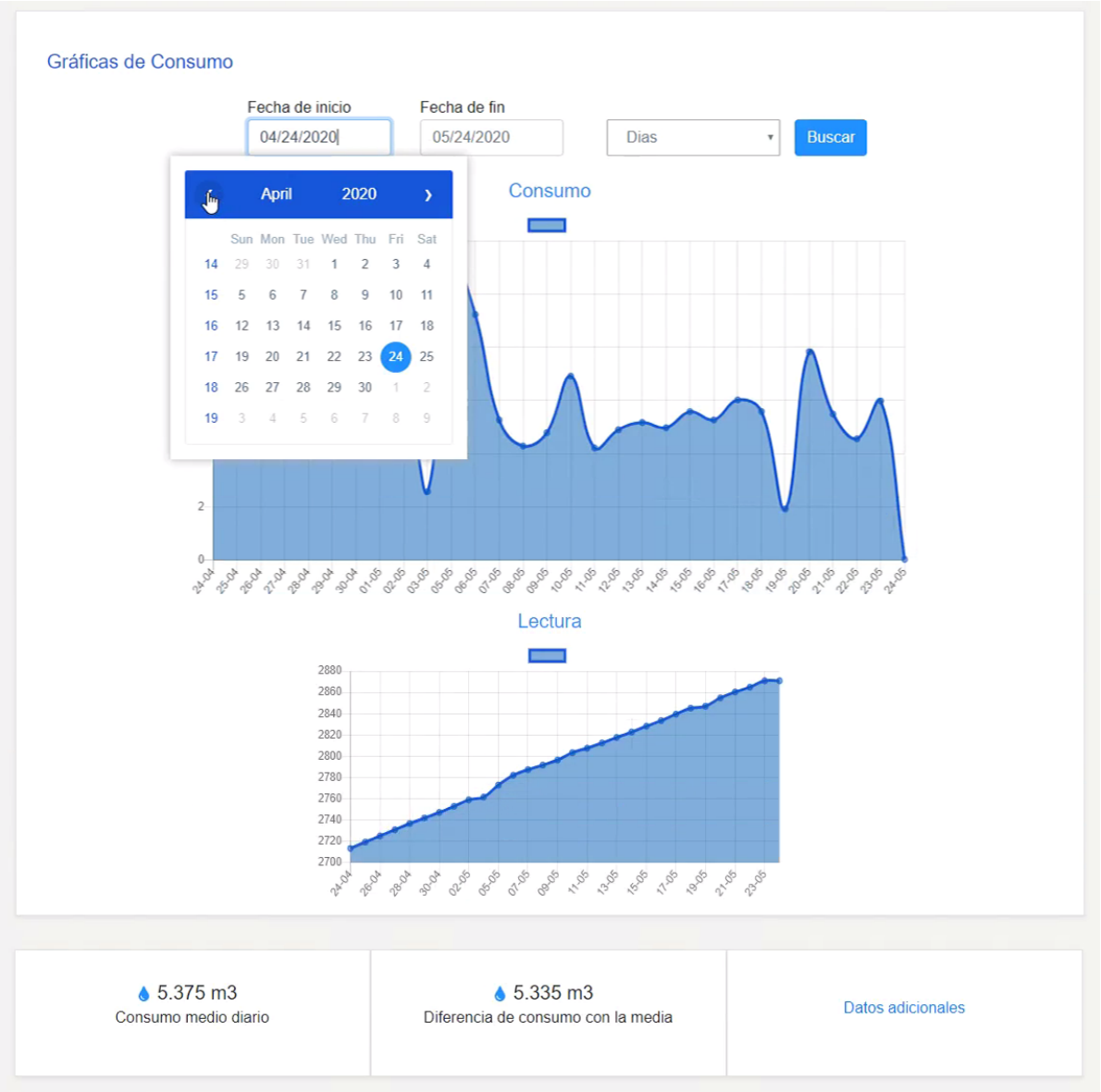

Figura 4.18: Componente con las lecturas del contador (Vista del contador)

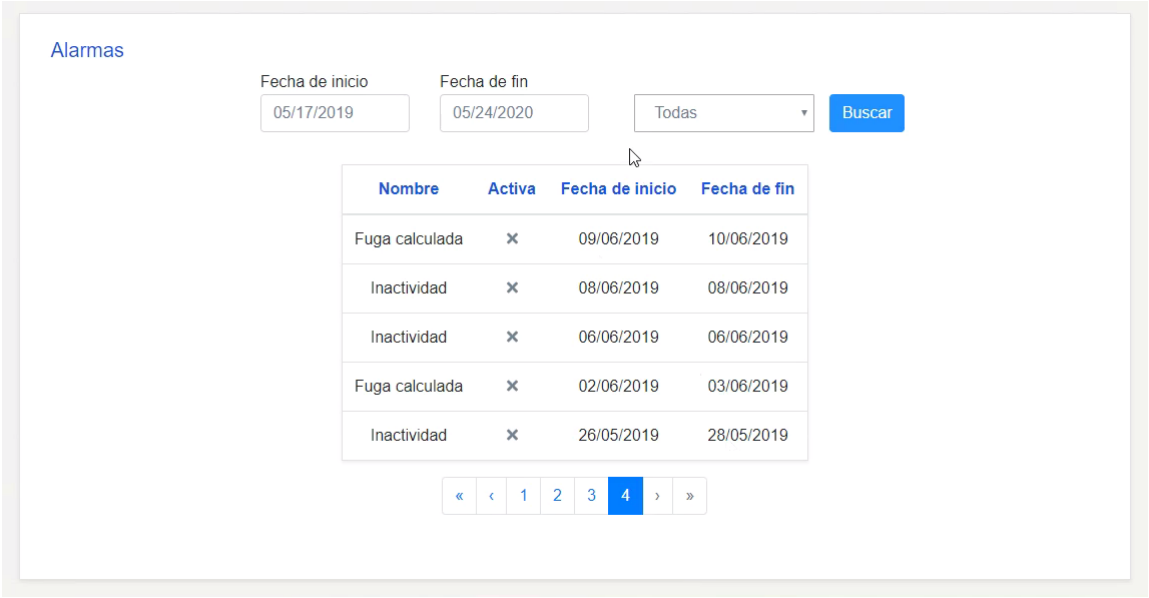

Figura 4.19: Componente con las alarmas del contador (Vista del contador)

4.2.4. Vista del mapa

La figura 4.20 muestra la vista del mapa que contiene las ubicaciones de los contadores de un usuario, los cuales están representados con marcadores que muestran la información básica del contador cuando se seleccionan.

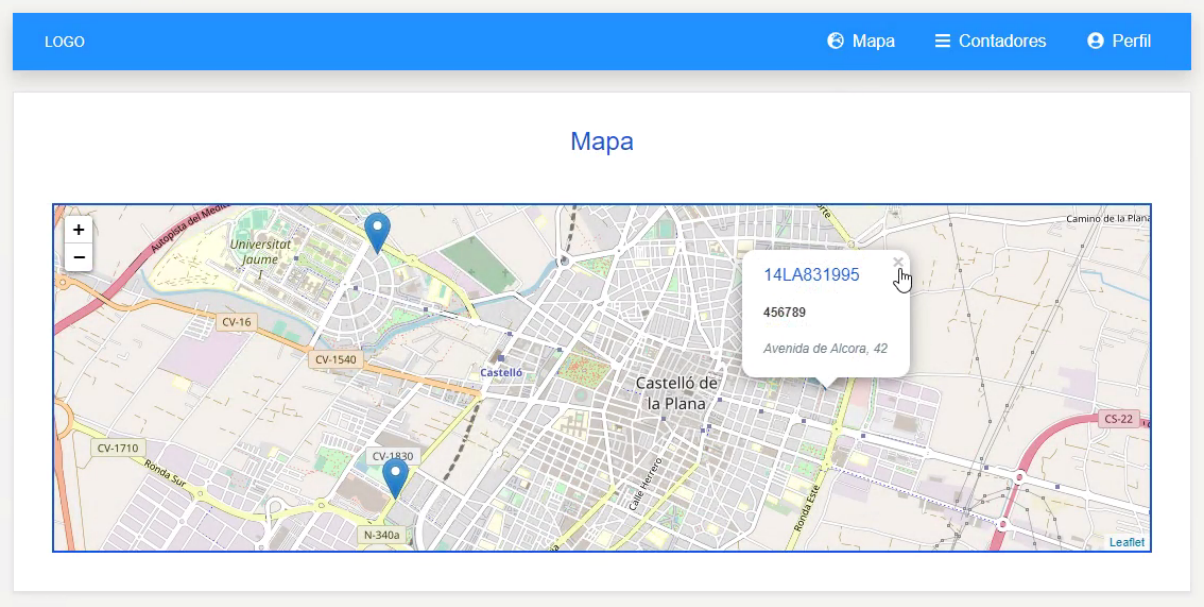

Figura 4.20: Vista del mapa con los marcadores de los contadores del usuario

4.2.5. Vista de la información adicional

Como se puede observar en la figura 4.21, la vista que muestra la información adicional de un contador es un formulario. Esto es debido a la comodidad de poder modificar en cualquier momento los datos, lo que también proporciona una libertad y flexibilidad al usuario para poder cancelar la operación en cualquier momento. Por otro lado, esta vista al igual que la del mapa, está conformada por un único componente.

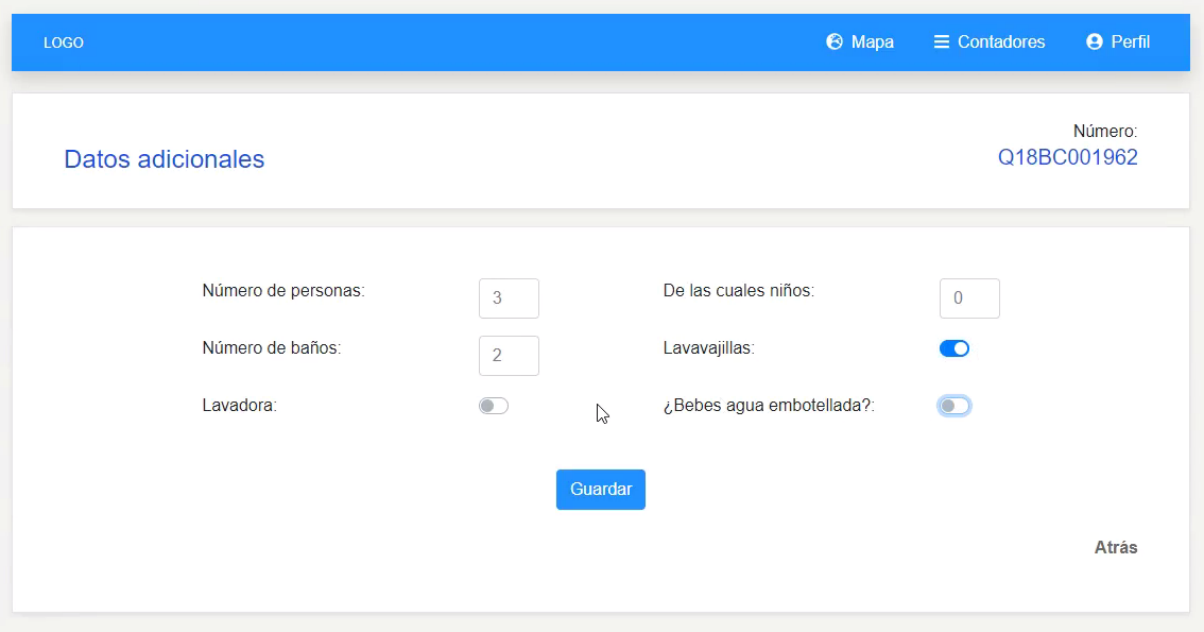

Figura 4.21: Vista del formulario con los datos adicionales de un contador

Capítulo 5

Verificación y validación

5.1. Pruebas de aceptación

Una parte fundamental en el desarrollo de un proyecto informático de calidad es el *testing*. Éste nos permite comprobar que la aplicación realice correctamente las funciones para las que ha sido desarrollada y que contenga la mínima cantidad de *bugs* o fallos. Normalmente, las empresas suelen contar con un departamento de QA (*Quality Assurance*) y/o *testers* que ayudan a cumplir este propósito.

Existen muchos tipos de pruebas con el fin de garantizar la calidad del producto *software*. Para este proyecto se definieron una serie de pruebas de aceptación, compuestas cada una por un escenario válido e inválido, creadas a partir de las historias de usuario contenidas en la pila del producto (véase tabla 2.1). Este tipo de pruebas van dirigidas tanto a usuarios como desarrolladores, para poder tener una visión general del comportamiento de la aplicación. Las tablas de la 5.1 a la 5.7 muestran las pruebas de aceptación creadas para probar las funcionalidades desarrolladas durante la estancia en prácticas.

Prueba de aceptación PA01	
Historia de usuario	Como usuario quiero loguearme en la plataforma para poder consultar la información de mis contadores de agua y mi consumo.

Prueba de aceptación PA01	
Escenario válido	El usuario tiene conexión con la base de datos. Given: ▪ Email: *cliente123@gmail.com* ▪ Contraseña: *password321* When: 1. El usuario solicita acceder a la aplicación. 2. La aplicación hace una petición a la base de datos del cliente con el email *cliente123@gmail.com*. 3. El sistema verifica la contraseña. Then: ▪ Salida esperada: se muestra la página principal de la aplicación. ▪ Estado esperado: se crea y almacena el token del usuario.
Escenario inválido	El usuario tiene conexión con la base de datos. Given: ▪ Email: *cliente123@gmail.com* ▪ Contraseña: *pkyefgeyb* When: 1. El usuario solicita acceder a la aplicación. 2. La aplicación hace una petición a la base de datos del cliente con el email *cliente123@gmail.com*. 3. El sistema invalida la contraseña. Then: ▪ Salida esperada: se muestra la página principal de la aplicación. ▪ Estado esperado: se crea y almacena el token del usuario.

Tabla 5.1: Prueba de aceptación PA01.

Prueba de aceptación PA02	
Historia de usuario	Como usuario quiero ver mi lista de contadores para saber sus datos.

Prueba de aceptación PA02	
Escenario válido	El usuario está logueado, tiene conexión con la base de datos y los microservicios que gestionan las alarmas y las lecturas. Given: ▪ Token del usuario: *7163jwk452* ▪ Id del usuario: *1533* When: 1. El usuario solicita ver el listado de contadores. 2. El sistema verifica el token del usuario. 3. La aplicación hace una petición a la base de datos. 4. La aplicación hace una petición al microservicio que gestiona las alarmas. 5. La aplicación hace una petición al microservicio que gestiona las lecturas. Then: ▪ Salida esperada: se muestra un listado con los datos de cada contador junto con el consumo del día, una pequeña gráfica y los iconos de las alarmas activas. ▪ Estado esperado: no se almacena ninguna información.
Escenario inválido	El usuario está logueado, tiene conexión con la base de datos pero no con los microservicios. Given: ▪ Token del usuario: *7163jwk452* ▪ Id del usuario: *1533* When: 1. El usuario solicita ver el listado de contadores. 2. El sistema verifica el token del usuario. 3. La aplicación hace una petición a la base de datos. 4. La aplicación hace una petición al microservicio que gestiona las alarmas. 5. La aplicación hace una petición al microservicio que gestiona las lecturas. Then: ▪ Salida esperada: se muestra un listado con los datos básicos de cada contador, no se muestran las alarmas y se indica con un guión que no se ha podido obtener la lectura. ▪ Estado esperado: no se almacena ninguna información.

Tabla 5.2: Prueba de aceptación PA02.

Prueba de aceptación PA03	
Historia de usuario	Como usuario quiero consultar la información de un contador para saber los datos del contador.
Escenario válido	El usuario está logueado y tiene conexión con la base de datos. Given: ▪ Token del usuario: *7163jwk452* ▪ Id del usuario: *1533* ▪ Id del contador: *Q7629UY760* When: 1. El usuario solicita ver la información de un contador. 2. El sistema verifica el token del usuario. 3. La aplicación hace una petición a la base de datos. Then: ▪ Salida esperada: confirmación de la petición, se muestra por pantalla el identificador, la póliza y la dirección del contador *Q7629UY760*. ▪ Estado esperado: no se almacena ninguna información en una base de datos.
Escenario inválido	El usuario está logueado pero no tiene conexión con la base de datos. Given: ▪ Token del usuario: *7163jwk452* ▪ Id del usuario: *1533* ▪ Id del contador: *Q7629UY760* When: 1. El usuario solicita ver la información de un contador. 2. El sistema verifica el token del usuario. 3. La aplicación intenta conectarse a la base de datos sin éxito. Then: ▪ Salida esperada: se muestra por pantalla el mensaje de error "En estos momentos no es posible mostrar la información, por favor inténtelo más tarde". ▪ Estado esperado: no se guarda ningún dato.

Tabla 5.3: Prueba de aceptación PA03.

Prueba de aceptación PA04	
Historia de usuario	Como usuario quiero elegir el período de tiempo para visualizar de manera gráfica el gasto de agua que indica un contador.
Escenario válido	El usuario está logueado y tiene conexión con la base de datos y al microservicio que gestiona las lecturas. Given: ▪ Token del usuario: *7163jwk452* ▪ Id del usuario: *1533* ▪ Id del contador: *Q7629UY760* ▪ Fecha inicio: *02/03/2020* ▪ Fecha fin: *24/05/2020* ▪ Intervalo: *semanas* When: 1. El usuario solicita saber el consumo realizado entre el día *02/03/2020* y *24/05/2020*, en intervalos de semanas. 2. El sistema verifica el período de tiempo dado. 3. El sistema verifica el token del usuario. 4. La aplicación hace una petición a la base de datos. 5. La aplicación hace una petición al microservicio que gestiona las lecturas. Then: ▪ Salida esperada: confirmación de la petición, se actualiza la gráfica mostrada por pantalla con los datos nuevos. ▪ Estado esperado: no se almacena ninguna información en una base de datos.

Prueba de aceptación PA04	
Escenario inválido	El usuario está logueado, tiene conexión con la base de datos y al microservicio que gestiona las lecturas. Given: ▪ Token del usuario: *7163jwk452* ▪ Id del usuario: *1533* ▪ Id del contador: *Q7629UY760* ▪ Fecha inicio: *09/12/2021* ▪ Fecha fin: *15/07/2019* ▪ Intervalo: *semanas* When: 1. El usuario solicita saber el consumo realizado del período incorrecto entre el *09/12/2021* y *15/07/2019*, en intervalos de semanas. 2. El sistema invalida el periodo de tiempo dado. Then: ▪ Salida esperada: se indica por pantalla el mensaje de error "Período de tiempo inválido". ▪ Estado esperado: no se actualizan los datos.

Tabla 5.4: Prueba de aceptación PA04.

Prueba de aceptación PA05	
Historia de usuario	Como usuario quiero visualizar las alarmas programadas de un contador en un período de tiempo para saber cuáles tengo activas y cuáles no.
Escenario válido	El usuario está logueado, tiene conexión con la base de datos y al microservicio que gestiona las alarmas. Given: ▪ Token del usuario: *7163jwk452* ▪ Id del usuario: *1533* ▪ Id del contador: *Q7629UY760* ▪ Fecha inicio: *02/03/2020* ▪ Fecha fin: *24/05/2020* ▪ Alarmas: *todas* When: 1. El usuario solicita saber todas las alarmas (activas e inactivas) de un contador entre el día *02/03/2020* y *24/05/2020*. 2. El sistema verifica el período de tiempo dado. 3. El sistema verifica el token del usuario. 4. La aplicación hace una petición a la base de datos. 5. La aplicación hace una petición al microservicio que gestiona las alarmas. Then: ▪ Salida esperada: confirmación de la petición, se actualiza la tabla mostrada por pantalla con los datos nuevos. ▪ Estado esperado: no se almacena ninguna información en una base de datos.

Prueba de aceptación PA05	
Escenario inválido	El usuario está logueado, tiene conexión con la base de datos pero no al microservicio que gestiona las alarmas. Given: ▪ Token del usuario: *7163jwk452* ▪ Id del usuario: *1533* ▪ Id del contador: *Q7629UY760* ▪ Fecha inicio: *02/03/2020* ▪ Fecha fin: *24/05/2020* ▪ Alarmas: *activas* When: 1. El usuario solicita saber las alarmas activas de un contador entre el día *02/03/2020* y *24/05/2020*. 2. El sistema verifica el período de tiempo dado. 3. El sistema verifica el token del usuario. 4. La aplicación hace una petición a la base de datos. 5. La aplicación intenta conectarse al microservicio que gestiona las alarmas sin éxito. Then: ▪ Salida esperada: se indica por pantalla el mensaje de error "No se han podido mostrar las alarmas del contador". ▪ Estado esperado: no se actualizan los datos.

Tabla 5.5: Prueba de aceptación PA05.

Prueba de aceptación PA06	
Historia de usuario	Como usuario quiero rellenar un formulario con información adicional de un contador para poder especificar mejor las condiciones que se deberían de tener en cuenta a la hora de mostrar datos estadísticos y predictivos de mi consumo de agua.

Prueba de aceptación PA06	
Escenario válido	El usuario está logueado y tiene conexión con la base de datos. Given: ▪ Token del usuario: *7163jwk452* ▪ Id del usuario: *1533* ▪ Id del contador: *Q7629UY760* ▪ Cantidad de personas: *4* ▪ Cantidad de baños: *2* ▪ Lavavajillas: *no* When: 1. El usuario solicita actualizar los datos adicionales relativos al contador *Q7629UY760*. 2. El sistema confirma que no hay errores en el formulario. 3. El sistema verifica el token del usuario. 4. La aplicación intenta conectarse a la base de datos sin éxito. Then: ▪ Salida esperada: confirmación de la petición, se actualizan los datos mostrados. ▪ Estado esperado: almacena la información en la base de datos y se actualizan los datos anteriores.
Escenario inválido	El usuario está logueado y tiene conexión con la base de datos. Given: ▪ Token del usuario: *7163jwk452* ▪ Id del usuario: *1533* ▪ Id del contador: *Q7629UY760* ▪ Cantidad de personas: *-56* ▪ Cantidad de baños: *sthsd* ▪ Lavavajillas: *no* When: 1. El usuario solicita actualizar los datos adicionales incorrectos relativos al contador *Q7629UY760*. 2. El sistema indica que hay errores en el formulario. Then: ▪ Salida esperada: se indica por pantalla que los campos de personas y baños son inválidos. ▪ Estado esperado: no se almacena ninguna información en la base de datos y no se actualizan los datos anteriores.

Tabla 5.6: Prueba de aceptación PA06.

Prueba de aceptación PA07	
Historia de usuario	Como usuario quiero consultar en un mapa la posición de mis contadores para así poder visualizar su ubicación geográfica.
Escenario válido	El usuario está logueado y tiene conexión con la base de datos. Given: ▪ Token del usuario: *7163jwk452* ▪ Id del usuario: *1533* When: 1. El usuario solicita ver la ubicación de sus contadores. 2. El sistema verifica el token del usuario. 3. La aplicación hace una petición a la base de datos. Then: ▪ Salida esperada: confirmación de la petición, se muestra por pantalla la ubicación del contador *Q7629UY760*. ▪ Estado esperado: no se almacena ninguna información en una base de datos.
Escenario inválido	El usuario está logueado pero no tiene conexión con la base de datos. Given: ▪ Token del usuario: *7163jwk452* ▪ Id del usuario: *1533* When: 1. El usuario solicita ver la ubicación de sus contadores. 2. El sistema verifica el token del usuario. 3. La aplicación intenta conectarse a la base de datos sin éxito. Then: ▪ Salida esperada: se muestra por pantalla el mapa sin ningún marcador y el mensaje de error "En estos momentos no es posible mostrar la ubicación de los sensores, por favor inténtelo más tarde". ▪ Estado esperado: no se almacena ninguna información en una base de datos.

Tabla 5.7: Prueba de aceptación PA07.

5.2. Pruebas unitarias y de integración

A nivel de *backend*, un miembro del grupo se encargó de realizar tanto pruebas unitarias como de integración. Con respecto a las primeras, se implementaron test unitarios para comprobar el correcto funcionamiento de métodos y clases de manera aislada. Puesto que algunos de estos métodos y clases tenían dependencias externas, como la base de datos y los microservicios, se usó un *mock*[1] llamado *Mokito*, que imitaba el comportamiento de estas dependencias.

Una vez hechas estas primeras comprobaciones, se procedió a analizar la interacción entre el software y su entorno real, es decir, se examinó su funcionamiento con la base de datos y los microservicios. Esta tarea la llevó a cabo el mánager del proyecto, encargado de supervisar que todas las funcionalidades implementadas por cada miembro del equipo quedaran correctamente integradas en la aplicación.

Con respecto al *frontend*, tanto Google como otros navegadores web disponen de herramientas para los desarrolladores. Con la finalidad de verificar el envío de datos y las consultas *HTTP* al *backend*, se utilizó el inspector de *Google Chorme* y la aplicación *Advanced REST client*. Esta extensión, pese a ser más simple que la aplicación de *Postman*, permitía hacer comprobaciones rápidas de las consultas al *backend* e informar de los errores producidos o de si se había realizado con éxito la operación. La figura 5.1 muestra el error 404 al proporcionarle una id incorrecta de un contador para obtener sus lecturas.

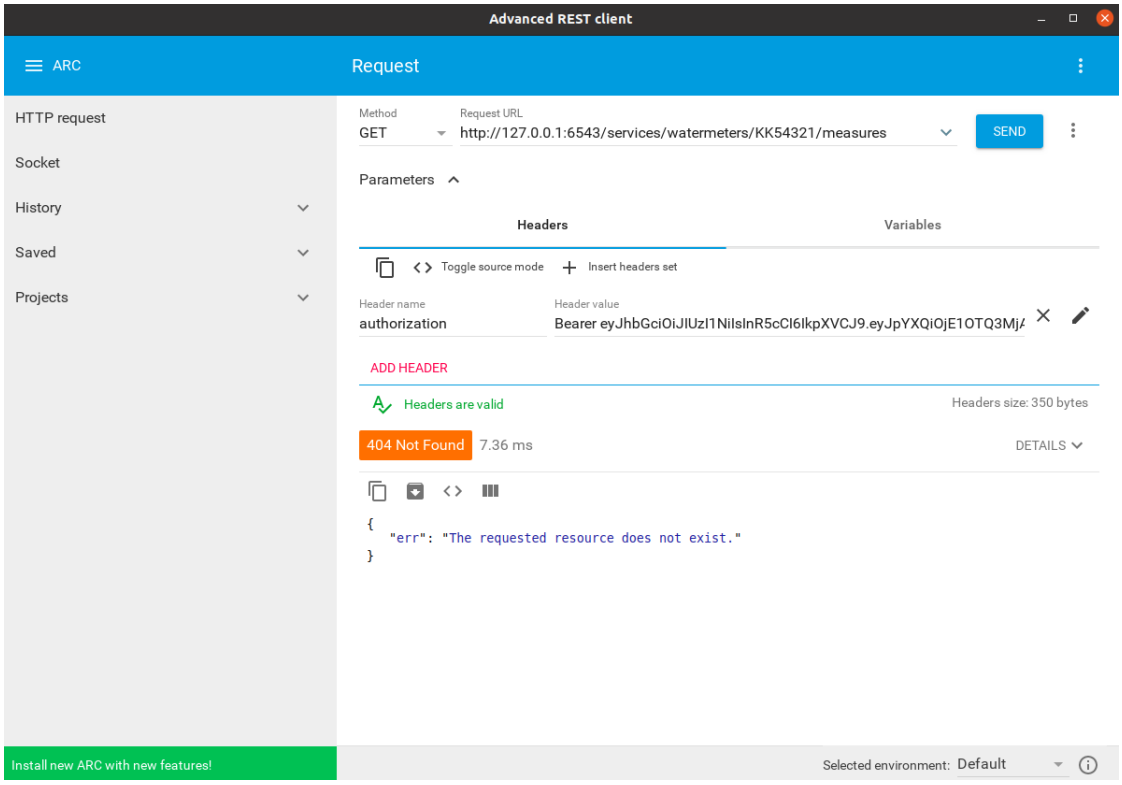

Figura 5.1: Ejemplo de petición fallida usando la aplicación *Advanced REST client*

[1]Son objetos preprogramados que simulan el comportamiento de objetos reales y se usan para probar y verificar el correcto funcionamiento de las llamadas a otros métodos [10].

Capítulo 6

Conclusiones

La realización de este proyecto ha sido una experiencia enriquecedora en varios aspectos. En primer lugar, he adquirido conocimientos de tecnologías con las que todavía no había trabajado, como el *framework Angular* y el lenguaje de programación *Typescript*.

También he tenido la oportunidad de vivir de primera mano el proceso que sigue la empresa IoTsens desde que genera la propuesta técnica de un producto *software* para un cliente, hasta que se empieza a implementar.

En lo que respecta a la experiencia laboral, trabajar y coordinarse con un equipo profesional en circunstancias adversas ha mejorado, sin duda alguna, mis capacidades de adaptación a situaciones inesperadas. Esto es especialmente importante por el hecho de haber realizado la mitad de mi estancia de prácticas de remotamente, ya que la pandemia del COVID-19 obligó a tener que permanecer en casa.

Por otro lado, me hubiera gustado tener la oportunidad de aplicar los conocimientos de *machine learning*, adquiridos durante la última semana de prácticas, a la aplicación. Desgraciadamente, esto no fue posible debido a que faltaban funcionalidades que implementar de mayor prioridad. Además, la parte de *machine learning* se consideró en todo momento una mejora del producto original. No obstante, el resultado final obtenido ha sido bueno y se han completado los objetivos esperados durante el período de estancia en prácticas.

Finalmente quería destacar que, en general, he aprendido mucho de esta experiencia y me ha ayudado a conseguir recientemente un puesto de trabajo como *IT Engineer* en el departamento de desarrollo de una empresa belga. Mi adaptación e integración en este nuevo entorno de trabajo y equipo ha sido rápida gracias a la experiencia adquirida durante las prácticas, a pesar de utilizar otras tecnologías.

Bibliografía

[1] https://iotsens.com/es/.

[2] https://www.grupogimeno.com/.

[3] https://www.atlassian.com/es/agile/kanban.

[4] https://www.atlassian.com/es/software/jira.

[5] Arquitectura de las aplicaciones webs. https://programacionwebisc.wordpress.com/2-1-arquitectura-de-las-aplicaciones-web/.

[6] Cocomo. https://web.archive.org/web/20171211201210/sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html.

[7] Golden rules of user interface design. https://theomandel.com/resources/golden-rules-of-user-interface-design/.

[8] Modelo cocomo (ingeniería de software). https://es.slideshare.net/yadithmiranda/modelo-cocomo-ingeniera-de-software.

[9] Oracle mysql database service. https://www.oracle.com/mysql/.

[10] Pruebas unitarias: ¿mock o stub? https://itblogsogeti.com/2015/03/26/desarrollo-pruebas-unitarias-trinitario-gomez-sogeti/.

[11] Qué es scrum. https://proyectosagiles.org/que-es-scrum/.

[12] Significado de url. https://www.significados.com/url/.

[13] ¿qué es un framework? https://neoattack.com/neowiki/framework/.

[14] ¿qué es una web responsive? https://www.digival.es/blog/que-es-una-responsive-web/.

[15] Rosselyn Barroyeta. Todo lo que debes saber sobre gitlab, la mejor alternativa a github. https://www.tekcrispy.com/2018/06/04/gitlab-repositorio-github/.

[16] Nacho Blanco. ¿qué patrón usa angular? mcv o mvvm. https://openwebinars.net/blog/que-patron-usa-angular-mvc-o-mvvm/.

[17] Andrés González. ¿qué es machine learning? https://cleverdata.io/que-es-machine-learning-big-data/.

[18] Jose M. Huerta. ¿qué es una épica y qué no es? https://josehuerta.es/gestion/agile/que-es-una-epica-y-que-no-es.

[19] César Krall. ¿qué es y para qué sirve uml? https://www.aprenderaprogramar.com.

[20] Luis Miguel López Magaña. ¿qué es spring framework? https://openwebinars.net/blog/que-es-spring-framework.

[21] Estanislao Echazú / Ramiro Rodríguez. *Primer glosario de comunicación estratégica en español*. fundéu BBVA, 2018.

[22] Juan Carlos Rubio. Qué es git y para qué sirve. https://openwebinars.net/blog/que-es-git-y-para-que-sirve/.

[23] Jorge Saiz. ¿qué son los puntos de historia? https://jorgesaiz.com/blog/puntos-historia/.

[24] Piotr Stefaniak. ¿qué es back end y front end? https://descubrecomunicacion.com/que-es-backend-y-frontend.

[25] Angel Luis Lozano Sánchez. Requisitos vs casos de uso vs historias de usuario. https://www.angellozano.com/requisitos-del-sistema-vs-casos-uso-vs-historias-usuario/.

[26] Dan Woods and Peter Thoeny. *Wikis For Dummies*. John Wiley Sons Ltd, 2007.

Anexo A

Prototipos de la aplicación

A.1. Prototipos de las vistas del cliente

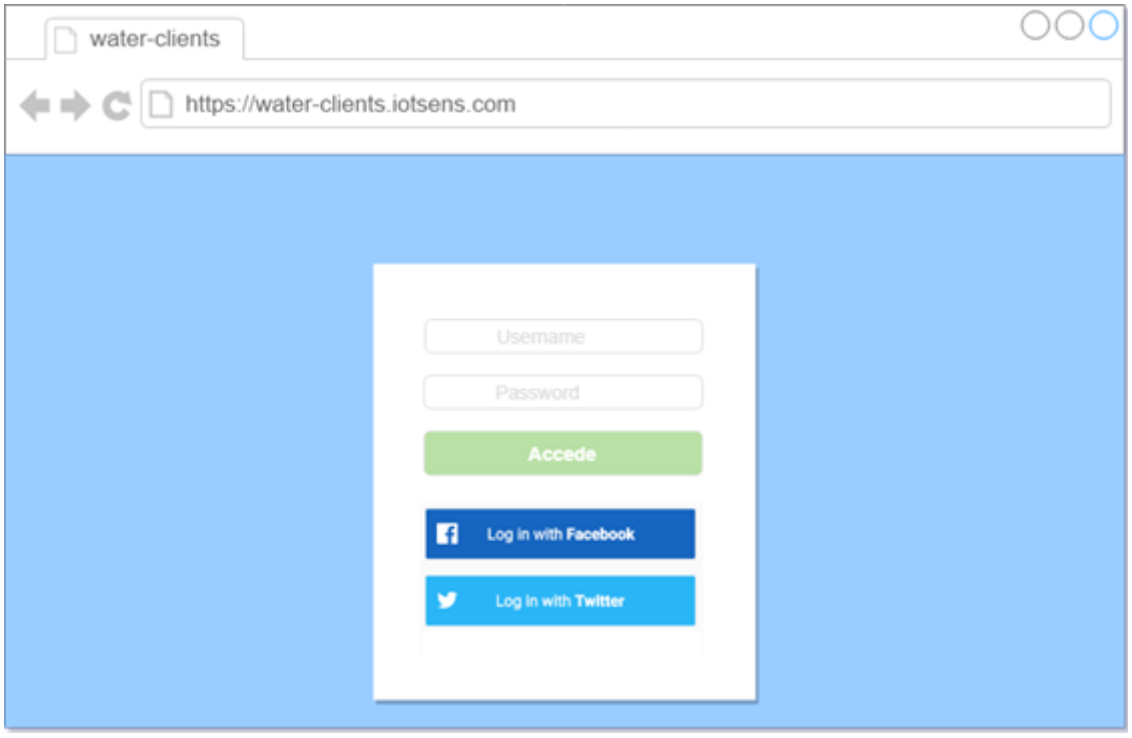

Figura A.1: Prototipo del *login* de la aplicación

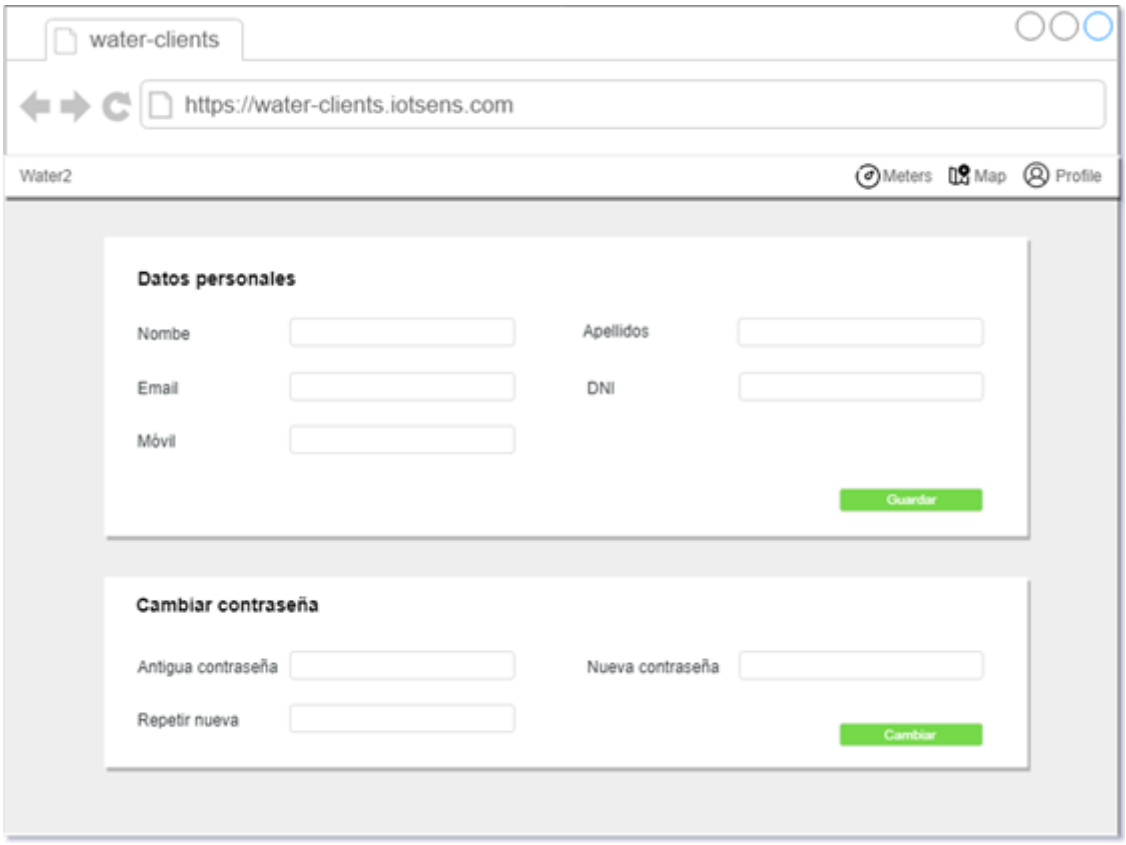

Figura A.2: Prototipo del panel del perfil de usuario

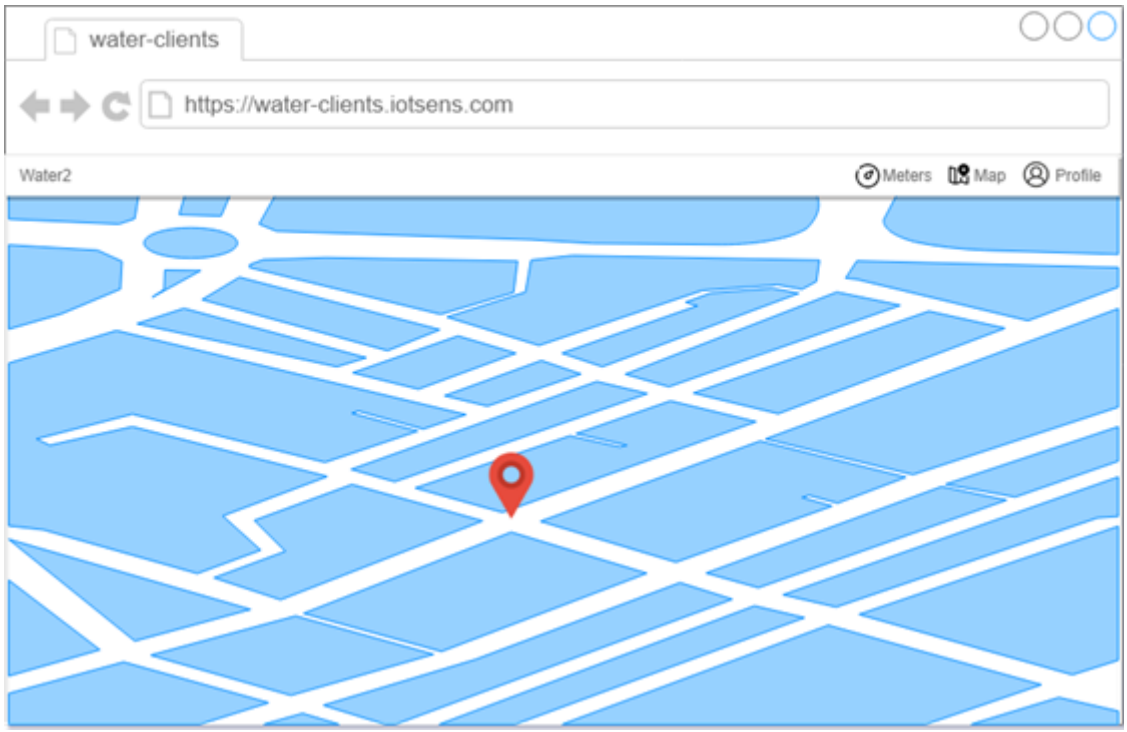

Figura A.3: Prototipo del mapa con la localización de los contadores

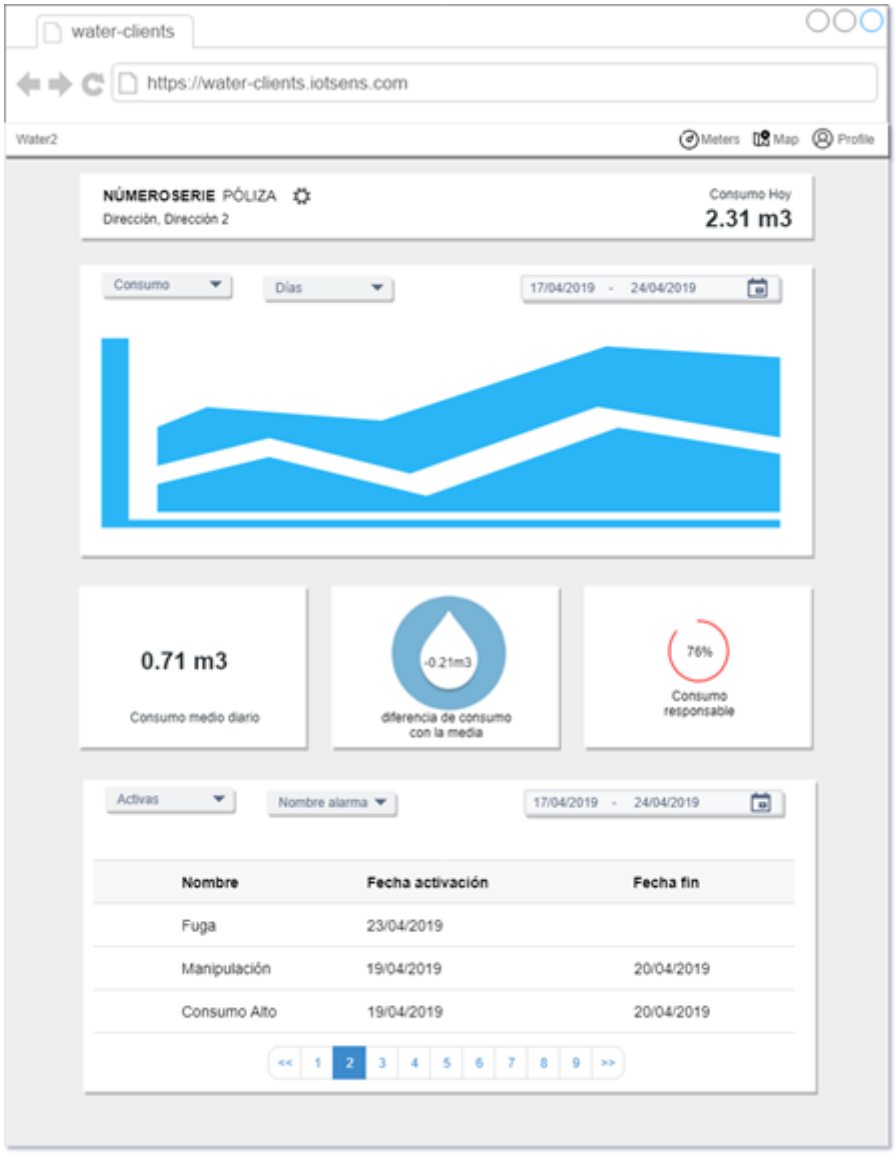

Figura A.4: Prototipo del panel de datos y alarmas del contador

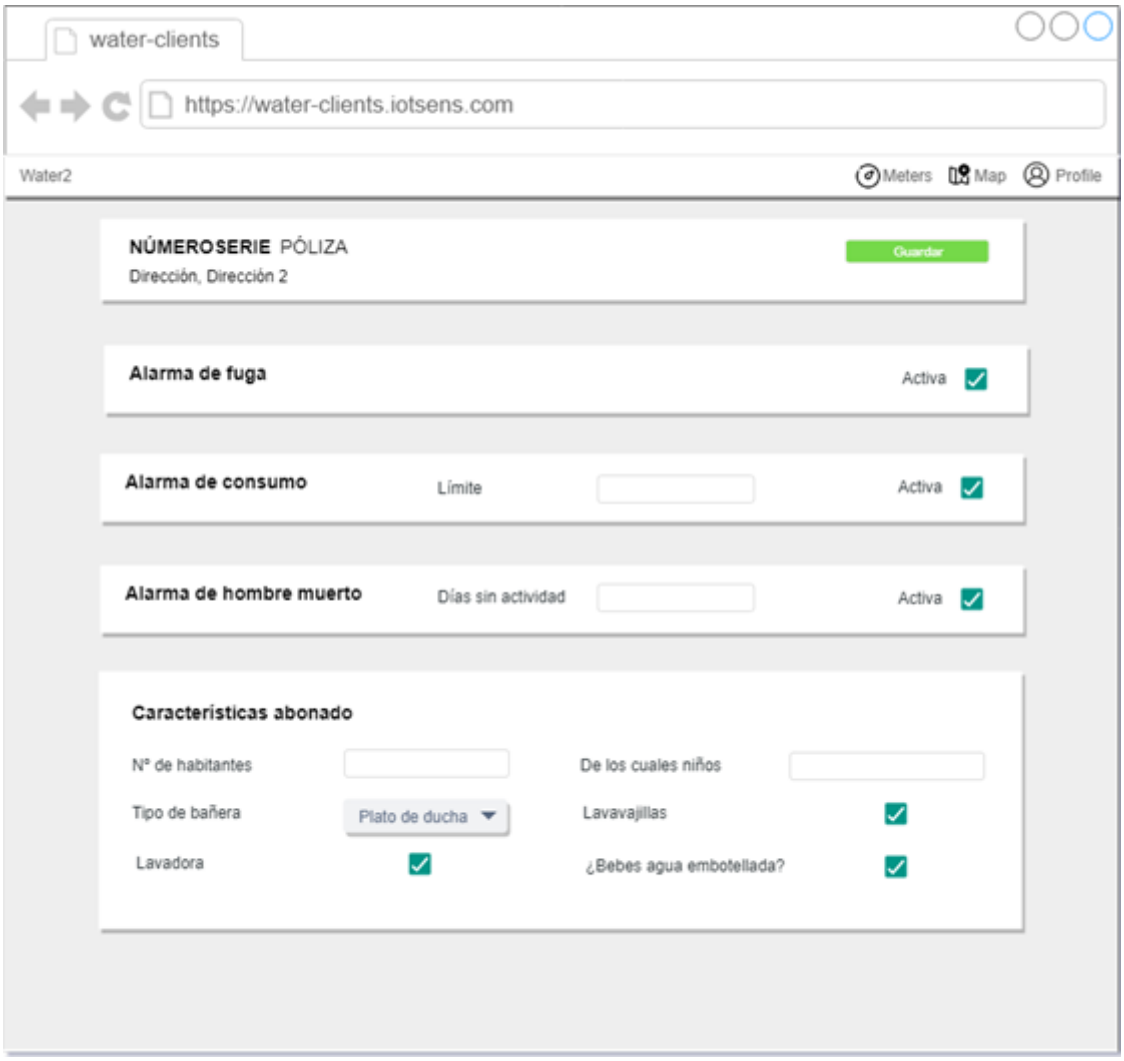

Figura A.5: Prototipo del formulario de datos adicionales del contador

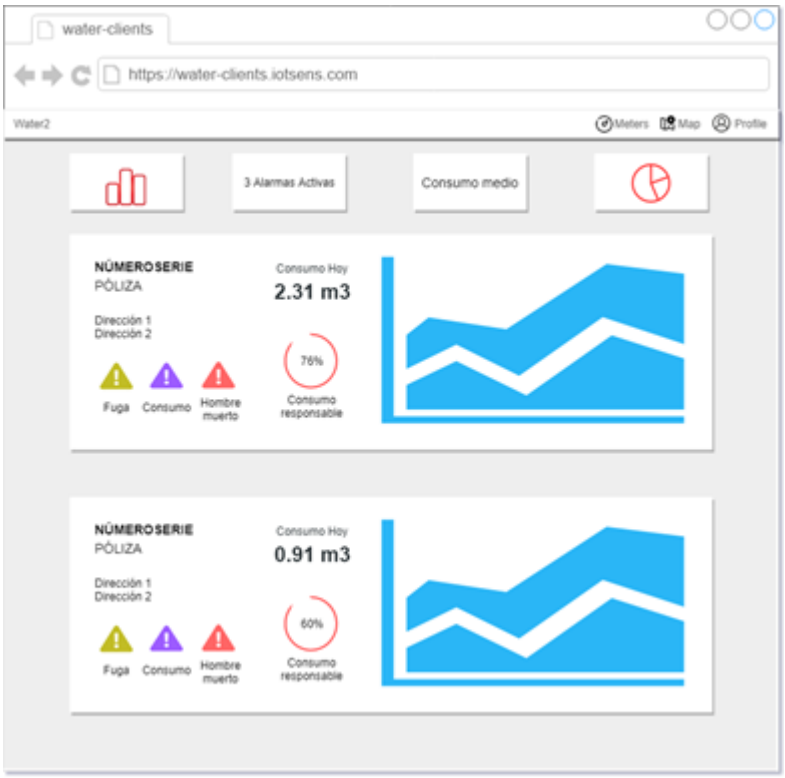

Figura A.6: Prototipo del listado de contadores