

Diseño y desarrollo de un sistema de representación y accesibilidad para OPEN API de Smart University



Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autor:

Jose Daniel Neme Jerez

Tutor/es:

José Vicente Berná Martínez

Lucía Arnau Muñoz



Universitat d'Alacant
Universidad de Alicante

Noviembre 2022

Resumen

El uso de datos abiertos es muy común hoy en día a través de OpenAPIs. El problema es que para hacer uso de una API hay que tener ciertos conocimientos técnicos. En este TFG se plantea el desarrollo de una plataforma que permita conectar APIs y hacer accesibles y entendibles los datos, comenzando a trabajar con la API de Smart University, accesible en:

<https://openapi.smartua.es/doc/>

Esta API permite recuperar datos de la plataforma Smart University, de una colección de datos, pero su visualización no es amigable. La idea es una plataforma donde se puedan visualizar datos a través de QR, de forma que se pueda generar un QR y al escanearlo con un terminal móvil se puedan observar datos de forma accesible y amigable que provengan de esta Open API.

Motivación, justificación y objetivo general

El último curso de mi grado en la universidad fue un mundo totalmente nuevo para mí. Nunca antes había estado en un curso donde la nota final depende del resultado del proyecto el cual te has dedicado a desarrollar durante todo el año académico junto con tus compañeros. Sin embargo, siempre recordaré con cariño este curso ya que ha sido el que más he disfrutado y en el que más he aprendido. Cada vez que avanzábamos en el proyecto y conseguíamos implementar nuevas funcionalidades, me sentía muy satisfecho y realizado con el trabajo que realizaba. Además, he tenido la oportunidad de conocer a profesores muy dedicados con su trabajo, a los que se les nota que les gusta lo que están haciendo. Todos estos factores me hicieron reflexionar sobre mi futuro. Ahora sé que quiero trabajar en un entorno donde me sienta útil, disfrute de lo que hago y en el que pueda aprender continuamente de las personas que encuentre en mi camino.

A lo largo de este curso, también era consciente de que tenía que realizar el TFG tarde o temprano. No obstante, esto me causaba mucho respeto ya que también era algo totalmente nuevo para mí. Al principio del curso, en una reunión donde los profesores explicaron los aspectos claves de esta asignatura, recomendaron que no nos matriculáramos hasta que no tuviéramos claro el tema y la fecha en la que queríamos presentar el proyecto. Dado que no se me ocurría ningún tema y tampoco tenía claro si iba a tener tiempo en realizarlo mientras me dedicaba al proyecto del curso, por lo que al principio decidí esperar.

A pocos días de la presentación final de todos los proyectos grupales, fue cuando mi actual tutor, José Vicente propuso ante toda la clase un proyecto para realizar y presentarlo como TFG. El proyecto captó mi interés debido a sus similitudes con lo que habíamos abordado en el último curso. Además, si se obtenía el resultado deseado, podría llegar a ser una herramienta útil para todos en la universidad. Parecía una señal para comenzar con el TFG justo en ese momento, por lo que decidí dar el paso y ofrecerme para realizarlo. Quise aprovechar la oportunidad para seguir aprendiendo y para demostrarme a mí mismo que soy capaz de llevar a cabo un proyecto de la índole de un TFG.

En definitiva, con este proyecto voy a ponerme a prueba para demostrarme que soy capaz de aplicar todo lo que he aprendido en la carrera. Con la orientación de mi tutor y mi esfuerzo confío en obtener un buen resultado. Este paso adelante me acercará a mi objetivo de alcanzar la constancia y pasión por el trabajo que he observado en los profesores y compañeros del último curso de la universidad.

Agradecimientos

En primer lugar, me gustaría agradecer a mis tutores José Vicente Berná Martínez y Lucia Arnau Muñoz. Me ha encantado trabajar con vosotros, voy a guardar esta experiencia con mucho cariño. Quiero llegar a ser algún día igual de profesional que vosotros.

También quiero agradecer a su equipo, en especial a Sergio Claramunt. Gracias por enseñarme y aconsejarme desde el minuto uno que nos pusieron en contacto.

A Alma, una persona increíble que me ha dado la universidad. Gracias por todos esos momentos que hemos vivido durante estos 4 años. Eres una de esas personas a las que admiro por su constancia y su forma de ser. Espero seguir compartiendo muchas más cosas contigo.

A mi pareja Sara, por estar siempre a mi lado y apoyarme durante todos estos meses. Gracias a ti he podido llegar hasta este momento. Te convertiste en una persona demasiado importante para mí en muy poco tiempo y sé que a tu lado voy a seguir consiguiendo cosas increíbles. Estoy muy orgulloso de ti, por todo lo que haces y por quién eres. Te quiero.

A mi grupo de amigos de toda la vida: Jesús, Marcos, Urrea, Roberto, Mateo y Lander. Sé que siempre puedo contar con vosotros y aunque a lo mejor no lo sepáis, el único hecho de teneros a mi lado me ha ayudado mucho a realizar este proyecto. Sois como una segunda familia para mí y espero poder seguir disfrutando de vuestra amistad para siempre. Haré todo lo que esté en mi mano para que así sea.

A mi familia, por confiar siempre en mí y darme el empujón que necesitaba para empezar el proyecto.

Citas

- ***"When do I Know I'm ready?"***

- ***"You won't. That's all it is. A leap of faith."***

Diálogo de la película: Spider-Man: Into the Spider-Verse

Directores: Peter Ramsey, Bob Persichetti, Rodney Rothman

Índice de contenidos

Resumen.....	1
Motivación, justificación y objetivo general	2
Agradecimientos.....	3
Citas.....	4
Índice de contenidos.....	5
Índice de figuras.....	7
Índice de tablas.....	11
1. Introducción.....	12
2. Planificación.....	13
3. Estado del arte	15
3.1. Primera llamada	15
3.2. Segunda llamada	17
3.3. Tercera llamada.....	20
4. Objetivos.....	22
5. Metodología.....	23
6. Implementación	25
6.1. Iteración 1. Definición del proyecto.	26
6.2. Iteración 2. Mockups del proyecto.	34
6.3. Iteración 3. Preparación del entorno de implementación y Base de Datos.	42
6.4. Iteración 4. Creación de ruta Login y su componente en el frontend.	47
6.5. Iteración 5. Creación de ruta y componente de los códigos QR.	54
6.6. Iteración 6. Creación del componente de configuración de un código QR.	60
6.7. Iteración 7. Creación del componente de configuración de las llamadas.	68
6.8. Iteración 8. Gestión de usuarios y primeras peticiones a Smart University.	73
6.9. Iteración 9. Vista de los códigos QR y multiidioma.	81
6.10. Iteración 10. Descargar códigos QR para su impresión.	98

7.	Entrega y puesta en marcha	113
7.1.	Versión 1.0	113
7.2.	Versión 1.1	122
7.3.	Versión 1.2	125
7.4.	Versión 1.3	130
8.	Pruebas y validación.....	134
8.1.	Caso de uso 1. Crear panel con la información de un sensor	134
8.2.	Caso de uso 2. Modificar el panel.....	138
8.3.	Caso de uso 3. Modificar el orden del panel.....	140
8.4.	Caso de uso 4. Desactivar el panel	141
8.5.	Caso de uso 5. Eliminar panel	142
9.	Resultados	144
9.1.	Producto final	144
9.2.	Coste temporal.....	144
10.	Conclusiones y trabajo futuro	147
10.1.	Objetivos alcanzados	147
10.2.	Trabajo futuro.....	147
	Referencias.....	149
	Apéndice.....	151

Índice de figuras

Figura 1. Página principal de la OpenAPI de Smart University.	15
Figura 2. Lista de los tags para seleccionar.	16
Figura 3. Respuesta de la primera llamada con el campo name seleccionado.	17
Figura 4. Fechas y filtros indicados en el cuerpo de la segunda llamada.	18
Figura 5. Primeros datos obtenidos en la segunda llamada de la OpenAPI.	19
Figura 6. Datos introducidos en la tercera llamada.	20
Figura 7. Dato obtenido en la última llamada de la OpenAPI.	21
Figura 8. Diagrama ilustrando como el énfasis relativo en las distintas disciplinas cambia a lo largo del proyecto.....	25
Figura 9. Primer boceto de la interfaz de gestión de QR propuesto por el product owner.	27
Figura 10. Primer boceto de la interfaz de configuración de una llamada propuesto por el product owner.	27
Figura 11. Wireframe de la página de gestión de QR.	33
Figura 12. Opciones del perfil en la cabecera de la aplicación.	34
Figura 13. Opciones del perfil y gestión en la barra lateral.	35
Figura 14. Mockup de la vista de un código QR en un dispositivo móvil.	36
Figura 15. Interfaz de inicio de sesión.	36
Figura 16. Mockup de la página principal con la barra lateral desplegada.	37
Figura 17. Mockup de la creación de códigos QR.	38
Figura 18. Formulario de llamada con la opción de todos los datos.	39
Figura 19. Formulario de llamada con la opción del máximo dato.	40
Figura 20. Página de edición de un usuario.	40
Figura 21. Caso de intentar salir del formulario sin guardar los cambios.	41
Figura 22. Caso de crear correctamente un código QR.	41
Figura 23. Vista global de las interfaces.	42
Figura 24. Esquema de la Base de Datos.	44
Figura 25. Estructura del backend del proyecto.	45
Figura 26. Archivo db.js encargado de la conexión y hacer peticiones a la base de datos.	46
Figura 27. Prueba de llamada a login desde Postman.	48
Figura 28. Creación del componente login desde la terminal.	49
Figura 29. Componente login implementada en la aplicación.	50

Figura 30. Alerta de error al intentar iniciar sesión.	51
Figura 31. Prueba de llamada a getUsers en Postman.	53
Figura 32. Comprobación de los campos de un código QR en la función UpdateQr.....	56
Figura 33. Estructura de la tabla de los códigos QR.	57
Figura 34. Mensaje modal que muestra el QR con más tamaño generado con SweetAlert2.	58
Figura 35. Interfaz de gestión de códigos QR implementado en la aplicación.	60
Figura 36. Prueba de llamada a getConsult desde Postman.	63
Figura 37. Formulario en modo edición.	64
Figura 38. Estructura de la tabla de las llamadas.	65
Figura 39. Eventos para el temporizador de la barra de búsqueda.....	67
Figura 40. Interfaz de configuración de un código QR.....	67
Figura 41. Select de la representación de los datos con la gráfica de gauge seleccionada.	70
Figura 42. Lista con los filtros de las llamadas.	71
Figura 43. Código donde se realiza el json de los filtros.	72
Figura 44. Interfaz de la configuración de las llamadas.	73
Figura 45. Tabla de gestión de los usuarios de la aplicación.....	76
Figura 46. Ruta de la función getData de Smart University y ejemplo de cuerpo que recibe la llamada.	77
Figura 47. Ejemplo de petición exitosa a la función getData de Smart University.	78
Figura 48. Ruta de la llamada getDataOperation de Smart University.	78
Figura 49. Ejemplo de petición exitosa de la función getDataOperation de Smart University.	79
Figura 50. Código de la paginación.	80
Figura 51. Botones de la paginación de las tablas de gestión de la aplicación.	80
Figura 52. Fechas adaptadas al formato y a la zona horaria del sistema en el que se ejecuta.	81
Figura 53. Select en la cabecera para cambiar el idioma de la aplicación.	83
Figura 54. Variables de entorno con los mensajes configurables.	85
Figura 55. Cálculo de la fecha relativa.	86
Figura 56. Obtención de la información necesaria para representar un máximo, mínimo o último dato.	87
Figura 57. Creación del JSON con los filtros aplicados a la llamada.	88
Figura 58. Obtención de la información necesaria para representar todos los datos disponibles en el rango de fechas proporcionada.	89
Figura 59. Objeto con la información para crear una gráfica de líneas o de barras.	90
Figura 60. Objeto con la información para crear una gráfica Gauge.	92
Figura 61. Objeto con la información para representar el valor de un dato en texto.	93

Figura 62. Interfaz de la vista de un código QR.	94
Figura 63. Tabla de la gestión de las llamadas de un QR con los botones para ordenar.	95
Figura 64. Formulario de cambio de contraseña.....	96
Figura 65. Archivo es.json con la traducción en español.	97
Figura 66. Archivo es.json con la traducción en inglés.	97
Figura 67. Ejemplo de elemento traducido.....	97
Figura 68. Interfaz de gestión de códigos QR en inglés.	98
Figura 69. Previsualización del código QR en el componente de configuración de un QR.	99
Figura 70. Dimensión de los diferentes formatos de papel de la serie A en mm.	100
Figura 71. Mensaje modal con la previsualización del contenido del PDF.	103
Figura 72. Tabla de la gestión de los QR con la columna para añadir QR a la lista y botón para descargarlos.....	104
Figura 73. Botón de descargar la lista de códigos QR indicando que se está generando el PDF. ...	105
Figura 74. PDF de la descarga de un solo QR con el tamaño A5.	106
Figura 75. PDF de la descarga de varios QR.	107
Figura 76. Interfaz de la gestión de los códigos QR con la funcionalidad de descargar códigos QR.	108
Figura 77. Botón de compartir en el componente de vista de los QR.	108
Figura 78. Componente de configuración de un QR al crear uno nuevo.	109
Figura 79. Tabla de gestión de las llamadas de un QR con el botón de duplicar.	110
Figura 80. Componente de configuración de una llamada al duplicar.	110
Figura 81. Formulario de configuración de un QR con la nueva distribución de los botones.	111
Figura 82. Código de la función asíncrona que se encarga de lanzar el mensaje modal.	112
Figura 83. Mensaje de carga de datos en el componente de la vista pública de los códigos QR. .	112
Figura 84. Ejemplo de parametrización de sentencia SQL.	114
Figura 85. Nueva estructura de la API.	115
Figura 86. Función del controlador de los usuarios que controla el rol del usuario que hace la petición.....	116
Figura 87. Validación del email en la ruta de los usuarios.	117
Figura 88. Validador de las contraseñas.	118
Figura 89. Información de la base de datos que se devolvía al crear un usuario.	118
Figura 90. Sentencia SQL que encripta y pasa a hexadecimal los identificadores de los QR.....	119
Figura 91. Nuevo enlace de la vista pública de los códigos QR.	120
Figura 92. Función auxiliar para desencriptar el id de los enlaces.	120
Figura 93. Alerta de error en la contraseña.	120

Figura 94. Gráficas que muestran solo el valor con los cambios (máximo dato).	121
Figura 95. Configuración del QR con la opción de mostrar el botón de compartir.	122
Figura 96. Selectores del color del valor y del fondo de las gráficas.	122
Figura 97. Nuevos ejemplos en el formulario de las llamadas, en este caso es una gráfica Gauge.	123
Figura 98. Nueva disposición de las gráficas. Ejemplo con 1 grande y 3 pequeñas.	124
Figura 99. Nueva disposición de las gráficas. Ejemplo 2 grandes y 1 pequeña en medio.	125
Figura 100. Componente de configuración de un QR al duplicar.	126
Figura 101. Componente de gestión de los códigos QR con el botón de duplicar.	127
Figura 102. Nueva forma de las gráficas que solo muestran un valor con icono.	128
Figura 103. Variable de entorno con la lista de los iconos.	129
Figura 104. Previsualización de una gráfica de solo el valor con el icono.....	129
Figura 105. Previsualización de una gráfica Gauge con el icono.	130
Figura 106. Primeros bocetos para el logo de la aplicación.	131
Figura 107. Logo para la aplicación.....	131
Figura 108. Campo del tiempo para refrescar en el formulario de configuración de un QR.	132
Figura 109. Función que crea el intervalo para refrescar el componente.	133
Figura 110. Código QR creado.	135
Figura 111. Llamada configurada con los datos del ejemplo.	136
Figura 112. Segunda llamada del ejemplo.	137
Figura 113. Resultado del panel configurado en el caso de uso.	138
Figura 114. Llamada actualizada para mostrar las últimas 24 horas.	139
Figura 115. Panel de control modificado.	140
Figura 116. Panel de control con el orden modificado.	141
Figura 117. Código QR desactivado.	142
Figura 118. Vista pública del QR cuando está desactivado.	142
Figura 119. Mensaje modal para confirmar el proceso de eliminación.....	143
Figura 120. Tiempo empleado en el proyecto por mes.	145
Figura 121. Top 10 de las tareas con más horas dedicadas, medidas en Clockify.....	146
Figura 122. Panel de control de XAMPP con Apache y MySQL activos.....	152
Figura 123. Pestaña de importar la base de datos en phpMyAdmin.....	152
Figura 124. Resultado de iniciar la API desde una terminal.	153
Figura 125. Resultado de iniciar el frontend desde una terminal.	153
Figura 126. Sección de los contenedores de Docker con el contenedor de prueba.	154
Figura 127. Grupo de los contenedores de la aplicación.	155

Índice de tablas

Tabla 1. Planificación temporal TFG	13
Tabla 2. Tabla de los requisitos funcionales.....	28
Tabla 3. Requisitos no funcionales.	32

1. Introducción

Hoy en día, donde la presencia de las tecnologías se hace notar en la mayoría de los ámbitos de las personas, los datos abiertos tienen un gran protagonismo. Los datos abiertos son los aquellos datos accesibles y reutilizables que están disponibles de forma libre para el uso de todo el mundo sin restricciones. No obstante, el uso de los datos abiertos puede ser complejo para los que no tienen familiaridad con ellos. Son los ingenieros los que se encargan de aprovechar esta filosofía de datos para acercar la información a la ciudadanía. Un ejemplo cotidiano se observa en los supermercados, donde en las entradas hay pantallas en las que indican la temperatura del recinto, el aforo de personas que hay en ese momento y más información que puede ser útil para los clientes.

Es en este contexto donde entra en juego Smart University. Smart University es una plataforma cuyo objetivo es ayudar a la comunidad universitaria brindándole todo tipo de información de la universidad. Esta plataforma utiliza sensores para medir diversos aspectos como la cantidad de CO₂ en un despacho, la temperatura y el consumo eléctrico. Los datos recopilados se ofrecen como datos abiertos a través de la OpenAPI de la plataforma, permitiendo su consulta tanto por parte de alumnos como profesores.

Jose Vicente, el tutor de este TFG y director de Smart University, quería que estos datos lleguen de forma abierta utilizando habilidades multimedia para que la consulta sea accesible. Aunque hasta el momento se podían consultar mediante gráficas creadas con Grafana [1], se necesitaban bastantes recursos y la visualización no era accesible ni amigable. La petición del tutor consiste en realizar una plataforma donde se puedan crear gráficas configurables y accesibles para mejorar la experiencia de consultar estos datos abiertos desde cualquier dispositivo. Un requisito fundamental es que las gráficas sean compatibles con los dispositivos móviles ya que se planea que en esta plataforma se puedan generar códigos QR para ponerlos en cualquier parte de la universidad y que puedan ser escaneados por todos. En definitiva, la solicitud del tutor consiste en desarrollar una plataforma en la que se pueda definir la información a mostrar y su formato, y poder hacer accesible esta información en forma de representaciones gráficas, utilizando códigos QR como método de enlace.

Por lo tanto, este TFG tiene como propósito poner en práctica todas las aptitudes y conocimientos adquiridos durante el grado de Ingeniería Multimedia para desarrollar la petición del tutor que, a partir de ahora, será reconocido como el product owner del proyecto.

2. Planificación

En la reunión inicial sobre el proyecto con el product owner se concretaron los objetivos y requerimientos principales de la aplicación. Es importante planificar las tareas a realizar antes de ponerse con el desarrollo para llegar a cumplir estos objetivos de forma organizada. Así se podrá saber si hay que dedicar más o menos tiempo a una tarea para que no afecte a la versión final del proyecto y que esté listo para la fecha de entrega. Hay que tener en cuenta que este TFG está pensado para entregarlo en la convocatoria C1, por lo que la fecha límite para entregarlo es en diciembre de 2023. Se empezó a trabajar en junio de 2023 de modo que, a partir de aquí, se pueden concretar las fechas límite de cada bloque de tareas.

El product owner propuso que se empezara el desarrollo del proyecto con el diseño e implementación de la aplicación. Además, se decidió que se suprimirían varios apartados de la memoria como el del diseño para juntarlo con el de la implementación en un mismo apartado que se redactaría al mismo tiempo que se trabaja en el proyecto. Este apartado se dividiría en iteraciones las cuales, a su vez se dividen en fases (fase de diseño, implementación, análisis...) como se puede observar más detalladamente en el apartado 6: Implementación. Una vez acabadas, se redactarán en la memoria y se enviarán al product owner para su corrección. Como en agosto es cuando la universidad cierra y comienzan las vacaciones, se propuso como fecha límite de esta fase, el último día del mes de julio. Los siguientes meses se dedicarían a la corrección de lo implementado, adición de funcionalidades que no hubiese dado tiempo en los dos meses anteriores, despliegue de la aplicación y a la finalización de la memoria.

Con todo esto en cuenta, la planificación temporal del TFG estaría organizada como se puede ver en la siguiente tabla.

*Tabla 1. Planificación temporal TFG
(Elaboración propia)*

Contenidos	Tiempo total	Fecha límite fin
Planificación	2 meses	31 julio
Objetivos		
Metodología		
Implementación		
Redactar iteraciones en la memoria		
Corrección de la implementación	2 meses	31 octubre
Añadir funcionalidades restantes		

Corrección de la implementación en la memoria		
Instrucciones instalación y despliegue Puesta en marcha de la aplicación Resumen, motivación, estado del arte Pruebas y validación Resultados Conclusiones y trabajo futuro Referencias, bibliografía y apéndices	1 mes	30 noviembre
Video del proyecto Agradecimientos, citas, índices	2 semanas	15 diciembre

3. Estado del arte

Como se explica en la introducción, la plataforma Smart University proporciona información de los sensores en forma de datos abiertos a través de su OpenAPI [2], donde se pueden realizar peticiones y obtener estos datos dependiendo de los filtros y rangos de fechas indicados. En este apartado se va a mostrar más en detalle esta OpenAPI con ejemplos y capturas, explicando los pasos para obtener la información deseada.

The screenshot shows the main page of the Smart University - Open API documentation. At the top, it says "SMART UNIVERSITY - OPEN API" with version "1.0.0" and "OAS 3.0". Below that, there's a note: "OPEN API documentation. Requests require a token to validate the authorization." and a link "Contact Sergio Claramunt Carriés". The main section is titled "Requests" and contains three items:

- GET /tag/{token}/{tag}** Get Values from Tag
- POST /data/{token}** Get Data
- GET /data/operation/{token}/time_start/{time_start}/time_end/{time_end} /{operation}/uid/{uid}/name/{name}** Get Data With Operation

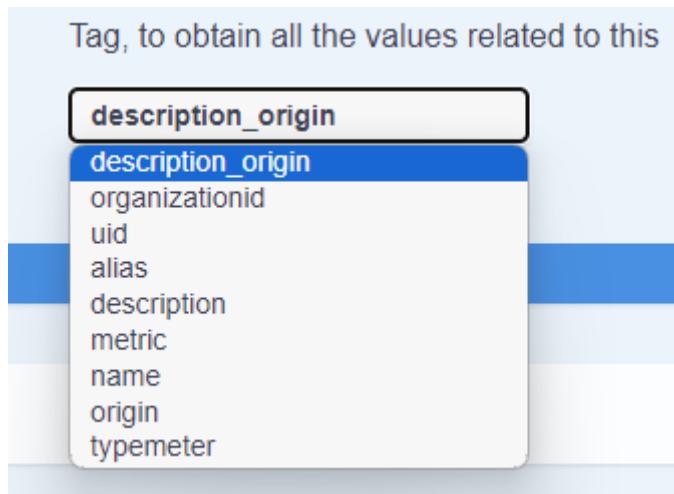
Figura 1. Página principal de la OpenAPI de Smart University.
(Fuente: <https://openapi.smartua.es/doc/>)

Lo primero que se indica al acceder a la OpenAPI es que, para poder realizar las peticiones, es necesario un token para validar la autorización. Este token debe ser entregado por uno de los administradores de la plataforma Smart University y solo puede ser utilizado por el usuario que lo ha solicitado. En el token se indica la colección de los datos de los que se va a obtener la información, es decir que, por ejemplo, hay tokens que sirven para obtener los datos de los sensores que miden el consumo eléctrico y otros para obtener información de la temperatura. En este caso se va a utilizar uno de los tokens del primer ejemplo, por lo que se van a obtener métricas del consumo eléctrico en la universidad.

3.1. Primera llamada

Es obligatorio indicar el token en las tres llamadas de la figura para realizar las peticiones correctamente. La primera llamada que hay sirve para obtener los tags de la colección. Los tags

vienen a ser los filtros que se pueden aplicar a las peticiones. En esta llamada se puede seleccionar uno de todos los filtros y se devolverá un arreglo con todos los valores que tiene en la colección.



*Figura 2. Lista de los tags para seleccionar.
(Fuente: <https://openapi.smartua.es/doc/>)*

En la figura se muestran todos los filtros que se pueden seleccionar, estos son comunes para todas las colecciones. A continuación, se explica qué es lo que indica cada uno:

- **Typemeter:** Nombre o marca del sensor o medidor.
- **Origin:** Información adicional sobre el sensor o medidor.
- **Description_origin:** Descripción de la ubicación del sensor.
- **Uid:** Identificador único del sensor.
- **Alias:** Alias del sensor.
- **Description:** Descripción del dato.
- **Metric:** Magnitud en la que se ha medido el dato.
- **Name:** Este campo identifica la dimensión del dato, la cual en algunas ocasiones son magnitudes como el CO₂, temperatura y humedad. En el caso de la colección del consumo eléctrico, solo se mide en kilovatios, por lo que en este campo se indica la cadencia en la que se ha medido el dato (15 minutos, 1 hora, 1 mes...).
- **Organizationid:** Nombre de la organización.
- **Time:** Fecha en la que se ha medido el dato. Este campo no se puede seleccionar, pero vendrá indicado en las respuestas de las siguientes llamadas.

Los siguientes campos están relacionados con la geolocalización de los sensores. Aunque no está la opción de seleccionarlos en la primera llamada, en las siguientes sí que se devolverán en la respuesta de las peticiones junto con los demás (como en el caso del campo “Time”).

- **Cota:** Altura donde está ubicado el sensor o medidor.
- **Lon:** Longitud de la posición geográfica del sensor o medidor.
- **Lat:** Latitud de la posición geográfica del sensor o medidor.

Response body

```
{  
  "columns": [  
    "key",  
    "value"  
  ],  
  "values": [  
    [  
      "name",  
      "15m"  
    ],  
    [  
      "name",  
      "1M"  
    ],  
    [  
      "name",  
      "1d"  
    ],  
    [  
      "name",  
      "1h"  
    ],  
    [  
      "name",  
      "1w"  
    ]  
  ]  
}
```

Figura 3. Respuesta de la primera llamada con el campo name seleccionado.
(Fuente: <https://openapi.smartua.es/doc/>)

3.2. Segunda llamada

La segunda llamada sirve para obtener un arreglo con datos de la colección comprendidos en el rango de fechas y filtros proporcionados en el cuerpo de la petición. Estos filtros son los comentados en la anterior llamada, por lo que se puede aprovechar para saber qué valores hay disponibles para cada filtro. Una vez se ejecuta la petición se obtendrá un arreglo con todos los datos, en los que para cada uno se indican sus tags acompañados del valor medido. Se ha descargado el JSON de respuesta para poder mostrarla mejor.

En el siguiente ejemplo, se muestra cómo se deben indicar las fechas y los filtros para que se obtenga la respuesta adecuada. Los filtros que se han aplicado son los identificadores de dos sensores y la cadencia en la que han medido los datos, que en este caso es cada 15 minutos.

The screenshot shows a POST request configuration for the endpoint `/data/{token}`. The title is "Get Data". The description is "Request to obtain `data` between given dates and optional filters".

Parameters section:

Name	Description
token * required string (path)	Token, to allow send the request

The value for the `token` parameter is set to `token`.

Request body section (required): `application/json`

The request body is defined as "Request body with dates and filters". The JSON content is:

```
{  
  "time_start": "2023-07-06T14:20:00Z",  
  "time_end": "2023-07-27T14:20:00Z",  
  "filters": [  
    {  
      "filter": "uid",  
      "values": [  
        "MLU00040001",  
        "MLU0200002"  
      ]  
    },  
    {  
      "filter": "name",  
      "values": [  
        "15m"  
      ]  
    }  
  ]  
}
```

At the bottom are "Execute" and "Clear" buttons.

Figura 4. Fechas y filtros indicados en el cuerpo de la segunda llamada.
(Fuente: <https://openapi.smartua.es/doc/>)

```

"columns": [
    "time",
    "uid",
    "value",
    "metric",
    "typemeter",
    "alias",
    "lat",
    "lon",
    "cota",
    "description",
    "description_origin",
    "name",
    "organizationid",
    "origin"
],
"values": [
    [
        "2023-07-06T23:00:00Z",
        "MLU02000002",
        1.47,
        "kWh",
        "SmartEnergy",
        "6339612",
        38.38366,
        -0.52556,
        0,
        "Consumo de energia en kWh cada 15 minutos",
        "Acom. Respaldo CPD en STI",
        "15m",
        "UA UNIVERSIDAD DE ALICANTE",
        "ElectricMeters UA"
    ],
    [
        "2023-07-07T04:00:00Z",
        "MLU02000002",
        1.473,
        "kWh",
        "SmartEnergy",
        "6339612",
        38.38366,
        -0.52556,
        0,
        "Consumo de energia en kWh cada 15 minutos",
        "Acom. Respaldo CPD en STI",
        "15m",
    ]
]

```

*Figura 5. Primeros datos obtenidos en la segunda llamada de la OpenAPI.
(Fuente: <https://openapi.smartua.es/doc/>)*

Como se puede observar, la respuesta viene dividida en dos arreglos. En el primero se indican el nombre de los tags y en el segundo se muestran los valores de esos campos respectivamente, para cada dato obtenido en el rango de fechas.

3.3. Tercera llamada

La tercera y última llamada sirve para obtener un único valor dentro del rango de fechas y filtros proporcionados, al igual que en la llamada anterior. La diferencia es que en esta petición solo hay disponibles dos filtros y son obligatorios. Estos son el identificador del sensor que ha medido el dato (UID) y la magnitud en la que lo ha hecho (name). Además, hay que indicar cuál de todos los datos dentro de ese rango de fechas es el deseado: el máximo, el mínimo o el último obtenido. La respuesta que se obtiene en esta petición tiene la misma estructura que en el caso anterior, pero solo se devolverá un dato en el arreglo de los valores.

En el siguiente ejemplo, se ha indicado que se quiere obtener el máximo dato del sensor “MLU00040001” medido cada 15 minutos, dentro de las fechas proporcionadas.

GET /data/operation/{token}/time_start/{time_start}/time_end/{time_end}/{operation}/uid/{uid}/name/{name} Get Data With Operation

Request to obtain **data** between given dates and filters. Data need to be applied one operation [LAST, MIN, MAX]

Parameters

Name Description

token * required string (path)	Token, to allow send the request <input type="text" value="token"/>
time_start * required string(\$date) (path)	Initial date to search data. Must be less than time_end . Format: YYYY-MM-DDTHH:MM:SSZ <input type="text" value="2023-05-19T05:00:00Z"/>
time_end * required string(\$date) (path)	End date to search data. Must be higher than time_start . Format: YYYY-MM-DDTHH:MM:SSZ <input type="text" value="2023-05-19T07:00:00Z"/>
operation * required string (path)	Operation to use on the data <input type="text" value="max"/>
uid * required string (path)	Filter data by value with tag uid <input type="text" value="MLU00040001"/>
name * required string (path)	Filter data by value with tag name <input type="text" value="15m"/>

Execute Clear

Figura 6. Datos introducidos en la tercera llamada.
(Fuente: <https://openapi.smartua.es/doc/>)

```
[{"tags": {"uid": "MLU00040001"}, "columns": ["time", "uid", "max", "metric", "typemeter", "alias", "lat", "lon", "cota", "description", "description_origin", "name", "organizationid", "origin"], "values": [[{"time": "2023-05-19T06:15:00Z", "uid": "MLU00040001", "max": 3.625, "metric": "kWh", "typemeter": "SmartEnergy", "alias": "6339598", "lat": "38.38787", "lon": "-0.51404", "cota": "0", "description": "Consumo de energia en kWh cada 15 minutos", "description_origin": "Ciencias 4", "name": "15m", "organizationid": "UA UNIVERSIDAD DE ALICANTE", "origin": "ElectricMeters UA"}]]
```

Figura 7. Dato obtenido en la última llamada de la OpenAPI.
(Fuente: <https://openapi.smartua.es/doc/>)

Como se ha podido observar, la información que devuelve la OpenAPI puede llegar a ser compleja y difícil de entender para la población. Es por eso por lo que el propósito de este proyecto es acercar esta información de manera accesible en forma de representaciones gráficas para que pueda ser entendible por todos los usuarios.

4. Objetivos

En este apartado se define el objetivo principal del proyecto y los subobjetivos que se van a llevar a cabo para conseguirlo. Si los objetivos quedan bien definidos desde un principio, luego será más sencillo saber si se va por buen camino y si se llegan a cumplir todas las metas propuestas.

El objetivo principal de este proyecto es diseñar y desarrollar una aplicación que facilite la visualización, accesibilidad y comprensión de los datos que provienen de la API de la plataforma Smart University.

A partir de este objetivo, nacen los siguientes subobjetivos:

- Permitir la generación de códigos QR en la plataforma para facilitar la visualización accesible y amigable de datos mediante el escaneo con dispositivos móviles.
- Facilitar la descarga de los códigos QR desde la plataforma, asegurando un proceso sencillo y configurable.
- Optimizar la visualización de los datos de Smart University para los dispositivos móviles.
- Permitir la configuración y personalización de la representación de los datos mediante gráficas.
- Diseñar interfaces específicas para cada rol de usuario dentro de la aplicación.
- Implementar medidas de seguridad para garantizar la integridad del sistema.
- Diseñar la infraestructura de la aplicación, con un enfoque especial en los aspectos de seguridad.

Estos subobjetivos recogen los aspectos clave para el éxito del proyecto por lo que su cumplimiento contribuirá al logro del objetivo principal. Gracias a ellos, se podrán obtener las funcionalidades principales que requerirá la plataforma y sus requisitos para proporcionar una experiencia eficiente.

5. Metodología

Para el desarrollo del proyecto se ha decidido trabajar con la metodología ágil Scrum. A pesar de que Scrum es un proceso en el que participa un equipo, en este caso se va a adaptar para el trabajo individual, por lo que no se realizarán daylis pero sí revisiones con el producto owner que sería el tutor.

Cada sprint o iteración de esta metodología tiene una duración de 1 semana. Dado que se dispone de 2 meses para llevar a cabo el desarrollo completo del proyecto, la duración de los sprints puede variar según sea necesario.

Para la correcta gestión y organización del proyecto y ayudar a aplicar la metodología, se va a hacer uso de 2 herramientas: Trello [13] y Clockify [14].

Por un lado, Trello es un software de administración de proyectos en el que se puede listar todas las tareas a realizar a lo largo del desarrollo del proyecto, dividiéndolas entre las diferentes iteraciones (gracias a las etiquetas). Trello permite crear tableros donde poder tener todas las tareas agrupadas en diferentes listas según el estado de estas. En este caso, las tareas se dividen en 4 listas según el estado de implementación en la que se encuentren.

- **Backlog:** todas las tareas previstas para la realización del proyecto pasan por esta primera lista. Las tareas que se encuentran en este estado, no se ha empezado con su desarrollo ni está previsto hacerlo en la iteración en la que se esté trabajando en un momento dado. Antes de pasarlas a la siguiente lista, se les asigna la etiqueta con la iteración correspondiente en la que se va a realizar.
- **To do:** en esta lista se encuentran las tareas en las que sí que está previsto el comienzo de su implementación en el sprint actual.
- **In progress:** a lo largo de las iteraciones, las tareas en las que se esté trabajando se encuentran en esta lista. Una vez que estas son finalizadas, se procede a su revisión y si todo está correcto se pueden pasar a la última lista.
- **Done:** en esta lista se encuentran todas las tareas que se han finalizado y revisado correctamente por lo que ya no deberían cambiar de estado ni moverse a otra lista.

Por otro lado, Clockify es un software de control horario el cual permite gestionar el tiempo de manera efectiva y es muy útil para medir el tiempo en el que se trabaja en cada tarea. Además, esta herramienta se puede integrar fácilmente con Trello, por lo que fue la opción preferida. En este software se ha creado un proyecto con las principales tareas o bloques que tiene el proyecto como,

por ejemplo, la creación de la API Rest o de la base de datos de la aplicación. Cada una de estas tareas se dividen en tareas más concretas las cuales son las que están agrupadas en el tablero de Trello. Un ejemplo de estas tareas puede ser la implementación de la función “getUsuarios” de la API Rest. Cuando se trabaja en una de estas tareas, en Clockify se especifica a cuál de estas tareas más generales es a la que pertenece y la iteración en la que se hace. Seguidamente se empieza a medir el tiempo pudiendo así saber con exactitud cuanto tiempo se le ha dedicado a cada bloque del proyecto.

Es importante destacar que estas dos herramientas se han utilizado a lo largo del último curso del grado por lo que existe un conocimiento previo sobre su funcionamiento y ha sido un factor determinante en la selección de estas.

6. Implementación

En esta sección se redacta todo el proceso de implementación llevado a cabo durante el desarrollo del proyecto. Como se comenta en el apartado de la metodología, para organizar eficientemente la implementación, se ha optado por utilizar la metodología Scrum, la cual se basa en iteraciones de 1 semana de duración.

Los subapartados de esta sección tienen una estructura similar. En ellos se describen primero los objetivos que se plantean cumplir en cada iteración. A continuación, se explica la fase de análisis, de diseño y de implementación que se ha realizado para poder cumplir con las expectativas de los mismos. En cada iteración puede ser que solo haya una fase de análisis y de diseño y en la siguiente solo de implementación, por lo que en todo momento se indica en cuales de estas fases se encuentra cada iteración. El flujo de trabajo durante las iteraciones será similar al diagrama de la siguiente figura.

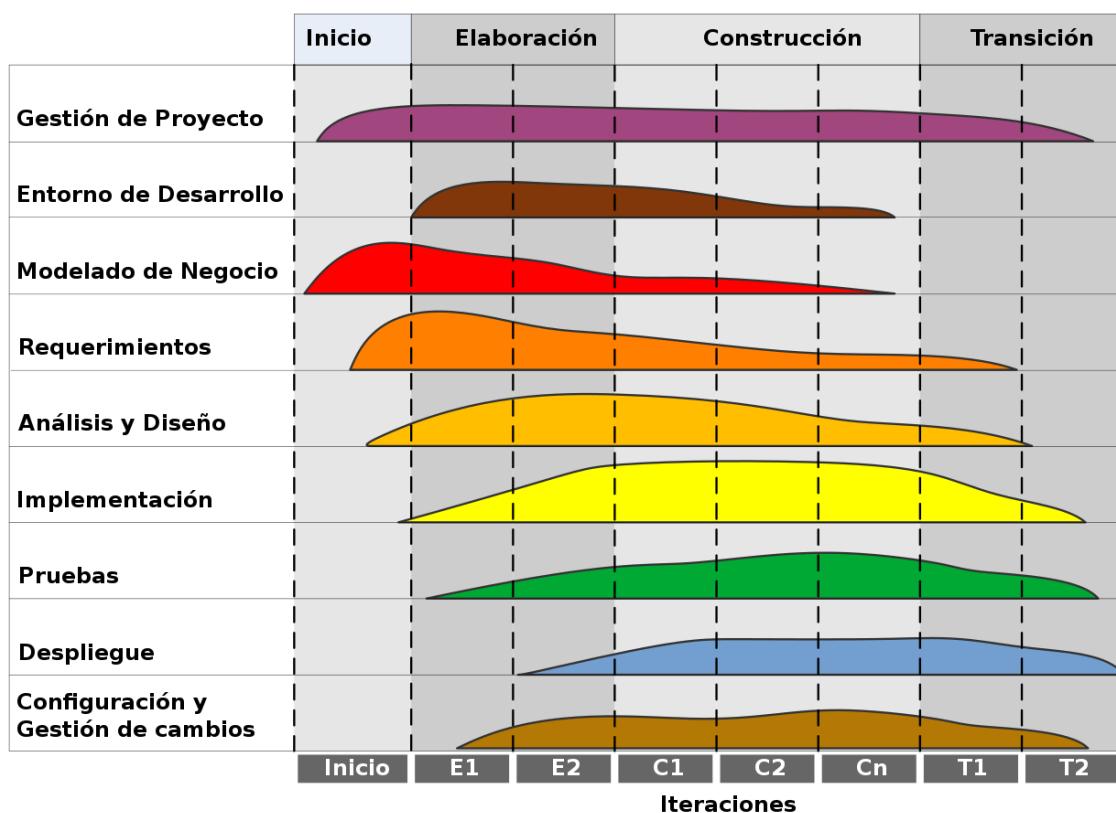


Figura 8. Diagrama ilustrando como el énfasis relativo en las distintas disciplinas cambia a lo largo del proyecto.

(Fuente: https://es.wikipedia.org/wiki/Proceso_unificado)

6.1. Iteración 1. Definición del proyecto.

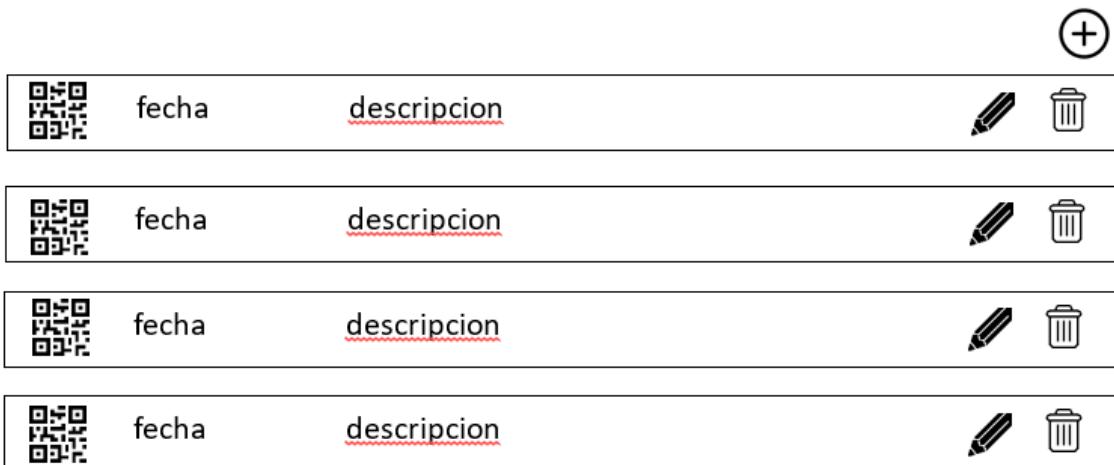
6.1.1. Fase de análisis.

Esta iteración comenzó con la primera reunión con el producto owner donde se comentaron los requerimientos básicos del proyecto. En ella se describió qué es lo que se quiere que haga la aplicación. Lo que se pide para este proyecto es una plataforma donde se puedan visualizar datos a través de códigos QR, de forma que se pueda generar un código QR y que al escanearlo con un dispositivo móvil se puedan observar datos de forma accesible y amigable. La idea es que estos datos provengan de la API de Smart University, la cual permite recuperar datos de esta plataforma a partir de una colección de datos, pero su visualización no es amigable.

Por lo tanto, se debe realizar una aplicación que permita tener una gestión de estos códigos y a su vez tener otra gestión de las consultas en cada código. Estas consultas son las llamadas a la API de Smart University, la cual devolverá los datos solicitados y será esta aplicación la que se encargará de hacer una vista accesible y entendible de estos cuando se escaneen con un terminal móvil.

En esta reunión se propusieron unos bocetos de las primeras versiones de las interfaces principales, en este caso, la interfaz de gestión de los códigos QR y la interfaz de configuración de las llamadas a la API. Esta primera ha de permitir al usuario poder hacer una gestión de sus propios códigos QR pudiéndolos crear, modificar, eliminar y desactivarlos en cualquier momento. Además, requiere que exista un tipo de paginación ya que se puede dar el caso en el que existan muchos códigos creados por un usuario y además los administradores pueden ver todos los códigos creados en la aplicación por lo que no habría una buena gestión si no existiese la paginación.

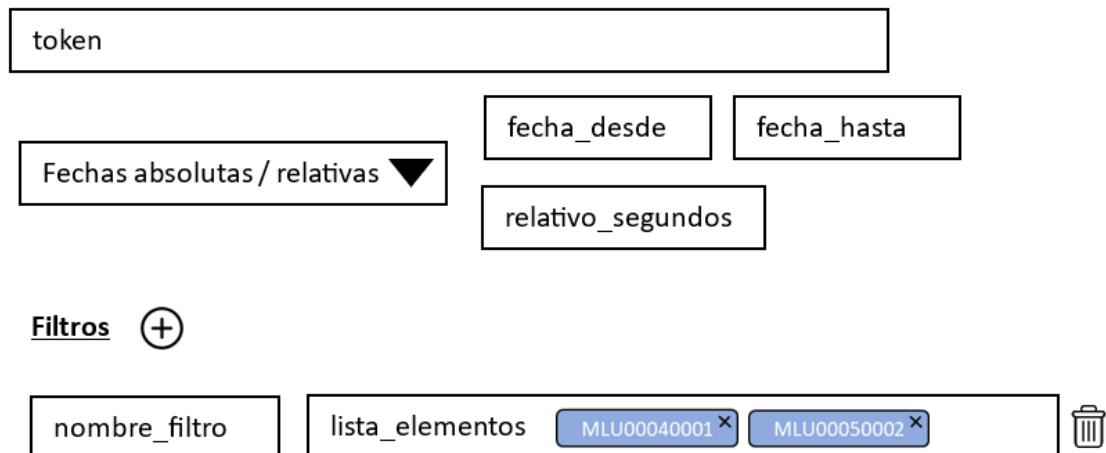
Otra de las principales funciones que se deben cumplir en esta interfaz es la posibilidad de imprimir estos códigos y elegir el formato en el que se quiere hacer. Por otro lado, la interfaz de configuración de las llamadas ha de permitir introducir el token necesario para poder realizar la llamada y también ha de permitir configurar los parámetros como, por ejemplo, las fechas, las cuales se pueden introducir de forma absoluta (fecha desde-hasta) o de forma relativa respecto al momento en que se haga la consulta (por ejemplo, desde hace 2 horas hasta ahora). También debe tener en cuenta que se puede solicitar un rango de datos en las fechas seleccionadas o simplemente un único dato numérico, por lo que el modo de representación se tiene que adaptar a estos datos.



Este boceto muestra una lista de cuatro elementos de QR. Cada elemento tiene un icono de QR, una columna para la fecha y otra para la descripción, y botones para editar y borrar.

	fecha	descripcion		

Figura 9. Primer boceto de la interfaz de gestión de QR propuesto por el product owner
(Fuente propia)



Este boceto muestra un formulario para configurar una llamada. Se incluyen campos para el token, fechas (fecha_desde, fecha_hasta, relativo_segundos), filtros (nombre_filtro) y una lista de elementos (lista_elementos).

token				
fechas absolutas / relativas ▼	fecha_desde	fecha_hasta		
	relativo_segundos			
Filtros +				
nombre_filtro	lista_elementos	MLU00040001 x	MLU00050002 x	

Figura 10. Primer boceto de la interfaz de configuración de una llamada propuesto por el product owner
(Fuente propia)

El sistema debe de estar protegido por un login pero está planteado para que solo se necesite el email y la contraseña por lo que no existe un perfil de usuario. Además, el sistema solo contempla dos roles de usuarios: el rol básico y el rol administrador. Todos los usuarios pueden hacer lo mismo

excepto crear nuevos usuarios, que solo lo pueden hacer los administradores, por lo que también se requiere una interfaz que permita la gestión de todos los usuarios registrados en la plataforma.

A partir de estos requerimientos básicos, se pueden definir los requisitos funcionales, que definen todas las funcionalidades que ofrece el sistema, y los requisitos no funcionales, que son los que están relacionados con el funcionamiento y diseño del sistema. Estos requisitos deben cumplir todos los objetivos que contempla la aplicación. Todos estos requisitos se agrupan en la tabla de a continuación y cada uno se identifica de la siguiente forma: RF-X (en el caso de los requisitos funcionales) o RNF-X (en caso de los no funcionales), donde X es el número de requisito de la lista.

También se indica a los usuarios a los que afecta cada requerimiento, los cuales, como se ha comentado anteriormente, existen los usuarios básicos y los usuarios administradores. Como los administradores pueden hacer todo lo que pueden hacer los usuarios básicos, en los requerimientos en los que únicamente pone que afecta a los usuarios básicos, se refiere a los dos tipos de usuario. Cuando un requerimiento vaya dirigido únicamente a los administradores, se indicará solo este tipo de usuario en la tabla.

Requisitos funcionales.

*Tabla 2. Tabla de los requisitos funcionales.
(Elaboración propia)*

Identificador	RF-1
Usuario	Básico
Descripción	Ofrecer una interfaz de inicio de sesión y acceso al sistema.

Identificador	RF-2
Usuario	Básico
Descripción	Ofrecer una interfaz de gestión de los códigos QR.

Identificador	RF-3
Usuario	Básico
Descripción	Creación de códigos QR, pudiendo especificar la descripción del código, su fecha de validez y su nombre y descripción de etiqueta. Estos últimos campos se verían a la hora de imprimir un código QR.

Identificador	RF-4
---------------	------

Usuario	Básico
Descripción	Activar y desactivar códigos QR para que se puedan escanear o no en un momento determinado. Aunque estén desactivados, los dueños de los códigos o los administradores sí que pueden escanearlos para realizar pruebas.

Identificador	RF-5
Usuario	Básico
Descripción	Imprimir códigos QR, pudiendo seleccionar el formato de impresión. Por ejemplo, seleccionar que se vea o no el nombre y la descripción de etiqueta.

Identificador	RF-6
Usuario	Básico
Descripción	Modificar códigos QR (cambiar nombre, fecha de validez, etc.)

Identificador	RF-7
Usuario	Básico
Descripción	Eliminar códigos QR. Como en el proyecto no se ha definido una papelera o algo similar, si se elimina se hará de forma permanente.

Identificador	RF-8
Usuario	Básico
Descripción	Buscar códigos QR a través de una barra de búsqueda.

Identificador	RF-9
Usuario	Básico
Descripción	Ofrecer una interfaz de gestión de las llamadas a la API que realiza cada código QR.

Identificador	RF-10
Usuario	Básico
Descripción	Añadir llamada a un código QR.

Identificador	RF-11
Usuario	Básico
Descripción	Activar o desactivar una llamada. A la hora de escanear el código QR, las consultas de las llamadas que se encuentran desactivadas no se realizan excepto por el dueño o los administradores.

Identificador	RF-12
Usuario	Básico
Descripción	Eliminar definitivamente llamadas.

Identificador	RF-13
Usuario	Básico
Descripción	Modificar llamadas.

Identificador	RF-14
Usuario	Básico
Descripción	Realizar una búsqueda entre las llamadas de un código QR.

Identificador	RF-15
Usuario	Básico
Descripción	Ofrecer una interfaz que permita configurar las llamadas a la API, pudiendo introducir el token necesario, seleccionar el rango de fechas, cantidad de datos y tipo de representación.

Identificador	RF-16
Usuario	Básico
Descripción	Aplicar filtros a la consulta a la API, pudiendo modificarlos y eliminarlos en cualquier momento.

Identificador	RF-17
Usuario	Básico

Descripción	Configurar parámetros de representación. Por ejemplo, en una gráfica de gauge, poder configurar colores en función de límites configurables a su vez.
-------------	---

Identificador	RF-18
Usuario	Básico
Descripción	Recuperar contraseña.

Identificador	RF-19
Usuario	Básico
Descripción	Cambiar contraseña.

Identificador	RF-20
Usuario	Básico
Descripción	Ofrecer una interfaz de vista de los datos de los códigos QR de forma accesible y amigable para poder ver principalmente desde los dispositivos móviles.

Identificador	RF-21
Usuario	Administrador
Descripción	Ofrecer una interfaz de gestión de los usuarios registrados en el sistema.

Identificador	RF-22
Usuario	Administrador
Descripción	Crear nuevos usuarios introduciendo su email y contraseña. Esta es la única forma de registrar usuarios por lo que no hay una interfaz de registro.

Identificador	RF-23
Usuario	Administrador
Descripción	Eliminar usuarios del sistema.

Identificador	RF-24
Usuario	Administrador
Descripción	Realizar búsqueda entre los usuarios del sistema.

Identificador	RF-25
Usuario	Administrador
Descripción	Modificar datos de los usuarios básicos como su límite de consultas y su rol.

Identificador	RF-26
Usuario	Administrador
Descripción	Llevar un registro de los códigos QR que se escanean desde un dispositivo móvil.

Requisitos no funcionales.

*Tabla 3. Requisitos no funcionales.
(Elaboración propia)*

Identificador	RNF-1
Usuario	Básico
Descripción	Ofrecer una buena experiencia de usuario. Diseñar las interfaces pensado en ofrecer servicios que requieran el mínimo de intervenciones posible de los usuarios y que generen una buena sensación en su uso.

Identificador	RNF-2
Usuario	Básico
Descripción	El sistema debe estar adecuadamente configurado para funcionar de forma optimizada y ofrecer el mejor rendimiento posible.

Identificador	RNF-3
Usuario	Básico
Descripción	La información personal de los usuarios debe estar almacenada de forma segura y protegida de ataques en todo momento y solo podrá ser accesible por el propio usuario y el administrador.

Identificador	RNF-4
Usuario	Básico
Descripción	Se debe asegurar que todos los datos introducidos por los usuarios permanezcan en la base de datos hasta que el usuario decida eliminarlos o sean eliminados del sistema por un administrador.

Identificador	RNF-5
Usuario	Básico
Descripción	Se tendrán en cuenta todos los aspectos legales relacionados con la política de privacidad, condiciones de uso y política de datos.

6.1.2. Fase de diseño.

Una vez definidos y los requerimientos básicos del sistema y propuesto una primera versión de lo que podía ser el diseño de las interfaces principales, se dio paso al desarrollo de los primeros bocetos de todas las interfaces necesarias para tratar de cumplir las expectativas de todos estos requerimientos. Se hicieron primero en papel a modo de una primera idea y guía para, posteriormente, poder crear los primeros wireframes en Balsamiq [4] y trabajar en la estructura de la aplicación.

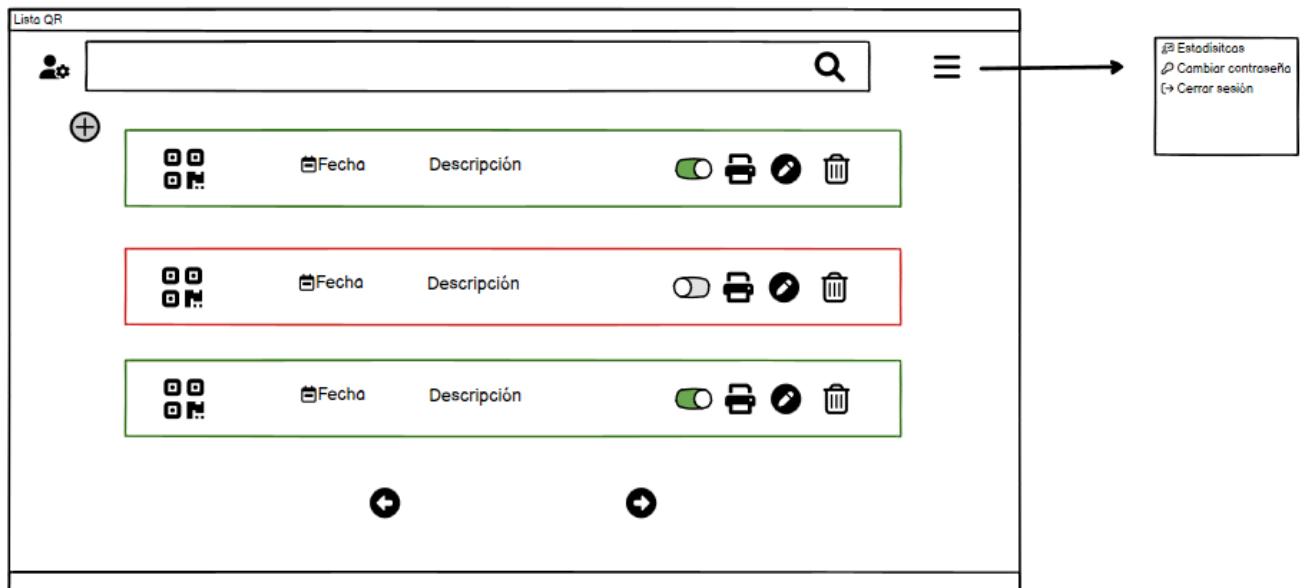


Figura 11. Wireframe de la página de gestión de QR.
(Fuente propia)

6.2. Iteración 2. Mockups del proyecto.

En esta iteración se continua con el diseño de interfaces a partir de los requerimientos recopilados en la fase de análisis de la iteración anterior, por lo se continua con la fase de diseño.

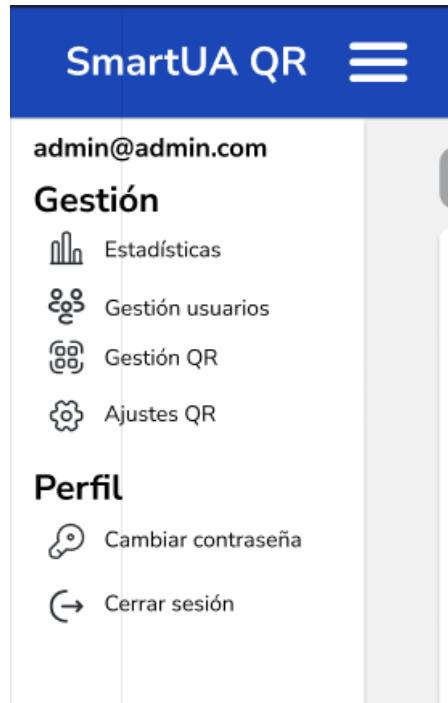
6.2.1. Fase de diseño.

Una vez creados todos los wireframes, se realizó una segunda reunión con el tutor para la revisión y corrección de la estructura de la aplicación. Se añadieron nuevas funcionalidades como la posibilidad de activar y desactivar llamadas de un QR, la cual ya se podía hacer con los propios QR. De esta manera no se tiene que borrar una llamada que no se desea que aparezca en la vista del QR, por el motivo que sea, perdiendo así todos los datos de esta. Otro elemento que se añadió fue la barra lateral. En un principio se pensó en poner las opciones del perfil como cambiar la contraseña, cerrar sesión o acceder a la gestión de los usuarios (en el caso de ser administrador) en la cabecera de la aplicación como se puede observar en la Figura 12.



Figura 12. Opciones del perfil en la cabecera de la aplicación.
(Fuente propia)

No obstante, al solo existir email y contraseña para todos los usuarios y no haber foto de perfil, se decidió poner estas opciones en una barra lateral además de la gestión de los QR y demás pudiendo acceder a ellas en cualquier momento.



*Figura 13. Opciones del perfil y gestión en la barra lateral.
(Fuente propia)*

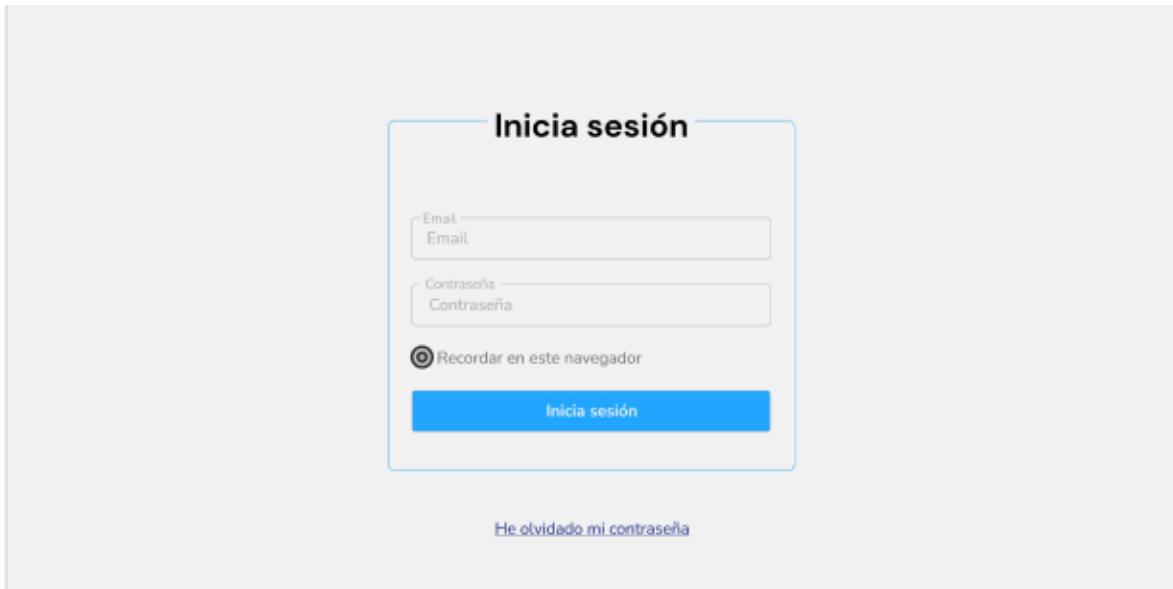
Dado el visto bueno, se procedió a realizar los mockups del proyecto en la aplicación Figma [5], un programa de edición y prototipado dirigido principalmente al diseño web, para así obtener unas interfaces más exhaustivas.

Aunque puede que sufran cambios a lo largo del desarrollo del proyecto, estos no afectarán excesivamente, por lo que estas son las interfaces en las que se va a basar el maquetado de la aplicación. Como el proyecto se basa en la creación de códigos QR, la interfaz de la vista de estos está pensada para los dispositivos móviles. Por otro lado, toda la gestión de los códigos y sus respectivas llamadas está pensada para realizarlos desde un ordenador, por lo que sus interfaces se han creado con el tamaño de una pantalla de escritorio. No obstante, todas las interfaces son responsivas y se podrían trabajar con ellas desde cualquier dispositivo.



*Figura 14. Mockup de la vista de un código QR en un dispositivo móvil.
(Fuente propia)*

En primer lugar, se pueden observar las interfaces de inicio de sesión y recuperación de la contraseña (Figura 15) en caso de que el usuario la olvide. Como el usuario administrador es el encargado de crear nuevos usuarios, no es necesario una página de registro.



*Figura 15. Interfaz de inicio de sesión
(Fuente propia)*

Une vez iniciada la sesión, se puede acceder a la página principal que en este caso es la gestión de los códigos QR. Desde esta sección se puede acceder a todas las funcionalidades de la aplicación, creando un nuevo código QR o accediendo a las opciones y ajustes en la barra lateral como se puede ver en la Figura 16.

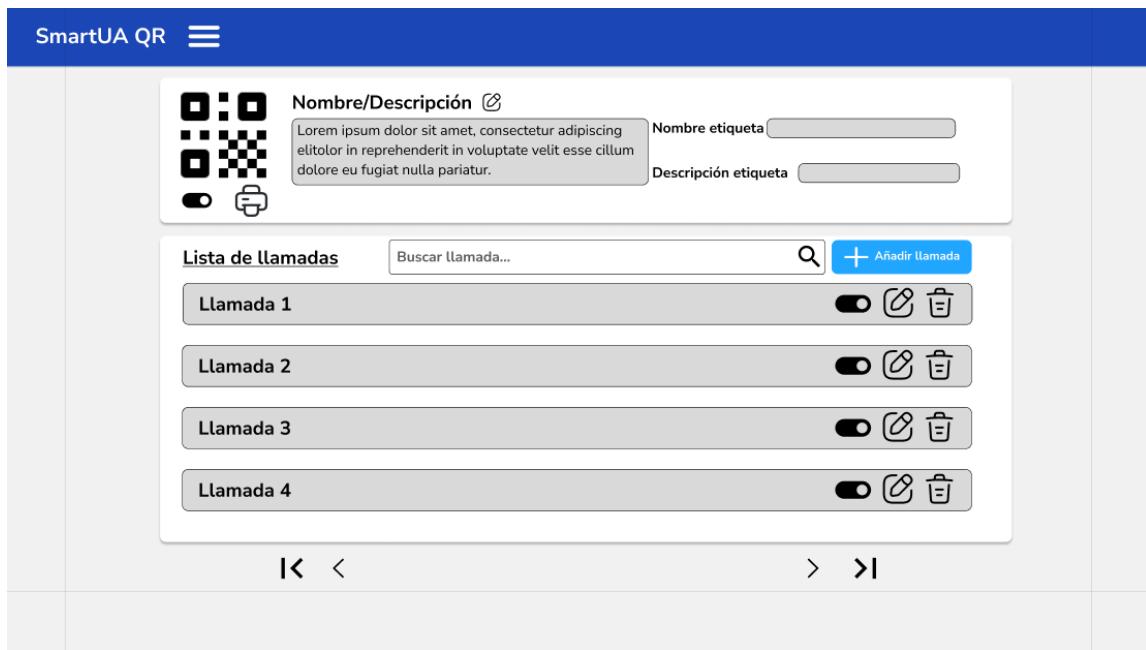
De cada código podemos ver su descripción y su fecha de validez, es decir, hasta qué fecha estará disponible la vista de este. Además, podemos activar o desactivarlos, imprimirlas, editarlas y eliminarlas. Finalmente, se puede escanear el código directamente gracias al hover (primer ícono de cada elemento en la lista de los códigos QR que se observa en la Figura 16) ya que amplía la imagen del QR haciéndolo legible para los dispositivos móviles.

The mockup shows the 'SmartUA QR' application interface. On the left, there's a sidebar with the user email 'admin@admin.com' and sections for 'Gestión' (Statistics, User Management, QR Management, QR Settings) and 'Perfil' (Change Password, Logout). The main area has a search bar 'Buscar QR ...' and a 'Añadir QR' button. Below is a table listing six QR codes, each with a preview icon, date (e.g., 28/04/2023), description, and a row of icons for edit, print, and delete. Navigation arrows at the bottom allow for viewing more items.

	Fecha	Descripción	Acciones
1	28/04/2023	Descripción	
2	03/03/2023	Descripción	
3	25/02/2023	Descripción	
4	15/02/2023	Descripción	
5	22/12/2022	Descripción	
6	02/12/2022	Descripción	

Figura 16. Mockup de la página principal con la barra lateral desplegada.
(Fuente propia)

Si se pulsa en el botón añadir QR, se accede a la página de edición de un código QR. En ella se puede añadir un nombre y una descripción, la cual posteriormente se podrá ver desde la página principal, y un nombre y descripción de etiqueta. Estos dos elementos sirven para la visualización de un código QR a la hora de su impresión, pudiendo elegir entre que solo se vea el propio código o que, además, se vea con su nombre y descripción de etiqueta para indicar a los usuarios qué muestra exactamente ese código.



*Figura 17. Mockup de la creación de códigos QR.
(Fuente propia)*

Desde esta sección se pueden añadir llamadas al QR, buscarlas, activarlas o desactivarlas, editarlas y eliminarlas.

Si vamos a añadir una nueva llamada, se pasaría al formulario de gestión de la llamada, en la que se rellenan con la información necesaria para indicar cuáles son los datos que queremos obtener. El diseño de este formulario está dividido en 4 pasos principalmente (como se aprecia en la Figura 18), pudiéndose añadir pasos extra en el caso de que sea necesario. En el primer paso se solicita el nombre de la llamada, el token necesario para poder hacer la petición a la API de Smart University, la cual debe ser entregada por un usuario administrador y el rango de fechas en el que se encuentran los datos que nos interesan. La fecha se puede introducir de dos formas distintas: de forma absoluta y de forma relativa. En la primera opción se debe introducir dos fechas: la fecha desde y la fecha hasta. En la segunda opción se puede indicar el tiempo en segundos que se quiere restar a la fecha del momento en el que se hace la consulta. Por ejemplo, se puede introducir 5 minutos en este campo, por lo que, a la hora de hacer la consulta, los datos que se devuelven están comprendidos entre la fecha en la que se realiza la petición y los 5 minutos que se le restan.

En el segundo paso simplemente hay que elegir si lo que se quiere es todos los datos o un máximo, mínimo o el último dato en las fechas seleccionadas. Dependiendo de lo que se seleccione los siguientes pasos cambian. Si se selecciona todos los datos, en el siguiente paso se especifican los filtros que queremos aplicar a la petición pudiendo añadir y eliminar. Si lo que se elige es un máximo, mínimo o el último dato, en el siguiente paso se tienen que llenar dos campos

obligatorios: el identificador del sensor y la magnitud que lee (un mismo sensor puede tener varias medidas como por ejemplo temperatura, humedad, CO₂).

Por último, se tiene que seleccionar el tipo de representación que, como se ha comentado, también varía según el número de datos que se han seleccionado. En el primer caso se puede seleccionar gráficas de barras o de líneas que permiten la representación de un gran número de datos. En el segundo caso se puede elegir entre un número simplemente o un gráfico de medidor radial. Estos son algunos ejemplos de representación de datos, pero se pueden ir añadiendo más formas para mostrar los datos si fuese necesario.

The screenshot shows the SmartUA QR call form interface. At the top, there are input fields for 'Nombre' and 'Token'. Below these are date range selection buttons: 'Fecha absoluta' (with 'Desde' and 'Hasta' sub-buttons), 'Fecha relativa' (radio button), and 'Fecha absoluta' (radio button). To the right, a dropdown menu shows 'Todos' at the top, followed by 'Max', 'Min', and 'Last'. A 'Filtros' section contains two entries, each with a 'Nombre_filtro' field and two 'ej_...' fields (ej_uid1/ej_uid2 and ej_filt1/ej_filt2) separated by a vertical line. Below this is a 'Tipo de representación' section with icons for a line chart and a bar chart. At the bottom right are 'Cancelar' and 'Guardar' buttons.

Figura 18. Formulario de llamada con la opción de todos los datos.
(Fuente propia)

The screenshot shows a search interface titled "SmartUA QR". At the top right is a menu icon (three horizontal lines). Below the title are two input fields: "Nombre" and "Token". Underneath these are date range controls: "Fecha absoluta" with arrows for "Desde" and "Hasta", and radio buttons for "Fecha absoluta" and "Fecha relativa". To the right is a dropdown menu set to "Max". Below this section is a "Filtros" panel containing two entries: "UID" and "Name", each with a corresponding input field. To the right is a "Tipo de representación" panel showing a large number "1" and a circular icon with a checkmark. At the bottom are red "Cancelar" and blue "Guardar" buttons.

*Figura 19. Formulario de llamada con la opción del máximo dato.
(Fuente propia)*

Hasta ahora, todas estas funcionalidades pueden ser realizadas por todos los usuarios, pero hay otras a las que solo se puede acceder si el usuario tiene el rol de administrador, como la gestión de los usuarios de la aplicación. En esta sección, los administradores pueden ver la lista de los usuarios registrados, pudiéndolos editar, eliminar o registrar nuevos. A parte de poder cambiar el email y la contraseña, a la hora de editar el perfil de un usuario, se puede limitar el número de consultas que un usuario con el rol base puede realizar.

The screenshot shows a user profile editing interface titled "SmartUA QR". At the top right is a menu icon. The main area displays a user profile for "User 1" with fields for "Email" and "Contraseña". Below these is a "Límite de consultas" input field. At the bottom are red "Cancelar" and blue "Guardar" buttons.

*Figura 20. Página de edición de un usuario.
(Fuente propia)*

Finalmente, quedan las interfaces que muestran casos que se pueden presentar a la hora de estar trabajando con la aplicación como el intentar salir de la gestión de una llamada sin guardar los cambios realizados como se ve en la siguiente figura, o los menajes de confirmación de una acción realizada, como puede ser la creación de un nuevo código QR.

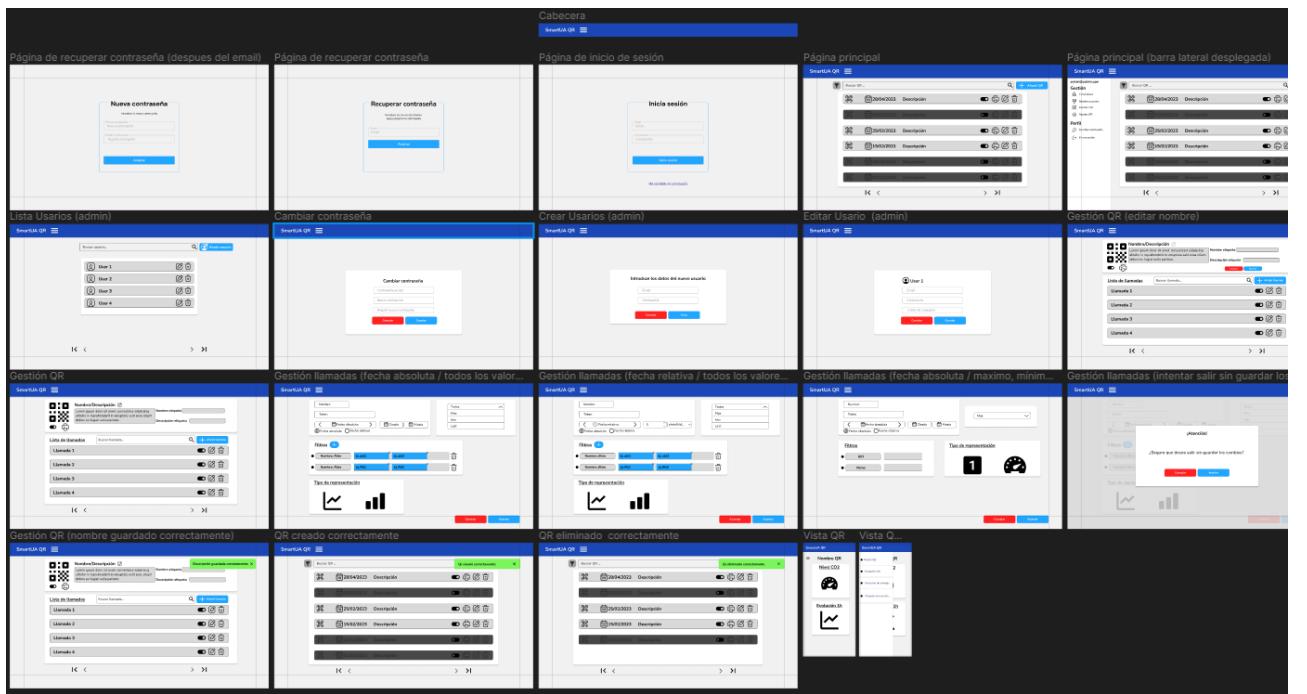


*Figura 21. Caso de intentar salir del formulario sin guardar los cambios.
(Fuente propia)*

Otro caso que se puede presentar mientras se trabaja con la plataforma es que se muestren alertas y mensajes que informan al usuario del resultado de una acción realizada como la creación de un nuevo código QR. Estas alertas están pensadas para que se muestren en la parte superior derecha de la pantalla, debajo de la cabecera, y que al cabo de unos segundos desaparezca automáticamente.



*Figura 22. Caso de crear correctamente un código QR.
(Fuente propia)*



*Figura 23. Vista global de las interfaces.
(Fuente propia)*

6.3. Iteración 3. Preparación del entorno de implementación y Base de Datos.

Una vez terminada la fase de diseño de interfaces se puede pasar a la implementación del proyecto por lo que se empieza con la fase de implementación. Para esta iteración se propuso como objetivo la preparación del entorno de desarrollo y se contempló la creación de una base de datos en MySQL en local con la aplicación PhpMyAdmin [11].

6.3.1. Fase de implementación.

Lo primero que se ha hecho es la preparación del entorno de desarrollo, comenzando por la creación de un repositorio en GitHub [12], con el nombre de My-university-QR [8], para realizar una correcta gestión de versiones de forma sencilla. Se va a trabajar con esta herramienta estructurando el repositorio principalmente en 2 ramas: la rama master y la rama develop. En la rama master se encuentra el código finalizado y revisado mientras que en la rama develop es en la que se trabaja y se sube el código de las funcionalidades estables a falta de su revisión. Por otro lado, se crean ramas extra para el desarrollo de las nuevas funcionalidades que se van añadiendo

según se avance en la implementación del proyecto. Por ejemplo, si se trabaja en una nueva función para el frontend del proyecto, se crea una rama llamada *feature/frontend-nombre_de_la_función*. En esta última rama se trabaja hasta completar el desarrollo propuesto para esta rama y se mergea con la rama *develop* para su revisión.

En cuanto a la estructura del proyecto, se ha decidido dividirlo en dos carpetas en el repositorio: *frontend* y *backend*. En la carpeta *backend* se encuentra todo el desarrollo relacionado con la API y la conexión con la base de datos mientras que en la carpeta *frontend* se encuentra todos los archivos de angular necesarios para crear y trabajar en las páginas y las peticiones a la API para su correcto funcionamiento.

El desarrollo del código se realiza en el editor Visual Studio Code [9] por las distintas facilidades que proporciona a la hora de programar gracias a las extensiones que tiene disponibles. Además, este editor hace más sencillo la conexión al repositorio de GitHub desde su propia interfaz, pudiendo trabajar directamente sin tener que abrir una terminal aparte, lo que hace que el flujo de trabajo sea más eficiente. Con simples pasos se pueden hacer commits y subir los cambios que se van realizando en el código además de tener un control de las ramas gracias a la extensión *GitGraph*.

Desde la terminal que brinda Visual Studio Code, se realizó toda la instalación de las herramientas y paquetes principales. Para el *backend*, como la API se desarrolla en Node.js, se instaló en su versión 19.0.0. Junto a esta herramienta, en la instalación, venía el paquete *npm* en su versión 8.19.2, el cual permite instalar los demás paquetes. Por otro lado, para el *frontend*, se instalaron los paquetes de *typescript*, versión 4.8.4 y de *angular-cli*, en su versión 16.0.5.

En cuanto a la base de datos, como se ha comentado, se ha creado en local con *PhpMyAdmin* instalando previamente la aplicación XAMPP [10] para poder realizar la conexión con la API de la aplicación. Según los requerimientos del proyecto, principalmente se necesita una tabla para los usuarios, otra para los códigos QR que crean estos y una última para almacenar las llamadas o consultas que realiza cada QR. En cuanto a los atributos de cada tabla, se irán añadiendo y modificando a medida que se avance en el desarrollo según se vayan necesitando para ajustarse a los requisitos de la aplicación. En esta iteración, para poder implementar el inicio de sesión, se necesitó que en la tabla *Usuario* se almacene el email y contraseña de cada uno, por lo que fueron los primeros atributos que se le añadieron. Posteriormente, se le añadieron los atributos rol, para la gestión de roles (básico y administrador) y límite de consultas, que indica el límite que tiene cada usuario para poder crear códigos QR. Este atributo tiene por defecto que el límite sea 10, pudiendo ser modificado únicamente por los administradores.

También se crearon las tablas *Códigos QR* y *Consultas* con unos atributos básicos que cambiarán en cuanto se trabajen con ellas. No obstante, en este momento ya se podía indicar las relaciones que tienen las tablas entre ellas. Un usuario puede crear muchos códigos QR y, a su vez, un código QR puede estar compuesto por muchas consultas por lo que en la base de datos existen dos relaciones uno a muchos entre la tabla *Usuarios* y *Códigos QR* y entre esta misma tabla y la de *Consultas*. Además, se le añadieron la regla borrar en cascada a estas relaciones de modo que si, por ejemplo, se elimina un código QR, se eliminan a su vez todas sus consultas sin tener que hacerlo todo manualmente desde el código una por una. Con todo esto, y a falta de modificar y adaptar los atributos de las tablas en la medida de lo necesario, el esquema de la base de datos sería la de la siguiente figura.



Figura 24. Esquema de la Base de Datos.
(Fuente propia)

Para empezar a trabajar en la API, se inicializó un proyecto con la orden `npm init` y seguidamente se instaló el paquete ExpressJS por las funcionalidades que brinda para no tener que programar desde cero todo el código. Posteriormente, se creó el archivo base del API llamado `index.js` donde se creó una primera aplicación que escucha al puerto 3000 y se instaló otro paquete llamado `nodemon` para poder trabajar sin tener que volver a lanzar la API cada vez que se realiza un cambio en el código. A continuación, con motivo de evitar tener todo el código agrupado en el mismo archivo `index.js` se estructuró el proyecto en dos carpetas: `routers` y `controllers`. En la carpeta `routers` se encuentran todas las declaraciones de las rutas y en la carpeta `controllers` se encuentran todos los controladores de estas rutas, es decir, las funciones CRUD (crear, leer, actualizar y borrar) como `getUsuarios` y `getCódigosQR`. Como en esta iteración se propuso comenzar con el login de la aplicación, se crearon el router y controller correspondientes a esta funcionalidad llamados con el nombre `auth.js`.

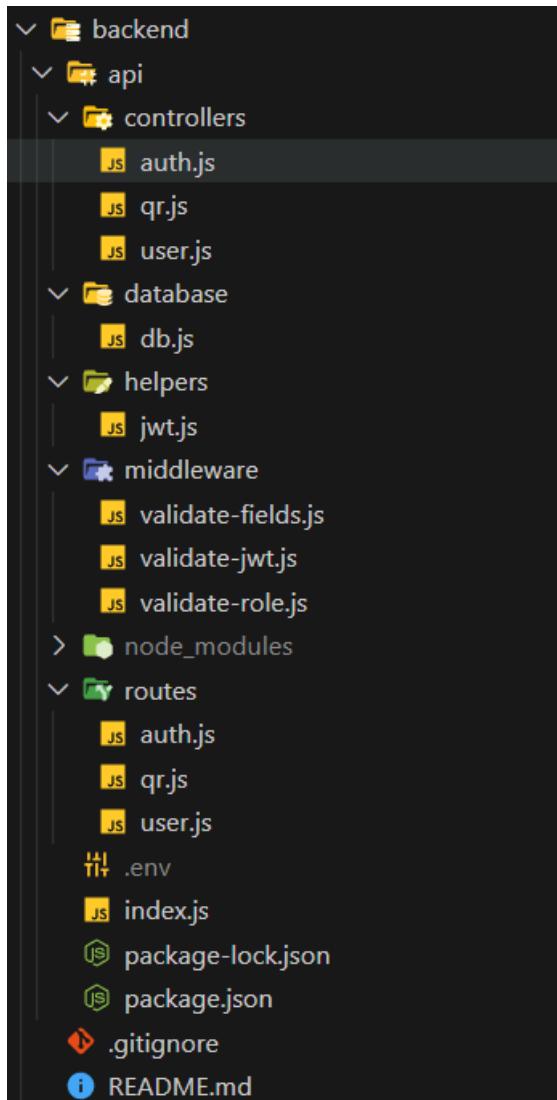


Figura 25. Estructura del backend del proyecto.
(Fuente propia).

En la carpeta backend, además de las dos carpetas mencionadas anteriormente, se van creando más con la finalidad de tener el código más ordenado. Para el caso de la conexión con la base de datos, se ha creado una carpeta llamada database, donde se encuentra el fichero encargado de realizar esta conexión (Figura 26). Para poder realizar este código se instaló el paquete de *mysql* con *npm*. Lo primero que se hace en este archivo es crear un objeto de conexión mysql con la función *createConnection ()*. A esta función se le pasa todos los datos de la base de datos: el host, el nombre de la base de datos y el usuario y su contraseña con los permisos para poder modificar la base de datos (en este caso el usuario root que viene por defecto en PhpMyAdmin).

Como todos estos datos pueden variar y sería muy ineficiente ir buscando en el código donde están e ir cambiándolos uno a uno, se ha creado un archivo aparte en la raíz del backend llamado `env` donde se declaran las variables de entorno. Estas variables se pueden llamar en cualquier parte del código y si por ejemplo se tuviese que cambiar una de ellas, solamente se tendría que modificar en este archivo. Fue necesaria la instalación del módulo `dotenv` para poder usar este archivo de variables.

Después de crear la conexión, se conecta a la base de datos y espera a que se le hagan peticiones a través de una consulta SQL. Para ello, en el mismo archivo se ha creado una función asíncrona que recibe una cadena de texto con una consulta (por ejemplo: `SELECT * FROM usuarios;`) y espera a la respuesta de la base de datos que es devuelta una vez recibida. Entonces, cada vez que se necesite realizar una petición a la base de datos, se le envía la consulta correspondiente a esta función sin necesidad de repetir código.

```
const mysql = require('mysql');
const util = require('util');

const connection = mysql.createConnection({
  host: process.env.HOST,
  database: process.env.DATABASE,
  user: process.env.USER,
  password: process.env.PASSWORD
});

// Conexión con la base de datos
connection.connect();

// Utilidad para promisifyar la función query y poder usar async/await
const queryAsync = util.promisify(connection.query).bind(connection);

// Función para realizar las consultas a la base de datos
async function dbConsult(query){
  try {
    const results = await queryAsync(query);
    return results
  } catch (error) {
    console.log("Error al realizar la consulta a la base de datos");
    throw error;
  }
}

module.exports = {dbConsult}
```

Figura 26. Archivo `db.js` encargado de la conexión y hacer peticiones a la base de datos.
(Fuente propia)

Aparte de la carpeta para la conexión de la base de datos, se han creado dos más: middleware y helpers (Figura 25). En la carpeta middleware se encuentran archivos con funciones cuya finalidad es hacer una serie de comprobaciones antes de realizar la petición a la API. En concreto, se realizan 3 comprobaciones en la mayoría de las rutas. La primera de todas es la comprobación de que el Json Web Token enviado sea válido. Este token se genera cada vez que un usuario inicia sesión en la plataforma como se comenta a continuación. Si no es válido directamente no se realizan las siguientes comprobaciones y se detiene la ejecución de la petición. La siguiente comprobación que se realiza es si los campos enviados por el cuerpo de la petición, en las funciones que la necesiten como por ejemplo el inicio de sesión, son los esperados por la API. En caso de que algún campo no sea válido, también se detiene la ejecución. Por último, está la función de validar los roles, la cual es útil para cuando se quiere crear o editar un usuario y se el campo rol que se introduce en la petición no existe o no es correcto en esta aplicación. En cuanto a la carpeta helpers, en ella se encuentra el archivo encargado de generar y devolver el token comentado anteriormente. Esta función genera un nuevo token cada vez que un usuario va a iniciar sesión a partir de su identificador en la base de datos y de su rol. Además, es necesario proporcionar una cadena de caracteres que sirve de secreto para firmar este token y saber que pertenece a la aplicación y una duración del token para indicar cuando caduca. Este secreto y duración se declaran en el archivo de las variables de entorno. Una vez generado, este se devuelve como respuesta a la petición.

6.4. Iteración 4. Creación de ruta Login y su componente en el frontend.

Esta iteración tenía como objetivo empezar con la implementación de las primeras rutas de la API de la aplicación con su respectiva interfaz en el frontend, por lo que se continua con la fase de implementación de la iteración anterior. A partir de ahora se trabaja de esta manera con las demás funcionalidades realizadas en las siguientes iteraciones. En este caso, se decidió empezar con la funcionalidad de inicio de sesión y la comprobación del token a la hora de intentar acceder a las rutas de la aplicación.

6.4.1. Fase de implementación.

La ruta base de la API de la aplicación, trabajando en localhost, es *localhost:3000/api* y a partir de esta salen todas las demás rutas. En el caso del router del login, se declara la ruta añadiéndole /login a la ruta base y así con las demás rutas que se van creando a lo largo del desarrollo del proyecto.

Esta ruta es una petición POST, ya que para poder iniciar sesión es necesario enviar el correo y la contraseña del usuario. En el controller se crea la función *login()*, donde primero se comprueba que existe el usuario en la base de datos con el email enviado y seguidamente se comprueba que la contraseña facilitada pertenece a este. En caso de que no exista el usuario o la contraseña no sea la correcta, se devuelve un mensaje donde se indica que el correo o la contraseña son incorrectos para no dar demasiada información por cuestiones de seguridad. En caso de que no haya ningún error, se procede a la creación del token y, una vez generado, este se devuelve como respuesta a la petición.

The screenshot shows a Postman request to `localhost:3000/api/login` using a POST method. The request body contains the following JSON:

```

1  {
2     "email": "userpass@gmail.com",
3     "password": "12345"
4 }

```

The response status is `200 OK`. The response body is:

```

1  {
2     "user": {
3         "idUser": 4,
4         "email": "userpass@gmail.com",
5         "role": 1,
6         "lim_consult": 0
7     },
8     "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1aWQiOjQsInJvbGUIOjEsImlhdCI6MTcwMjM4MTM2NiwiZXJ0uu-PfoHW0ApRW25lJpuu5oK0vPHdivigFeNX99g4"

```

Figura 27. Prueba de llamada a *login* desde Postman.
(Fuente propia)

Una vez terminadas las rutas y funciones necesarias para realizar el login en la parte del backend, se comenzó a realizar la página correspondiente en angular. Para empezar, se generó el proyecto en la carpeta frontend con el comando `ng new` e indicando que sí que vamos a usar Angular routing en el proyecto. Para no tener que realizar todas las páginas desde cero, se hace uso de una plantilla llamada NiceAdmin [7], que proporciona todos los complementos necesarios para la realización de una aplicación web. Por lo que, al crear los complementos en angular, se puede pasar directamente el código HTML y CSS adaptándolo a las necesidades del proyecto y directamente desarrollar la

lógica de estos en typescript. Las páginas de la aplicación están agrupadas según si son públicas, como la página login o la página de vista de los códigos QR o privadas, como la página de gestión de estos códigos.

Como la página de login es pública, se crea el componente dentro de la carpeta de las páginas públicas. Una vez creado con el comando `ng g c /pages/public/login --skip-tests`, se procedió a pasar todo lo necesario de la plantilla a este componente y realizar las peticiones necesarias a la API. El comando ‘`--skip-tests`’ se utiliza al crear un componente para evitar generar archivos de prueba automatizados junto con el componente. A partir de ahora, todos los componentes que sean necesarios se crearan de la misma forma y se obtendrá un resultado como el de la siguiente figura.

```
PS C:\Users\Neme\Desktop\UNI\TFG\app\frontend> ng g c page/public/login --skip-tests
Node.js version v19.0.0 detected.
Odd numbered Node.js versions will not enter LTS status and should not be used for pro
CREATE src/app/page/public/login/login.component.html (20 bytes)
CREATE src/app/page/public/login/login.component.ts (198 bytes)
CREATE src/app/page/public/login/login.component.css (0 bytes)
```

*Figura 28. Creación del componente login desde la terminal.
(Fuente propia)*

En este caso se debe realizar la petición POST a la ruta login de la API y pasarle los campos email y contraseña del formulario. Para poder realizar esta petición fue necesaria la creación de un servicio. Los servicios permiten encapsular toda la lógica que tiene que ver con la comunicación con el backend de forma que puede ser reutilizable desde cualquier componente de la aplicación. Estos servicios se encuentran dentro de la carpeta services, por lo que primero se ejecutó el comando `ng g s services/user` y se generó el archivo user.service.ts encargado de realizar las peticiones relacionadas con los usuarios. Los servicios se generan de la misma forma que los componentes y se obtiene un resultado similar al de la Figura 28. Lo que hace el servicio de usuario, a la hora de iniciar sesión, es realizar la petición pasándole los campos introducidos a través del formulario de la página login. Si el backend contesta sin ningún error, es el mismo servicio el que se encarga de almacenar el token que devuelve la API en el almacenamiento local del servidor para poder acceder a él en cualquier momento. Además, si en el formulario (Figura 29) el usuario marca que quiere que se le recuerde en la aplicación, también almacena su correo y la siguiente vez que entre aparecerá directamente su correo en el formulario. Si todo sale bien, se redirige a la página principal que en este caso es la de la gestión de los códigos QR. En caso de que haya algún error, se muestra una alerta amigable en pantalla informando al usuario de lo que ha pasado. De esta forma se cumpliría con el primer requisito funcional del proyecto (RF-1 en la Tabla 2).

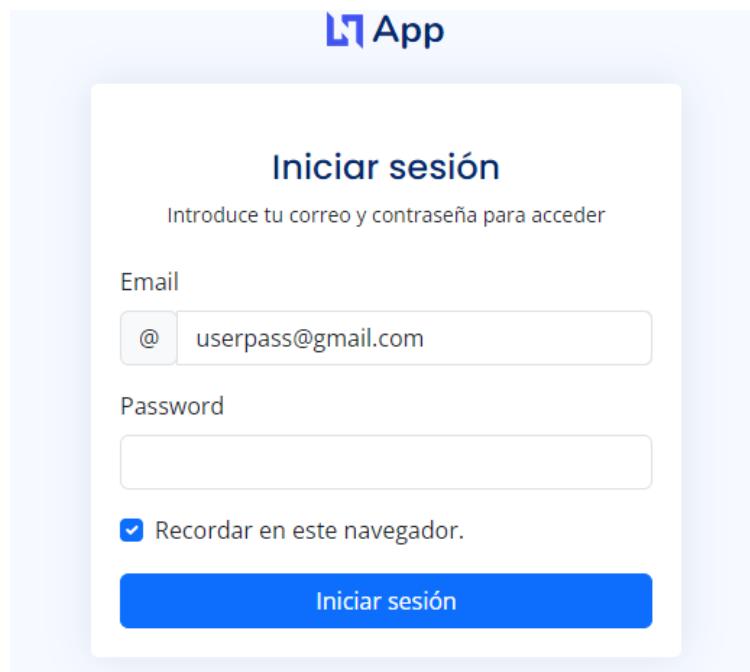


Figura 29. Componente login implementada en la aplicación.
(Fuente propia)

Esta alerta está ubicada en la parte superior derecha de la pantalla y desaparece por si sola al cabo de unos segundos para no tener que quitarla manualmente. La idea es utilizar estas alertas para ir informando a los usuarios de cada cosa que suceda mientras trabaja en la aplicación. Esta alerta es un complemento a parte que se ha generado al igual que el complemento de la página login, pero este posee su propio servicio. Con lo que cada vez que se necesite mostrar una alerta, simplemente hay que llamar a este servicio, el cual se encarga crearlo y mostrarlos con el mensaje que se le pase por parámetro. Hay distintos tipos de alertas disponibles: en caso de que todo haya ido bien, en caso de que haya sucedido algún error, en caso de querer informar al usuario y en caso de querer mostrar una advertencia. Cada alerta tiene su propio color por lo que es fácil de diferenciar que objetivo tiene cada uno de estos tipos de alerta.

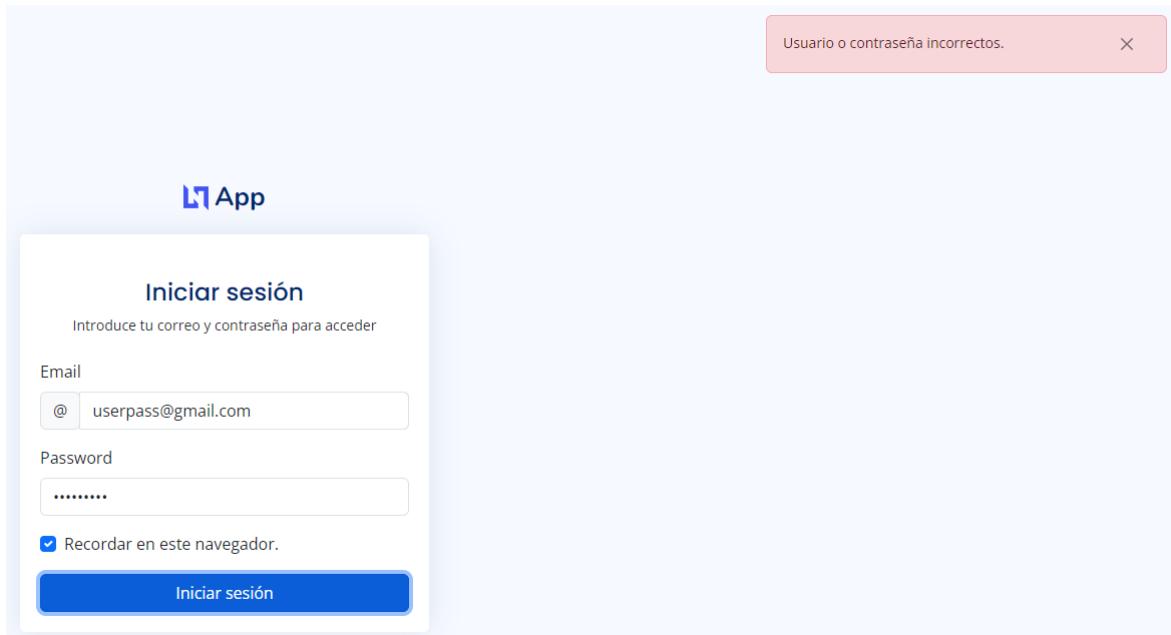


Figura 30. Alerta de error al intentar iniciar sesión.
(Fuente propia)

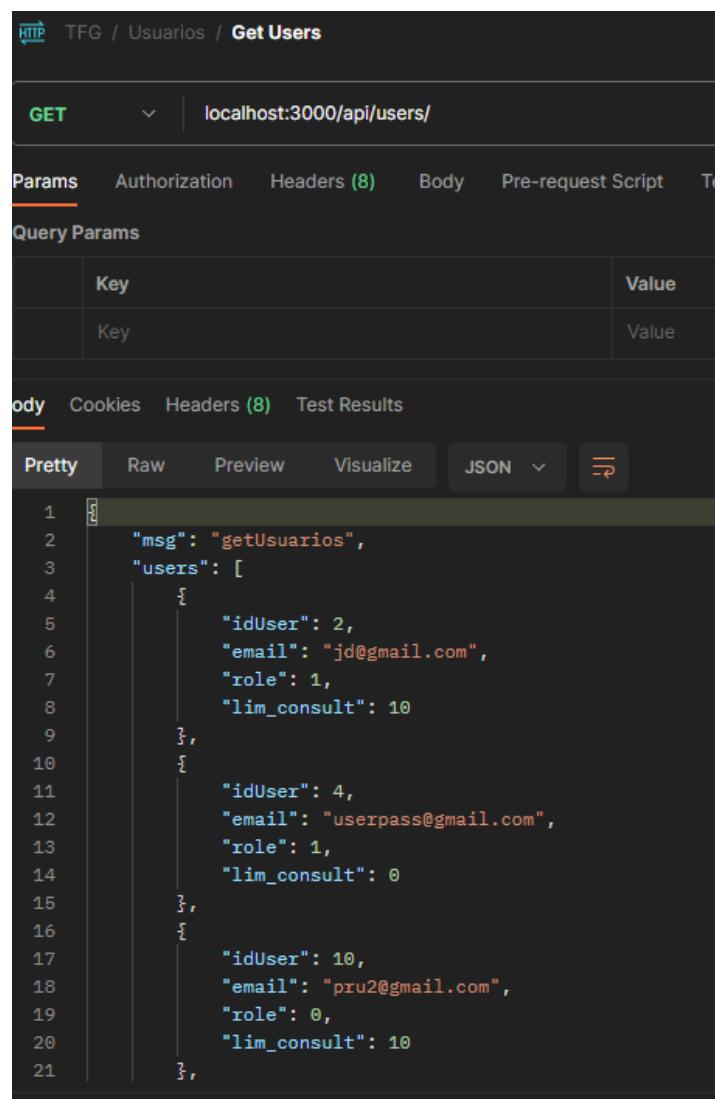
Aprovechando que ya estaba disponible la ruta de login en la API, en esta iteración también se realizó una función para comprobar la validez del token cada vez que se intente acceder a una página privada. De esta forma, se garantiza que no se pueda acceder a la aplicación simplemente añadiendo un token inválido en el almacenamiento local del navegador. Como esta función se declara en el mismo controlador del inicio de sesión, su ruta es la misma que la del login pero añadiéndole `/token` (`localhost:3000/api/login/token`). Cuando se llama, lo primero que hace es comprobar que se ha enviado un token y lo verifica con el secreto para comprobar si el token está firmado correctamente. Cuando se verifica un token, se puede obtener el identificador y el rol del usuario con los que se generó, por lo que, si todo sale bien se comprueba que el usuario con el identificador obtenido existe en la base de datos. Si no existe, se envía un mensaje de error simplemente indicando que el token no es válido, mientras que si no hay ningún error, se vuelve a generar el token para renovar la fecha de validez el tiempo indicado en las variables de entorno, en este caso 24 horas. Gracias a un guard que se ha generado en angular llamado `auth.guard.ts`, se pude llamar a esta función cada vez que se intente acceder a una ruta de la aplicación, por lo que si hay un error, este guard es el encargado de evitar que se realice la navegación.

Por último, también se han creado la ruta y el controlador para los usuarios. De esta forma, se podía comenzar a realizar pruebas con el inicio de sesión con ejemplos de usuarios cada uno con sus propios datos personales. La ruta en la API es `localhost:3000/api/users` y desde ella se puede llamar a las siguientes funciones CRUD de usuarios:

- **GetUsers:** método GET, donde se devuelven todos los usuarios de la base de datos paginados. El número de usuarios que se devuelven por página se indica en el archivo de las variables de entorno y el índice por el que se quiere empezar a devolver los usuarios se envía por parámetro a la petición. En esta llamada simplemente se define la consulta SQL que se va a realizar a la base de datos y se le envía a la función encargada de realizarlo. Si todo sale bien se devuelve un mensaje con los usuarios paginados y se indica el índice y usuarios por página que ha devuelto.
- **GetUserById:** método GET, donde se devuelve un único usuario que tenga el identificador pasado por parámetro. Si no existe un usuario con el id facilitado, se envía un error indicando que no se ha encontrado ningún usuario. En caso de que sí exista, se devuelve en la respuesta de la petición.
- **CreateUsers:** método POST, donde se crean nuevos usuarios en la base de datos. Como en esta aplicación, son los administradores los únicos que pueden crear usuarios, lo primero que se realiza en esta función es comprobar que el rol del usuario que ha realizado la petición sea el de administrador. Seguidamente, se comprueba que el email del nuevo usuario no está ya en uso y se crea una cadena aleatoria para encriptar la contraseña antes de subirla a la base de datos. Las funciones encargadas de generar esta cadena y de encriptar se han obtenido importando la biblioteca *Bcryptjs* en el proyecto. Una vez se realiza todo esto, se inserta el nuevo usuario en la base de datos con el email y contraseña encriptada. El otro atributo de los usuarios, el límite de consultas, por defecto se pone a 10 cuando se insertan.
- **UpdateUsers:** método PUT, donde se actualizan los atributos que se pasan por el cuerpo de la petición. Lo primero que se hace es mirar que exista el usuario con el identificador facilitado y si no hay errores se comienza a montar la consulta SQL para realizar los cambios. En el caso de los usuarios los campos que se pueden actualizar son su email, su rol y su límite de consultas. En el caso de la contraseña, se ha creado una función a parte para evitar que se actualice por error en este método. Entones, se comprueba cuáles de los tres campos que se pueden cambiar se han enviado por el cuerpo de la petición y dependiendo de cuál se envíe, la consulta es de una forma u otra. Cabe destacar que el rol y el límite de las consultas solo puede ser modificado por un administrador, por lo que en la consulta se añadirán estos campos para su actualización únicamente cuando la petición sea realizada por un usuario administrador.
- **DeleteUsers:** método DELETE, donde se elimina el usuario que coincide con el identificador pasado por parámetro. En este método solo se comprueba primero que el usuario a

eliminar exista y si no hay errores se procede con el borrado. Como en la base de datos se indicó que en las relaciones entre las tablas esté la regla de eliminar en cascada, si se elimina un usuario que tiene códigos QR, estos se eliminarán al mismo tiempo. Lo mismo pasa con las llamadas de cada QR del usuario.

- **ChangePassword:** método POST que se encarga de actualizar la contraseña del usuario. En esta función se debe pasar por el cuerpo de la petición la contraseña actual y la nueva por la que se va a sustituir y lo primero que se comprueba es que estos dos campos no coincidan. A continuación, se comprueba que la actual contraseña coincide con la contraseña almacenada en la base de datos. Si no se genera ningún error, lo último que se hace es encriptar la nueva contraseña, de la misma forma que se hace a la hora de crear un usuario, y se genera la consulta SQL para actualizarla.



The screenshot shows a Postman interface with the following details:

- HTTP Method:** GET
- URL:** localhost:3000/api/users/
- Params:** None
- Query Params:** None
- Body:** None
- Headers:** (8)
- Test Results:** None
- Pretty:** Selected
- Raw:** None
- Preview:** None
- Visualize:** None
- JSON:** Selected

```

1
2     "msg": "getUsuarios",
3     "users": [
4         {
5             "idUser": 2,
6             "email": "jd@gmail.com",
7             "role": 1,
8             "lim_consult": 10
9         },
10        {
11            "idUser": 4,
12            "email": "userpass@gmail.com",
13            "role": 1,
14            "lim_consult": 0
15        },
16        {
17            "idUser": 10,
18            "email": "pru2@gmail.com",
19            "role": 0,
20            "lim_consult": 10
21        }
]

```

Figura 31. Prueba de llamada a getUsers en Postman.
(Fuente propia)

6.5. Iteración 5. Creación de ruta y componente de los códigos QR.

Esta iteración tiene como objetivo la creación de las rutas y sus funciones CRUD de los códigos QR almacenados en la base de datos y de su respectivo componente en el frontend. Lo primero que se realizó fue un análisis de la interfaz de la gestión de los códigos QR (Figura 16) para ver lo que requiere y empezar a implementar desde ahí.

6.5.1. Fase de análisis.

Como ya se ha comentado, se necesita una interfaz donde se va a realizar toda la gestión relacionada con los códigos QR. Esta interfaz debe ser accesible y tiene que poder funcionar correctamente con el mínimo de intervención del usuario. Para resolver estas necesidades, se pensó en realizar una tabla donde se agrupan todos los QR creados. Cada fila de esta tabla pertenece a un QR distinto y en ella se puede realizar funciones básicas como activarlos o desactivarlos, editarlos, eliminarlos y descargarlos individualmente. Como puede llegar un punto donde haya demasiados QR creados por un usuario, también se requiere una paginación de estos, los cuales se agrupan de 10 en diez y de esta forma poder navegar entre ellos de forma accesible. Por este mismo motivo, también se hace necesaria la implementación de una barra de búsqueda de códigos QR, para poder acceder a un código en concreto de manera más directa y sencilla. Con todos estos requerimientos en mente y la guía del mockup de la Figura 16, se empezó con la implementación.

6.5.2. Fase de implementación.

Para comenzar a trabajar en esta interfaz, primero se realizó la creación de su router y controlador, como ya se ha realizado con la interfaz del inicio de sesión. Como se hizo en la iteración anterior, se declaró la ruta añadiéndole /qr a la ruta base de la API, por lo que la ruta para realizar todas las peticiones relacionadas con los códigos QR es *localhost:3000/api*. Para atender a estas peticiones, en el router se declaran los métodos GET, para la obtención de los QR; POST, para la creación; PUT, para la modificación; y DELETE, para la eliminación, cada uno con sus validaciones. Estas validaciones son las funciones que se encuentran en la carpeta middleware del backend (Figura 25). Puede que haya una función que no requiera de alguna de estas validaciones como, por ejemplo, en el método GET no es necesario validar los campos ya que en este tipo de peticiones no se envía nada por el cuerpo de la petición. Pero la validación del token es común a todas las peticiones que se realizan a partir de ahora, ya que este componente pertenece a la parte privada de la aplicación.

Una vez todo se valida correctamente, se llaman a las funciones definidas en el controlador, las cuales son:

- **GetQr:** método GET, donde se devuelven todos los códigos QR paginados de un usuario de la base de datos. En caso de que sea un administrador el que realiza la petición, se devuelven todos los códigos del sistema. Al igual que con los usuarios, el número de códigos que se devuelven por página se indica en el archivo de las variables de entorno y el índice por el que se quiere empezar a devolver los códigos se envía por parámetro a la petición. En esta llamada se define la consulta SQL comprobando si el rol del usuario que realiza la petición es administrador. En caso de que no lo sea, se indica en la consulta que solo devuelva los códigos que sean del propio usuario. Si todo sale bien se devuelve un mensaje con los códigos QR paginados y se indica el índice y códigos por página que ha devuelto.
- **GetQrById:** método GET, donde se devuelve un único código QR que tenga el identificador pasado por parámetro. Si no existe un QR con el id facilitado, se envía un error indicando que no se ha encontrado ningún código. En caso de que sí exista, se devuelve en la respuesta de la petición.
- **CreateQr:** método POST, donde se crean nuevos códigos QR en la base de datos. En este caso cualquier usuario puede crear códigos, por lo que en esta función no se comprueba el rol del usuario. Cuando se crea un nuevo QR, se accede a la interfaz de la configuración de este (Figura 17) con los datos por defecto definidos en la base de datos. Por esta razón, no es necesario enviar ningún campo por el cuerpo de la petición. Simplemente es necesario el id del usuario para indicar que es suyo, pero este id se obtiene directamente desde el backend gracias al token. Si no hay ningún error, se incrusta el nuevo código en la base de datos y se devuelve su id generado.
- **UpdateQr:** método PUT, donde se actualizan los atributos que se pasan por el cuerpo de la petición. Como con los usuarios, primero se comprueba si existe el código con el identificador facilitado y si no hay errores se comienza a montar la consulta SQL para realizar los cambios. En este caso, los campos de los QR que se pueden actualizar son su descripción, su nombre y descripción de etiqueta, su fecha de validez y si se encuentra activado o desactivado. Se comprueba los campos que se envían por la petición para hacer la consulta SQL de una forma u otra, como en la función de actualizar los usuarios. En este caso no hay atributos que solo puedan modificar los administradores, por lo que en esta función tampoco se revisa el rol del usuario y una vez montada la consulta, se envía a la base de datos para realizar las modificaciones.
- **DeleteQr:** método DELETE, donde se elimina el código QR que coincide con el identificador pasado por parámetro. En este método solo se comprueba que el código a eliminar exista y si no hay errores se procede con el borrado. Como se comenta anteriormente, si se

elimina un código QR que tenga llamadas, estas también se eliminarán gracias a la regla de eliminar en cascada.

```
// Extrae los campos que se pueden enviar por el cuerpo de la petición para realizar comprobaciones
let { description, tagName, tagDescription, date, activated} = req.body;
let updateQuery = `UPDATE ${process.env.QRTABLE} SET `;

// En este array se van almacenando todos los campos a actualizar
let updateFields = [];

// Dependiendo de los campos que se envíen la query es de una forma u otra.
if(description){
    updateFields.push(`description = '${description}'`);
}
if(tagName){
    updateFields.push(`tagName = '${tagName}'`);
}
if(tagDescription){
    updateFields.push(`tagDescription = '${tagDescription}'`);
}
if(date){
    updateFields.push(`date = '${date}'`);
}
if( activated === 1 || activated === 0 ){
    updateFields.push(`activated = '${activated}'`);
}

// Se unen los campos enviados por la petición con una coma en el caso que haya más de uno
updateQuery += updateFields.join(',');
updateQuery += ` WHERE idQr=${uid}`;

// Se actualiza.
qr = await dbConsult(updateQuery);
```

Figura 32. Comprobación de los campos de un código QR en la función UpdateQr.
(Fuente propia)

A continuación, se comenzó a implementar el componente de la interfaz de la página principal, la gestión de los códigos QR, en el frontend. Para ello, se creó un nuevo componente en angular desde la terminal de la misma forma que se hizo con el componente de inicio de sesión, solo que, en este caso, el componente se encuentra dentro de la carpeta de las páginas privadas de la aplicación. Una vez generado, se aprovechó el código que proporciona la plantilla para empezar a implementar su código HTML, en el archivo *home.component.html*.

La plantilla proporciona una serie de tablas que se usaron para agrupar todos los QR de manera sencilla. La tabla está organizada de la siguiente forma: la primera fila de la tabla se encuentra el nombre de cada columna, es decir, indica qué es lo que muestra cada columna. En la primera columna se indica que ahí se muestran los códigos QR para escanearlos, la segunda columna muestra la fecha de validez de los códigos, la tercera muestra una descripción de estos y en las siguientes columnas se encuentran los botones para activarlos o desactivarlos, imprimirlos,

editarlos y eliminarlos en este orden. En las siguientes filas es donde se muestran todos los códigos QR obtenidos desde la base de datos.

Tus códigos QR						
	Fecha validez	Descripción	Activar/Desactivar	Imprimir	Editar	Eliminar
	8/7/2023	Tercera Prueba: frontend2	<input checked="" type="checkbox"/>			

Figura 33. Estructura de la tabla de los códigos QR.
(Fuente propia)

Para poder obtener los códigos QR de los usuarios almacenados en la base de datos se requiere la creación de otro servicio para encapsular la comunicación con la API sobre toda la lógica relacionada con los códigos QR. Se generó de la misma manera que con el servicio de los usuarios en la iteración anterior y se encuentra en la misma carpeta services. En este servicio se agrupan todas las peticiones a la ruta de los códigos QR. Lo primero que se necesita es la petición que devuelve todos los códigos, por lo que en el servicio se define la función encargada de realizarla. En esta función, simplemente se realiza la petición GET indicándole la ruta y pasando por la cabecera de esta el token del usuario almacenado. El backend contesta con la lista de los QR solicitados si no ha habido ningún error. En el archivo de la lógica del componente, llamado *home.component.ts*, se llama a este servicio para poder realizar la petición y la lista de códigos obtenidos se almacena en un array. Como cada fila de la tabla tiene la misma estructura, se puede iterar a partir del array y de esta forma solo es necesario escribir el código HTML de una fila de la tabla. Angular se encarga de recorrer el array y crear una fila por cada QR almacenado.

Una vez podemos ver los códigos listados en el frontend, se puede pasar a realizar las demás funcionalidades relacionadas con su gestión. Lo siguiente que se realizó, fue la instalación de la librería *angularx-qrcode* [15] a través de npm como ya se ha hecho otras veces. Esta librería es la encargada de generar los códigos QR que se podrán escanear con los dispositivos móviles. Basta con poner la etiqueta *qrcode* en el código HTML del componente y agregarle principalmente los atributos *qrdta*, que indica el enlace que se obtiene al escanearlo y *width*, que indica el tamaño del código QR. El enlace de cada código de la tabla debe redirigir a la página de vista de cada uno, como se puede observar en su mockup de la Figura 14, por lo que cada código contendrá un enlace del estilo (mientras se trabaja en local): *localhost:4200/view/idCódigoQr*.

En cuanto el tamaño de estos, cuando se muestran en cada fila no puede ser muy grande porque romperían la tabla, por eso tienen un tamaño de 60 por 60 píxeles. Este tamaño es muy pequeño para poder escanear los códigos con un dispositivo móvil de manera sencilla, por lo que se ha

implementado un mensaje modal, que aparece al clicar en estos y muestra el QR con un tamaño de 300 por 300 que facilita la lectura de estos. Este mensaje modal se ha realizado con la librería *SweetAlerts2* [16] que permite generar mensajes modales configurables, responsivos y accesibles. Todos los mensajes modales de la aplicación se realizan con esta librería a partir de ahora.



Figura 34. Mensaje modal que muestra el QR con más tamaño generado con SweetAlert2.
(Fuente propia)

En este punto, quedaban las funcionalidades de activar/desactivar, editar, eliminar y crear códigos. Se decidió que la funcionalidad de descarga se realizaría en una fase más avanzada de la implementación, cuando la vista de los códigos QR ya estuviese hecha. En cuanto a crear y editar códigos, como estos botones redirigen a la página de configuración de un código QR (mockup en la Figura 17) se decidieron completar en la siguiente iteración, mientras se creaba este componente. No obstante, ya se podía definir las llamadas a la API, desde el servicio creado anteriormente, para realizar estas peticiones. Como se ha comentado antes, al pulsar en crear un nuevo código, se redirige a la página de configuración de este con datos por defecto, por lo que en el servicio no hay que pasarle ningún campo por el cuerpo de la petición. A la función de actualizar los códigos se le

pasa el formulario con los campos a modificar. En ambas funciones también es necesario pasar el token del usuario.

La función de activar y desactivar simplemente es una llamada a *updateQr* de la API pasándole por el cuerpo de la petición el valor que debe tener el campo *activated* de la base de datos, que indica si un código QR está activado o no. En caso de que esté activado, en la base de datos se indica con un 1 mientras que los que estén desactivados, tendrán un 0. Por defecto, se crean con el estado desactivado. Cuando se pulsa en el botón activar/desactivar, si está activo se le pasa a la petición el valor 0 para que se desactive en la base de datos y viceversa. Si todo sale bien, se le indica con las alertas que la acción se ha realizado correctamente.

Para la eliminación de los códigos, se crea una llamada al servicio y este se encarga de enviar el id del QR que se quiere eliminar a la API. Antes de que se realice la eliminación, se lanza un mensaje modal al usuario para que confirme que desea eliminar el código QR. Si se pulsa en cancelar, no se realiza la eliminación mientras que si se pulsa en eliminar se elimina en la base de datos y se le indica al usuario que se ha eliminado correctamente con una alerta.

La barra de búsqueda y la paginación son otras dos funcionalidades que faltan en esta interfaz, pero ya que en la interfaz de configuración de un código QR también se necesitan para la búsqueda y paginación de las llamadas que tenga, se decidió realizarlas para las dos cuando este último componente se complete. La plantilla también proporciona el código HTML necesario para estas dos funcionalidades.

Por último, también se realizaron las primeras versiones de la cabecera y barra lateral de la aplicación. De esta manera, se puede empezar a obtener la estructura común que tendrá la aplicación en la parte privada. En cuanto a la cabecera, está compuesta por un logo el cual será cambiado más adelante por el de la aplicación y el botón para desplegar o esconder la barra lateral. Por otro lado, la barra lateral está compuesta por las opciones del perfil, donde se encuentran los botones para cambiar la contraseña y cerrar sesión; y las opciones de gestión, donde se encuentran la página de gestión de los códigos QR y la gestión de los usuarios para los administradores. Se irán añadiendo opciones a la barra lateral según se vayan haciendo. Desde el archivo de la lógica de la aplicación, llamado *app.component.ts* se controla que la cabecera y la barra lateral solo se muestren en la parte privada de la aplicación, para que no se muestre por error en las páginas públicas como por ejemplo la del inicio de sesión.

Con todas las funcionalidades realizadas en esta iteración, la página de gestión de los códigos QR queda de la siguiente forma.

The screenshot shows the NiceAdmin application's QR code management interface. On the left, there is a sidebar with 'GESTIÓN' and 'PERFIL' sections. Under 'GESTIÓN', there are dropdown menus for 'Components', 'Forms', 'Tables', 'Charts', and 'Icons'. Under 'PERFIL', there are links for 'Cambiar contraseña' and 'Cerrar sesión'. The main content area is titled 'Códigos QR' and shows a search bar with 'Buscar códigos QR...' and a magnifying glass icon. A blue button '+ Añadir QR' is located in the top right. Below the search bar, the title 'Tus códigos QR' is displayed. A table lists six QR codes with columns: 'Fecha validez', 'Descripción', 'Activar/Desactivar' (with a toggle switch), 'Imprimir' (green icon), 'Editar' (blue icon), and 'Eliminar' (red icon). The table rows contain the following data:

Fecha validez	Descripción	Activar/Desactivar	Imprimir	Editar	Eliminar
8/7/2023	Tercera Prueba: frontend2	<input checked="" type="checkbox"/>			
27/6/2023	Se va a probar update en este código Qr	<input checked="" type="checkbox"/>			
9/7/2023	Otro QR de prueba	<input checked="" type="checkbox"/>			
4/7/2023	'Descripción del código QR'	<input checked="" type="checkbox"/>			
5/7/2023	QR con pruebas realizadas	<input checked="" type="checkbox"/>			
7/7/2023	Descripción del código QR	<input checked="" type="checkbox"/>			

Figura 35. Interfaz de gestión de códigos QR implementado en la aplicación.
(Fuente propia)

6.6. Iteración 6. Creación del componente de configuración de un código QR.

Esta iteración tiene como objetivo la implementación de la interfaz de configuración de un código QR, la cual se puede acceder pulsando en el botón de editar un QR o añadiendo uno nuevo desde la página principal como se puede ver en la Figura 35. Como en la iteración anterior, primero se realizó un análisis para ver los requerimientos de esta interfaz con la guía de su mockup de la Figura 17.

6.6.1. Fase de análisis.

En esta interfaz es donde se puede realizar todas las modificaciones y ajustes a un código QR, por lo que se necesita que todas estas operaciones se puedan realizar de manera clara y sin muchas complicaciones. Además, como un único QR puede tener cero o muchas llamadas, también se necesita poder tener una gestión de estas de forma parecida a la que se tiene en la página de inicio con los QR. Por esta razón, en esta iteración es necesario también la implementación de las rutas y funciones CRUD, que tienen que ver con las llamadas, en la API de la aplicación.

Esta iteración tendrá una estructura similar a la anterior en cuanto a la implementación: primero se realizan todas las rutas y funciones necesarias para el correcto funcionamiento en el backend y seguidamente se realiza el componente y el servicio para las peticiones de las llamadas en el frontend.

Por último, como en esta interfaz hay otra gestión, también es necesaria la barra de búsqueda y paginación que se planteó con la interfaz principal. Como su funcionamiento es el mismo en las dos interfaces, en esta iteración también se cubre la implementación de la barra de búsqueda adaptándola a las necesidades de cada componente. La paginación, por otro lado, se realiza más adelante.

Cabe destacar un cambio que se realizó en esta iteración a partir de una funcionalidad creada en la anterior. Como ya se sabe, los códigos QR que se encuentran en la tabla se pueden escanear con el mensaje modal que aparece si se pulsa en ellos. No obstante, si se desactivaban el mensaje modal ya no se mostraba por lo que no se podían escanear a menos que se volviesen a activar. Esto suponía un problema a la hora de gestionar los códigos QR ya que, si se quieren realizar pruebas con uno antes de que se muestren, lo normal es tenerlo desactivado hasta que se finalicen las pruebas, pero hasta ahora no se podría leer hasta que no se active. Por esta razón, a partir de esta iteración, aunque los códigos se encuentren desactivados, estos se podrán escanear únicamente por los dueños y administradores para poder realizar las pruebas que se deseen.

6.6.2. Fase de implementación.

Como en las iteraciones anteriores, primero se realizó la creación del router y controlador para las llamadas de cada código QR. En este caso, la ruta declarada para estas funciones en la API es *localhost:3000/api/consult*. Como con los códigos QR, en el router se declaran los métodos GET, POST, PUT y DELETE que atenderán a las peticiones y las validaciones que requiere cada uno, siendo común para todos, la validación del token. Una vez todo está validado de forma correcta, se llaman a las funciones del controlador:

- **GetConsult:** método GET, donde se devuelven todas las llamadas de un código QR paginadas de la base de datos. Al igual que con los métodos de los códigos QR y usuarios, el número de llamadas que se devuelven por página se indica en el archivo de las variables de entorno y el índice por el que se quiere empezar a devolver las llamadas se envía por parámetro a la petición. Si no se envía nada, por defecto es 0. En esta función no se comprueba el rol del usuario que realiza la petición, por lo que la consulta SQL simplemente necesita el identificador del código QR del que se va a devolver sus llamadas. Este identificador se debe facilitar pasándolo por parámetro a la hora de realizar la petición. Si

todo sale bien se devuelve un mensaje con las llamadas paginadas y se indica el índice y número de llamadas por página que ha devuelto.

- **GetConsultById:** método GET, donde se devuelve una única llamada que tenga el identificador pasado por parámetro. Funciona como con los usuarios y códigos QR: si no existe la llamada en la base de datos se envía un error indicando que no se ha encontrado mientras que si existe, se devuelve en la respuesta de la petición.
- **CreateConsult:** método POST, donde se crean nuevos códigos QR en la base de datos. En este caso tampoco se comprueba el rol del usuario ya que cualquiera puede crear llamadas en la aplicación. De la misma forma que pasa con los códigos QR, cuando se crea una nueva llamada, se accede a la interfaz de la configuración de esta (Figura 18) con los datos por defecto definidos en la base de datos. Por esta razón, en esta petición tampoco es necesario enviar ningún campo por el cuerpo a excepción del identificador del código QR al que pertenece la llamada. Si no hay ningún error, se incrusta la nueva llamada en la base de datos y se devuelve su id generado.
- **UpdateConsult:** método PUT, donde se actualizan los atributos que se pasan por el cuerpo de la petición. Como hasta ahora, primero se comprueba si existe la llamada con el identificador facilitado y se comienza a montar la consulta SQL para la modificación. En este caso, los campos de las llamadas que se pueden actualizar son su nombre, el token necesario para realizar la petición a la API de SmartUniversity, las fechas desde y hasta, los filtros que se aplican a la petición, el tipo de representación y si se encuentra activada o desactivada. Como con la actualización de los usuarios, se comprueba los campos que se envían por la petición para hacer la consulta SQL de una forma u otra. En esta función tampoco hay que comprobar el rol del usuario, por lo que una vez se monta la consulta, se envía a la base de datos y se actualiza la llamada.
- **DeleteConsult:** método DELETE, donde se elimina la llamada que coincide con el identificador pasado por parámetro. En este método solo se comprueba que el código a eliminar exista y si no hay errores se procede con el borrado. En este caso, si se elimina una llamada no se elimina nada más aparte ya que no hay más relaciones en la base de datos.

The screenshot shows a POSTMAN interface with the following details:

- HTTP Method:** GET
- URL:** localhost:3000/api/consult?idQr=3
- Params:** idQr = 3
- Body:**
 - Pretty
 - Raw
 - Preview
 - Visualize
 - JSON

```

1  {
2      "msg": "getConsult",
3      "consult": [
4          {
5              "idConsult": 1,
6              "name": "Llamada 1 frontend",
7              "token": "1111111112",
8              "dateFrom": "2023-07-12T00:25:44.000Z",
9              "dateTo": "2023-07-12T00:26:00.000Z",
10             "filters": "{}",
11             "activated": 1,
12             "qrCode": 3
13         },
14         {
15             "idConsult": 6,
16             "name": "Nombre de la llamada",
17             "token": "Token de la llamada",
18             "dateFrom": "2023-07-06T12:21:30.000Z",
19             "dateTo": "2023-07-06T12:21:30.000Z",
20             "filters": "",
21             "activated": 1,
22             "qrCode": 3
23         }
24     ],
25     "page": {
26         "desde": 0,
27         "registrop": 10,
28         "total": 2
29     }
30 
```

Figura 36. Prueba de llamada a getConsult desde Postman.
(Fuente propia)

Una vez la parte del backend está completa, se procede a la implementación de la interfaz de la configuración de un código QR en la parte del frontend. Como se ha hecho hasta ahora, primero se creó un nuevo componente de la misma forma que en la iteración anterior, ya que esta interfaz

también pertenece a la parte privada de la aplicación. Esta interfaz está estructurada en dos partes: la parte de la configuración del propio QR y la parte donde se listan todas las llamadas que realiza.

Por un lado, en la primera parte es donde se pueden realizar todas las modificaciones necesarias al código QR y también se pueden activar o desactivar y descargarlos. Para ello, esta parte se encuentra dividida en dos columnas principales. En la primera columna se encuentra el código QR con un tamaño más grande que el de la página principal para poder escanearlo directamente y en la parte inferior de este se encuentran los botones para activarlo o desactivarlo y descargarlo. En la segunda columna se encuentra el formulario que permite introducir los cambios que se desean aplicar al QR. Cuando se entra en esta interfaz, los campos de este formulario están llenados con la información almacenada en la base de datos y se encuentran en modo de solo lectura, es decir, no se pueden modificar. Para poder actualizar los campos, se debe pulsar en el botón de editar que se encuentra al lado del título del formulario y los campos se podrán modificar. Esto simplemente se ha hecho aplicando o eliminando el atributo *disabled* en los inputs del formulario según se quiera modificar o no los campos. Además, cuando se pulsa en el botón de edición, en la parte inferior del formulario aparecen dos botones para cancelar, en caso de no querer realizar ninguna modificación finalmente, y guardar, en caso de querer enviar la petición de modificación. Si se pulsa en el botón de cancelar, a los inputs del formulario se les vuelve a aplicar el atributo *disabled* para que solo se puedan leer y no cambiar. Si se pulsa en el botón de guardar, se realiza la petición al servicio de los códigos QR y este se encarga de enviar los campos a modificar al backend de la aplicación. Si todo se realiza correctamente, se le informa al usuario con una alerta. Lo mismo pasa si sucede algún error en el proceso.

Datos del QR

Nombre/Descripción	Nombre de etiqueta
Tercera Prueba: frontend2	<input type="text" value="tres"/>
	Descripción de etiqueta
	<input type="text" value="tres"/>
	Fecha de validez
	<input type="text" value="07/07/2023"/> <input type="button" value=""/>
	<input type="button" value="Cancelar"/> <input type="button" value="Guardar"/>

Figura 37. Formulario en modo edición.
(Fuente propia)

Por otro lado, en la segunda parte es donde se puede realizar la gestión de las llamadas que tiene cada código QR de forma similar a la de los códigos QR en la página principal. En esta parte también se ha aprovechado el código HTML de las tablas que proporciona la plantilla. Esta tabla tiene la misma estructura que la de los códigos QR: la primera fila de la tabla indica qué es lo que muestra cada columna, solo que en esta ocasión se necesitan menos columnas. La primera columna indica el nombre de la llamada y las tres siguientes son los botones para activar o desactivar, editar y eliminar en ese orden. Las siguientes filas muestran todas las llamadas obtenidas desde la base de datos.

Llamadas del código QR		+ Añadir Llamada	
Nombre	Activar/Desactivar	Editar	Eliminar
Llamada 1 frontend	<input checked="" type="checkbox"/>		

*Figura 38. Estructura de la tabla de las llamadas.
(Fuente propia)*

Para la obtención de la lista de las llamadas es necesario la creación de otro servicio que encapsule todas las funciones que realizan las peticiones al backend relacionadas con estas. Se generó el servicio llamado *consults.service.ts* de la misma forma que los demás servicios y se almacena en la misma carpeta. La primera función del servicio es la que realiza la petición GET para obtener la lista de las llamadas que tiene un código QR. Como con el GET de los QR, esta función simplemente realiza la petición GET indicando la ruta de la API para las llamadas y pasando el token por la cabecera. Si todo sale bien, se obtiene la lista de las llamadas solicitadas. De forma similar, en el archivo de la lógica del componente, llamado *qr.component.ts*, se llama a esta función del servicio y las llamadas obtenidas se almacenan en un array para poder iterar en la tabla y escribir el código HTML de una fila de la tabla como se hizo en la iteración anterior.

El botón de activar y desactivar tiene el mismo funcionamiento que el de los códigos QR. Se realiza una llamada a *updateConsult* en la API pasándole el valor que va a tener el campo *activated*: 1 si se va a activar o 0 si se va a desactivar. Si la función se realiza correctamente, se le informa al usuario con las alertas.

El botón de eliminar las llamadas también es el mismo que con los códigos QR. Se realiza una llamada al servicio y este realiza una petición a *deleteConsult* facilitándole el id de la llamada a eliminar. En esta interfaz, también se lanza un mensaje modal antes de realizar la petición para la confirmación la eliminación de las llamadas.

Los botones para editar y añadir llamadas redirigen a la página de configuración de una llamada (mockup en la Figura 18) la cual se realiza en la siguiente iteración por lo que estos botones funcionarán cuando esta página se realice. No obstante, como se comentó en la iteración anterior, los botones de editar y añadir códigos QR redirigen a la interfaz realizada en esta iteración por lo que ya se pueden conectar. En el caso de añadir códigos QR, se llama al servicio que realiza la petición a la función *createQR* y esta crea una nueva entrada en la tabla de los códigos QR de la base de datos con los datos por defecto y devuelve su identificador generado. Este identificador es usado por la lógica del componente para redirigir al usuario a la página de configuración del código QR creado. Por otro lado, en el caso de editar códigos QR, se obtiene el identificador del código a editar del array donde se almacenan los QR de la tabla y se redirige a la página de configuración de ese código con su identificador. En los dos casos, cuando carga la interfaz de configuración se realiza la llamada a la obtención del código QR a partir de su identificador (*getQrById*) para poder mostrar todos sus datos, y a la llamada para obtener todas las llamadas del QR (*getConsult*).

Por último, queda la implementación de la barra de búsqueda. Ha sido necesaria la modificación de los métodos GET que devuelven la lista de los códigos QR y la de sus llamadas para su funcionamiento. Simplemente se ha añadido una condición en las dos funciones para comprobar si se envía una consulta de búsqueda por parámetro a la hora de realizar la petición. Si la hay, la consulta SQL se modifica para que añada que busque los QR y llamadas que coincida con la descripción, en caso de los QR, o con el nombre, en el caso de las llamadas con la ayuda del operador LIKE.

En el frontend, se ha añadido un input que funciona de barra de búsqueda en las dos tablas. Cuando cargan los dos componentes en las que se encuentran, desde el archivo de la lógica de estos, se le añaden dos eventos: cuando se pulsa en una tecla y cuando se deja de pulsar en esa tecla. Se hace uso de un temporizador para que, al escribir en la barra de búsqueda, no se hagan peticiones a la base de datos por cada letra que se añada a la consulta. El temporizador espera un segundo antes de realizar la búsqueda. Si se pulsa en alguna tecla, el temporizador se reinicia y cuando se deja de pulsar en esa tecla vuelve a contar. Si en un segundo no se añaden más letras, se realiza la búsqueda. De esta forma se evita realizar peticiones innecesarias que podrían afectar al rendimiento de la aplicación.

```

// Eventos para hacer que la búsqueda se haga al pasar un tiempo solo y no hacer una petición cada vez que se introduce una letra
const searchFieldElement = this.renderer.selectRootElement( '#searchField' );

searchFieldElement.addEventListener('keyup', () =>{
  clearTimeout(this.timer);
  this.timer = setTimeout(() =>{
    this.search();
  }, 1000)
});

searchFieldElement.addEventListener('keydown', () =>{
  clearTimeout(this.timer);
});

```

*Figura 39. Eventos para el temporizador de la barra de búsqueda.
(Fuente propia)*

Con todas las funcionalidades realizadas en esta iteración, la página de configuración de un código QR queda de la siguiente forma.

The screenshot shows a web-based configuration interface for a QR code. At the top, there's a header with a menu icon and the title "Código QR". Below the header, a breadcrumb navigation shows "Home / Dashboard". A large section titled "Configuración del QR" displays a QR code with a black border. To the right of the QR code, there are input fields for "Nombre/Descripción" containing "Tercera Prueba: frontend2", and checkboxes for "Nombre de etiqueta" (checked, with value "tres") and "Descripción de etiqueta" (unchecked, with value "tres"). There's also a date field for "Fecha de validez" set to "07/07/2023". Below this section, there's a toggle switch and a print icon. At the bottom, there's a table titled "Llamadas del código QR" with columns for "Nombre", "Activar/Desactivar", "Editar", and "Eliminar". It lists two entries: "Llamada 1 frontend" (activated, edit icon blue, delete icon red) and "Nombre de la llamada" (activated, edit icon blue, delete icon red). Above the table is a search bar with placeholder "Buscar códigos QR..." and a "Buscar" icon, along with a blue button labeled "+ Añadir Llamada".

*Figura 40. Interfaz de configuración de un código QR.
(Fuente propia)*

6.7. Iteración 7. Creación del componente de configuración de las llamadas.

Esta iteración tiene como objetivo la implementación de la interfaz de configuración de las llamadas que realiza un código QR, a la cual se puede acceder desde la interfaz de configuración de un código QR. Esta interfaz se ha realizado en la iteración anterior como se puede ver en la Figura 40 y la idea es que, si se pulsa en el botón de crear o editar una llamada del QR, se redirija a esta nueva interfaz y poder configurar las llamadas. Primero se realiza un análisis para ver los requerimientos que precisa esta interfaz con la guía de los mockups de la Figura 18 y de la Figura 19.

6.7.1. Fase de análisis.

En esta interfaz es donde se puede realizar todas las modificaciones y ajustes a las llamadas de un código QR, por lo que, como en la interfaz de la iteración anterior, se necesita que todas estas operaciones se puedan realizar de forma clara y concisa. En este caso, la configuración que se puede realizar es más grande. Esta interfaz debe permitir configurar el nombre de la llamada, el token necesario para poder realizar las peticiones a la API de Smart University, el modo de introducir las fechas ya sea de manera absoluta o de manera relativa, la cantidad de datos que se quiere obtener ya sean todos los datos o un máximo, mínimo o el último dato en el rango de fechas seleccionado, los filtros que se desean aplicar a la petición y, por último, el tipo de representación en el que se muestran los datos obtenidos en la interfaz de visualización de un código QR.

Ya que esta interfaz solo necesita trabajar con los datos de las llamadas, y su servicio se realizó en la iteración anterior, en esta iteración, se comienza directamente con la creación e implementación del componente en el frontend.

Mientras se trabajaba en esta interfaz, se decidió cambiar la forma de estructurar el formulario de la que está en los mockups para evitar lo máximo posible el scroll vertical. Como se puede apreciar en los mockups, por ejemplo, en el de la Figura 18, la interfaz está dividida en varias partes. Esto se hizo así para tratar guiar en la configuración de las llamadas en los distintos pasos que forman el formulario. Pero al realizarse de esta forma, a la hora de implementarlo en el componente, estas partes ocupaban o mucho o casi nada de espacio por lo que la estructura de la interfaz quedaba muy poco consistente y requería desplazar mucho la pantalla verticalmente para poder apreciar todas las opciones de configuración. Por esta razón, se decidió agrupar más el formulario, aprovechando al máximo los espacios libres para evitar este desplazamiento y que la configuración de las llamadas sea más sencilla y coherente.

Por último, en esta interfaz se ha añadido un nuevo atributo a la tabla de las llamadas en la base de datos llamado *chart* para poder almacenar el tipo de representación deseado. Este atributo se almacena en la base de datos con un número, el cual indica el tipo de gráfica seleccionada para la representación. De momento, en la aplicación hay disponibles cuatro tipos de representación: gráfica lineal, representada con un 0 en la base de datos; gráfica de barras, representada con un 1; gráfica de gauge, representada con un 2 y simplemente un número, representado con un 3. De esta forma, si se añaden más modos de representación, en la base de datos basta con indicarlos con los números que siguen en la lista. Las dos primeras gráficas están diseñadas para utilizarlas cuando se obtenga más de un dato mientras que las dos últimas gráficas se utilizarán cuando se obtenga el dato máximo, mínimo o último.

6.7.2. Fase de implementación.

Como se ha comentado en la fase de análisis, la implementación de esta iteración comienza con la creación de un nuevo componente en el proyecto. Se crea de la misma forma que siempre, en la carpeta de los componentes de la parte privada de la aplicación y se obtienen los archivos necesarios para el componente el cual se llama *consult-form.component*.

Una vez creado el componente, se comenzó a construir el formulario. Lo primero que se añadió fueron los inputs del nombre y el token de la consulta ya que son del mismo tipo texto. Los siguientes campos que se añadieron son los de la fecha. Se barajó varias posibilidades para permitir elegir entre introducir las fechas de forma absoluta o de forma relativa. Al final se decidió utilizar dos pestañas, una para cada forma de seleccionar las fechas. Por defecto, siempre sale seleccionada la pestaña de la fecha absoluta, pero en cualquier momento se puede cambiar a la otra pestaña. Cada pestaña tiene dos inputs, en la de la pestaña absoluta, hay dos de tipo *datetime* los cuales permiten introducir las fechas acompañadas de una hora en concreto. Si la fecha ‘desde’ es mayor que la fecha ‘hasta’ a la hora de intentar actualizar la llamada, no se realiza la llamada al servicio encargado de hacer la petición para guardar los cambios y se usa una alerta para informar al usuario del error. La pestaña de la fecha relativa tiene un input de tipo *number* para introducir la cantidad de tiempo que se le quiere restar a la fecha en la que se hace la petición, y otro de tipo *select* para seleccionar la unidad del tiempo que se quiere usar (segundos, minutos, horas o días).

La siguiente opción que se encuentra es la selección del tipo de representación para los datos que se van a solicitar. Es un input de tipo *select* en el que se muestran las gráficas disponibles para la representación. De esta forma, se pueden ir añadiendo más gráficas según se necesite. Esta lista de gráficas está acompañada por una imagen que cambia según la opción seleccionada para ver una aproximación de cómo se vería el resultado. La lista puede variar según la cantidad de datos que se

van a solicitar para esas fechas: si se quiere obtener todos los datos disponibles entre esas fechas, las gráficas disponibles son, hasta el momento, la de líneas y la de barras ya que son las que mejor se adaptan para representar una gran cantidad de datos. Si lo que se quiere es obtener el mayor, el menor o el último dato en ese rango de fechas, las gráficas que aparecen en la lista son la gráfica de gauge, o simplemente el valor del dato que se quiere mostrar.

Selecciona la gráfica para representar los datos

Gráfica de gauge

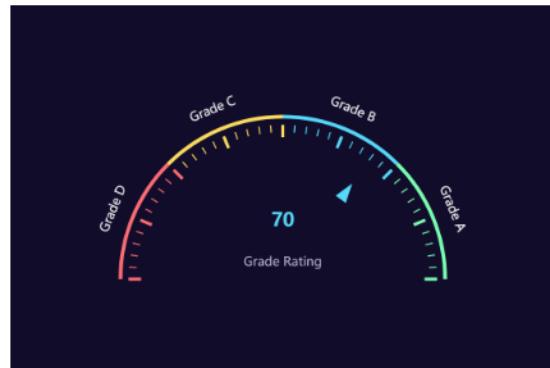


Figura 41. Select de la representación de los datos con la gráfica de gauge seleccionada.
(Figura propia)

Por último, se encuentra la opción de, como se ha comentado, seleccionar la cantidad de datos que se quieren obtener en las fechas indicadas. Se trata de otro *select* pero con 4 opciones: todos los datos, el dato con el valor máximo, el dato con el valor mínimo o el último dato disponible en esas fechas. Dependiendo de la opción seleccionada, a continuación, se pueden introducir los filtros que se le aplicarán a la petición, de una forma u otra. Si se selecciona la opción de todos los datos, aparecerá una sección en la que se muestra una lista con los filtros que la llamada tiene en esos momentos (o vacía si se acaba de crear). En esta sección se pueden ir añadiendo o eliminando los filtros que se deseen hasta un máximo de 12, que es la cantidad máxima de filtros que acepta Smart University (uid, description_origin, lat, lon, organizationid, alias, cota, description, metric, name, origin, typemeter).

Cantidad de datos

Filtros +

Filtro 1	<input type="text" value="uid"/> <input type="text" value="MLU00040001"/>	Delete
Filtro 2	<input type="text" value="name"/> <input type="text" value="15m"/>	Delete
Filtro 3	<input type="text" value="lat"/>	Delete
Filtro 4	<input type="text" value="lon"/>	Delete
Filtro 5	<input type="text" value="description"/>	Delete

Figura 42. Lista con los filtros de las llamadas.
(Fuente propia)

Por otro lado, si se selecciona el máximo, mínimo o el último dato, la sección cambia y aparecen únicamente dos campos pertenecientes al identificador del sensor del que se quiere extraer el dato y la magnitud en el que se mide. Estos dos campos son obligatorios y no se podrán salvar los cambios hasta que no se introduzca un valor en los dos inputs.

Para añadir, se han declarado observables por cada formulario creado en la lógica de este componente. En total hay 4 formularios que se usan en la interfaz. El primer formulario es el que se encarga de almacenar los datos principales que requieren las llamadas y que se introducen en los inputs comentados anteriormente. El segundo formulario se encarga de agrupar los filtros que se van añadiendo si se ha seleccionado la opción de solicitar todos los datos en las fechas elegidas y posteriormente, cuando se va a actualizar la llamada, todos los filtros aplicados se añaden al primer formulario que es el que se envía al servicio de las llamadas para que este lo utilice en el cuerpo de la petición a la API. En la base de datos se guardan los filtros aplicados como un json siendo la clave el nombre del filtro a aplicar y el valor lo que se introduce en los inputs. Antes de actualizar la llamada, se monta un json recorriendo todos los filtros seleccionados y se guardan en el campo llamado *filters* del primer formulario.

El tercer formulario es parecido al anterior ya que se encarga de almacenar los valores introducidos en los dos campos que hay que rellenar si se selecciona la opción de solicitar el máximo, el mínimo o el último dato. Antes de actualizarse la llamada estos campos también se añaden al primer formulario en el campo *filters* para que se almacenen en la base de datos. En este caso, el json se monta con solo dos claves uid (identificador) y name (magnitud), por lo que dependiendo de la cantidad de datos que se quieran obtener, varía la forma de hacer el json de los filtros.

```
// Montar el campo de los filtros.
let json = `{

  if(Number(this.firstForm.value.operation) > 1){

    // Se comprueba que se hayan introducido los campos necesarios
    if(!this.operationForm.valid){
      this.alertService.error("Los campos UID y Magnitud deben tener un valor");
      return;
    }

    Object.keys(this.operationForm.controls).forEach((key, index) => {
      json += `${key}":"${this.operationForm.get(key)?.value}`;

      if(index !== Object.keys(this.operationForm.controls).length - 1){
        json += ',';
      }
    });
  }
  else{
    Object.keys(this.filterForm.controls).forEach((key, index) => {
      json += `${key}":"${this.filterForm.get(key)?.value}`;

      if(index !== Object.keys(this.filterForm.controls).length - 1){
        json += ',';
      }
    });
  }

  json += `}`;
  // Se añade este campo al primer formulario que es el que se envia al update
  this.firstForm.setControl('filters', new FormControl(json));
}
```

*Figura 43. Código donde se realiza el json de los filtros.
(Fuente propia)*

Por último, el cuarto formulario se encarga de almacenar el tiempo que se le quiere restar al tiempo en el que se realiza la petición si se selecciona introducir las fechas de forma relativa. La cantidad de tiempo introducida ya sea en segundos, minutos, horas o días, se trasforma a milisegundos para poder restárselo al tiempo actual en ese momento. Una vez se realizan los cálculos y se crean las nuevas fechas con los resultados, se modifican los campos que almacenan las dos fechas en el primer formulario.

Como se ha comentado al principio, los observables que se han declarado detectan cualquier cambio que se realicen en estos formularios para que el botón de guardar los cambios se muestre disponible y se pueda interactuar con él, ya que, cuando no hay ningún cambio, este sale deshabilitado para no realizar operaciones innecesarias.

Con todas las funcionalidades realizadas en esta iteración, la página de configuración de las llamadas que realiza un código QR queda de la siguiente forma.

The screenshot displays the configuration interface for a call. At the top, there are fields for 'Nombre' (Name) containing 'Llamada 1 frontend' and 'Cantidad de datos' (Data Type) set to 'Todos'. Below this is a 'Token' field containing a long string of characters. To the right, there's a section for 'Filtros' (Filters) with five filter slots. The first slot has 'uid' selected with value 'MLU00040001'. The second slot has 'name' selected with value '15m'. The third slot has 'lat' selected. The fourth slot has 'lon' selected. The fifth slot has 'descripti' selected. Below the filters is a section for date selection with 'Fecha absoluta' (Absolute Date) selected, showing 'Desde' (From) as '24/07/2023 10:22:52' and 'Hasta' (To) as '24/07/2023 10:27:52'. A bar chart is shown below these controls. At the bottom right are 'Cancelar' (Cancel) and 'Guardar' (Save) buttons.

*Figura 44. Interfaz de la configuración de las llamadas
(Fuente propia)*

6.8. Iteración 8. Gestión de usuarios y primeras peticiones a Smart University.

Esta iteración tiene como objetivo la implementación de los componentes necesarios para realizar tanto el listado y gestión de los usuarios como la creación y edición de estos. También se propone realizar las primeras peticiones a la API de Smart University y recibir los primeros datos para, en la siguiente iteración, implementar la interfaz de vista de los códigos QR, donde se visualizarán todos estos datos. Como en todas las iteraciones hasta ahora, primero se realiza un análisis para obtener los requerimientos que necesitan estas interfaces.

6.8.1. Fase de análisis.

En la interfaz del listado de los usuarios se va a realizar una gestión de estos, por lo que los requisitos para implementarla serán muy parecidos a los de la interfaz de la gestión de los códigos QR (Figura 35) y de la interfaz de gestión de las llamadas de un QR (Figura 40). Por esta razón, los elementos que se utilizarán para implementarla se obtendrán de estos componentes y se adaptarán para poder realizar una gestión de los usuarios de forma correcta.

En cuanto a las interfaces de creación y edición de usuarios, en realidad son iguales solo que en el caso de la creación, el formulario aparecerá vacío mientras que en el de la edición, el formulario estará relleno con los datos del usuario a editar. El formulario que compone estas dos interfaces se puede observar en el mockup de la Figura 20, donde se solicita el email y la contraseña del usuario a crear y el límite de códigos QR que puede crear el usuario. Si no se desea aplicar ningún límite, basta con poner un 0 en ese campo. Esta opción está indicada en el formulario. También se ha decidido poner la opción seleccionar el rol del usuario en la plataforma en estas interfaces. De momento, como se ha comentado anteriormente, solo están contemplados dos roles, el básico y el administrador, por eso al principio se implementó un input de tipo check para otorgar o eliminar los permisos de administrador. No obstante, como en un futuro se pueden requerir más roles, se ha decidido utilizar un input de tipo select para poder elegir entre todos los posibles roles que se añadan a la aplicación.

Por otro lado, para empezar a obtener los primeros datos de Smart University, es necesaria la creación de un nuevo servicio que se encargue de realizar las peticiones necesarias diferenciando el tipo de dato solicitado en cada caso. Desde un principio, se pensó en realizar el servicio de forma que fuera este el que se encargara de realizar las peticiones a la API de Smart University directamente. El problema de esto es que las peticiones a esta API requieren que se facilite el token necesario para poder recibir los datos a través de la url de la misma. Esto causa que este token pueda ser visto por cualquier usuario y que lo pueda obtener fácilmente por lo que supondría una brecha en la seguridad de la aplicación. Para solucionarlo, se cambió el servicio para que las peticiones que realice sean a la API de la propia aplicación y que sea esta la encargada de realizar todas las operaciones y peticiones necesarias y que devuelva todos los datos requeridos. De esta forma, todos los datos necesarios para realizar las llamadas, como el token, no son visibles para los usuarios. Por esta razón, en esta iteración también se realiza la creación de una nueva ruta y de su controlador en la API de la aplicación.

Por último, esta iteración también se aprovechó para añadir la paginación en la gestión de los códigos QR, en las de las llamadas que tiene cada código y en la de los usuarios implementada en

esta iteración. Además, también se arregló un problema que había a la hora de obtener las fechas de las llamadas de la base de datos de la aplicación. Se devolvían dos horas por debajo de las que estaban almacenadas en la base de datos, por lo que se hizo uso de la librería *date-fns* [17] para así obtener las fechas en el formato y zona horaria de forma correcta. Esta librería proporciona un conjunto de herramientas para manipular fechas de JavaScript en un navegador.

6.8.2. Fase de implementación.

Como se comenta en la fase de análisis, el componente de la gestión de los usuarios es prácticamente igual a los otros componentes de gestión que tiene la aplicación. Como siempre, primero se genera el componente, llamado *users*, dentro de la carpeta de la zona privada de la aplicación, ya que esta gestión solo podrá ser realizada por los administradores. A su vez, este componente se almacena en una carpeta en la que se agrupan los componentes que se realizan en esta iteración, ya que todos están dedicados a los usuarios. Una vez generados los archivos del componente, se procede a copiar el código de la tabla de cualquiera de las ya implementadas anteriormente y se adapta a los datos de los usuarios. En este caso, la tabla está formada por cuatro columnas en las que se indican el email del usuario, su rol en la aplicación y los botones para editarlos o eliminarlos en este orden.

Para poder mostrar todos los datos de cada usuario en esta tabla es necesario la llamada a un servicio que devuelva todos los usuarios registrados en la base de datos. Para ello, se aprovechó el servicio de los usuarios creado en la iteración 4 (*user.service.ts*) y se añadieron las todas las funciones necesarias para llamar a las funciones CRUD de los usuarios en la API de la aplicación. Como en las otras tablas, una vez se obtienen todos los usuarios, estos se almacenan en un array en el archivo de la lógica de este componente (*users.component.ts*) para que de esta forma sea la tabla la que itere entre todos los usuarios obtenidos y los muestre con sus respectivos datos.

Lista de usuarios			
Inicio / Gestión de usuarios <div style="float: right; margin-right: 10px;"> <input placeholder="Buscar usuarios..." type="text"/> 🔍 </div>			
Crear nuevo usuario			
Usuarios			
Email	Rol	Editar	Eliminar
usu@gmail.com	user	📝	🗑
jd@gmail.com	admin	📝	🗑
userpass@gmail.com	admin	📝	🗑
pruUpdate@gmail.com	user	📝	🗑
pru2@gmail.com	user	📝	🗑
formusu@gmail.com	user	📝	🗑
cambioApi@gmail.com	admin	📝	🗑
josedanielnemejerez@gmail.com	admin	📝	🗑

Figura 45. Tabla de gestión de los usuarios de la aplicación.
(Fuente propia)

El botón de eliminar a un usuario hace lo mismo que en las demás tablas: al pulsar en él lanza un mensaje modal para confirmar la eliminación del usuario y si se confirma, se realiza una llamada a la función *deleteUser* del servicio. Un usuario no se puede eliminar a sí mismo.

Los botones de crear y editar un usuario redirigen a su respectivo componente, comentados en la fase de análisis. Comparten el mismo formulario solo que cuando se crea un usuario este está vacío y llama a la función *createUser* del servicio mientras que cuando se edita un usuario este aparece rellenado con sus datos correspondientes y llama a la función *updateUser* del servicio pasándole el id del usuario a editar.

Una vez se tenía los componentes necesarios para poder gestionar a los usuarios de la aplicación, se procedió a realizar las primeras llamadas a la API de Smart University. Para ello, lo primero que se realizó fue la creación de un nuevo servicio llamado *university.service.ts* el cual es el que se encarga de realizar las peticiones a la API de la aplicación pasándole los datos necesarios para poder obtener los datos, que la API se encargará de devolver. Para se pueda realizar esto, fue necesario añadir una nueva ruta con su respectivo controlador (ambos con el nombre de *smartuni.js*), el cual está formado por las funciones que se encargan de realizar la petición a Smart University. La nueva ruta declarada para atender a estas peticiones es *localhost:3000/api/smartuni*.

El controlador está formado únicamente por dos funciones POST las cuales reciben por el cuerpo de la petición la información, que envía el servicio, necesaria para poder realizar la llamada a Smart University. Para realizar las peticiones desde la API, se instaló a través de npm la librería Axios [18] la cual permite realizar peticiones o llamadas al contenido de un enlace HTTP.

La primera función es *getData*, la cual se encarga de devolver los datos de Smart University comprendidos en las fechas y filtros proporcionados. Esta función se encarga de realizar la petición a la función POST correspondiente de la API de Smart University la cual requiere que se le envíe por el cuerpo de la petición un JSON con la información del rango de las fechas en las que se encuentran los datos y los filtros que se quieren aplicar a la llamada. La propia API de Smart University pone un ejemplo de cómo debe ser el JSON.

The screenshot shows the API documentation for the `POST /{token}/getData` endpoint. The title is "Get Data". Below it, a description states: "Request to obtain `data` between given dates and optional filters". The "Parameters" section includes a table:

Name	Description
token * required string (path)	Token, to allow send the request <input type="text" value="token"/>

The "Request body" section is described as "Request body with dates and filters". It includes a "Example Value" link and a "Schema" link. The schema is shown as a JSON object:

```
{
  "time_start": "2023-05-17T05:18:38Z",
  "time_end": "2023-05-19T05:18:38Z",
  "filters": [
    {
      "filter": "uid",
      "values": [
        "uid1",
        "uid2",
        ...
      ]
    }
  ]
}
```

Figura 46. Ruta de la función *getData* de Smart University y ejemplo de cuerpo que recibe la llamada.
(Fuente: <https://openapi.smartua.es/doc>)

Si todo sale bien, se obtiene un json con los datos requeridos con la estructura del siguiente ejemplo que también proporciona su API.

```
{
  "columns": [
    "time",
    "uid",
    "value",
    "metric",
    "typemeter",
    "alias",
    "lat",
    "lon",
    "cota",
    "description",
    "description_origin",
    "name",
    "organizationid",
    "origin"
  ],
  "values": [
    [
      "timeValue",
      "uidValue",
      0,
      "metricValue",
      "typemeterValue",
      "aliasValue",
      "latValue",
      "lonValue",
      "cotaValue",
    ]
  ]
}
```

Figura 47. Ejemplo de petición exitosa a la función getData de Smart University.
(Fuente: <https://openapi.smartua.es/doc/>)

Como se puede ver, el json obtenido contiene dos claves. La primera clave llamada columnas, contiene el nombre de toda la información asociada al dato como, por ejemplo, *time* que se refiere a la fecha en la que se midió. La segunda clave llamada valores, contiene todos los valores respectivos a los nombres de la primera clave incluido el propio valor del dato.

La otra función de la API de la aplicación es getDataOperation, la cual devuelve el máximo, mínimo o último dato comprendido en las fechas y filtros proporcionados. En esta función se hace una petición GET a Smart University, por lo que, en este caso, el token, las fechas y los filtros se facilitan a través de la propia url de la petición.

GET	<code>/{{token}}/time_start/{{time_start}}/time_end/{{time_end}}/operation /{{operation}}/uid/{{uid}}/name/{{name}}/getDataOperation</code>
-----	--

Figura 48. Ruta de la llamada getDataOperation de Smart University.
(Fuente: <https://openapi.smartua.es/doc/>)

Como se puede ver a continuación, si todo sale bien, el resultado es muy similar al del ejemplo anterior solo que, en este caso, solo se devolvería un objeto valor en la clave de los valores ya que se está solicitando un único dato.

```
{
  "tags": {
    "uid": "string"
  },
  "columns": [
    "time",
    "uid",
    "value",
    "metric",
    "typemeter",
    "alias",
    "lat",
    "lon",
    "cota",
    "description",
    "description_origin",
    "name",
    "organizationid",
    "origin"
  ],
  "values": [
    [
      "timeValue",
      "uidValue",
      0,
      "metricValue",
      "typemeterValue",
      "aliasValue",
      "latValue",
      "lonValue",
      "cotaValue",
      "descriptionValue",
      "description_originValue",
      "nameValue",
      "organizationidValue",
      "originValue"
    ]
  ]
}
```

Figura 49. Ejemplo de petición exitosa de la función `getDataOperation` de Smart University.
(Fuente: <https://openapi.smartua.es/doc/>)

Estas funciones se llaman dependiendo del tipo de llamada seleccionada en el formulario de configuración de una llamada (Figura 44). Si se ha seleccionado que la llamada devuelva todos los datos en el rango de fechas proporcionado, se llamará a la función `getData`, mientras que, si se ha seleccionado que únicamente devuelva un dato con el último, máximo o mínimo valor, se llamará a la función `getDataOperation`. Como se comenta en la iteración anterior, si se selecciona el máximo, mínimo o último, solo se pueden introducir dos filtros para indicar el identificador del sensor que mide el dato y la magnitud en la que lo mide y son obligatorios. Por lo tanto, solo serían estos dos filtros los que se pasan por la url de la petición.

Por último, como se ha comentado en la fase de análisis, en esta iteración se implementó la paginación a todas las tablas de gestión de la aplicación realizadas a lo largo de las iteraciones anteriores y esta. También se arregló un problema a la hora de obtener las fechas desde la base de datos.

La paginación se ha implementado de la misma forma para todas las tablas, esta se encuentra en la parte inferior de cada una. En el momento de obtener la información necesaria para mostrarla en

cada tabla, se calcula el número de páginas que debe haber. La API devuelve los elementos de cada tabla de 10 en 10, por lo que, para saber el número de páginas, se divide el total de elementos que hay en la base de datos para esa tabla y se divide entre 10 y con el resultado se crea un array con los números de las páginas ordenados empezando por el 1. La plantilla también proporciona la base para poder implementar los botones de la paginación. Entonces, una vez tenemos el número de páginas que se deben mostrar, se puede iterar el array que almacena el número de estas y mostrar el mismo número de botones con su número de página correspondiente.

```
<!-- Paginación -->
<nav aria-label="Page navigation example">
  <ul class="pagination justify-content-center">
    <li class="page-item" [ngClass]="{'disabled': numPage === 0}">
      <button class="page-link" aria-label="Previous" title="Anterior" (click)="pageQr(numPage - 1)">
        <span aria-hidden="true">&laquo;</span>
      </button>
    </li>
    <li class="page-item" *ngFor="let pag of pageArray; let i = index">
      <button title="Página {{i+1}}" [ngClass]="numPage === i ? 'btn btn-primary' : 'page-link'" (click)="pageQr(i)">{{pageArray[i]}}</button>
    </li>
    <li class="page-item" [ngClass]="{'disabled': numPage + 1 === page}">
      <button class="page-link" aria-label="Next" title="Siguiente" (click)="pageQr(numPage + 1)">
        <span aria-hidden="true">&raquo;</span>
      </button>
    </li>
  </ul>
</nav>
```

*Figura 50. Código de la paginación.
(Fuente propia)*

A parte de los botones con el número de páginas, a los dos extremos laterales de estos, se han añadido dos botones para ir a la página anterior o a la siguiente. En una variable se almacena el número de la página en la que está el usuario, de esta forma, se puede controlar que, si se está en la primera página, no se pueda ir a una anterior y viceversa al estar en la última página. El resultado de la paginación de las tablas de gestión de la aplicación es la siguiente.



*Figura 51. Botones de la paginación de las tablas de gestión de la aplicación.
(Fuente propia)*

En cuanto al problema con las fechas, este no se había detectado antes ya que la hora no se revisaba en las demás interfaces. Es a la hora de elegir el rango de fechas en la interfaz de configuración de una llamada (Figura 44), en la que se debe elegir también la hora y cuando se entraba en esta interfaz, las fechas que salían en los inputs estaban atrasadas dos horas de las seleccionadas

anteriormente. Como se ya se ha dicho, se resolvió con la librería *date-fns* [17], la cual con su función *format*, permite pasar un objeto *Date* de JavaScript al formato y zona horaria que se desee. A partir de ahora se usa esta función cada vez que se obtiene una fecha desde la base de datos.

```
// Se adaptan las fechas
this.consult.dateFrom = new Date(this.consult.dateFrom);
this.consult.dateFrom = format(this.consult.dateFrom, "yyyy-MM-dd'T'HH:mm:ss.SSS");

this.consult.dateTo = new Date(this.consult.dateTo);
this.consult.dateTo = format(this.consult.dateTo, "yyyy-MM-dd'T'HH:mm:ss.SSS");
```

Figura 52. Fechas adaptadas al formato y a la zona horaria del sistema en el que se ejecuta.
(Fuente propia)

6.9. Iteración 9. Vista de los códigos QR y multiidioma.

Una vez ya se pueden obtener los datos que proporciona Smart University, se puede pasar a la implementación de la visualización de estos. Esta iteración tiene como objetivo la creación del componente para la interfaz de vista de los datos de los códigos QR. En ella se podrán ver estos datos representados mediante gráficas para que se puedan ver de forma accesible y amigable en un dispositivo móvil. Además del objetivo principal, también se aprovechó esta iteración para añadir migas de pan y la posibilidad de cambiar el idioma de la aplicación. Finalmente, también se añadió un nuevo componente para el cambio de contraseña de los usuarios.

Como siempre, a continuación, se encuentra la fase de análisis para conocer los requisitos que requieren estos componentes, con la guía, para la interfaz principal de esta iteración, del mockup de la Figura 14.

6.9.1. Fase de análisis.

Como se ha comentado, la interfaz de vista de los códigos QR debe mostrar los datos de forma accesible y amigable en los dispositivos móviles, por esta razón, se ha decidido que la representación de estos datos sea realizada con la ayuda de la librería ECharts [19]. ECharts es una librería de JavaScript que permite la rápida construcción de soluciones de visualización de datos basadas en web. Esta permite, entre otras cosas, embeber gráficas de rápida carga en un entorno web, importar los datos a analizar desde simples fuentes de datos como JSON y representar datos en tiempo real. Estas características se adecuan con los requisitos de la aplicación, por lo que fue la librería elegida para pasar a representar los datos de los QR.

La interfaz de vista de los QR está compuesta por el nombre del código QR como título seguido por las gráficas ordenadas de arriba abajo por orden de la fecha de creación. Estas igualmente se podrán ordenar de la forma que se prefiera. Como las gráficas están pensadas para verse desde un dispositivo móvil, estas deben ser responsivas y adecuarse correctamente al tamaño de los dispositivos, ocupando el espacio necesario para que se pueda ver e interactuar con los datos de manera sencilla. A su vez, deben detectar el tipo y la cantidad de datos que deben representar según lo seleccionado en el formulario de configuración de las llamadas.

Esta interfaz también debe saber reconocer si el código del que se solicita su visualización se encuentra activado o desactivado y que su fecha de validez no haya caducado. En caso de que esté desactivado o caducado, la interfaz no debe permitir la visualización e informar al usuario de forma clara de lo que sucede. También debe comprobar que, si el código está activado, tenga por lo menos una llamada y que estén activadas y sin ningún error en su configuración. En el caso de que alguna llamada se encuentre en alguno de esos casos, pero las demás no, esta no se debe mostrar, dejando hueco a las llamadas correctas.

Como se ha comentado, el orden de las llamadas se puede modificar y se verá reflejado en esta vista, de esta forma se puede elegir que códigos son los que se desea que aparezcan antes. Para ello, en la parte del backend, se ha añadido una nueva columna en la tabla de las llamadas de la base de datos de la aplicación. Esta columna tiene el nombre de *orderConsult* y almacena un número que indica el orden de las llamadas de un código QR. La API es la que se encarga de asignar el orden a cada nueva llamada que se crea. Por otro lado, en el frontend, se han añadido dos nuevas columnas a la tabla de la gestión de las llamadas de un QR, en la interfaz de la configuración de este (Figura 40). En estas columnas se encuentran dos nuevos botones, uno para bajar de puesto la llamada y otro para subirla.

Se han añadido también 3 nuevas columnas la misma tabla: *typeDate*, *number* y *unit*. Como se comenta en la implementación de la configuración de una llamada en la iteración 7, hay dos formas de seleccionar el rango de fechas del que se quieren obtener los datos: de forma absoluta y relativa. Si se selecciona un rango absoluto, simplemente se tiene que realizar la búsqueda de los datos siempre en esa misma fecha, mientras que si se selecciona el rango relativo, cada vez que se realice la petición, se deberá realizar el cálculo de este con la cantidad y unidad del tiempo seleccionada en el formulario y además esta selección tiene que verse reflejada en el componente de esta iteración. Por esta razón se han añadido estas columnas. *TypeDate* almacena el tipo de fecha que se ha seleccionado (0 para la absoluta y 1 para la relativa), *number* almacena la cantidad y *unit* la unidad de tiempo que se le quiere restar a la fecha y hora en la que se realiza la petición si se

selecciona la fecha relativa. Por lo tanto, cuando se vaya a visualizar un código QR, la aplicación debe saber detectar qué tipo de rango de fechas debe mostrar.

Los requisitos de la interfaz del cambio de contraseña son muy similares a varios de los formularios ya creados en la aplicación. Es necesario un formulario que permita introducir la antigua y la nueva contraseña, esta última dos veces para confirmarla. Para que todo se valide y se pueda realizar el cambio correctamente, se hace uso de la función de la API llamada *changePassword*, la cual se encuentra en el controlador de los usuarios y se realizó en la iteración 4.

En cuanto al multiidioma, para que se pueda cambiar el idioma en cualquier momento, se ha decidido añadir un *select*, con los idiomas disponibles en la aplicación, en la cabecera de la aplicación para que sea visible en todo momento. En la siguiente fase se explica cómo se hace todo el proceso para poder cambiar el idioma.

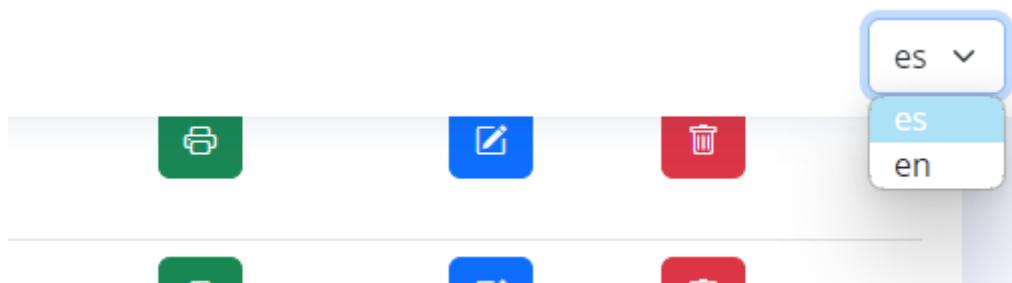


Figura 53. Select en la cabecera para cambiar el idioma de la aplicación.
(Fuente propia)

6.9.2. Fase de implementación.

Para empezar con la implementación de la interfaz de vista de los QR, primero se genera un nuevo componente como siempre y se guarda en la carpeta de las páginas públicas, ya que esta interfaz puede ser vista por cualquier usuario que escanee el código QR con su dispositivo móvil.

A su vez, se decidió generar otro componente para las gráficas. Este componente, funciona de hijo para el componente de la vista de los códigos QR para que, de esta forma, simplemente se le pueda pasar la información necesaria de cada gráfica y esta se encargue de usarla para su representación. Este componente también se guarda en la carpeta de los componentes públicos.

Entonces el componente padre básicamente está compuesto por el título del código QR y la sección donde se muestran todas las gráficas. La lógica de este componente es la que se encarga de realizar la petición a la API para que le devuelva toda la información necesaria para montar las gráficas. Esta información se la pasa al componente hijo, el cual se compone del div que contiene cada gráfica.

Por cada gráfica recibida en la petición, se crea una instancia del componente hijo, el cual una vez recibe la información, la lógica de este se encarga de detectar qué tipo de gráfica debe representar y la monta con los datos proporcionados para, finalmente, renderizarla y mostrarla en la vista.

La petición que realiza el componente padre debe ser pública y no pedir ningún token ya que la vista de los QR debe poder ser vista por cualquier usuario que escanee el código con su dispositivo móvil. Por lo tanto, se ha creado una función en la lógica que se encarga de llamar al servicio de los códigos QR para poder realizar esta petición a la API y, para ello, se ha creado una nueva función en el controlador de los códigos QR.

Esta función se llama *viewQR* y solo necesita que se le proporcione el identificador el QR del que se quiere obtener los datos de sus llamadas si es que las tiene. Lo primero que realiza esta función es comprobar todos los posibles casos que puede haber. Nada más obtiene el id, busca el QR en la base de datos y comprueba que exista. En el caso de que no se encuentre, la función se detiene y no comprueba nada más. Si existe, lo siguiente que se comprueba es que el QR no se encuentre desactivado o caducado. Si todo sale bien, procede a obtener sus llamadas en la base de datos y comprueba que si no está vacío. En el caso de que tenga llamadas comienza a recorrerlas y comprueba que estas estén activadas. En el caso de que no lo estén o que estén mal configuradas ya sea porque el token de esta no es el correcto o no se obtiene ningún dato de esta, pasa directamente a la siguiente en el array en el que se almacenan. Si ocurre que todas las llamadas están desactivadas o no se encuentran configuradas correctamente, la API devolverá un mensaje indicándolo.

En todas las comprobaciones que se han comentado, si resulta que un QR no pasa alguna de estas, la función se detiene y la API se encarga de devolver un mensaje al frontend indicando lo que sucede. Estos mensajes se aprovechan para poder detectar cual es la comprobación que el QR no ha pasado satisfactoriamente y poder montar un mensaje en el HTML del componente para que el usuario lo pueda leer. En las variables de entorno de la parte del frontend, se ha creado una por cada caso y dependiendo del mensaje recibido por parte de la API, se muestra uno u otro. De este modo, estos mensajes pueden ser configurados fácilmente por si en algún momento se quieren cambiar o indicar otra cosa al usuario.

```

    //Mensaje si el QR no esta activado
messNoActive: "El código QR no se encuentra activo en estos momentos. Intentelo más tarde.",
//Mensaje si el QR no tiene ninguna llamada activa (tambien se obtiene si todas las llamadas no estan configuradas correctamente)
mess0Active: "El código QR no tiene ninguna llamada activa o no estan configuradas correctamente en estos momentos.",
//Mensaje si el QR no existe
messNoExists: "Este código QR no existe",
//Mensaje si la fechad e validez del QR ha caducado
messExpired: "El código QR ha superado la fecha de validez.",
//Mensaje si el QR no tiene llamadas
messEmpty: "El código QR no tiene llamadas.",
defaultDes: "Descripción del código QR"

```

*Figura 54. Variables de entorno con los mensajes configurables.
(Fuente propia)*

Una vez se obtienen las llamadas y estas pasan las comprobaciones, se comienza a montar el cuerpo de la petición que se enviará a la API de Smart University. La información que se obtendrá a partir de esta petición se guarda en un objeto el cual será el que se devuelva al frontend y contenga todo lo necesario para poder mostrar las gráficas. Este objeto se compone por el título del QR y un array que almacena la información de cada gráfica.

Lo primero que se comprueba es el tipo de fecha que la llamada tiene seleccionada. Si la fecha es absoluta, simplemente se obtiene el rango de fechas a partir de las columnas *dateFrom* y *dateTo* las cuales son las fechas ‘desde’ y ‘hasta’ que son las que se usarán directamente. En cambio, si el tipo de fecha seleccionada es la relativa, se procede a realizar la resta a la fecha en la que se realiza la petición con el número y unidad de tiempo en el que está configurada. Para ello primero se comprueba que unidad es la que se ha seleccionado. Hasta ahora se puede elegir entre cuatro unidades de tiempo: segundos, minutos, horas, y días. Hay que pasar cada unidad a milisegundos, por lo que, dependiendo de la unidad seleccionada, el cálculo será de una forma u otra. Para ello se ha creado un array que almacena la cantidad por la que hay que multiplicar dependiendo de la unidad de tiempo seleccionada: 1.000 para los segundos, 60.000 para los minutos, 360.000 para las horas y 86.400.000 para los días. Una vez se obtiene la cantidad que se desea restar, se crea un objeto Date del resultado de restarle a la fecha en la que se realiza la petición el resultado obtenido y la fecha ‘desde’ será lo que se obtenga de este cálculo mientras que la fecha ‘hasta’ será la fecha en la que se realiza la petición. De esta forma cada vez que se vaya a visualizar el QR, se realizará este cálculo y si, por ejemplo, se ha seleccionado que se quiere obtener los datos de hace 15 minutos, se le restarán a la hora en la que se escanea con el móvil, obteniendo cada vez resultados distintos en el tiempo.

```

// Se comprueba que tipo de fecha tiene la llamada Absoluta / Relativa
// Si es la relativa se hacen los calculos
if(consult.typeDate === 1){
    let now = new Date();
    let result;
    let num = consult.number;

    // Se pasa el numero introducido a milisegundos
    num *= time[consult.unit - 1];

    result = new Date(now.getTime() - num);

    // Se establecen las fechas con el resultado de la resta
    consult.dateFrom = format(result, "yyyy-MM-dd'T'HH:mm:ss.SSS") + 'Z';
    consult.dateTo = format(now, "yyyy-MM-dd'T'HH:mm:ss.SSS") + 'Z';
}

```

*Figura 55. Cálculo de la fecha relativa.
(Fuente propia)*

Una vez se tienen el rango de fechas, se comprueba el tipo de operación de la llamada: si se quiere obtener el dato máximo, mínimo o último o si se quiere obtener todos los datos en ese rango de fechas. En el primer caso, se crea un objeto con toda la información que la petición necesita: el token de la llamada, el rango de fechas, el tipo de dato que se quiere obtener (máximo, mínimo o último) y el identificador y magnitud del sensor del que se quiere recuperar el dato. A continuación, se realiza una petición a la función *getDataOperation* (creada en la iteración anterior) de la API de la aplicación que es la que se encarga de realizar la petición a la API de Smart University con la información del objeto que se ha creado. Esta petición devuelve la información del dato solicitado como en el ejemplo de la Figura 49. De toda la información recibida, se almacena la necesaria para poder representar el dato en una gráfica y se guarda en la última posición del array de las gráficas: el nombre de la llamada, su descripción, el tipo de dato (máximo, mínimo o último), el valor del dato, su magnitud y la unidad métrica. Si todo sale bien se pasa a la siguiente llamada del QR.

```

// Se comprueba que tipo de operacion tiene
if(consult.operation > 1){
    // Max, min, last

    // Pasamos los filtros a JSON
    consult.filters = JSON.parse(consult.filters);

    let data = {
        token: consult.token,
        dateFrom: consult.dateFrom, //cambiar por cons.dateFrom
        dateTo: consult.dateTo,
        operation: op[consult.operation - 2],
        uid: Object.values(consult.filters)[0],
        name: Object.values(consult.filters)[1]
    }

    // Se realiza la peticion a Smart University
    let res = await axios.post(` ${process.env.URLAPI}/smartuni/operation` , data);
    data = res.data.result;
    //console.log(data)

    // Rellenar el objeto con los datos de la llamada
    charts.push({
        title: consult.name,
        description: data.values[0][data.columns.indexOf('description')],
        type: consult.chart,
        values: [data.values[0][data.columns.indexOf(op[consult.operation - 2])]],
        name: data.values[0][data.columns.indexOf('name')],
        metric: data.values[0][data.columns.indexOf('metric')]
    });
}

```

*Figura 56. Obtención de la información necesaria para representar un máximo, mínimo o último dato.
(Fuente propia)*

Por otro lado, si lo que se solicita en la llamada son todos los datos en el rango de fechas proporcionado, se crea un JSON (como el de la Figura 46) que servirá como cuerpo de la petición a Smart University. El token y el rango de fechas se obtienen directamente desde la llamada, pero el array de los filtros hay que montarlo a partir de los propios filtros que tiene la llamada almacenados en la base de datos. Para ello, se recorre entre los filtros que tenga la llamada y por cada filtro se añade su valor, o valores en caso de que tenga varios, y se monta el JSON para que quede igual que en el ejemplo.

```

else{
    // Todos los datos disponibles

    // Se comienza a montar el cuerpo de la petición
    let body = `{"token": "${consult.token}", "time_start": "${consult.dateFrom}",
        "time_end": "${consult.dateTo}", "filters": [`;

    // Añadir los filtros
    if(consult.filter !== ''){
        // Pasamos los filtros a JSON
        consult.filters = JSON.parse(consult.filters);

        Object.entries(consult.filters).forEach((key, index) => {
            // Comprobar si tienen muchos valores una misma clave
            key[1] = key[1].split(',');

            body += `{"filter": "${key[0]}", "values": [`;
            key[1].forEach((elem, index) => {
                body += `"${elem}"`;

                if(index !== key[1].length - 1){
                    body += ','
                }
            });
            body += `]}`;
        });

        if(index !== Object.entries(consult.filters).length - 1){
            body += ','
        }
    });
}

body += ']}';
body = JSON.parse(body);

```

*Figura 57. Creación del JSON con los filtros aplicados a la llamada.
(Fuente propia)*

Una vez se tiene el JSON, se realiza una petición a la función *getData* (creada también en la iteración anterior) de la API de la aplicación, la cual se encarga de realizar la petición a la API de Smart University usando el JSON proporcionado por el cuerpo de la petición. Como con la función anterior, el resultado que se obtiene contiene toda la información de los datos solicitados, como se puede ver en el ejemplo de la Figura 47. A continuación, por cada identificador seleccionado en los filtros, se almacenan todos los valores de sus datos que Smart University ha devuelto en el rango de fechas y se almacenan en un array junto con el propio identificador y el tipo de gráfica que se va a crear (gráfica de líneas o gráfica de barras). Cuando se almacenan todos los datos de todos los identificadores, se añaden en el array del objeto que se va a devolver junto con el nombre de la

llamada, su descripción, las fechas disponibles en ese rango de fechas y la magnitud y la unidad métrica de los datos. Este tipo de llamadas se pueden realizar sin agregar ningún filtro a la petición, pero puede pasar que no se obtenga ningún resultado ya que se tendrían que devolver una gran cantidad de datos.

```
// Se realiza la petición a Smart University
let res = await axios.post(`${process.env.URLAPI}/smartuni/`, body);
let data = res.data.result;

// Montar el objeto de las series

// Primero se obtienen los uid presentes en los filtros
let ids;

body.filters.map((id) => {
  if(Object.values(id)[0] === 'uid'){
    ids = Object.values(id)[1]
  }
})

let seriesData= [];
ids.forEach((id) => {
  // Se filtran los arrays por cada uid y se obtienen sus valores
  let series = data.values.filter((array) => array[data.columns.indexOf('uid')] === id)
    .map((array) => array[data.columns.indexOf('value')]);

  seriesData.push({
    name: id,
    data: series,
    type: type[consult.chart]
  })
});

// Se guardan las fechas
let dates = data.values.map((subarray) => subarray[data.columns.indexOf('time')]);

// Rellenar el objeto con los datos de la llamada
charts.push([
  title: consult.name,
  description: data.values[0][data.columns.indexOf('description')],
  type: consult.chart,
  ids: ids,
  values: seriesData,
  dates: dates,
  name: data.values[0][data.columns.indexOf('name')],
  metric: data.values[0][data.columns.indexOf('metric')]
]);
```

*Figura 58. Obtención de la información necesaria para representar todos los datos disponibles en el rango de fechas proporcionada.
(Fuente propia)*

Una vez la petición devuelve toda la información de las gráficas a mostrar, el componente de las vistas de los QR itera entre las gráficas devueltas y crea un componente hijo por cada una. El

componente padre les pasa los datos de la gráfica que tienen que crear y estos inicializan un canvas donde se representarán los datos, con la ayuda de las funciones que proporciona la librería ECharts [19]. Una vez se encuentra todo preparado para la representación, comprueban qué tipo de gráfica, entre las cuatro que están disponibles en la aplicación por ahora, es la que tienen que crear. En ECharts, hay una función llamada *setOption* a la que se le pasa un objeto con toda la información necesaria según el tipo de gráfica que se quiere mostrar, por lo tanto, este objeto será de una forma u otra según el tipo seleccionado. Como las gráficas de líneas y barras es muy similar, el objeto que se tiene que pasar a la función es la misma diferenciándose únicamente en el tipo. El ejemplo siguiente muestra como es el objeto en estos dos casos.

```

option = {
  title: {
    text: this.data.title
  },
  tooltip: {
    trigger: 'axis'
  },
  legend: {
    data: this.data.ids,
    top: '10%'
  },
  xAxis: {
    type: 'category',
    data: this.data.dates
  },
  yAxis: {
    type: 'value'
  },
  grid: [
    top: '20%', // Espacio en la parte superior de la grafica
  ],
  dataZoom: [
    {
      type: 'inside',
      start: 0,
      end: 100
    },
    {
      start: 0,
      end: 100
    }
  ],
  series: this.data.values
};

graph.setOption(option);

```

Figura 59. Objeto con la información para crear una gráfica de líneas o de barras.
(Fuente propia)

Como se puede observar, el objeto contiene las propiedades que se quieren mostrar en la gráfica. Las propiedades presentes en este objeto son las siguientes:

- **Title:** muestra un título en la parte superior y se usa para mostrar el nombre de la llamada.
- **Tooltip:** sirve para que cuando se pulse en un punto de la gráfica, se muestre el valor exacto que se ha medido en esa fecha.
- **Legend:** activa una leyenda para mostrar lo que se desee, en este caso se usa para mostrar los identificadores de los sensores presentes en la representación y si se pulsa en alguno de estos, desaparecen o aparecen en la gráfica.
- **xAxis:** sirve para indicar qué es lo que se quiere representar en el eje de las abscisas. En este caso se usa para representar las fechas.
- **yAxis:** sirve para indicar qué es lo que se quiere representar en el eje de las ordenadas. En este caso es de tipo valor para que sea un eje numérico.
- **Grid:** con *top* se le indica que deje un poco de espacio entre la gráfica y la parte superior del contenedor en la que se dibuja.
- **dataZoom:** se utiliza para hacer zoom en un área específica, lo que permite al usuario investigar datos en detalle. En el objeto se utiliza dos tipos de zoom. El *inside*, integra las funcionalidades del zoom de datos dentro del sistema de coordenadas permitiendo al usuario hacer zoom o desplazarse por el sistema arrastrando, moviendo el ratón o tocando con el dedo (en caso de los dispositivos móviles). El segundo, el cual no se indica el tipo porque es el que viene por defecto, es el *slider* que proporciona una barra deslizante especial, en la que se puede hacer zoom o desplazarse por el sistema de la misma forma que en el ejemplo anterior.
- **Series:** es la propiedad donde se introducen todos los datos a mostrar en la gráfica. En este caso aquí es donde se introducen los valores obtenidos desde Smart University.

El siguiente caso es el de la gráfica Gauge, la cual se usa para representar un único valor con una especie de velocímetro. La forma de crear el objeto varía un poco, sobre todo en la forma de indicar el valor del dato a mostrar.

```

// Gauge
option = {
  tooltip: {
    formatter: ` {a} <br/> {b} : {c}`
  },
  title: {
    text: this.data.title
  },
  series: [
    {
      name: this.data.description,
      type: this.type[this.data.type],
      progress: {
        show: true
      },
      detail: {
        valueAnimation: true,
        fontSize: 20,
        formatter: '{value}'
      },
      axisLabel: {
        fontSize: 10
      },
      data: [
        {
          value: this.data.values[0],
          name: this.data.metric
        }
      ]
    }
  ]
}

graph.setOption(option);

```

*Figura 60. Objeto con la información para crear una gráfica Gauge.
(Fuente propia)*

Las propiedades son similares a las utilizadas en el ejemplo anterior, solo que en este caso se necesitan menos ya que se está representando un único valor. Por esa razón, en la propiedad series se puede ver que tiene más detalle que simplemente indicar el dato a mostrar. Gracias a esto se puede seleccionar el tamaño de la fuente del dato y de la unidad en la que está medida y activar propiedades como una animación que se reproduce nada más inicializar la gráfica.

Por último, se encuentra el caso de la gráfica que solo muestra el valor del dato obtenido. En este caso, se utiliza un texto para representar el dato y la descripción de este para saber qué es lo que

indica. Por lo tanto, en el objeto se indica de qué forma es en la que se quiere que aparezca el texto y el dato a mostrar.

```
// Solo el valor
option = {
  title: {
    text: `${this.data.values[0]} ${this.data.metric}`,
    subtext: `${this.data.title}: \n\n ${this.data.description}`,
    left: "center",
    top: "center",
    width: 2,
    textStyle: {
      fontSize: 30
    },
  },
  media: [
    query: {
      maxWidth: 360,
    },
    option: {
      title: {
        subtextStyle: {
          fontSize: 10
        }
      }
    }
  ],
  {
    query: {
      minWidth: 361,
    },
    option: {
      title: {
        subtextStyle: {
          fontSize: 15
        }
      }
    }
  ]
}
```

Figura 61. Objeto con la información para representar el valor de un dato en texto.
(Fuente propia)

En este último caso, se puede ver que la propiedad principal aquí es el título, en el que se indica de qué forma se quiere que se muestre en pantalla. Gracias a las propiedades como *top* y *left* se puede

hacer que el texto se renderice justo en la mitad del contenedor y con la propiedad `fontSize` se puede seleccionar el tamaño de la fuente de este. Además, gracias a la propiedad `media`, se pueden cambiar las propiedades que se requieran para poder adecuar el texto al tamaño de la pantalla del dispositivo que se esté usando para ver las gráficas. Como se puede ver, en este ejemplo se modifica el tamaño de la fuente según el ancho del contenedor, el cual varía dependiendo del dispositivo en el que se muestre.

De esta forma ya se podrían representar todas las gráficas con los datos que se obtienen a partir de la API de Smart University. En la siguiente figura se muestra un ejemplo del componente implementado en esta iteración con 3 gráficas.



Figura 62. Interfaz de la vista de un código QR.
(Fuente propia)

Como comenta en la fase de análisis, en esta iteración también se ha añadido la posibilidad de ordenar las gráficas, desde la tabla de gestión de las llamadas de un QR, para poder elegir el orden en el que se quiere que aparezcan cuando se visualizan. Cada uno de los botones que se han añadido (el de subir y el de bajar el orden) tiene su propia función. Estas funciones son muy similares ya que intercambian la posición entre dos llamadas y llaman al servicio de los códigos QR para realizar una petición a la función que los actualiza y cambiar así el orden en la base de datos. La diferencia es que el botón de bajar en la lista intercambia el orden entre la llamada seleccionada y la que tiene justo en la siguiente posición, mientras que el botón de subir la intercambia por la llamada que tiene en la posición anterior. La primera llamada de la lista de la página no se puede subir de posición y la última no se puede bajar.

Llamadas del código QR		+ Añadir Llamada				
Nombre	Activar/Desactivar	Editar	Eliminar	Bajar	Subir	
Consumo de energía del sensor MLU00040001	<input checked="" type="checkbox"/>					
Gráfica de barras	<input checked="" type="checkbox"/>					
Gráfica de líneas con dos sensores	<input checked="" type="checkbox"/>					
Nombre de la llamada	<input type="checkbox"/>					

« 1 »

Figura 63. Tabla de la gestión de las llamadas de un QR con los botones para ordenar.
(Fuente propia)

En cuanto al cambio de contraseña, como también se comenta en la fase de análisis, se trata de un formulario similar a los que ya hay en la aplicación. En este, se debe introducir la contraseña actual y la nueva dos veces para confirmarla. Una vez se confirma el cambio, las dos contraseñas se envían al backend para que se realicen todas las comprobaciones necesarias tal y como se explica en la función *changePassword* de la iteración 4.

The screenshot shows a user interface for changing a password. At the top, there is a title: "Introduce los datos del nuevo usuario" with a user icon. Below the title are three input fields: "Contraseña actual" (current password), "Nueva contraseña" (new password), and "Repetir nueva contraseña" (repeat new password). Each input field has a corresponding empty text area below it. At the bottom of the form are two buttons: a red "Cancelar" (Cancel) button on the left and a blue "Cambiar contraseña" (Change password) button on the right.

*Figura 64. Formulario de cambio de contraseña.
(Fuente propia)*

Finalmente se implementó la posibilidad de cambiar el idioma de la aplicación entre español e inglés. Para ello, se creó un nuevo componente en la carpeta layouts junto con el componente de la cabecera y de la barra lateral. Este componente es el que muestra los idiomas disponibles en la aplicación y se encarga de cambiarlo. Después de generar el componente, en el archivo de las variables de entorno se agregó un array donde se almacenan los idiomas. De momento solo está el español, que es el que se encuentra por defecto, y el inglés, pero de esta manera se podrían agregar todos los idiomas que se quisieran. Este array se almacena en la lógica del componente para que en el HTML se pueda crear un select mostrando todas las opciones de idiomas.

A continuación, para que cuando se seleccione un idioma se vea en el frontend, es necesaria la instalación de una librería de internacionalización para Angular llamada ngx-translate [20]. Esta librería permite cambiar el idioma de los elementos que se seleccionen según el idioma elegido. Los elementos que se seleccionan para ser traducidos deben indicarse en un archivo JSON con el nombre del idioma. Por ejemplo, el archivo de las palabras en español debe llamarse *es.json*. Estos archivos se almacenan en la carpeta assets y dentro de esta, en una carpeta llamada i18n.

Una vez se tienen los archivos, se pueden llenar los JSON con las traducciones. Las claves sirven para indicar el nombre de los elementos a traducir y en sus valores se indica la traducción que se va a visualizar. Estos son los JSON de los dos idiomas con algunos de los elementos a traducir.

```

{
  "titulo.home": "Códigos QR",
  "boton.Qr": "Añadir QR",
  "table.Qr": "Tus códigos QR",
  "table.date": "Fecha validez",
  "table.title": "Título",
  "table.onoff": "Activar/Desactivar",
  "table.print": "Imprimir",
  "table.edit": "Editar",
  "table.delete": "Eliminar",
  "search.Qr": "Buscar códigos QR",
  "msg.home": "No has creado ningún código QR todavía."
}

}

```

*Figura 65. Archivo es.json con la traducción en español.
(Fuente propia)*

```

{
  "titulo.home": "QR Codes",
  "boton.Qr": "Add QR",
  "table.Qr": "Your QR Codes",
  "table.date": "Validity date",
  "table.title": "Title",
  "table.onoff": "Activate/Deactivate",
  "table.print": "Print",
  "table.edit": "Edit",
  "table.delete": "Delete",
  "search.Qr": "Search QR codes",
  "msg.home": "You have not created any QR code yet."
}

```

*Figura 66. Archivo es.json con la traducción en inglés.
(Fuente propia)*

Por último, queda ir al HTML de los componentes que tengan elementos para traducir y sustituirlos por el nombre del elemento de los JSON anteriores que contengan la traducción que se desee. En el constructor del componente se declara el idioma por defecto con la función `setDefaultLang()` que proporciona la librería, que en este caso es el español, por lo que se le pasa el string 'es'. A continuación, se muestra un ejemplo de la traducción del título de la página de inicio de la aplicación.

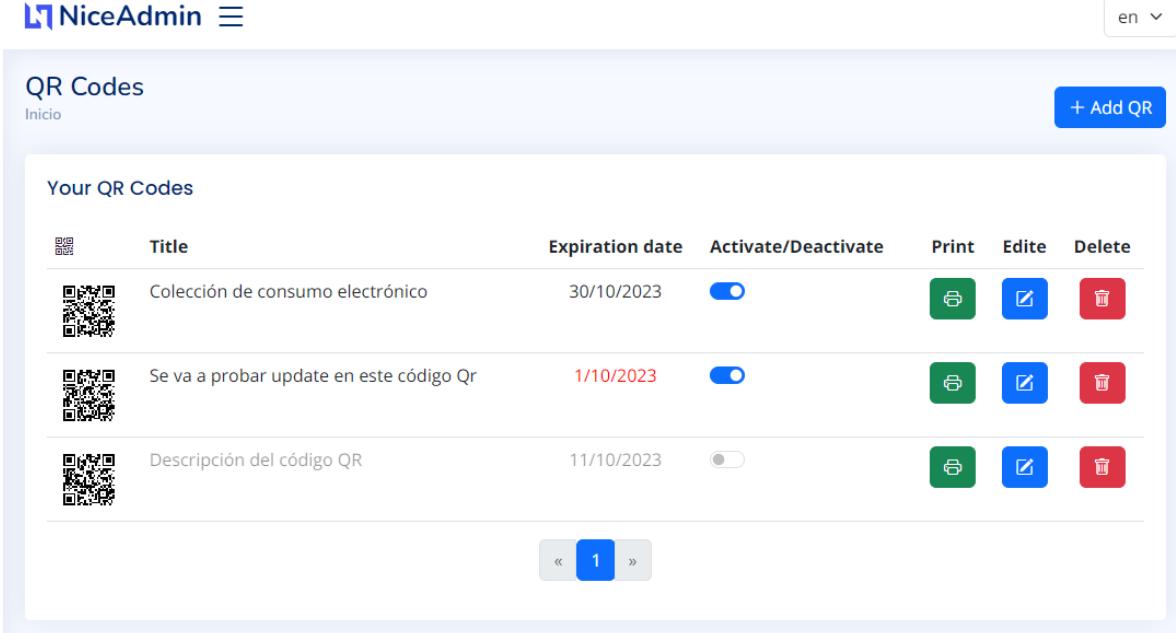
```


<h1>{{'titulo.home' | translate}}</h1>
  <nav>


```

*Figura 67. Ejemplo de elemento traducido.
(Fuente propia)*

Finalmente se crea otra función que detecte el cambio de idioma a través del select y con la función `use()`, también de la librería, se cambia el idioma y los elementos marcados como en el ejemplo anterior se verán traducidos en el idioma correspondiente.



The screenshot shows a web application titled "NiceAdmin" with a "QR Codes" section. The interface includes a header with a logo, language selection ("en"), and a navigation menu. Below the header, there's a breadcrumb trail ("Inicio") and a button to "+ Add QR". The main content area is titled "Your QR Codes" and displays a table with three rows of QR code data. Each row includes a QR code icon, a title, an expiration date, an activation/deactivation switch, and three action buttons for "Print", "Edit", and "Delete". The titles in the table are: "Colección de consumo electrónico" (Expiration: 30/10/2023), "Se va a probar update en este código Qr" (Expiration: 1/10/2023), and "Descripción del código QR" (Expiration: 11/10/2023). At the bottom of the table, there is a pagination control showing page 1 of 1.

QR	Title	Expiration date	Activate/Deactivate	Print	Edit	Delete
	Colección de consumo electrónico	30/10/2023	<input checked="" type="checkbox"/>			
	Se va a probar update en este código Qr	1/10/2023	<input checked="" type="checkbox"/>			
	Descripción del código QR	11/10/2023	<input type="checkbox"/>			

Figura 68. Interfaz de gestión de códigos QR en inglés.
(Fuente propia)

6.10. Iteración 10. Descargar códigos QR para su impresión.

En esta iteración se realiza la implementación de la última funcionalidad principal: descargar los códigos QR para poder imprimirlos. También se ha añadido la funcionalidad de duplicar las llamadas de un QR y la de compartir las vistas de estos, en las que se pueden observar las gráficas, a través de un enlace.

Además, se aprovecha para llevar a cabo las correcciones a la aplicación que van surgiendo en las reuniones con el product owner, a lo largo del desarrollo de las iteraciones. Los requisitos de estas funcionalidades y los de las correcciones que se plantearon en la última reunión se recogen en la fase de análisis.

6.10.1. Fase de análisis.

En la etapa de la definición del proyecto se plantearon dos formas de poder descargar los QR para su impresión: de forma individual, donde solo se descargaría un solo QR y de forma grupal, donde se podría seleccionar los QR de la lista que se quieren descargar.

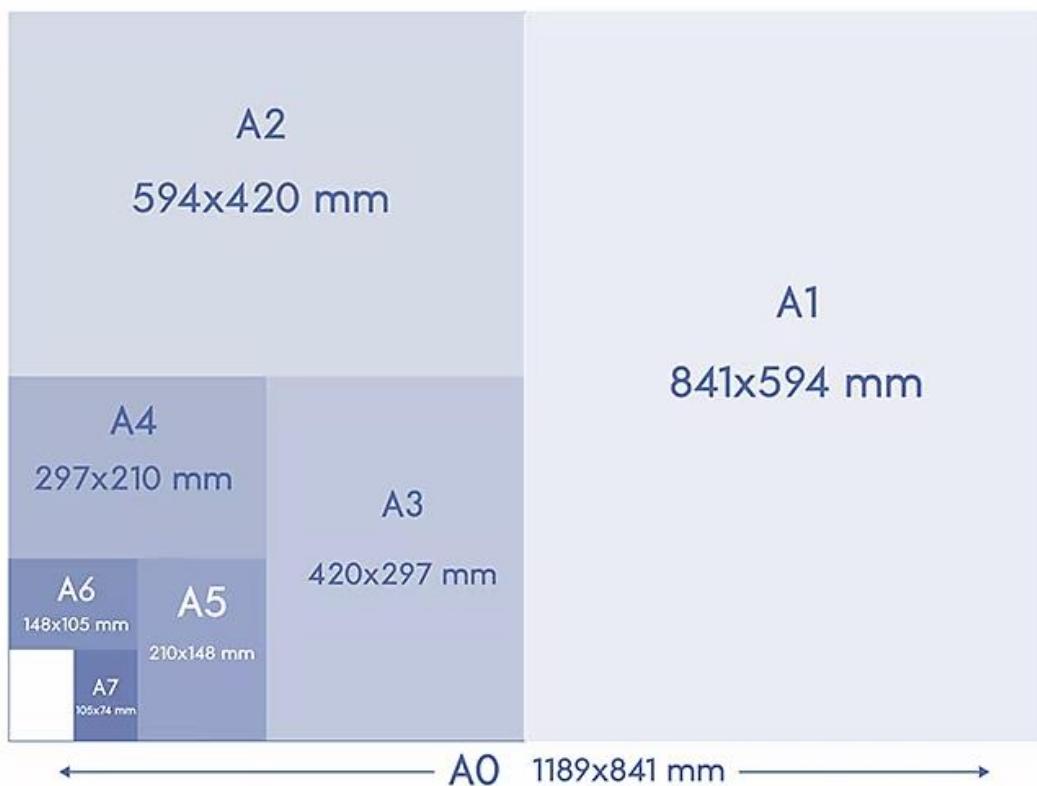
Para realizar esto, se ha decidido que a la hora de descargarlos se genere un PDF que contenga los códigos seleccionados. Este archivo contendría el QR generado a la hora de crearlo y el nombre y descripción de etiqueta indicados en la configuración de estos, donde también se puede ver un ejemplo de cómo sería el resultado.



*Figura 69. Previsualización del código QR en el componente de configuración de un QR.
(Fuente propia)*

Para la descarga de un solo QR, los usuarios pueden seleccionar el tamaño de la hoja del PDF. El tamaño se puede elegir entre cuatro opciones: A4, el más común a la hora de imprimir hojas de papel; así como A5, A6 y A7, los cuales reducen su superficie a la mitad del anterior respectivamente. Una vez se genera el PDF, el QR se ajustará al tamaño de la hoja para ocupar el máximo espacio y que se pueda ver y escanear correctamente.

En el caso de la descarga de varios QR, no se brinda la opción de seleccionar el tamaño individualmente. En su lugar, los QR se agrupan en una hoja A4 en grupos de cuatro. Si es necesario, se agregarán páginas adicionales para acomodar todos los QR seleccionados.



*Figura 70. Dimensión de los diferentes formatos de papel de la serie A en mm.
(Fuente: <https://www.adobe.com/es/creativecloud/design/discover/a4-format.html>)*

Para poder realizar correctamente la descarga de estos PDF, en esta iteración se han descargado dos nuevas librerías. Estas son *jsPDF* [21] y *html2canvas* [22].

La librería *jsPDF* es la encargada de generar el PDF con el contenido deseado. Ofrece una gran cantidad de opciones para generarlo, pero las que son necesarias en este caso son la opción de elegir el tamaño de la hoja del archivo, la de agregar una nueva página, la de agregar imágenes y la de descargarlo en el dispositivo que se esté utilizando.

Esta librería se complementa con *html2canvas*, la cual permite hacer una captura a las páginas web, o a parte de ellas, directamente desde el navegador de los usuarios. Esta captura se basa en el DOM de la página en la que se utiliza, por lo que no es una captura como tal, sino una imagen que se genera a partir de las propiedades que obtiene a partir de este. Por lo tanto, con esta última librería se obtendrá una captura de los códigos QR con el nombre y descripción de etiqueta y se añadirá al PDF con las funciones que proporciona *jsPDF*. De esta forma se podrán generar los PDF con los códigos QR seleccionados para su posterior impresión.

Otra de las funcionalidades que se han añadido permite a los usuarios compartir enlaces a las vistas públicas de los códigos QR donde se muestran las gráficas. Esta característica se ha desarrollado aprovechando las capacidades proporcionadas por el objeto *Navigator* de JavaScript y su función *share()* a la cual se le puede pasar por parámetro un objeto que contenga los datos a compartir. Los valores que interesan compartir en este componente son: el título de la página, el título del código QR que se está compartiendo y su enlace.

La otra funcionalidad implementada en esta iteración es la posibilidad de duplicar las llamadas de los QR. Esta nueva función permite la creación de nuevas llamadas a partir de los datos de otra llamada ya creada y no tener que empezar desde cero. De este modo si se quiere obtener una gráfica parecida a otra que ya hay creada, pero con filtros diferentes, simplemente se tendría que duplicar y adaptarla.

Esta nueva funcionalidad se aprovecha de una de las correcciones aplicadas en esta iteración, la cual afecta a la forma de crear los propios códigos QR y sus llamadas. Previamente, estos se creaban directamente al pulsar en el botón de añadir de su respectivo componente y en el backend se creaban a partir de datos por defecto definidos en la base de datos. Esto se ha cambiado ya que lo correcto es permitir al usuario llenar previamente los formularios de los QR y llamadas con los datos que deseé o cancelar la propia creación.

Para solucionarlo, se ha cambiado el comportamiento de los botones de crear; ahora redirigen al componente respectivo y muestra el formulario vacío para comenzar a rellenarlo. Una vez se llenan como mínimo los campos obligatorios (los cuales se indican en la fase de implementación), se pueden guardar. Al pulsar en el botón de guardar, se llamará a la función del respectivo servicio que se encarga de crear un nuevo elemento a partir de los datos obtenidos de los campos llenados (los que son opcionales, se llenan con datos por defecto). De esta forma, los códigos QR y llamadas no se crean hasta que se llenan sus formularios y se guardan.

Cuando se duplica una llamada, en lugar de directamente crear una nueva con sus mismos datos, se redirige al componente de configuración de las llamadas y los campos se llenan con los mismos datos para que editar aquellos que se deseen. Una vez editado, se puede crear de la misma forma, al pulsar en el botón de guardar.

Las demás correcciones que se han realizado en esta iteración no han representado una carga significativa de trabajo ya que se han centrado en la optimización de la interfaz de usuario del frontend. Estas han consistido en:

- **Reorganización de botones:** En el componente de configuración de códigos QR, se ha reubicado algunos botones para que sigan el mismo diseño que el de la configuración de las llamadas.
- **Confirmación de cambios:** Se ha implementado un mensaje modal que solicita la confirmación antes de que el usuario continúe navegando por la aplicación cuando haya cambios sin guardar. Este modal se activa en componentes con formularios, como la configuración de códigos QR o la creación/edición de un usuario. Si se intenta cerrar sesión, también se muestra este modal, permitiendo al usuario confirmar antes de continuar con el proceso.
- **Indicador de cargas:** Se ha agregado un mensaje en la vista pública de los códigos QR que informa al usuario que se están cargando los datos necesarios para mostrar las gráficas. Este mensaje desaparecerá una vez que todos los datos se obtengan correctamente.

6.10.2. Fase de implementación.

Para empezar con la funcionalidad de la descarga de los QR, lo primero que hay que tener en cuenta es que la forma de implementarla varía según el componente donde se descargan y la cantidad de los QR.

La descarga de los QR se puede realizar desde el componente de gestión de los códigos QR (Figura 35) y el componente de configuración de un QR (Figura 40). Como se comenta en la fase de análisis, el PDF se genera a partir de una captura al código QR realizada con la librería *html2canvas* la cual necesita que el elemento a capturar se encuentre presente en el DOM del componente. En el componente de configuración de un QR, se encuentra el propio QR con su nombre y descripción de etiqueta, por lo que la captura se puede realizar directamente. No obstante, en el componente de gestión de los QR estos no están en el DOM por lo que no se puede realizar una captura.

Para solucionar esto se han aprovechado los mensajes modales de *SweetAlert2* [16]. Cada vez que se pulsa en el botón de descargar un código QR, se crea un mensaje modal en el que se muestra una previsualización del contenido del PDF, que en este caso es el QR con su nombre y descripción de etiqueta y el tamaño seleccionado en el que se va a descargar. De esta manera, una vez que se pulse en confirmar, se puede aprovechar el código HTML creado para el modal y utilizarlo para realizar la captura. Como *html2canvas* no puede realizar capturas a los mensajes modales de *SweetAlert2*, lo que se hace es adjuntar este código al DOM del componente y obtener la imagen.

El problema de esto es que al adjuntarlo directamente sería visible para todos los usuarios y rompería con la estructura del componente, por lo que una vez se descarga, se tendría que eliminar

el elemento de la página. Aunque se realice de esta manera, se vería como el elemento aparece durante un breve período de tiempo y luego desaparece, por lo que se le aplica una propiedad de CSS para que no sea visible para el usuario, pero sí para la librería.

La propiedad de CSS que se utiliza se llama *clip-path* la cual crea una región de recorte que establece qué parte de un elemento debe mostrarse. Las partes de elemento que se encuentren dentro de la región se mostrarán, mientras que las partes que estén fuera no. Esta propiedad se combina con la función de CSS *circle* que se usa en este tipo de propiedades para crear formas básicas. Esta función requiere como parámetro el radio del círculo, por lo que si se le indica que el radio sea de 0 todo el elemento se ocultará para el usuario. De esta forma se puede realizar la captura al QR desde los dos componentes.



Figura 71. Mensaje modal con la previsualización del contenido del PDF.
(Fuente propia)

Hasta ahora, solo se puede capturar un QR individualmente pero también se requiere que se puedan capturar más de un QR para poder descargar varios de una misma vez. La descarga de más de un QR se realiza desde el componente de la gestión de los códigos QR, en el que se ha agregado una nueva columna a la tabla que sirve para añadir a una lista el QR seleccionado. Esta lista indica los códigos QR que se van a descargar, por lo que también se ha agregado otro botón en la parte superior de la tabla que sirve para generar el PDF con todos los QR añadidos en esta lista. Este botón también indica la cantidad de los QR seleccionados.

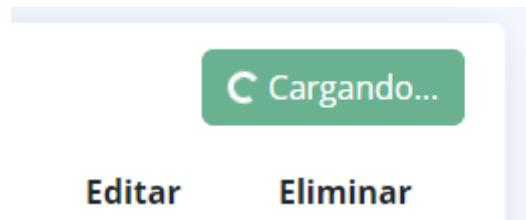
Descargar	Añadir a lista	Editar	Eliminar

Figura 72. Tabla de la gestión de los QR con la columna para añadir QR a la lista y botón para descargarlos.
(Fuente propia)

Para que no se pierda toda esta información al navegar entre componentes, se ha creado un nuevo servicio llamado `print.service.ts` para que se encargue de almacenar y gestionar la lista. Este servicio cuenta con un array donde se almacenan los QR a imprimir y funciones para añadir, eliminar y comprobar si algún QR se encuentra en la lista.

Finalmente, para hacer la captura a todos los QR de la lista, se recorre y se realiza lo mismo que se hace cuando solo se descarga un QR añadiendo una nueva página al PDF cada cuatro códigos QR.

Este proceso puede tardar un poco por lo que se aprovecha el botón para indicar al usuario que el PDF se está generando.



*Figura 73. Botón de descargar la lista de códigos QR indicando que se está generando el PDF.
(Fuente propia)*

Una vez se realizar las capturas en todos los casos se continua con la creación del PDF con la ayuda de *jsPDF*. La forma de hacerlo es muy similar en ambos casos diferenciándose en que en el individual se puede elegir el tamaño de la página del PDF y en el de varios no.

En el primer caso, el tamaño de la imagen de la captura del QR se adapta al tamaño seleccionado y el nombre del archivo coincide con el título del QR indicado en su configuración. En el último caso, las capturas tienen el tamaño justo (contando con los márgenes) para que quepan cuatro en una página de tamaño A4.



*Figura 74. PDF de la descarga de un solo QR con el tamaño A5.
(Fuente propia)*



Figura 75. PDF de la descarga de varios QR.
(Fuente propia)

Para poder seleccionar del tamaño del PDF, en la tabla de los códigos QR de la base de datos, se agregó una nueva columna llamada *sizePrint* en el que se indica el tamaño seleccionado (por defecto es el A4). También se agregó un nuevo campo al formulario del componente de configuración de un QR, en la sección de los datos de impresión, para poder seleccionarlo y editarlo desde la aplicación. De esta forma estaría implementada la descarga de los códigos QR en la aplicación.

Códigos QR		Buscar códigos QR...	+ Añadir QR				
Tus códigos QR							
Imagen QR	Título	Fecha validez	Activar/Desactivar	Editar	Descargar	Añadir a lista	Eliminar
	Colección de consumo electrónico	30/10/2023	<input checked="" type="checkbox"/>				
	Se va a probar update en este código Qr	1/10/2023	<input type="checkbox"/>				
	asdudad	28/10/2023	<input type="checkbox"/>				
	adawawd	21/10/2023	<input type="checkbox"/>				

« 1 »

Figura 76. Interfaz de la gestión de los códigos QR con la funcionalidad de descargar códigos QR.
(Fuente propia)

Siguiendo con las demás funcionalidades, para que la que permite compartir enlaces a las vistas públicas de los códigos funcione, se le pasa por parámetro al método `share()` la información indicada en la fase de análisis. Este método invoca el mecanismo de compartir nativo del dispositivo que se esté utilizando, lo que determinará los destinos disponibles para compartir. Esta función se activa si se pulsa en el botón que se ha agregado en el componente de la vista de los QR.



Figura 77. Botón de compartir en el componente de vista de los QR.
(Fuente propia)

La siguiente funcionalidad implementada es la de duplicar llamadas de un código QR. Como se comenta en la fase de análisis, se ha desarrollado a partir del cambio realizado en la forma de crear los QR y sus llamadas. Para que todo esto funcione, se ha cambiado las rutas de los componentes

de configuración de estos. En el caso de la configuración de los QR, cuando se vaya a crear uno nuevo, la ruta, en lugar de tener el identificador de un nuevo QR en la base de datos, tendrá un 0 para indicar que primero se tiene que llenar el formulario para posteriormente crearlo en la base de datos. En el caso de las llamadas es muy similar ya que también habrá un 0 cuando se vaya a crear una nueva. La diferencia es que ahora, además del identificador de la propia llamada, al final de la ruta también se indica el identificador del QR para que al crear una nueva se le pueda indicar al backend a qué código QR pertenece.

En ambos casos, los componentes no mostrarán varios elementos hasta que no se termine con el proceso de creación. Un ejemplo de esto, en el componente de configuración de un QR, es que la previsualización del QR o la tabla de gestión de sus llamadas no será visible. Una vez se crea, estos elementos aparecen en pantalla.

Código QR

Inicio / Configuración de código QR

Rellene los datos del nuevo QR

Datos del QR	Datos de impresión
Título *	Nombre de etiqueta
Título del QR	Nombre de etiqueta
Fecha de validez *	Descripción de etiqueta
dd/mm/aaaa	Descripción de etiqueta

[Cancelar](#) [Guardar](#)

*Figura 78. Componente de configuración de un QR al crear uno nuevo.
(Fuente propia)*

Como se puede ver, el formulario tiene dos campos obligatorios: el título y la fecha de validez. En el componente de configuración de las llamadas también hay campos obligatorios como el token y el rango de fechas ya que esta información es necesaria para realizar la petición de los datos. Hasta que no se rellenen estos campos no se podrán crear los códigos QR y sus llamadas.

Para duplicar las llamadas se ha agregado una nueva columna con un botón en la tabla de las llamadas de los códigos QR. Cuando se pulsa en estos se redirige al componente de configuración de las llamadas y para ello la ruta en lugar de indicar al final el identificador del QR, indica el identificador de la llamada a duplicar. De esta forma se puede obtener toda la información de la llamada y llenar el formulario con esta. Para diferenciar las llamadas, por defecto al campo del

nombre se le agrega la palabra (copia). A partir de aquí se puede editar de la forma que se desee y no se creará en la base de datos hasta que no se pulse en el botón de guardar.

Llamadas del código QR		Buscar llamadas...	<input type="button" value=""/>	+ Añadir Llamada			
Nombre		Activar/Desactivar	Editar	Duplicar	Eliminar	Bajar	Subir
Consumo máximo de energía del sensor MLU00040001	<input checked="" type="checkbox"/>						
Gráfica de barras	<input checked="" type="checkbox"/>						
Gráfica de líneas con dos sensores	<input checked="" type="checkbox"/>						
Prueba	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
sdsda	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Consumo máximo de energía del sensor MLU00040001 (copia)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

*Figura 79. Tabla de gestión de las llamadas de un QR con el botón de duplicar.
(Fuente propia)*

Configuración de la llamada

Inicio / Configuración de código QR / Configuración la llamada

Crear llamada a partir de una copia

<p>Nombre</p> <input type="text" value="Consumo máximo de energía del sensor MLU00040001 (copia)"/>	<p>Cantidad de datos</p> <div style="border: 1px solid #ccc; padding: 2px;">Máximo</div>
<p>Token</p> <div style="border: 1px solid #ccc; padding: 2px; height: 40px; overflow: auto;">eyJhbGciOiJIUzI1NilsInR5cCI6IkpXVCJ9.eyJpYXQiOjE2ODk3NjQxOTF9.ZyZoVHlgFTwneop0pxn0yW059FUmTi92bnUlklPHmQ</div>	<p>Filtros</p> <p>UID</p> <input type="text" value="MLU00040001"/> <p>Magnitud</p> <input type="text" value="15m"/>
<input type="checkbox"/> Fecha absoluta <input type="checkbox"/> Fecha relativa	
<input style="background-color: #e00000; color: white; border: 1px solid #e00000; padding: 2px 10px; border-radius: 5px; margin-right: 10px;" type="button" value="Cancelar"/> <input style="background-color: #008000; color: white; border: 1px solid #008000; padding: 2px 10px; border-radius: 5px;" type="button" value="Guardar"/>	
<p>Desde</p> <div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">19/05/2023 05:00 <input type="button" value=""/></div> <p>Hasta</p> <div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">19/05/2023 07:00 <input type="button" value=""/></div>	
<p>Selecciona la gráfica para representar los datos</p> <div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">Solo el valor <input type="button" value=""/></div> <div style="border: 1px solid #ccc; width: 100px; height: 100px; background-color: black; display: inline-block; vertical-align: middle; text-align: center; font-size: 2em; color: white;">70</div>	

*Figura 80. Componente de configuración de una llamada al duplicar.
(Fuente propia)*

A continuación, se comenta la implementación de las demás correcciones propuestas en esta iteración.

En cuanto a la reorganización de los botones del componente de configuración de un QR, como se puede ver en la Figura 40 y en la Figura 44 los botones de ambos formularios no guardan ninguna similitud. Además, hasta ahora para poder editar el código QR había que pulsar en el botón de “Editar configuración” mientras que en el formulario de las llamadas se podía editar directamente al pulsar en el botón de edición de la tabla de gestión de estas llamadas. Por esta razón, se ha cambiado de lugar los botones de guardar y cancelar del formulario al final de este para que guarde la misma estructura. El botón de descargar el QR se ha movido también a la parte inferior de la previsualización de este para aislarlo de los demás e indicar claramente su funcionamiento. También se ha eliminado el “modo edición” de este componente por lo que ya no hay que pulsar en un botón para poder comenzar a editar los datos del QR.

Código QR

Inicio / Configuración de código QR

Configuración del QR

Datos del QR

Activado

Título

Fecha de validez

Datos de impresión

Nombre de etiqueta

Tamaño impresión

Descripción de etiqueta

Primer QR

Descripción de etiqueta del primer QR

Volver

Figura 81. Formulario de configuración de un QR con la nueva distribución de los botones.
(Fuente propia)

La siguiente corrección tiene que ver con la anterior ya que se encarga de no perder los cambios sin guardar de los formularios. Cuando había cambios sin guardar en los formularios y se navegaba a otro componente, todos estos cambios se perdían. Para evitar esto, se ha creado un mensaje modal como se comenta en la fase de análisis. Para implementarlo se ha creado un nuevo guard llamado `changes.guard.ts` para que se encargue de comprobar en los componentes que tengan un formulario que, cuando se quiera navegar a otro componente, no hayan cambios sin guardar. En los componentes que tienen formularios se ha creado una función asíncrona a la que el guard llama cada vez que detecte que se quiere navegar fuera del componente. Cuando se llama a esta función,

lanza el mensaje modal para que el usuario confirme la navegación. Si lo confirma el guard permite continuar con la navegación, mientras que si lo cancela no lo permite. La función es asíncrona para que la aplicación espere a la respuesta del usuario para continuar con el proceso de navegación.

```
async checkExit(){
  let res = true;
  if(this.hasChanges){
    await Swal.fire({
      icon: "warning",
      title: "¡Cambios sin guardar!",
      text: "¿Desea continuar sin guardar los cambios?",
      showCancelButton: true,
      cancelButtonText: 'Cancelar',
      confirmButtonText: 'Confirmar',
      confirmButtonColor: '#198754',
      reverseButtons: true
    }).then((result) => {
      if(result.isConfirmed){
        res = true;
      }
      else{
        res = false;
      }
    })
  }

  return res;
}
```

Figura 82. Código de la función asíncrona que se encarga de lanzar el mensaje modal.
(Fuente propia)

Por último, tenemos la corrección del indicador de carga en el componente de la vista pública de los códigos QR. De manera similar a lo que se ha realizado con el botón de descarga de la lista de los QR seleccionados (Figura 73), se ha agregado un mensaje que indica que se están obteniendo los datos. Este desaparecerá una vez los obtenga todos dejando paso a las gráficas que los representan.

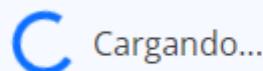


Figura 83. Mensaje de carga de datos en el componente de la vista pública de los códigos QR.
(Fuente propia)

7. Entrega y puesta en marcha

Una vez acabada la fase de implementación del proyecto, era hora de entregar el producto al product owner para ponerlo en marcha. Pero antes de eso fue necesario la revisión de lo desarrollado para ver que todo iba correctamente. Uno de los integrantes del equipo del product owner, Sergio Claramunt, administrador de sistemas, se encargó personalmente de revisar el backend de la aplicación. Detectó todos los cambios que se debían hacer al código para que el backend fuera todo lo seguro y funcional posible ya que el proyecto va a estar en explotación y cualquiera puede acceder. También se corrigieron y añadieron funcionalidades al frontend de la aplicación para adaptarlo a estos cambios.

Como se puede ver en el apartado anterior, backend y frontend han sido implementados a la misma vez, por lo que algunas de las funcionalidades de la API solo se adaptan a los requerimientos del frontend y no eran lo bastante seguras para controlar en su totalidad todas las posibilidades. Esto creaba un problema en la seguridad ya que cualquiera con intención de hacer daño puede acceder a las vulnerabilidades de la aplicación y aprovecharse de lo que no se está controlando. A continuación, se comentan todas estas modificaciones y ajustes que se llevaron a cabo antes del despliegue de la aplicación.

Se ha decidido dividir este apartado en las versiones de la aplicación. Cada vez que se realice una iteración para revisar y añadir funcionalidades, la versión de la aplicación se actualizará. Esto se puede ver reflejado en la barra lateral de la parte privada y en la vista pública de los códigos QR. De este modo se puede tener un control de que la aplicación esté actualizada, una vez esté en producción.

7.1. Versión 1.0

7.1.1. Backend

En el Backend de la aplicación es donde se han realizado más cantidad de cambios en cuanto a la seguridad. Todas estas modificaciones se han llevado a cabo con las recomendaciones del equipo del product owner para que el despliegue en la plataforma de Smart University se hiciese de forma correcta.

Lo primero y más importante a cambiar era la librería con la que se realizaba la conexión a la base de datos. Hasta ahora, la librería que se utilizaba era *mysql* [24] que cumplía con lo básico que se

necesitaba para realizar las conexiones y peticiones a la base de datos. No obstante, esta librería se encuentra obsoleta, ya que la última actualización se hizo hace 4 años. Además, esta librería daba errores de compatibilidad al cambiar de gestor de base de datos. Como se puede ver en el apartado de la implementación, para el desarrollo del proyecto se ha utilizado como base de datos MariaDB en Xampp, la cual cumple con los requisitos para este tipo de proyectos. Sin embargo, cuando se está en un entorno más complejo en el que se da servicio a los clientes se puede quedar algo escueto en cuanto a seguridad y operabilidad. Por esta razón, al desplegar la aplicación se necesitaba cambiar el gestor a MySQL, el cual ofrece más prestaciones para estos casos. Al intentar cambiarlo *mysql* daba errores por lo que no era funcional.

Por todas estas razones, era necesario el cambio a su segunda versión llamada *mysql2* [25] la cual se encuentra activa y recibiendo actualizaciones. Esta nueva librería cubre las vulnerabilidades de la antigua y brinda facilidades a la hora de utilizarla. Entre estas facilidades, se encuentra la de que permite realizar funciones asíncronas sin necesidad de tener que hacerlas manualmente de otras formas más complejas, que es lo que se estaba haciendo anteriormente. También ayuda a mejorar la seguridad de la conexión con la base de datos al permitir parametrizar las sentencias SQL y sanear las entradas de datos de estas.

```
const userByEmail = async(email) => {
  try {
    const query = `SELECT idUser, email, role, lim_consult FROM ${process.env.USERTABLE} WHERE email= ?`;
    const paramsQuery = [email];
    const [user] = await dbConsult(query, paramsQuery);

    return user.length === 0 ? null : user[0];
  } catch (error) {
    throw error;
  }
}
```

*Figura 84. Ejemplo de parametrización de sentencia SQL.
(Fuente propia)*

Como se puede ver en la figura, las sentencias SQL se han modificado para aplicar la parametrización. La parametrización se indica con los ‘?’ y los parámetros se agrupan en el array *paramsQuery*, que en este ejemplo solo contiene uno. De esta forma se consigue que las peticiones se realicen de la manera esperada, aunque se intenten realizar ataques de inyección SQL.

A raíz del cambio de librería y como otra recomendación para mejorar la escalabilidad y legibilidad del código, se separaron las peticiones a la base de datos de los controladores, que hasta ahora se realizaba todo conjuntamente. Para ello, se ha creado otra carpeta con el nombre de *dao* donde se agrupan las funciones que realizan las peticiones a cada una de las tablas de la base de datos. Estas

se agrupan en las funciones dirigidas únicamente a trabajar con la información de los usuarios, los códigos QR y las llamadas de cada uno. Una de las nuevas funciones es precisamente el ejemplo presentado en la figura anterior.

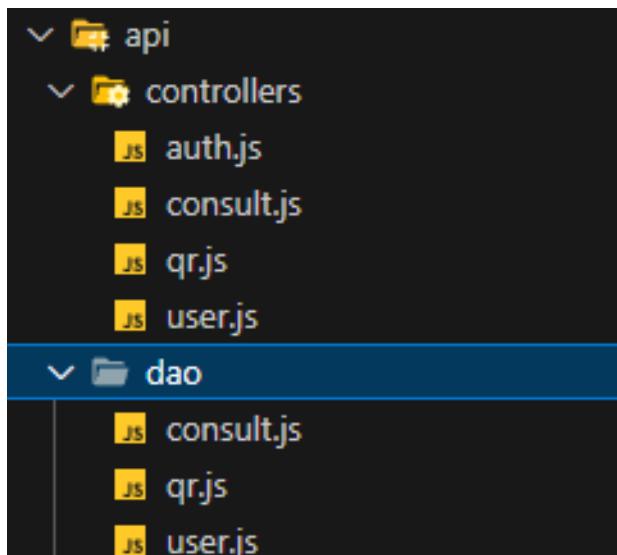


Figura 85. Nueva estructura de la API.
(Fuente propia)

Ahora en los controladores se realizan todas las comprobaciones de los datos obtenidos antes de involucrar a la base de datos para hacer cualquier acción. Una vez todo está correcto, es cuando se llaman a las funciones de *dao* para que obtenga o almacene los datos correspondientes.

En el caso de devolver información, por ejemplo, al listar un usuario, antes se devolvía toda la información, incluida la contraseña que, aunque esté hasheada, puede romperse fácilmente. Además, tampoco se comprobaba el rol de los usuarios a la hora de realizar este tipo de peticiones, por lo que cualquier usuario podía acceder a esta información privada generando una gran vulnerabilidad en la aplicación. Para solucionarlo se revisó y corrigió todas las funciones de los controladores. De esta manera se controla qué información pueden recibir y qué acciones pueden realizar los usuarios dependiendo de su rol. En la siguiente figura se muestra un ejemplo, donde se controla que solo los usuarios administradores pueden listar y acceder a la información de los usuarios del sistema.

```

const getUserById = async( req , res ) => {
    // Se extrae el id del usuario desde el path
    const uid = req.params.id;
    try {

        // Solo los usuarios administrador pueden listar usuarios
        if(req.role !== 1){
            res.status(403).json({
                msg: 'Solo los administradores pueden listar usuarios'
            });
            return;
        }

        const user = await userById(uid);

        if(user !== null){
            res.status(200).json({
                msg: 'getUsuario',
                user: user
            });
            return;
        }
        // Si no se encuentra
        else{
            res.status(404).json({
                msg: 'No se ha encontrado al usuario'
            });
        }
    } catch (error) {
        console.error(error);

        res.status(500).json({
            msg: 'Error devolver usuario'
        });
    }
}

```

Figura 86. Función del controlador de los usuarios que controla el rol del usuario que hace la petición.
(Fuente propia)

Todavía faltaba solucionar el problema de la contraseña. Para que se dejase de enviar la contraseña por la respuesta de la petición, junto con los demás datos, se modificaron las funciones *dao* encargadas de devolver la información de los usuarios. En sus sentencias SQL ahora se indica cuáles son los datos que se quieren obtener excluyendo a la contraseña para que esta información privada no salga de la base de datos. No obstante, hay algunas ocasiones, como a la hora de iniciar sesión, que es necesario comprobar si la contraseña proporcionada coincide con la de la base de datos, por

lo que en estos casos sí que es necesario obtener la contraseña. Para solucionarlo, se ha creado una función auxiliar en el *dao* de los usuarios que únicamente devuelve la contraseña hasheada un usuario. Esta función auxiliar únicamente se llama desde las funciones que necesitan hacer comprobaciones de la contraseña como en el inicio de sesión y en el cambio de contraseña.

En el caso de almacenar la información, también era necesario realizar algunas modificaciones, concretamente a la hora de crear un nuevo usuario. No se controlaba que el email y contraseña introducidos en el formulario fuesen válidos antes de almacenarlos. En la base de datos podía haber un usuario con un email “asdasdasd” y contraseña “1”. Para controlar el email, en la ruta de los usuarios donde ya se comprobaba que en el POST el campo email no podía estar vacío, se añadió otra comprobación para que el email sea válido.

```
router.post('/', [
  check('email', 'El campo email es obligatorio').notEmpty(),
  check('email', 'El email debe ser válido').isEmail(),
  check('password', 'El campo password es obligatorio').notEmpty(),
  validateJWT,
  validateFields,
  validateRole
], createUsers);
```

Figura 87. Validación del email en la ruta de los usuarios.
(Fuente propia)

Para la contraseña, se añadió la librería *password-validator* [26]. Esta librería permite crear un validador con los requisitos que se deseen y pasárselle una contraseña para ver si los cumple. Una de las recomendaciones del equipo es que se obligue a unos mínimos por seguridad. En este caso se valida que la contraseña tenga 8 caracteres como mínimo, una mayúscula y minúscula, al menos 2 dígitos y que no tenga espacios. A este validador se le llama en todas las funciones que trabajan con la contraseña para almacenarla en la base de datos. Si la contraseña no cumple con los requisitos, se indica como respuesta en la petición y no almacena nada.

```

// Funcion que comprueba si la contraseña cumple los requisitos
const checkPassword = (password) => {
    const shcema = new passwordValidator();

    shcema
        .is().min(8)
        .has().uppercase()
        .has().lowercase()
        .has().digits(2)
        .has().not().spaces();

    return shcema.validate(password);

}

```

*Figura 88. Validador de las contraseñas.
(Fuente propia)*

Otro de los problemas de seguridad que había al almacenar o modificar la información en la base de datos era que, si todo salía bien, se devolvía demasiada información de la base de datos. Toda esta información no era necesaria para el frontend y podía generar una vulnerabilidad al exponer mucha información de la operación realizada en la base de datos, por lo que se eliminó de la respuesta. En el caso de la modificación de la información, en algunos casos es necesario obtener el identificador del elemento que se ha actualizado, por lo que en estos casos únicamente se devuelve ese dato.

```

{
    "msg": "postUsuarios",
    "user": {
        "fieldCount": 0,
        "affectedRows": 1,
        "insertId": 4,
        "info": "",
        "serverStatus": 2,
        "warningStatus": 0,
        "changedRows": 0
    }
}

```

*Figura 89. Información de la base de datos que se devolvía al crear un usuario.
(Fuente propia).*

La última modificación que se realizó en el backend, en esta versión, es en las peticiones que se realizaban a la OpenAPI de Smart University. Para ello, desde un principio se creó una nueva ruta llamada *smartuni*. El problema era que estas rutas solo eran utilizadas por la propia API de la aplicación y estaban expuestas ya que no se controlaba si había token o no al ser una petición usada en la parte pública de la aplicación. Se decidió eliminar esta ruta y mover estas dos peticiones a funciones auxiliares en el controlador de los códigos QR. De esta manera solo este controlador puede acceder a estas peticiones y no se exponía una ruta de la API innecesariamente.

7.1.2. Frontend

En el frontend de la aplicación también se tuvieron que realizar algunos cambios para adaptarlo a las modificaciones del backend y aumentar la seguridad.

El primer cambio que se realizó fue en los enlaces de las vistas públicas de los códigos QR. Hasta este momento, estos enlaces eran del estilo *localhost/view/1*, donde 1 indica el identificador del código QR a mostrar. Cualquiera podría recorrer todos los QR y acceder a su información simplemente cambiando el número del final. Es por eso por lo que, para que los enlaces de vista pública de los QR no sean “tan públicas”, se decidió cambiar la estructura del enlace. Ahora, en lugar de poner el identificador autonumérico que se genera en la base de datos, se pone el mismo id, pero encriptado y pasado a hexadecimal. Para conseguir esto, se cambiaron las sentencias SQL de las funciones *dao* que se encargan de devolver y listar la información de los códigos QR.

```
const query = `SELECT LOWER(HEX(AES_ENCRYPT(idQr, '${process.env.CODE}'))) AS uid,
```

Figura 90. Sentencia SQL que encripta y pasa a hexadecimal los identificadores de los QR.
(Fuente propia)

Como se puede ver en la figura, se indica en la sentencia que se devuelva una nueva propiedad, la cual contiene el identificador encriptado, en la respuesta. La función *AES_ENCRYPT* se encarga de encriptar el primer parámetro, usando una clave indicada en el segundo parámetro y devuelve una cadena de binarios que contiene el valor encriptado. En este caso la clave se define en las variables de entorno. Posteriormente, la cadena obtenida se pasa a hexadecimal con la función *HEX* para poder utilizar la cadena resultante en el enlace. La función *LOWER* pasa todos los caracteres obtenidos en la función anterior a minúsculas.

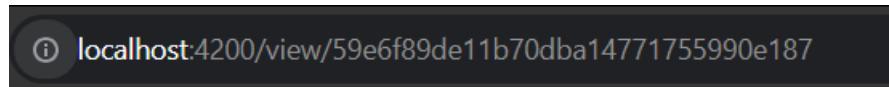


Figura 91. Nuevo enlace de la vista pública de los códigos QR.
(Fuente propia)

Se ha añadido también una función auxiliar que se encarga de hacer la misma operación, pero al revés para desencriptar el identificador del enlace y poder llamar a la base de datos con el identificador correcto. Si esta función devuelve un null, significa que el valor indicado en el enlace no corresponde con ningún identificador de los QR. De esta manera ya no se puede acceder a los códigos tan fácilmente.

```
// Desencriptar uid
const decrypt = async(uid) => {

  try {
    const query = `SELECT AES_DECRYPT(UNHEX('${uid}'), '${process.env.CODE}') AS idQr`;
    let res = await dbConsult(query);

    return res[0][0].idQr === null ? null : res[0][0].idQr.toString();
  } catch (error) {
    throw error;
  }
}
```

Figura 92. Función auxiliar para desencriptar el id de los enlaces.
(Fuente propia)

Otro de los cambios que se han realizado para adaptarse a los cambios del backend es a la hora de introducir contraseñas en los formularios que lo requieren. Como ahora se valida que las contraseñas cumplan con los requisitos que se comentan antes, si se introduce por los formularios una contraseña que no es válida, se le indicará al usuario mediante una alerta el problema.

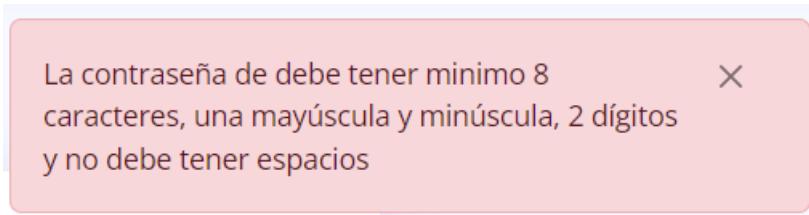


Figura 93. Alerta de error en la contraseña.
(Fuente propia)

También se aprovechó esta etapa de revisión para añadir nuevas funcionalidades. Una de ellas tiene que ver también con la vista pública de los códigos QR, concretamente con las gráficas que se solo muestran el valor del dato. Se les ha añadido la fecha en la que se midió el dato en la parte superior de la gráfica. Además, dependiendo de si se elige mostrar el máximo, mínimo o último dato, la gráfica tendrá un color de fondo u otro. Este color se puede cambiar desde las variables de entorno de la parte del frontend, preferiblemente indicando colores suaves para que se puedan ver correctamente.

Lo último que se les ha añadido a estas gráficas es la opción de elegir el número de decimales a mostrar del valor, en la configuración de las llamadas. Se ha añadido una nueva columna en la tabla de las llamadas de la base de datos para indicar el número de decimales seleccionado. Esto es porque en Smart University, hay sensores que los valores que miden pueden llegar a tener hasta 16 decimales. Con esta opción se da la posibilidad de poder configurar las gráficas dependiendo del valor que se va a mostrar. Con gráficas que muestran la temperatura, con uno o dos decimales ya sería suficiente, mientras que en otras donde lo que se mide es más complejo y preciso, puede ser útil mostrar más decimales. Esto también se aplica a las gráficas gauge.



*Figura 94. Gráficas que muestran solo el valor con los cambios (máximo dato).
(Fuente propia)*

El último cambio que se realizó, en esta versión, es la posibilidad elegir si se quiere mostrar o no el botón de compartir los enlaces públicos de los QR, para casos concretos en los que no se quiera compartir o que simplemente no sea visible el botón. Para ello, se ha añadido una nueva columna en la tabla de los códigos QR de la base de datos llamada *share*. En esta columna se indica con un booleano si se quiere o no mostrar el botón. En el componente de configuración de un QR, se ha añadido un nuevo campo con un botón como el de activar y desactivar los QR para poder elegir. Si el botón se encuentra desactivado, en la vista pública no aparecerá la opción para compartir.

Configuración del QR

The screenshot shows a configuration interface for a QR code. It is divided into two main sections: 'Datos del QR' (QR Data) and 'Datos de impresión' (Print Data).
In the 'Datos del QR' section:

- 'Activado' (Enabled) switch is turned on.
- 'Título' (Title) input field contains 'Consumo electrónico'.
- 'Fecha de validez' (Validity Date) input field contains '30/11/2023'.
- 'Mostrar botón compartir' (Show share button) toggle switch is off.

In the 'Datos de impresión' section:

- 'Nombre de etiqueta' (Label name) input field contains 'Primer QR'.
- 'Tamaño impresión' (Print size) dropdown menu shows 'A4'.
- 'Descripción de etiqueta' (Label description) input field contains 'Prueba'.

At the bottom right are two buttons: a blue 'Volver' (Back) button and a green 'Guardar' (Save) button.

Figura 95. Configuración del QR con la opción de mostrar el botón de compartir.
(Fuente propia)

7.2. Versión 1.1

Esta versión se aprovechó para añadir dos nuevas funcionalidades relacionadas con las gráficas.

7.2.1. Backend

La primera que se implementó fue la posibilidad de seleccionar el color del fondo y del valor de las gráficas para aumentar así la personalización de estas. Esto se añadió únicamente en las que representan un solo valor. Para ello, se añadieron dos columnas en la tabla de las llamadas en la base de datos llamadas *colorVal*, para almacenar el color del valor y *colorBack*, para almacenar el color del fondo. En la parte del frontend, se añadieron dos inputs nuevos, en la selección del tipo de gráfica para escoger estos colores.

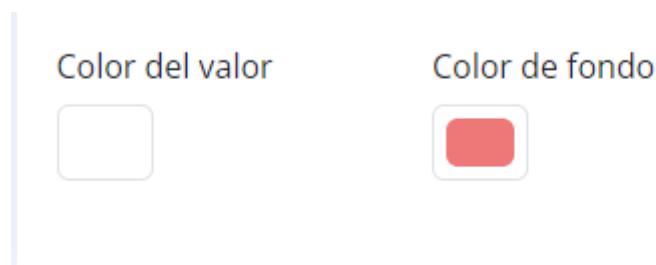


Figura 96. Selectores del color del valor y del fondo de las gráficas.
(Fuente propia)

7.2.2. Frontend

Hasta ahora, cuando se seleccionaba un tipo de gráfica, se mostraba al lado una imagen de ejemplo de estas como se puede ver en la Figura 41. Al ser una imagen estática, no se podía previsualizar las gráficas con los colores seleccionados, por lo que se decidió cambiar la forma en la que se mostraban estos ejemplos. Para que se pudieran reflejar estos cambios, se comenzó a utilizar la librería ECharts [19] para crear ejemplos de cada tipo de gráfica de manera similar a lo que se hace en el componente de vista pública de los QR, pero más simplificado. De esta forma, cada vez que se aplique un cambio, como en este caso al cambiar los colores, simplemente hay que refrescar las gráficas con estos cambios para que se vean reflejados y poder saber cómo será el resultado final. Además, si se necesitara añadir más opciones de personalización, se podrán mostrar con esta nueva forma de crear los ejemplos.

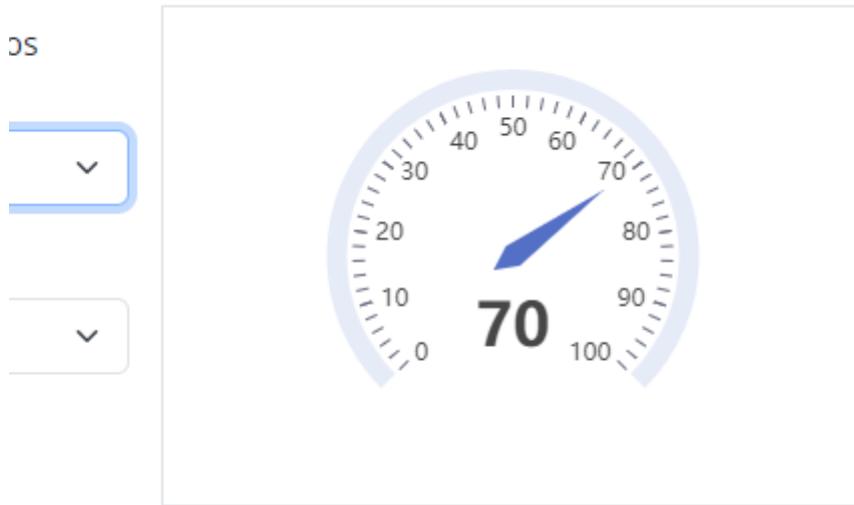


Figura 97. Nuevos ejemplos en el formulario de las llamadas, en este caso es una gráfica Gauge.
(Fuente propia)

La segunda modificación está relacionada con la maquetación de la vista pública de los códigos QR. Aunque desde un principio, esta vista estaba pensada para dispositivos móviles, al acceder desde un ordenador, la maquetación de las gráficas no era muy accesible. Esto sucedía ya que las gráficas que solo representaban un valor y las de Gauge, ocupaban toda la pantalla dejando mucho espacio útil sin utilizar. Para solucionar esto, se decidió aplicar las clases de Bootstrap para posicionar las gráficas de forma más amigable, para que se adaptase correctamente tanto en dispositivos móviles como en ordenadores.

La estructura que se eligió para disponer las gráficas es la siguiente: si antes se creaba una fila por cada gráfica, ahora las gráficas que solo muestran un valor ocupan un tercio de estas filas. Es decir, que si hay tres gráficas seguidas que únicamente representan un valor, ocuparan una fila entera, mientras que las gráficas de líneas y de barras ocupan toda la fila ya que, al representar más cantidad de datos, necesitan más espacio. En los casos en los que una gráfica pequeña esté entre dos grandes, esta seguirá ocupando un tercio de la fila, pero estará sola en el centro.

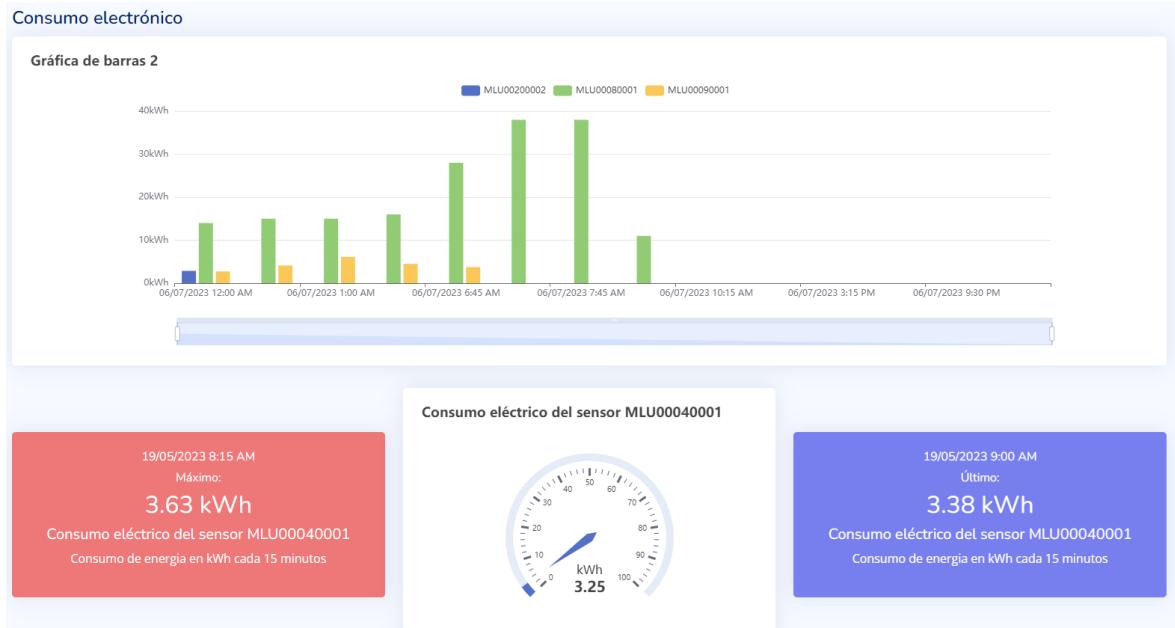


Figura 98. Nueva disposición de las gráficas. Ejemplo con 1 grande y 3 pequeñas.
(Fuente propia)



Figura 99. Nueva disposición de las gráficas. Ejemplo 2 grandes y 1 pequeña en medio. (Fuente propia)

7.3. Versión 1.2

Esta versión se implementó ya que mientras se probaba la aplicación, se observó que se podían añadir dos nuevas funcionalidades que mejorarían la experiencia de usuario.

7.3.1. Backend

Por el lado del Backend, en esta versión lo único que se ha tenido que hacer es añadir una nueva columna a la tabla de las llamadas en la base de datos. Esta columna se llama *icons* y se encarga de almacenar el índice del ícono seleccionado, el cual se aplicará a su gráfica. Esto se explica en la sección del Frontend.

7.3.2. Frontend

La primera es la de duplicar códigos QR. De la misma manera que se hace con las llamadas, se vio que era necesario que también se pudiera hacer con los QR ya que, en algunos casos se podría requerir crear un nuevo QR con las mismas llamadas que otro, pero cambiando únicamente algunos datos. Para no tener que crear todo esto otra vez desde cero, se implementó la posibilidad de

duplicar los códigos QR, añadiendo el botón de duplicar en la tabla de gestión de estos. Para que el componente pueda saber cuándo se va a duplicar, se ha añadido un asterisco seguido del identificador del QR a duplicar. Por lo tanto, cuando se pulse en este botón, el componente obtendrá los datos del QR a duplicar y los mostrará en el formulario. Antes de crearlo, se podrá editar algunos datos, como el nombre o la fecha de validez. Sin embargo, no se podrán editar las llamadas ni acceder a su vista pública hasta que no se cree.

The screenshot shows a user interface for managing QR codes. At the top, there's a header bar with the title "Código QR" and a breadcrumb navigation "Inicio / Configuración de código QR". Below this is a form titled "Crear código Qr a partir de una copia". The form is divided into two main sections: "Datos del QR" and "Datos de impresión". In the "Datos del QR" section, there are fields for "Título" (Consumo electrónico (copia)), "Fecha de validez" (29/12/2023), and a checkbox "Mostrar botón compartir". In the "Datos de impresión" section, there are fields for "Nombre de etiqueta" (Primer QR), "Descripción de etiqueta" (Prueba), and a dropdown "Tamaño impresión" (A4). At the bottom of the form are "Cancelar" and "Guardar" buttons. Below the form is a table titled "Llamadas del código QR" with columns: "Nombre", "Editar", "Duplicar", "Eliminar", "Bajar", and "Subir". The table lists several entries, each with a checkbox in the "Editar" column and a green "Duplicar" button. A blue "Nuevo" button is located at the top right of the table. At the bottom of the table are navigation buttons for page 1.

Nombre	Editar	Duplicar	Eliminar	Bajar	Subir
Gráfica de barras 2	<input checked="" type="checkbox"/>				
Consumo eléctrico del sensor MLU00040001	<input checked="" type="checkbox"/>				
Consumo eléctrico del sensor MLU00040001	<input checked="" type="checkbox"/>				
Consumo eléctrico del sensor MLU00040001	<input checked="" type="checkbox"/>				
Gráfica de líneas con dos sensores pru	<input checked="" type="checkbox"/>				

Figura 100. Componente de configuración de un QR al duplicar.
(Fuente propia)

Una vez creado, se obtiene el identificador del nuevo QR y el componente se ocupa de obtener y mostrar sus datos y llamadas. De esta forma ya se puede duplicar cualquier código QR.

Códigos QR		Buscar códigos QR...		+ Añadir QR				
Tus códigos QR								
	Título	Fecha validez	Activar/Desactivar	Editar	Duplicar	Descargar	Añadir a lista	Eliminar
	Consumo electrónico	29/12/2023	<input checked="" type="checkbox"/>					
	VOC1	1/1/2030	<input checked="" type="checkbox"/>					
	VOC1 (copia)	1/1/2030	<input type="checkbox"/>					

« 1 »

Figura 101. Componente de gestión de los códigos QR con el botón de duplicar.
(Fuente propia)

Otra cosa que se observó haciendo pruebas en la aplicación, es que se podían añadir iconos en las gráficas que solo muestran un valor para mejorar su accesibilidad. ECharts permite aplicar iconos en algunas de sus gráficas por lo que se quiso aprovechar. El problema es que su forma de añadirlos es limitada ya que solo acepta imágenes o la propiedad *path* [27] de los SVG de los iconos. Otro problema que había era que al intentar poner los iconos en las gráficas que solo muestran un valor, todo se desconfiguraba. Esto sucedía ya que se utilizaba la propiedad del título y subtítulo, las cuales solo aceptan texto, para representar estas gráficas. Se les podía aplicar un ícono, pero la única forma de hacerlo era mediante imágenes y no cuadraba correctamente con el resto del texto.

Para solucionar estos problemas, lo primero que se hizo fue cambiar la forma de realizar estas gráficas. Ahora, cuando se necesite crear una gráfica de este tipo, se realiza directamente en HTML con la ayuda de Bootstrap, permitiendo mucha más libertad a la hora de personalizarla. Además, se decidió aplicar los iconos mediante los *path* para poder aplicarlos tanto en las gráficas de ECharts como en HTML. En este último caso, si se ha seleccionado un ícono, simplemente se añade una etiqueta SVG indicándole el *path* del ícono para que aparezca junto con el valor.



Figura 102. Nueva forma de las gráficas que solo muestran un valor con ícono.
(Fuente propia)

Para seleccionar un ícono se añadió un nuevo campo en el formulario de las llamadas en el que se muestra una lista de los íconos disponibles en la aplicación. Esta lista se recoge en una variable de entorno para poder añadir un ícono en cualquier momento sin tener que hacer nada en los componentes. Esta variable es un array de objetos, donde cada objeto representa un ícono y tienen las mismas propiedades. Cabe recalcar que los íconos que se han introducido en la variable son de Bootstrap Icons [28], donde se indica toda la información que estos objetos necesitan:

- **Name:** nombre del ícono que aparece en la lista para seleccionarlo.
- **Html:** código HTML del ícono que aparecerá en la lista. Este código es del tipo “” y se debe introducir de este modo ya que las etiquetas *options* de los *select* son muy estrictos y solo aceptan texto. Para solucionarlo, se le ha aplicado al select la fuente que usan los íconos de Bootstrap para que se puedan ver. Es por este motivo que la lista sale con una fuente distinta a la del formulario. En Bootstrap Icons, estos códigos se encuentran en la sección “Code point”.
- **Path:** necesario para que aparezcan los íconos en las gráficas. Se encuentran en los SVG de los íconos. Si un ícono tiene varios path, se ponen juntos separándolos con un espacio.

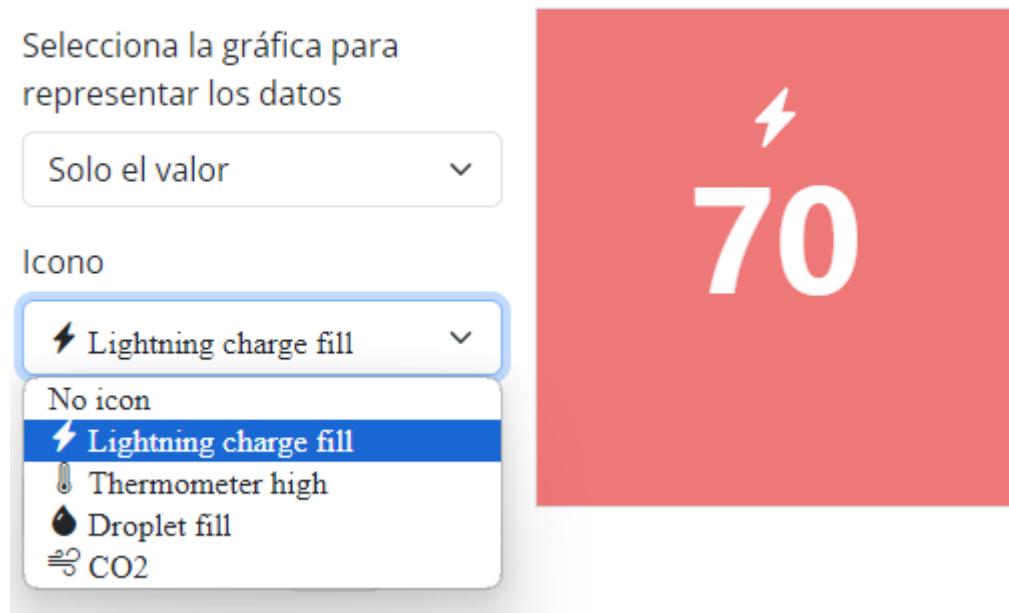
```

icons: [
  {
    name:"No icon",
    html:"",
    path:""
  },
  {
    name:"Lightning charge fill",
    html:"&#xF46C;",
    path:"M11.251.068a.5.5 0 0 1 .227.58L9.677 6.5H13a.5.5 0 0 1 .364.843l-8 8.5a.5.5 0
  },
  {
    name:"Thermometer high",
    html:"&#xF5CE;",
    path:"M9.5 12.5a1.5 1.5 0 1 1-2-1.415V6.5a.5.5 0 0 1 1 0v4.585a1.5 1.5 0 0 1 1 1.415
  },
  {

```

*Figura 103. Variable de entorno con la lista de los iconos.
(Fuente propia)*

Así, los iconos se podrán ver desde el formulario de las llamadas y las gráficas. Además, también se ha aprovechado esta estructura para aplicar los iconos en las gráficas Gauge, y gracias a la nueva forma de mostrar los ejemplos de las gráficas en el formulario, se pueden previsualizar para ver como quedarían en el resultado final.



*Figura 104. Previsualización de una gráfica de solo el valor con el ícono.
(Fuente propia)*

Selecciona la gráfica para representar los datos

Gráfica de gauge

Icono

Droplet fill

No icon

Lightning charge fill

Thermometer high

Droplet fill

CO2

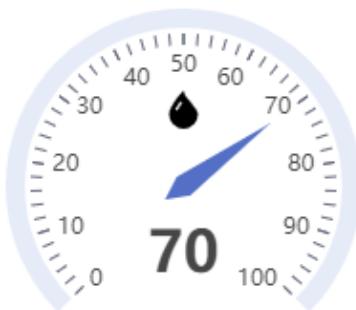


Figura 105. Previsualización de una gráfica Gauge con el ícono.
(Fuente propia)

7.4. Versión 1.3

En esta versión se han introducido dos novedades. Como en la parte del backend, lo único que se ha añadido es una nueva columna en la tabla de los códigos QR de la base de datos, se va a pasar directamente a explicar las modificaciones del frontend. La función de esta columna se muestra a continuación.

7.4.1. Frontend

Lo primero que se ha agregado, es un logo para la aplicación. Hasta ahora se estaba usando el logo por defecto que viene con la plantilla, por lo que era necesario crear una propia.

Al principio, se pensó en crear un logo que representara dos funcionalidades principales: generar códigos QR y visualizar los datos en gráficas. Con esta idea en mente se realizaron algunos bocetos. La primera idea fue intentar realizar el logo con la forma de un código QR con una flecha en el interior para simular una gráfica. No obstante, esta forma de realizar el logo era muy complicada y podía quedar un resultado confuso, por lo que finalmente se decidió descartar esta idea.

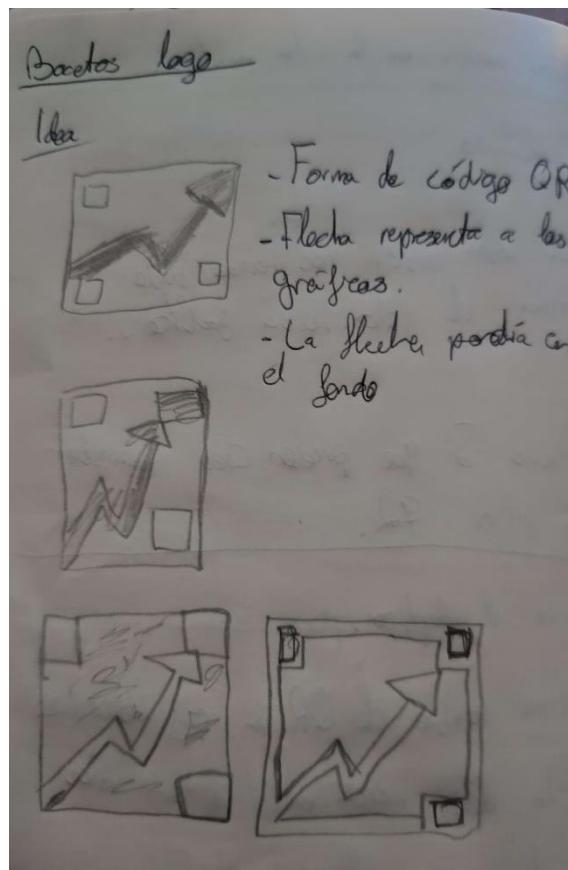


Figura 106. Primeros bocetos para el logo de la aplicación.
(Fuente propia)

Se decidió darle un enfoque más sencillo, con formas más suaves. Con esta nueva perspectiva se decidió utilizar un solo ícono para representar a la aplicación. Se propuso el ícono del QR ya que simboliza perfectamente la aplicación al ser una de sus funcionalidades principales. Mediante un ícono simplificado de un código QR contenido en un círculo para entornar las formas suaves y sencillas, se llegó al resultado que se ve en la siguiente figura.



Figura 107. Logo para la aplicación.
(Fuente propia)

La segunda novedad que se ha agregado es la funcionalidad que las vistas públicas se recarguen automáticamente. Ahora se puede seleccionar el tiempo en segundos para recargar las gráficas desde el componente de configuración de un QR. Si no se quiere aplicar esta funcionalidad basta con introducir un 0 en el campo.

Configuración del QR

Datos del QR Activado

Título

Fecha de validez

Tiempo en segundos para recargar panel

Mostrar botón compartir

Figura 108. Campo del tiempo para refrescar en el formulario de configuración de un QR.
(Fuente propia)

Como se ha mencionado al principio, se ha añadido una columna a la base de datos llamada *refresh*. Esta sirve para almacenar los segundos introducidos en el formulario. La API utilizará este dato para agregarlo al objeto que se devuelve en la función *viewQR()* para que el componente de la vista pública pueda saber cuál es el intervalo de segundos que debe esperar para actualizar los datos de las gráficas. En caso de que el intervalo sea 0, el componente solo realiza la petición una sola vez y habrá que refrescarlo manualmente.

A la hora de refrescar, el componente primero borra todos los componentes hijo habida creado en el intervalo anterior. Seguidamente vuelve a realizar la petición a la API para volver a crear estos componentes con los datos actualizados.

De esta forma, si se desea crear un panel con gráficas cuya información cambia continuamente, se puede activar esta opción para tener siempre los últimos datos disponibles en pantalla.

```
async viewQr(){

    // Si ya hay un timer, se elimina para evitar errores
    // y que espere a que el panel reciba todos los datos
    if(this.intervalID){
        clearInterval(this.intervalID)
    }

    try {
        let qr: any = await lastValueFrom(this.qrService.viewQr(this.idQr));

        this.loading = false;
        this.charts = qr.res.charts;
        this.qr = qr.res.titleQr;
        this.showShare = qr.res.share;
        this.interval = qr.res.interval;

        // Se pone el nombre del QR como title de la página
        document.title = this.qr;

        this.chartsQr();

        // Se crea el timer
        if(this.interval > 0){
            this.intervalID = setInterval(() => {
                this.containerRef.clear();
                this.loading = true;
                this.viewQr();
            }, this.interval * 1000)
        }
    }
}
```

Figura 109. Función que crea el intervalo para refrescar el componente.
(Fuente propia)

8. Pruebas y validación

En esta sección, se van a definir los principales casos de uso para validar que todo lo desarrollado durante la fase de implementación y despliegue funciona correctamente.

Se va a explicar el comportamiento del sistema mediante la interacción con el usuario en cada caso de uso, detallando los pasos y las excepciones que se pueden encontrar durante su ejecución. Para ello se propone un ejemplo de un usuario que quiere crear un panel de Smart University. Siguiendo los pasos de cada caso de uso, se puede recrear este ejemplo si se accede a la aplicación.

8.1. Caso de uso 1. Crear panel con la información de un sensor

El usuario se encuentra en la página de inicio y quiere crear un panel a partir de los datos obtenidos por un sensor llamado “sensor-voc-5”. Más concretamente, en el panel quiere mostrar la evolución en el tiempo de la temperatura y la máxima en las últimas 12 horas medidas por el sensor.

Acciones para llevar a cabo el caso de uso:

1. Pulsar en el botón de añadir QR en la parte superior de la tabla de gestión de los QR. El sistema le llevará a la página de configuración de un QR, para comenzar con el proceso de creación.
2. Rellenar los campos del formulario con la información que quiere para el QR. Como mínimo deberá llenar los campos obligatorios: título y fecha de validez. Hasta que no se llenen estos dos campos, el sistema no permitirá finalizar el proceso de creación. Una vez se crea el QR, se mostrarán las demás opciones de configuración, entre ellas la gestión de sus llamadas.

Configuración del QR

Datos del QR

Título Desactivado

Fecha de validez

Tiempo en segundos para recargar panel 0 para no recargar

Mostrar botón compartir

Datos de impresión

Nombre de etiqueta Tamaño impresión

Descripción de etiqueta

Caso de uso



Primer caso de uso

Llamadas del código QR

Buscar llamadas...

Nombre	Activar/Desactivar	Editar	Duplicar	Eliminar	Bajar	Subir
No has creado ningún código QR todavía.						

*Figura 110. Código QR creado.
(Fuente propia)*

3. Para conseguir lo que quiere, el usuario debe añadir dos llamadas al QR. Por lo tanto, el siguiente paso es pulsar en el botón de añadir llamada en la parte superior de la tabla de gestión de las llamadas. El sistema dirigirá al usuario a la página de configuración de una llamada.
4. Para la evolución de la temperatura, el usuario va a utilizar una gráfica de líneas. Lo primero que va a hacer es llenar el nombre y el token que le ha dado un administrador de Smart University para poder realizar la petición a la colección del “sensor-voc-5”.
5. Introducir el rango de fechas. En este caso introduce la fecha de manera absoluta, poniendo en los dos casos el mismo día, pero diferentes horas.
6. Como la gráfica lineal sale seleccionada por defecto, no toca este campo y pasa directamente a introducir los filtros necesarios.
7. Para la primera gráfica necesita dos filtros, el identificador (uid) y la magnitud (name). En primer campo indicar el id del sensor que en este caso es “sensor-voc-5” y en el segundo, al querer obtener la temperatura, indicar de la siguiente forma, “Temperatura”.
8. Pulsar en el botón de guardar. Si se ha llenado todo correctamente, el sistema le indica mediante una alerta que la llamada se ha creado correctamente.

Formulario de configuración Desactivado

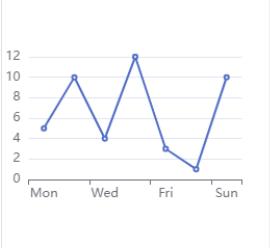
Nombre	Cantidad de datos
<input type="text" value="Evolución temperatura"/>	<input type="button" value="Todos"/>
Token	Filtros +
<pre>eyJhbGciOiJIUzI1NilsInR5cCI6IkpXVCJ9.eyJpYXQiOjE3MDEyNTM5MTR9.jHNbXYKMK-errnM3lgEydWjQ1BICjBCwmhWXcUKPIL4</pre>	Filtro 1 <input type="button" value="uid"/> <input type="text" value="sensor-voc-5"/> Delete
Fecha absoluta	
Desde	Hasta
<input type="text" value="12/12/2023 10:00"/> Calendar	<input type="text" value="12/12/2023 18:00"/> Calendar
Filtro 2	
<input type="button" value="name"/> <input type="text" value="Temperatura"/> Delete	
Selecciona la gráfica para representar los datos	
<input type="button" value="Gráfica lineal"/> ▼	
Volver Guardar	

Figura 111. Llamada configurada con los datos del ejemplo.
(Fuente propia)

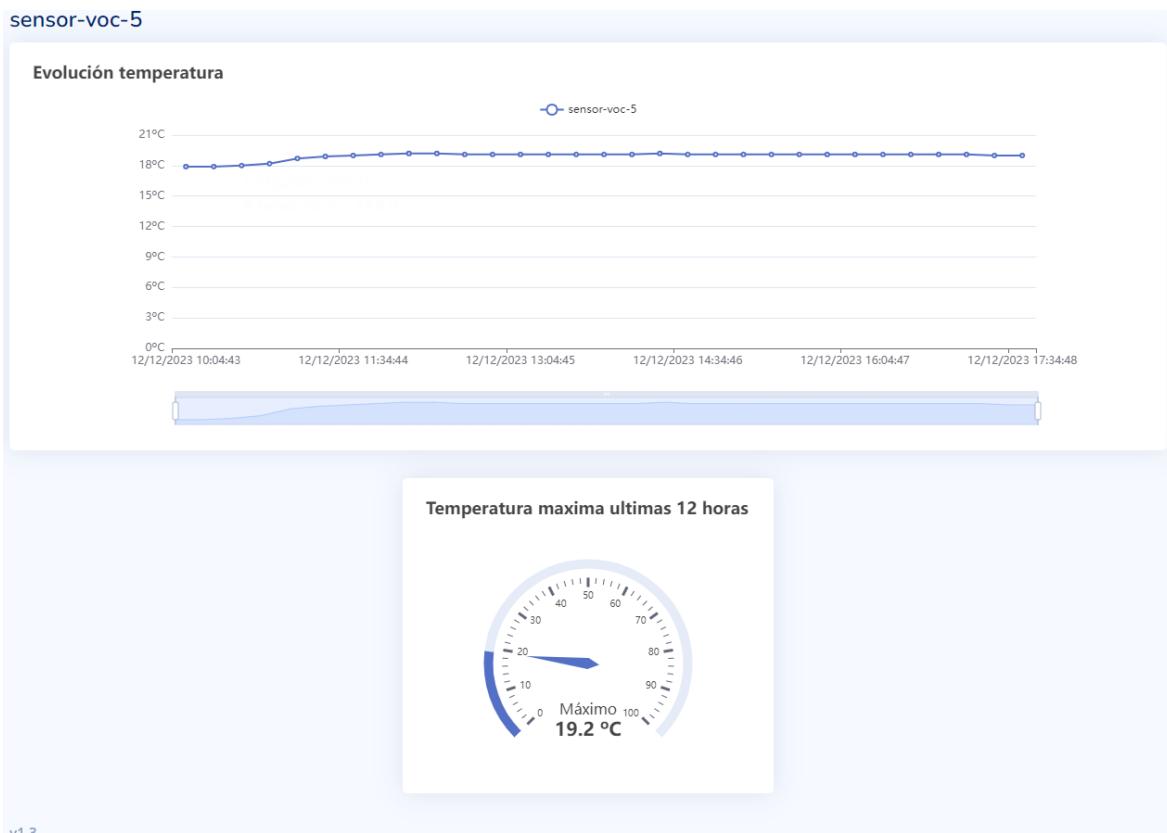
9. El usuario quiere crear otra llamada con el mismo token. El siguiente paso es duplicar la llamada. El sistema le dirigirá a la configuración de la llamada con los datos de la primera.
10. Modifica la llamada para obtener el resultado deseado. Esta vez introduce las fechas de forma relativa, poniendo un 12 en el primer campo y seleccionando las horas en el segundo.
11. En el campo de la cantidad de datos, selecciona la opción del máximo dato.
12. En el campo de las gráficas selecciona el de Gauge.
13. En los filtros obligatorios, indicar la misma información de la llamada anterior que en el paso 7. En este caso, se indica que el dato se muestre con 2 decimales.
14. Guardar la llamada.

Formulario de configuración **Activado**

Nombre	Cantidad de datos
<input type="text" value="Temperatura maxima ultimas 12 horas"/>	<input type="text" value="Máximo"/>
Token	Filtros
<input type="text" value="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOjE3MDEyNTM5MTR9.jHNbXYKMK-errnM3lgEydwjQ1BICjBCwmhWXcUKPIL4"/>	UID: <input type="text" value="sensor-voc-5"/> Magnitud: <input type="text" value="Temperatura"/> Decimales: <input type="text" value="2"/>
<input type="checkbox"/> Fecha absoluta	<input checked="" type="checkbox"/> Fecha relativa
Cantidad	Unidad de tiempo
<input type="text" value="12"/>	<input type="text" value="Horas"/>
<input type="button" value="Volver"/> <input type="button" value="Guardar"/>	
Selecciona la gráfica para representar los datos Gráfica de gauge <input type="button" value="▼"/> Icono: <input type="button" value="No icon"/>	
	

*Figura 112. Segunda llamada del ejemplo.
(Fuente propia)*

15. En la configuración del QR, activar el propio QR y las llamadas que ha creado para que se puedan visualizar.
16. Pulsar en el enlace a la vista pública (“Visitar enlace” en la Figura 110) y ver como se representan los datos que ha solicitado.



*Figura 113. Resultado del panel configurado en el caso de uso.
(Fuente propia)*

8.2. Caso de uso 2. Modificar el panel

El usuario desea modificar el panel porque ahora quiere que en lugar de mostrarse la temperatura máxima de las últimas 12 horas, sean de las últimas 24 horas. Además, quiere añadir una gráfica que muestre el valor de la temperatura mínima de las últimas 24 horas.

Acciones para llevar a cabo el caso de uso:

1. Pulsar en el botón de editar del código QR en la tabla de gestión de la página de inicio.
2. Pulsar en el botón de editar de la llamada que muestra las últimas 12 horas en la tabla de gestión de las llamadas.
3. Hay dos formas de cambiar a 24 horas. Cambiando el 12 por un 24, ya que estaban seleccionadas las horas o cambiando el 12 por un 1 y seleccionando los días.
4. Pulsar en el botón de guardar.

Formulario de configuración **Activado**

Nombre

Cantidad de datos

Máximo

Token

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOjE3MDEyNTM5MTR9.jHNbXYKMKc-errnM3lgEydWjQ1BICjBCwmhWXcUKPIL4
```

Filtros

UID	<input type="text" value="sensor-voc-5"/>
Magnitud	<input type="text" value="Temperatura"/>
Decimales	<input type="text" value="2"/>

Fecha absoluta

Fecha relativa

Volver **Guardar**

Cantidad

Unidad de tiempo

Horas

Selecciona la gráfica para representar los datos

Gráfica de gauge

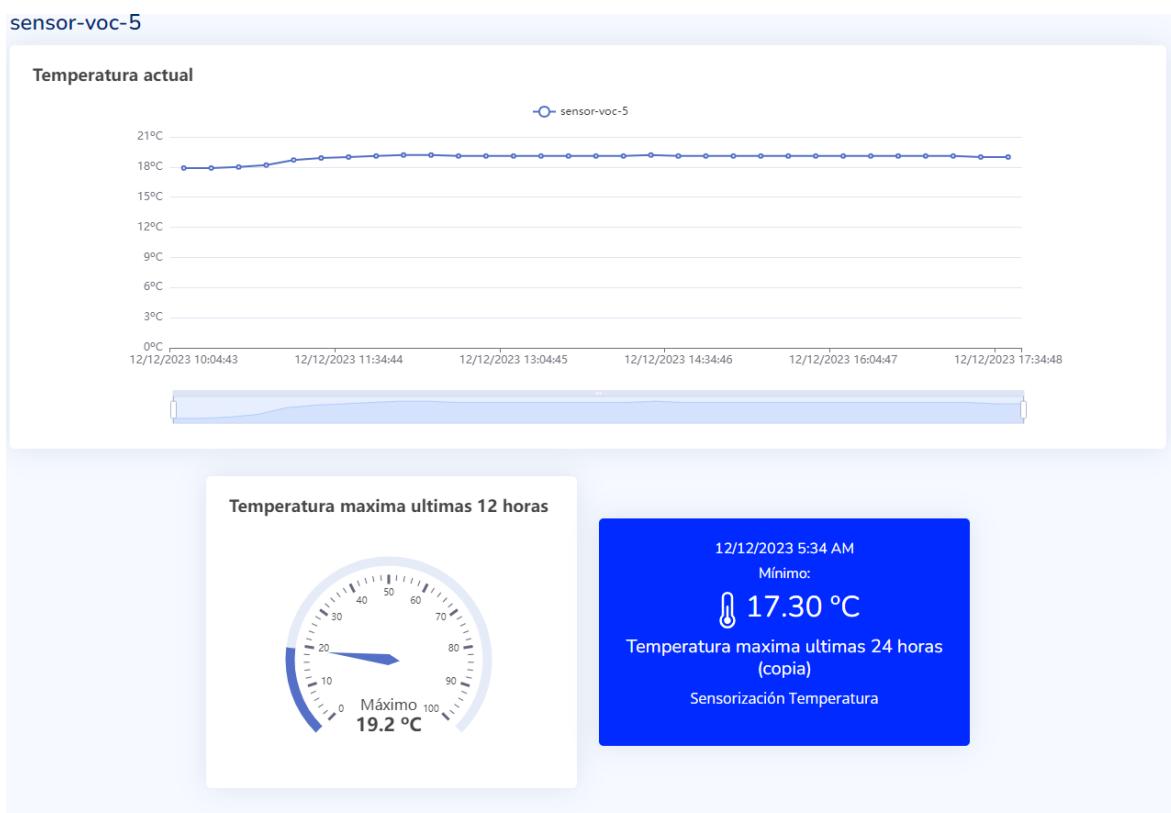
Icono

No icon



*Figura 114. Llamada actualizada para mostrar las últimas 24 horas.
(Fuente propia)*

5. Dirigirse a la gestión de las llamadas y duplicar la que se acaba de modificar.
6. En la sección de las gráficas, cambiar la de Gauge por la que solo muestra el valor.
7. Añadir un ícono y cambiar el color del fondo y del valor si se desea.
8. Seleccionar el mínimo dato en el campo de la cantidad de datos.
9. Pulsar en el botón de guardar.
10. Activar la nueva llamada.
11. Dirigirse de nuevo a la vista pública del QR y ver como se han aplicado las modificaciones correctamente.



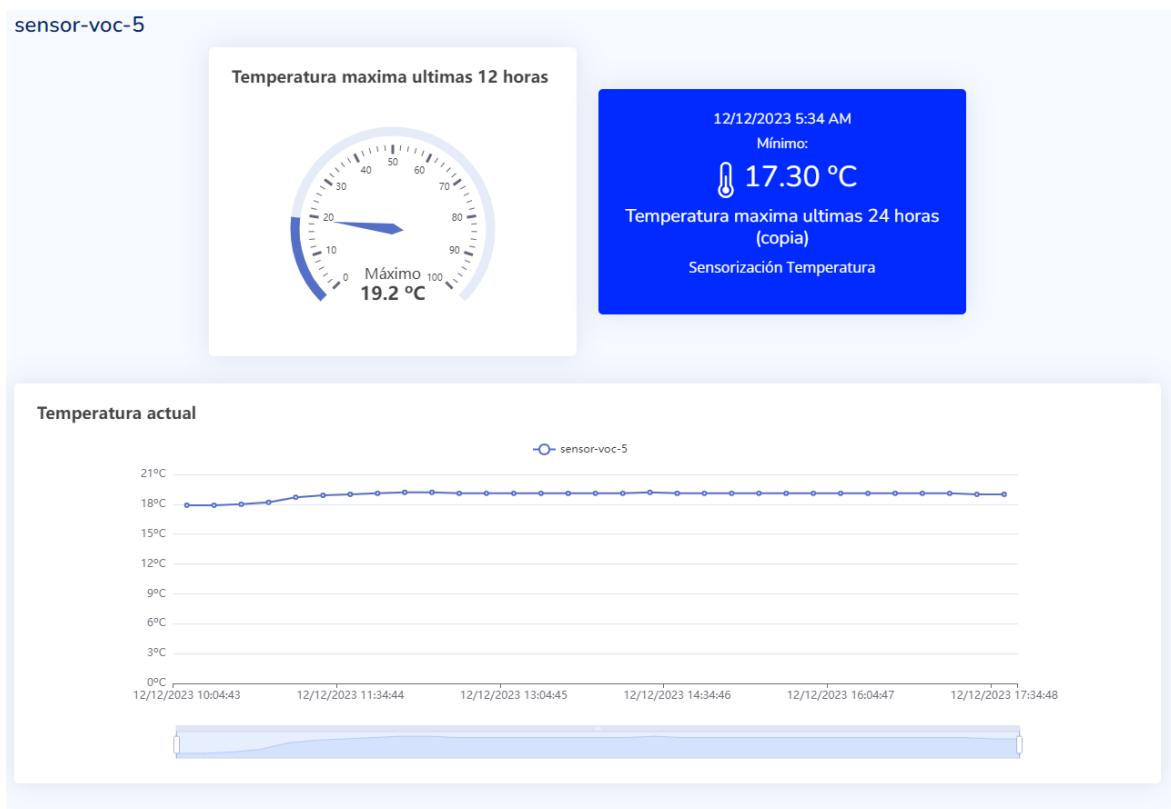
*Figura 115. Panel de control modificado.
(Fuente propia)*

8.3. Caso de uso 3. Modificar el orden del panel

Ahora el usuario quiere cambiar el orden en el que se ven las gráficas del panel. Quiere que la gráfica de líneas se sitúe en la última posición.

Acciones para llevar a cabo el caso de uso:

1. Dirigirse a la página de configuración del QR pulsando en el botón de editar.
2. En la tabla de gestión de las llamadas, pulsar en el botón de la columna de bajar orden. Como es la primera, se debe bajar dos veces. El sistema notificará al usuario que el orden se ha modificado correctamente.
3. Dirigirse a la vista pública del QR y ver cómo ha cambiado el orden.



*Figura 116. Panel de control con el orden modificado.
(Fuente propia)*

8.4. Caso de uso 4. Desactivar el panel

Ahora el usuario quiere desactivar el panel para que no se pueda visitar, ya que desea hacer algunas modificaciones antes. Esto se puede realizar desde la página de inicio y la página de configuración del QR.

Acciones para llevar a cabo el caso de uso:

1. Dirigirse a la página de inicio y situar el QR en la tabla de gestión.
2. Pulsar en el check de la columna Activar/Desactivar.
3. El sistema notifica al usuario que se ha desactivado correctamente el QR mediante una alarma.

Códigos QR		Buscar códigos QR...		Código QR desactivado correctamente				
Tus códigos QR		Descargar lista						
Imagen QR	Título	Fecha validez	Activar/Desactivar	Editar	Duplicar	Descargar	Añadir a lista	Eliminar
	Consumo electrónico	29/12/2023	<input checked="" type="checkbox"/>					
	VOC1	1/1/2030	<input checked="" type="checkbox"/>					
	VOC1 (copia)	1/1/2030	<input checked="" type="checkbox"/>					
	sensor-voc-5	26/12/2023	<input checked="" type="checkbox"/>					

« 1 »

*Figura 117. Código QR desactivado.
(Fuente propia)*

4. Dirigirse a la vista pública del QR y ver como el panel ya no es visible. En su lugar se muestra un mensaje informando al usuario de la situación.

sensor-voc-5

El código QR no se encuentra activo en estos momentos. Inténtelo más tarde.

*Figura 118. Vista pública del QR cuando está desactivado.
(Fuente propia)*

8.5. Caso de uso 5. Eliminar panel

El panel ya no se encuentra en uso y el usuario quiere eliminarlo definitivamente.

Acciones para llevar a cabo el caso de uso:

1. Situarse en la página de inicio.
2. Pulsar en el botón de eliminar del QR.
3. El sistema lanza un mensaje modal al usuario para pedirle su confirmación.

4. Pulsar en confirmar.
5. El sistema notifica al usuario que el proceso de eliminación se ha realizado correctamente mediante una alarma.
6. Ver como el QR ya no aparece en la tabla de gestión de los códigos QR.

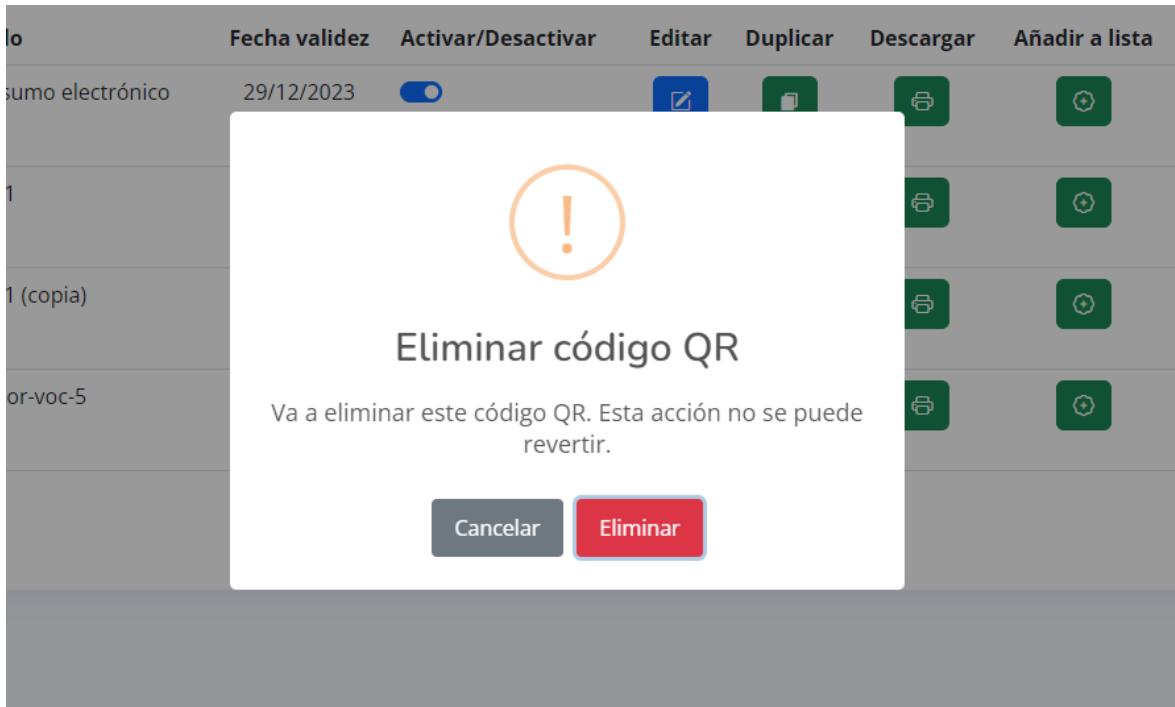


Figura 119. Mensaje modal para confirmar el proceso de eliminación.
(Fuente propia)

9. Resultados

En este apartado se van a comentar y analizar los resultados obtenidos con un producto final entregado al product owner y puesto en producción. Se van a examinar también los costes temporales que ha supuesto el desarrollo de este proyecto.

9.1. Producto final

El producto final alcanzado se aproxima a lo planeado inicialmente, superando incluso las metas establecidas. En un principio, se ideó un sistema donde poder generar códigos QR, a los que añadirle llamadas configurables para representar los datos de Smart University, y posteriormente imprimirlas para poder colocarlos en cualquier lugar de la universidad. Se puede corroborar que el sistema cubre este objetivo principal ya que sus funcionalidades cumplen con los requisitos detallados en la Tabla 2. Sin embargo, a medida que avanzaba el desarrollo surgieron nuevas funcionalidades no previstas inicialmente pero que fueron esenciales para mejorar la experiencia de usuario.

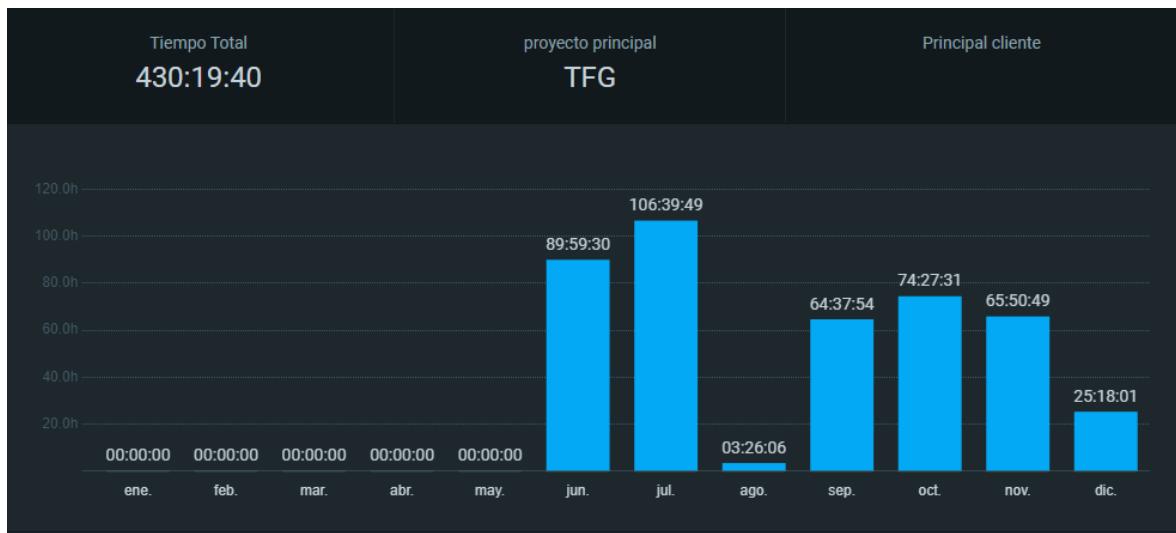
Un ejemplo de estas es la funcionalidad de duplicar, la cual surgió probando la aplicación y viendo que para hacer dos llamadas o códigos QR similares, diferenciándose únicamente por datos concretos, se tenían que crear desde cero. La inclusión de esta funcionalidad no solo facilitó este proceso, si no que añadió un valor significativo al producto final.

En resumen, en producto final no solo cumple los objetivos principalmente establecidos, sino que también se ha ido enriqueciendo de nuevas funcionalidades que le han añadido un valor destacable y ha mejorado la experiencia al trabajar con el sistema.

Cabe destacar que, una vez finalizado la fase de Entrega y puesta en marcha, el producto se puso en producción y se puede probar desde este [enlace](#).

9.2. Coste temporal

A continuación, se van a exponer el coste temporal del proyecto con los datos medidos a través de Clockify. Se va a examinar cual es la fase que ha supuesto más peso y ha necesitado más horas de dedicación. En la siguiente figura, se muestra una gráfica que distribuye el tiempo total por meses.



*Figura 120. Tiempo empleado en el proyecto por mes.
(Fuente propia)*

Se puede observar que se empezó a trabajar desde junio y que se ha finalizado en diciembre, ya que es el mes acordado para la entrega de este trabajo. El mes que ha requerido más tiempo de trabajo por diferencia ha sido junio. Esto se debe a que, en esas fechas, el proyecto se encontraba en la fase de implementación. También se puede destacar, que en agosto casi no se realizaron horas, ya que como se comenta en el apartado Planificación, este mes es en el que cierra la universidad por el comienzo de las vacaciones.

Se ha dedicado un total de 430 horas en el desarrollo completo del proyecto. Pese a que se pensaba terminar con la fase de implementación en los primeros dos meses de trabajo, después de las vacaciones se revisó y se plantearon correcciones y nuevas funcionalidades a implementar, por lo que esta fase se alargó más de lo planeado. Esto se puede ver en el top 10 de las tareas a las que más tiempo se les ha dedicado, Figura 121. De todas formas, esto no supuso un retraso para el proyecto ya que al únicamente tener pendiente la asignatura del Trabajo de Fin de Grado, se le ha podido dedicar más horas de las mínimas estipuladas por la Universidad de Alicante para superarla, 300.

Actividades más registradas	Top 10 ▾
Revision Frontend para despliegue • TFG: Corrección y revisión	26:44:49
Corregir API para despliegue • TFG: Corrección y revisión	21:43:54
Crear gráficas de echart con los datos ... • TFG: Creación de las páginas en el fr...	17:26:03
Pasar los bocetos a Balsamiq/Figma • TFG: Bocetos de las interfaces	17:01:03
Crear página configuración de llamada • TFG: Creación de las páginas en el fr...	15:11:37
Redactar Sprint 8 (9 en la memoria) • TFG: Redactar sprints/iteraciones e...	13:10:40
Redactar Sprint 9 (10 en memoria) • TFG: Redactar sprints/iteraciones e...	12:53:51
Implementar botón imprimir • TFG: Corrección y revisión	12:33:32
Cambiar la llamada que realiza las vist... • TFG: Creación de las páginas en el fr...	08:59:42
Implementar duplicar llamada • TFG: Creación de las páginas en el fr...	08:52:32

*Figura 121. Top 10 de las tareas con más horas dedicadas, medidas en Clockify.
(Fuente propia)*

Se puede apreciar que la tarea que más aparece en esta tabla es la de *Corrección y revisión*, la cual empezó al volver de las vacaciones. Cabe destacar que la primera posición la ocupa las revisiones que se realizaron al frontend para poder desplegarla y ponerla en producción. Esta está seguida de las revisiones que se hicieron al backend con el mismo objetivo.

10. Conclusiones y trabajo futuro

Para finalizar con la memoria, en este apartado se van a comentar las conclusiones obtenidos a partir de los objetivos alcanzados y del trabajo que ha quedado por hacer y mejorar.

10.1. Objetivos alcanzados

Se puede decir sin lugar a duda, que el proyecto ha alcanzado los objetivos establecidos desde su inicio. Si nos dirigimos al apartado de Objetivos y vemos cual era el objetivo principal, se puede ver con claridad que el sistema desarrollado cumple con sus expectativas.

La aplicación es capaz de permitir la personalización de paneles informativos con gráficas que representan cualquier dato obtenido de la plataforma Smart University. También permite generar códigos QR para poder compartir los paneles informativos de manera sencilla. Da total libertad de seleccionar el acceso a los paneles y sus gráficas en todo momento. Además, las funcionalidades nuevas que han ido surgiendo durante el desarrollo, han incrementado el valor del sistema y la utilidad para los usuarios.

A partir de ahora, cualquier usuario, perteneciente o no a la comunidad universitaria, podrá acceder a los paneles informativos para visualizar todos los datos de manera amigable y accesible, en cualquier momento. De este modo, se soluciona el problema planteado al inicio del proyecto, permitiendo a todas las personas comprender los datos de Smart University, anteriormente de difícil interpretación.

Finalmente, con la implementación exitosa del proyecto y su puesta en producción, comienza la fase de su uso activo. El feedback de los usuarios, que vayan escaneando los QR y accediendo a los paneles informativos, es un recurso muy valioso para evaluar y continuar mejorando el sistema.

10.2. Trabajo futuro

Pese a que se ha conseguido alcanzar la mayoría de las funcionalidades y objetivos establecidos, hay algunos otros que no se han podido implementar. A continuación, se recoge una lista de estos:

- **Creación de una tabla para los logs en la base de datos.** Cada acción que se realizase en el sistema se registraría en esta tabla.

- **Sección analítica para los administradores.** Esto se realizaría a partir del punto anterior para que los administradores pudiesen analizar cualquier aspecto del sistema. Un ejemplo sería poder ver el número de veces que se ha escaneado un código QR en concreto.
- **Seleccionar una imagen para los QR.** De esta forma se podría agregar cualquier imagen que queramos para que funcione de logo del QR, aumentando así la personalización.
- **Recuperar contraseña.** Como esta funcionalidad no se ha llegado a implementar, por el momento son los administradores los que tienen que editar las contraseñas de los usuarios y volverlas a proporcionar en caso de olvido.

Esta es una pequeña lista de las principales funcionalidades que se pensaron desde el inicio, pero que al final no se han llegado a implementar. No obstante, esta lista puede aumentar con las necesidades que vayan surgiendo a partir del feedback de los usuarios.

Para finalizar, me gustaría decir que me siento muy orgulloso de lo que se ha conseguido con este proyecto. Jamás pensé que, en mi paso por la universidad, iba a participar en un proyecto que fuese de utilidad para todas las personas y mucho menos que se fuera a poner en producción.

No me arrepiento para nada de haber escogido este proyecto para mi TFG, ya que, gracias a él y a mis tutores, he aprendido y mejorado muchos aspectos vistos durante mi recorrido por el grado. Además, me ha servido para demostrarme a mí mismo de lo que soy capaz y de que puedo seguir mejorando si me lo propongo. Estoy convencido de que esta experiencia me servirá para seguir aprendiendo y trabajando en mi futuro profesional.

Referencias

1. Grafana Labs. *Grafana: The Open Observability Platform* | Software libre que permite la visualización y el formato de datos métricos. Disponible en: <https://grafana.com/>
2. Documentación de la OpenAPI de Smart University de la Universidad de Alicante. Disponible en: <https://openapi.smartua.es/doc/>
3. *Especificación de Requisitos según el estándar de IEEE 830*. Disponible en: <https://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf>
4. Balsamiq. Herramienta para crear wireframes. Disponible en: <https://balsamiq.com/>
5. FigMa: The Collaborative Interface Design Tool | Programa de edición y prototipado dirigido principalmente al diseño web. Disponible en: <https://www.figma.com/>
6. Postman. Herramienta para análisis de API REST. En forma de plugin para Chrome y aplicación. Disponible en: <https://www.getpostman.com/>
7. NiceAdmin. Plantilla de administración y paneles basada en la última versión del framework Bootstrap. Disponible en: <https://bootstrapmade.com/nice-admin-bootstrap-admin-html-template/>
8. Jdnj1. (2023). GitHub - jdnj1/My-University-QR. Repositorio del proyecto disponible en: <https://github.com/jdnj1/My-university-QR>
9. *Visual Studio Code - Code Editing. Redefined.* (2021, 3 noviembre). Disponible en: <https://code.visualstudio.com/>
10. XAMPP installers and downloads for Apache Friends. Disponible en: <https://www.apachefriends.org/es/index.html>
11. Contributors, P. phpMyAdmin. Disponible en: <https://www.phpmyadmin.net/>
12. *GitHub: Let's build from here*. GitHub. Disponible en: <https://github.com>
13. *Gestiona los proyectos de tu equipo desde cualquier lugar | Trello*. Disponible en: <https://trello.com/es>
14. Clockify. (2017, 1 septiembre). *Clockify - Software de control del tiempo GRATIS*. Disponible en: <https://clockify.me/es/>
15. NPM: angularx-qrcode. Librería de generación de códigos QR. Disponible en: <https://www.npmjs.com/package/angularx-qrcode>

16. SweetAlert2. Librería de mensajes modales accesibles. Disponible en:
<https://sweetalert2.github.io/>
17. Modern JavaScript Date Utility Library. Disponible en: <https://date-fns.org/>
18. NPM: Axios. Cliente HTTP basado en promesas para node.js y el navegador. Disponible en:
<https://www.npmjs.com/package/axios>
19. Apache ECharts. Librería de visualización de datos. Disponible en:
<https://echarts.apache.org/en/index.html>
20. Ngx-Translate. Librería de internacionalización para Angular. Repositorio de GitHub disponible en: <https://github.com/ngx-translate/core>
21. jsPDF. Librería para generar archivos PDF. Documentación disponible en:
<https://raw.githack.com/MrRio/jsPDF/master/docs/index.html>
22. HTML2Canvas. Capturas con JavaScript. Disponible en: <https://html2canvas.hertzen.com/>
23. Docker Desktop: The #1 containerization tool for developers | Docker. (2023, 29 agosto). Disponible en: <https://www.docker.com/products/docker-desktop/>
24. NPM: MySql. Librería de node.js para MySQL (Obsoleta). Disponible en:
<https://www.npmjs.com/package/mysql>
25. NPM: Mysql2. Versión más reciente de la librería de node.js para MySQL. Disponible en:
<https://www.npmjs.com/package/mysql2>
26. NPM: Password-validator. Librería de node.js para validar contraseñas. Disponible en:
<https://www.npmjs.com/package/password-validator>
27. Paths - SVG: Scalable Vector Graphics | MDN. (2023, 1 noviembre). MDN Web Docs.
<https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Paths>
28. Bootstrap Icons. Biblioteca de iconos de Bootstrap. Disponible en:
<https://icons.getbootstrap.com/>

Apéndice

Este apéndice se aprovecha para dejar el enlace al repositorio del proyecto e indicar las instrucciones de cómo configurarlo para su despliegue por si cualquiera quiere probarlo.

El enlace del repositorio en GitHub es el siguiente: <https://github.com/jdnj1/My-university-QR>.

El repositorio está compuesto principalmente por dos carpetas llamadas Backend y Frontend. La primera carpeta contiene todo el código relacionado con las funciones de la API de la aplicación y con la conexión de la base de datos. Dentro de esta hay una carpeta llamada *bd* donde se encuentra el script SQL con la estructura de la base de datos, el cual se puede importar en cualquier base de datos relacional. Esta carpeta cuenta también con un archivo de texto llamado *variablesEnt.txt* donde se encuentran las variables de entorno necesarias para el funcionamiento del backend.

En cuanto a la carpeta Frontend, esta contiene todos los archivos y código relacionado con el lado del cliente de la aplicación. En ella se encuentran todos los componentes y los servicios que usan, además de los guards que se encargan de controlar la navegación por los componentes.

Por último, se ha añadido una nueva carpeta llamada Docker, la cual contiene un archivo llamado *docker-compose.yml* que se encarga de crear los contenedores necesarios para arrancar el proyecto en la aplicación de Docker [23]. Esta forma de arrancar la aplicación se explica a continuación.

Para probar la aplicación desde el dispositivo de forma local hay dos formas. No obstante, en los dos casos se tiene que crear un archivo con la extensión *env* para las variables de entorno dentro de la carpeta *api*. Pegando las variables del archivo de texto *variablesEnt.txt* en este archivo se podrá comenzar con la puesta en marcha de la aplicación. También se deja a continuación un usuario con el rol de administrador para poder entrar en la parte privada de la aplicación.

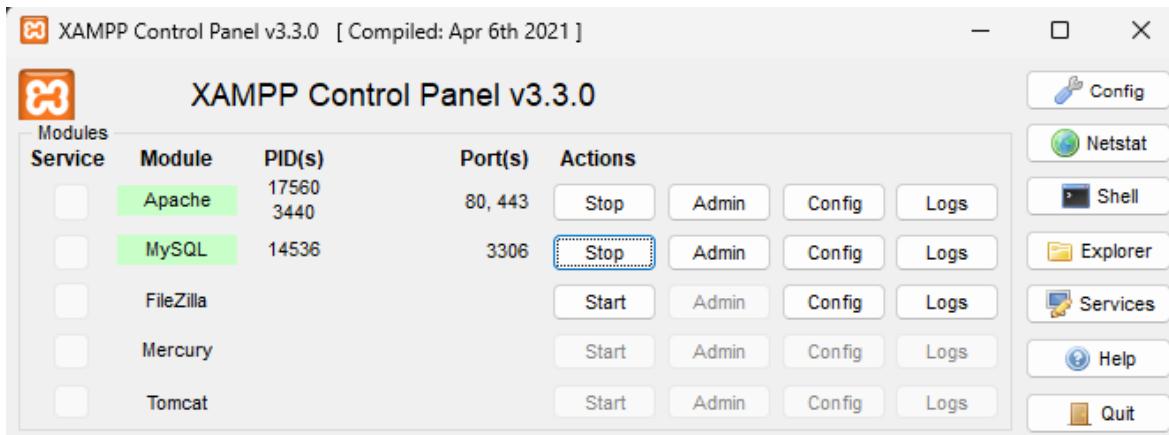
- Email: userpass@gmail.com
- Contraseña: 12345

Con este usuario se puede entrar en la parte privada donde se pueden visitar y editar los códigos QR que ya hay creados o crear los que se deseen (este usuario no tiene límites de creación de QR).

El primer paso en ambos métodos es clonar el repositorio: `git clone https://github.com/jdnj1/My-university-QR.git`

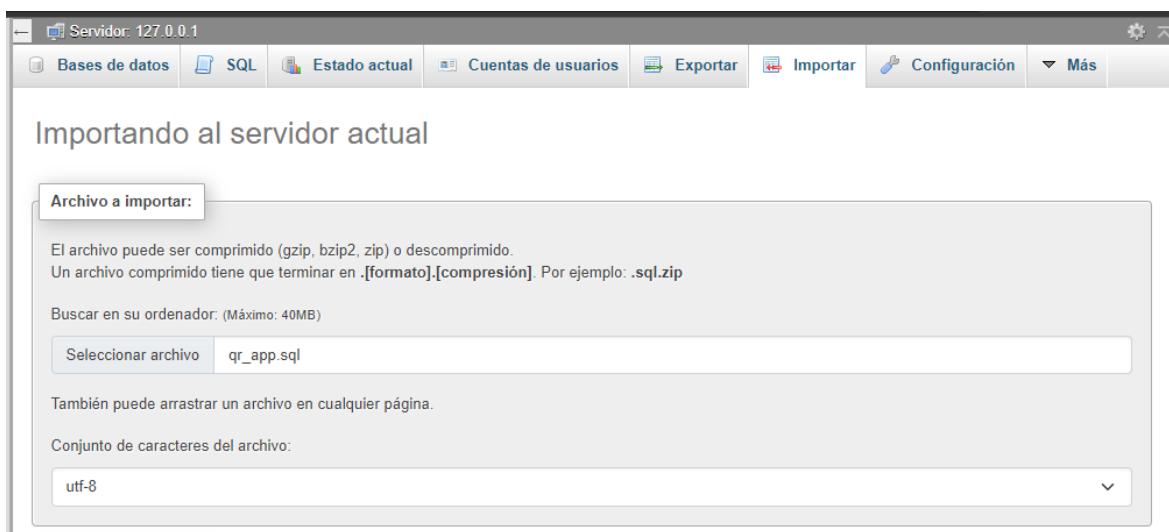
En la primera forma es necesario el uso de dos terminales, una abierta dentro de la carpeta de la API de la aplicación la cual se encuentra en *backend/api* y otra abierta dentro de la carpeta Frontend. Es importante aclarar que con este método se necesita crear un entorno de desarrollo

web local en el equipo para que todas las conexiones con la API y la base de datos puedan funcionar correctamente. Una opción popular para poder crear este tipo de entornos es XAMPP, el cual es un software fácil de instalar y configurar y es el que se ha usado durante el desarrollo del proyecto. Dentro de XAMPP solo es necesario arrancar Apache y MySQL.



*Figura 122. Panel de control de XAMPP con Apache y MySQL activos.
(Fuente propia)*

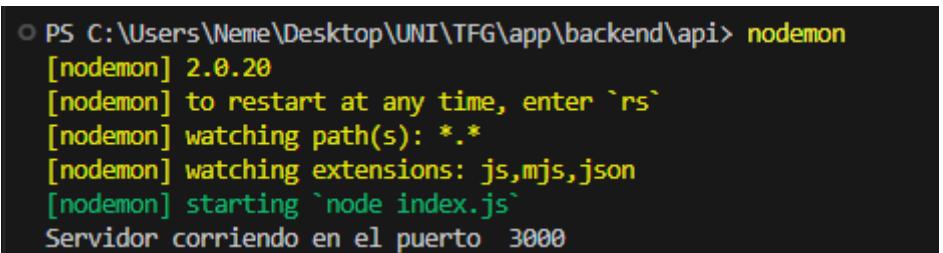
También es necesario importar el archivo SQL a la base de datos que proporciona XAMPP. Por lo tanto, cuando XAMPP esté activo se podrá acceder a phpMyAdmin [11] e importar la base de datos en el siguiente enlace: <http://localhost/phpmyadmin/>. Simplemente hay que crear una base de datos con el nombre de *qr_app* y dirigirse a la pestaña de importación.



*Figura 123. Pestaña de importar la base de datos en phpMyAdmin.
(Fuente propia)*

Una vez están las dos terminales en los directorios, el entorno de desarrollo esté activado y la base de datos se haya creado, se empieza a poner en marcha la aplicación. Primero se debe arrancar la

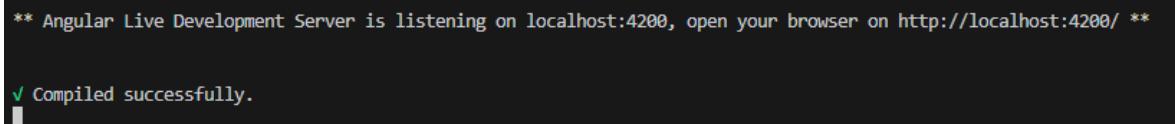
API para poder iniciar sesión y para ello lo primero que hay que hacer es instalar los paquetes que utiliza el backend con el comando *npm install*. Cuando se hayan instalado todos los paquetes se debe poner el comando *nodemon* en la terminal (también se puede poner en marcha con la orden *npm start* solo que si se realiza algún cambio en el código se tendría que volver a iniciar la API para que se vean los cambios). Si todo sale bien se indicará en la terminal con la frase “Servidor corriendo en el puerto 3000”.



```
PS C:\Users\Neme\Desktop\UNI\TFG\app\backend\api> nodemon
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Servidor corriendo en el puerto 3000
```

Figura 124. Resultado de iniciar la API desde una terminal.
(Fuente propia)

A continuación, se arranca la parte del frontend de manera similar. Como en el caso anterior, primero se instalan los paquetes con *npm install* y luego hay que ejecutar el comando *ng serve* y esperar unos instantes. Cuando la terminal indique que todo se ha compilado correctamente, se puede abrir la aplicación en local con el siguiente enlace: <http://localhost:4200/>. Se abrirá la aplicación en la página de inicio de sesión en la que se tienen que introducir los datos del usuario facilitado para poder acceder.



```
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
[✓] Compiled successfully.
```

Figura 125. Resultado de iniciar el frontend desde una terminal.
(Fuente propia)

Como se ha comentado antes, la otra forma de arrancar la aplicación en local se hace con Docker. Con este método no es necesaria la instalación de los paquetes ni la creación de la base de datos ya que los contenedores de Docker se encargan de todo ello. Para arrancar la aplicación de esta forma primero hay que instalar la aplicación Docker Desktop [23]. Una vez está instalada en el equipo se requerirá la creación de una cuenta de usuario. Posteriormente, se podrá comenzar a usar la aplicación. En la pestaña de los contenedores se puede observar que ya hay un contenedor creado de prueba el cual se puede arrancar pulsando en el botón de la columna *Actions*.

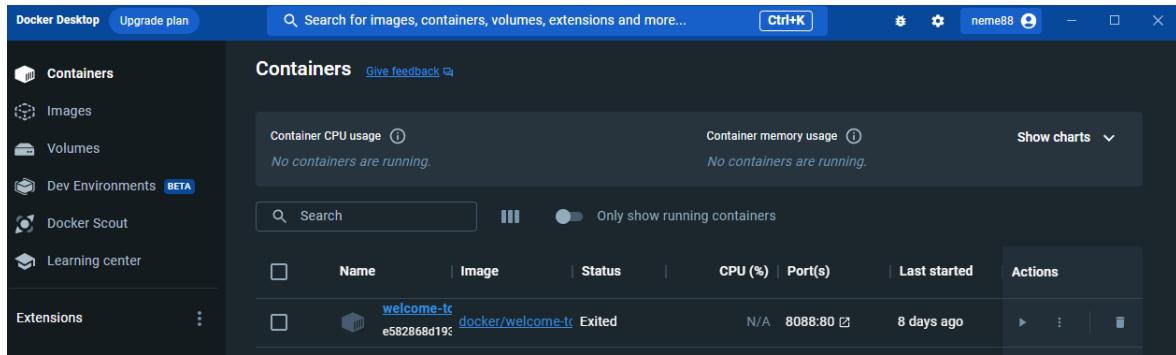


Figura 126. Sección de los contendedores de Docker con el contenedor de prueba.
(Fuente propia)

Estos contenedores son los que se van a encargar de arrancar la aplicación. Primero hay que crear los contenedores que la aplicación necesita para funcionar. En total se tienen que crear 3 contenedores: uno para la API, otro para la base de datos y otro para el frontend. Docker se encarga de crearlos automáticamente. Es necesario comentar que, para que el contendedor de la base de datos funcione correctamente, en el archivo de las variables de entorno hay que cambiar la variable HOST de *localhost* a *mariadb* ya que el contenedor de la base de datos se crea a partir de su imagen.

Para ello, se tiene que abrir una terminal dentro de la carpeta Docker del proyecto mencionada anteriormente. Una vez allí se ejecuta el siguiente comando: *docker-compose up -d*. Este comando creará los contenedores necesarios para poner en marcha esta aplicación. Una vez el proceso finaliza, se podrá observar que en la sección de los contenedores se han creado los tres contenedores y se han agrupado. También se podrá observar que se han arrancado después de su creación, no obstante, puede ser que en la columna del estado del contenedor de la API indique que no se ha iniciado correctamente. Esto se debe a que la API necesita que primero se inicie el contenedor de la base de datos para poder conectarse a esta. Si se pulsa manualmente en el botón de iniciar funcionará correctamente. Finalmente queda entrar en el mismo enlace que en el método anterior (<http://localhost:4200/>) y se podrá observar como la aplicación se ha puesto en marcha. Ahora, cada vez que se quiera probar la aplicación, simplemente habrá que pulsar en el botón de iniciar del grupo de los contenedores.

	Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
□	docker		Exited	0%		15 minutes ago	▶ ⚙️ ⚡
□	mariadb	mariadb	Exited	0%	3306:3306 ↗	15 minutes ago	▶ ⚙️ ⚡
□	angular	docker-angular	Exited (137)	0%	4200:4200 ↗	15 minutes ago	▶ ⚙️ ⚡
□	api	docker-api	Exited (1)	0%	3000:3000 ↗	15 minutes ago	▶ ⚙️ ⚡

Figura 127. Grupo de los contenedores de la aplicación.
(Fuente propia)