Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

Diseño e implementación de una herramienta online y colaborativa para el análisis temporal de la fiabilidad de encuestas, sondeos y estimación de indicadores

Autor: Francisco Javier García López

Tutor: Javier Macías Guarasa

2022

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

Trabajo Fin de Grado

Diseño e implementación de una herramienta online y colaborativa para el análisis temporal de la fiabilidad de encuestas, sondeos y estimación de indicadores

Autor: Francisco Javier García López

Tutor: Javier Macías Guarasa

Tribunal:

Presidente: Julio Pastor Mendoza

Vocal 1º: Ángel Llamazares Llamazares

Vocal 2º: Javier Macías Guarasa

Fecha de depósito: 21 de septiembre de 2022

Resumen

El objetivo es desarrollar una herramienta informática en la que el uso colaborativo de distintos usuarios lleve a poder identificar la fiabilidad de distintos sondeos, estimaciones y predicciones. Se ha analizado el problema para poder identificar con exactitud los requisitos necesarios. Con los requisitos identificados se ha diseñado la herramienta en forma de aplicación web empleando metodologías y tecnologías actuales y finalmente se ha implementado para que ofrezca todas las funcionaldiades necesarias para que los requisitos identificados cumplan el objetivo principal y ofrezca los resultados deseados. Finalmente se ha desplegado para que sea posible su acceso desde cualquier sitio con un acceso a internet.

Palabras clave: Seguimiento, sondeos, estimadores, fiabilidad, Blazor.

Abstract

The objective is to develop a software in which the collaborative use of different users leads to being able to identify the confidence of different surveys, estimates and predictions. The problem has been analyzed in order to accurately identify the necessary requirements. With the identified requirements, the software has been designed in the form of a web application using current methodologies and technologies and finally it has been implemented so that they offer all the necessary functionalities so that the identified requirements meet the main objective and offer the desired results. Finally, it has been deployed so that it can be accessed from anywhere with internet access.

Keywords: tracking, surveys, estimators, reliability, Blazor.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Nadie aprende sin desear aprender[1].

Simone Weil

1.1 Contextualización

Actualmente todas las encuestas, estimaciones, previsiones o sondeos son ofrecidos por fuentes concretas, como puede ser la previsión de crecimiento del euríbor por distintos bancos, o sondeos electorales tanto por medios de comunicación como fuentes oficiales. Una vez estas fuentes generan esas predicciones son pocas las que hacen una comparación final, comparándolas con los valores reales que finalmente se obtienen. En casos puntuales, dicho análisis a posteriori puede ser comentado de forma independiente en tertulias o artículos, con comparaciones muy dispersas y que ofrecen poca información contrastable.

Existen algunas páginas web que tienen una finalidad similar a la de la herramienta que en este proyecto se pretende realizar, como por ejemplo https://electocracia.com/. Una página agregadora de fuentes que las compara y muestra la evolución de las mismas. Pero que se limita a elecciones generales y son los propios propietarios de la página web los encargados de meter los datos y generar los distintos gráficos mostrados. En base a esos gráficos la web añade artículos y todo tipo de análisis estadístico y político.

Por lo tanto aunque incluye información, no es colaborativa y por muchas fuentes que añada siempre serán las que se quieren añadir manualmente por parte de los propietarios. Puede que por intereses propios o por popularidad de las fuentes seleccionadas. El caso es que si se busca una herramienta colaborativa ha de quedar en la mano de los usuarios el poder añadir esa información, distintas fuentes, complementar sondeos que faltan de fuentes ya presentes, etc. Y en base a toda esa información generar los gráficos y distintos datos estadísticos. La unión de todos los usuarios crean el conjunto de datos que compondrán las estadísticas de la herramienta colaborativa, haciendo imposible pensar que no haya transparencia porque queda en mano de cada uno añadir los datos o fuentes que no están aún presentes para crear un mapa completo de los datos. Que todo usuario pueda introducir información buscando una visibilidad podría a la vez llevar a tener datos inventados o de dudosa procedencia, por lo que será necesario que exista un sistema de verificación llevado a cabo por los propios usuarios para que igual que todos pueden introducir información, también se puedan corregir unos a otros en busca de un equilibrio entre libertad y calidad.

[1] Simone Wil, La gravedad y la gracia, Editorial Trotta, 2007.

1.2 Motivación

La motivación principal para afrontar este proyecto y diseñar y desarrollar una herramienta radica en el adjetivo “”colaborativa”. No se está buscando simplemente crear una fuente de información que consultar para poder contrastar datos, sino que se busca que sea creada entre todo el mundo de forma colaborativa buscando que todos pueden poner todo tipo de fuentes existentes, de forma que todas puedan compararse entre sí y también ver hasta qué punto han resultado ser fiables.

De esta forma podemos tener fuentes en las que existan intereses económicos, políticos o incluso sociales muy diferentes y todos sean tenidos en cuenta. También hace que la transparencia sea total, haciendo que no se pueda plantear dudas respecto a los datos o fuentes elegidas, pues cada uno podrá registrarse y añadir las fuentes y datos que considera que faltan y sin importantes.

La herramienta tendrá un sistema de validación para los datos registrados para evitar repetición de datos o datos mal introducidos, pero sólo cumplirá esa función. Cualquier dato bien introducido será aceptado. Esta fase de validación se plantearía con administradores aceptando o descartando datos, pero en caso de identificarse como no satisfactoria buscando esa transparencia total se podría replantear, tanto permitiendo un sistema que requiera varios votos de varios administradores para aceptar o rechazar un dato, o incluso hacer accesible a los usuarios esa votación.

1.3 Objetivos

La herramienta resultante deberá ser capaz de proporcionar la funcionalidad suficiente para permitir introducir todo tipo de sondeos y estimaciones así como previsiones de cualquier tipo, como previsiones macroeconómicas, PIB, paro, etc. Y lo tendrá que hacer de una forma que todos estos datos sean posibles de configurar por el usuario, sin que se tengan que diseñar de forma independiente y exclusiva. La herramienta ha de ser capaz de, a partir de una sencilla configuración por el usuario, aceptar esos datos concretos y distinguirlos de otros. Y a la vez de permitir introducir distintos tipos de datos, tendrán que ser divididos por fuentes y ser válidos para ser tratados en gráficos estadísticos comparativos.

Todas estas posibles configuraciones deberán suponer pantallas independientes para que luego esos datos configurados puedan ser usados o simplemente consultados por otros usuarios en otras pantallas con otras funcionalidades.

1.4 Estructura de la obra

La realización del trabajo se dividirá en distintos capítulos, siendo los siguientes: objetivos, análisis, diseño, implementación y finalmente el despliegue.

1.4.1 Objetivos

Es el presente capítulo, que ha sido empleado como introducción al proyecto que se va a desarrollar, se realiza una introducción el problema planteado, los ámbitos en los que interviene y la idea general que se desarrollará para poder afrontarlo y resolverlo. Y cómo la herramienta que se comienza lo podrá resolver. En este punto sólo detalla el resultado que se aspira conseguir y en los siguientes cómo afrontarlo para solventarlo y realizarlo para obtener los resultados deseados. Se da por hecho que el problema se afrontará realizando algún tipo de software que lo solventará, pero serán los siguientes capítulos en los que se decidirá la mejor forma de implementarlo para que los resultados sean satisfactorios.

1.4.2 Análisis

En el análisis se identificarán los requisitos necesarios en la herramientas, es decir, la información que ha de aparecer y cómo interactuar con la misma; el aspecto superficial que se desea que tenga, para hacerse una idea de cómo será la herramienta y su esperada interactividad; los datos que van a usarse en las distintas funcionalidades y una vez identificados estos datos ver las distintas opciones para resolver el problema para finalmente identificar la forma más apropiada de afrontar el problema. Al final del análisis ya se debe tener claro cómo va a ser finalmente la herramienta, qué cosas se van a poder realizar, su forma y qué tecnologías van a ser empleadas para ayudar en la resolución y realización de la herramienta.

1.4.3 Diseño

En el diseño una vez ya se sabe cómo va a ser la herramienta y qué tecnologías emplear, se hará el diseño general de la herramienta. Es decir, se identificará cada pantalla y/o funcionalidad, cómo se diseñará informáticamente y de qué elementos, interacciones y datos se compondrá. Es decir, en este capítulo tendremos que tener terminado el esqueleto general de la herramienta, pero no su creación concreta. Realmente este diseño estará basado en un método concreto de desarrollar software, no tanto en una tecnología o lenguaje concreto. La tecnología elegida se basará en ese modelo, pero el diseño será independiente de ella.

1.4.4 Implementación

En la implementación se detallará cómo se tendrá que codificar cada parte y cómo tendrá que relacionarse. No es el código como tal, que se podrá añadir como un anexo. Será detallar los componentes, clases, etc. y cómo interaccionan cuando son invocados, de forma que quede claro el movimiento de datos y cómo se resuelven las distintas consultas. La información que se ofrezca en este capítulo ha de dejar claro toda la interacción de los distintos componentes software así como el acceso a datos del sistema gestor de bases de datos. Aunque no será el objetivo de este proyecto, al haber estructurado el proyecto en componentes y a la vez cada componentes en funciones con una función concreta permitiría distribuir el trabajo en varias personas para que cada una se encargue de la codificación de cada tarea y luego poder funcionar perfectamente en conjunto. También serán necesarios otros detalles relacionados con la programación en sí, como la gestión interna de sesiones, localizaciones, interacción entre distintas partes, y el acceso a los datos por parte de la herramienta.

1.4.5 Despliegue

En el despliegue todo el desarrollo de la herramienta ha sido completado y se detalla la publicación para que sea accesible por los usuarios. El despliegue puede suponer bases de datos, aplicaciones, hostings, etc. En definitiva, se hace accesible y se prepara para que la herramienta comience a usarse tal y como originalmente había sido planteada.

1.4.6 Resumen

En los objetivos se ha planteado qué es lo que se desea conseguir, en análisis qué tenemos y qué necesitamos para resolver el problema, en el diseño hemos abordado el problema y lo hemos dividido en bloques manejables, en la implementación se ha detallado la composición de cada trozo, y en el despliegue finalmente lo hemos visto completado. El principal objetivo de esta división del trabajo es identificar

todas las necesidades antes de empezar a desarrollar para que cada fase se base en la anterior para ser realizada. Este planteamiento no impide que si se detecta algún detalle que no había sido tenido en cuenta no pueda ser actualizado, pero sí podría suponer un detalle si esa alteración del diseño es demasiado profunda e importante sobre el resultado final.

1.5 Conclusiones

Tras haber identificado lo que se desea conseguir, es necesario ahora darle forma. Para eso será necesario plasmar las ideas en algo concreto y que se pueda dividir de forma que empiece a ser el esqueleto de la herramienta y cómo esa división ayudará a un futuro desarrollo.

Capítulo 2

Análisis del Sistema

Encontrar una dificultad extraordinaria para realizar una acción ordinaria es un favor por el que hay que estar agradecido [1].

Simone Weil

2.1 Introducción

Tras comentar la herramienta que se desea desarrollar, el siguiente paso necesario, una vez se tiene claro el objetivo principal, que no es otro que poder introducir datos y poder almacenarlos y cruzarlos de distintas maneras, es analizar las distintas funcionalidades que tendrá la herramienta, con esas funcionalidades se planteará una primera aproximación visual con diseños de las distintas pantallas, con esas pantallas y funcionalidades se determinará la forma de la herramienta, y finalmente se diseñará la estructura superficial de datos con la que se guardará toda la información.

2.2 Estructura de datos

Antes de determinar los requisitos funcionales es necesario saber, sin necesidad de entrar en detalle, cómo se va a estructurar la herramienta de cara a organizar la información, de forma que pueda archivarse de una forma que sea accesible y fácilmente localizable además de permitir que los datos puedan relacionarse de cara a mostrar conclusiones.

Para determinar cómo estructurar la información hay que tener en cuenta los siguientes elementos:

- **Sondeos y predicciones no definitivas:** Las distintas fuentes ofrecerán datos tanto basándose en encuestas, análisis de datos históricos, análisis de expertos, etc. Pero no serán los datos "reales" que sucederán.
- **Sondeos y predicciones periódicos:** Las distintas fuentes ofrecerán actualizaciones de sus predicciones, pudiendo ser una o muchas.
- **Distintos valores:** Un sondeo o predicción no tiene que limitarse a un simple dato, o un dato puede llevar asociados otros datos. Por lo que se deberán tener en cuenta esa posibilidad.

[1] Simone Wil, La gravedad y la gracia, Editorial Trotta, 2007.

- **Valores sobre distintos elementos:** Los datos ofrecidos por las fuentes podrían ser sobre un sólo actor, como podría ser la previsión de crecimiento del PIB de un país, pero también sobre varios actores, como por ejemplo unas elecciones en las que se presentan muchos candidatos.
- **Datos agregados sobre un nodo:** Cada dato estará asociado a un nodo que los agrupará todos y que estarán condicionados a un resultado final o un dato real, como los resultados de unas elecciones o el dato real de crecimiento del PIB.
- **Nodo dependiente de un nodo superior:** A la vez ese nodo es algo que se realizará periódicamente, como pueden ser elecciones generales cada x años o un informe de crecimiento de PIB semestral, se esta manera es necesario que a la vez estén agrupados sobre un nodo superior que los permita relacionarlos.

Con estos puntos tenidos en cuenta se determina que la información y eje principal de la herramienta será la siguiente (tabla 2.1):

Tipos	Los tipos son el nodo más alto y sobre el que se agrupan todos los datos dependientes.	Ejemplo: Elecciones generales de España
Ediciones	Son las subdivisiones de los tipos y sobre los que se van a capturar datos y compararse.	Ejemplo: Elecciones generales de España 2019
Entidades Participantes	Son las entidades a las que se le asignan valores.	PSOE y PP
Medidas	Es el elemento que identifica qué dato se está indicando que tiene la entidad participante.	Ejemplo: Diputados y votos
Registros	Son los datos concretos de los sondeos o estimaciones que ofrecen las distintas fuentes.	Ejemplo: Barómetro de antena3 en septiembre 2022 para las próximas elecciones generales de España
Fuentes	Son los proporcionadores originales de la distinta información de la que se suministrará la herramienta.	Ejemplo: Sigmados para antena3

Tabla 2.1: Estructura base de la herramienta

Tomemos la imagen de la figura 2.1 para explicar con un ejemplo la organización de la información principal que tendrá la herramienta.

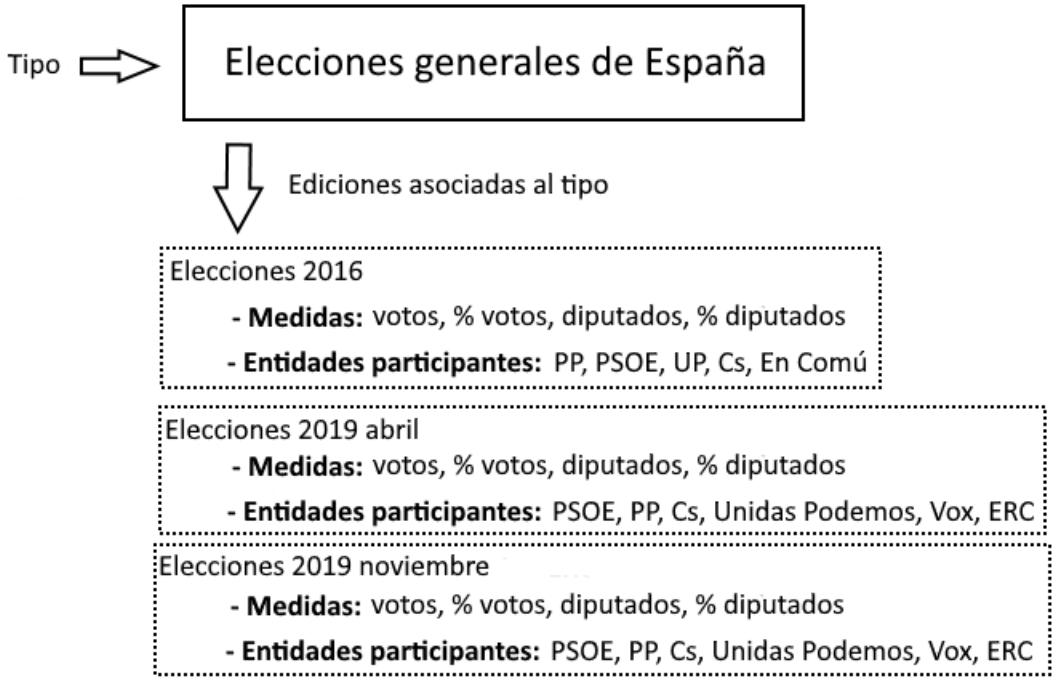

Figura 2.1: Ejemplo de cómo se almacenarían las distintas elecciones generales de España

Las elecciones generales de España son un tipo. Este tipo será el nodo principal sobre el que colgarán todos los datos que se guarden sobre elecciones generales. Posteriormente para empezar a introducir datos será necesario indicar qué elecciones concretas son sobre las que vamos a añadir datos de sondeos electorales. Entonces se crearán ediciones tal como sea necesario para poder diferenciar los sondeos de unas u otras elecciones. En la imagen, por ejemplo, aparecen tres ediciones: la edición 2016 y las dos celebradas en 2019. Cada una de estas ediciones se configura de forma independiente ya que pueden variar los partidos que se presentan, y de ahí de de donde se toma la idea de Entidades Participantes, que en el caso del ejemplo son partidos políticos que concurren a las elecciones concretas de ese año/edición. También será necesario identificar lo que se llama medidas, que es la definición de los datos que se podrán asociar a las entidades participantes, como en ejemplo, votos, diputados y las medidas en porcentaje. Finalmente cuando un medio lance algún sondeo sobre estas elecciones el dato se podrá guardar fácilmente como registros asociados a entidades participantes y medidas, que pertenecen a una edición y a la vez esa edición a un tipo. En capítulos posteriores este diseño base se enriquecerá añadiendo más información que sea identificada en los casos de uso, pero la base se respetará al considerase eficaz para el ordenamiento de la información y sobre la que se podrán relacionar otros datos.

El ejemplo se ha explicado sobre sondeos electorales al ser una forma sencilla y visual de entenderse la estructura, pero es adaptable a cualquier tipo de estimación: por ejemplo, la previsión de crecimiento del PIB: se tendría como tipo “Crecimiento PIB España”, ediciones cada año, entidades participantes una sola siendo “PIB”, y medida “Porcentaje crecimiento”, que perfectamente podría ser negativo.

La estructura de los datos se ha pensado de forma que sea totalmente configurable para todo tipo de sondeos, estimaciones o previsiones y que al introducir todo tipo de datos queden ordenados, para cuando se implemente poder usar los datos de forma sencilla y cruzarlos para llegar al objetivo deseado de sacar conclusiones en base a los datos y a lo que finalmente ha sucedido.

2.3 Requisitos funcionales

Con la idea base de cómo se va a organizar la información y con una idea clara de qué es lo que ha de realizar la herramienta, se determinarán las distintas funcionalidades que serían necesarias para poder interaccionar y llegar a cargar la información, administrarla y mostrarla procesada. Para determinar las funcionalidades se identificarán los distintos requisitos funcionales que tendrá la herramienta online. Puesto que la herramienta estará compuesta de distintas pantallas, que a la vez separarán las distinta funcionalidades, se identificarán esos primeros requisitos funcionales identificando esas pantallas, que gestionarán distintos elementos.

- **Módulo Dashboard:** Al iniciar sesión se entrará en un módulo que mostrará el resumen de los tipos, ediciones y registros creados por el usuario.
- **Módulo Tipos:** El módulo de los tipos proporcionará toda la gestión sobre la creación de tipos.
- **Módulo Entidades participantes:** Las elecciones participantes serán las entidades a las que se asignarán los datos. Se crearán en este módulo.
- **Módulo Ediciones:** Esos tipos sobre los que se guardan los datos almacenados pueden repetirse periódicamente, por lo que será necesario que esos tipos a la vez se dividan en ediciones. Desde este módulo se gestionará la creación de las distintas ediciones y su configuración con medidas y entidades participantes, además de los rangos en esas medidas.
- **Módulo Usuarios:** Desde este módulo se validarán a los usuarios que se han registrado pero aún no tienen acceso y se modificarán los roles de los usuarios creados en el sistema.
- **Módulo Carga Registros Manual:** Se podrán añadir de distintas fuentes datos manualmente. La herramienta mostrará los campos esperados y el usuario los rellenará en base a esas fuentes.
- **Módulo Carga Registros WEB:** La herramienta como complemento a la funcionalidad manual ofrecerá diferentes ayudas para que la introducción de datos desde las fuentes sea más guiada y sencilla encargándose la herramienta de detectar los datos en la página web. No será una pantalla automática, sino que deberá ser supervisada por el usuario, pero en la medida de lo posible ahorrará trabajo manual.
- **Módulo Carga Registros OCR:** Realizará una función de apoyo similar a la carga WEB, pero en este caso analizará imágenes buscando datos en vez de analizar una página web.
- **Módulo Resultados:** En el módulo Resultados se introducirán los resultados finales sobre los datos estimados o previstos de las distintas ediciones, y en base a estos datos se calcularán qué fuentes son más o menos fiables.
- **Módulo Visualizador:** En el módulo visualizador se podrán ver todos los datos almacenados de manera que adquieran una dimensión distinta al ser ensamblados y mostrados en conjunto, pudiendo ver visualmente distinta información y conclusiones a las que se llega procesando los datos de los que dispone la herramienta y que previamente han sido introducidos por los usuarios.
- **Módulo Validación:** Aquí se validarán que registros introducidos son válidos o no. Sólo los válidos serán tenidos en cuenta para ser mostrados.

A continuación se identificarán los requisitos funcionales concretos de cada uno de los módulos:

2.3.1 Requisitos funcionales del módulo Dashboard

Es el módulo principal que hace de pantalla de inicio una vez el usuario haya iniciado sesión. Mostrará los tipos, ediciones y registros creados por el usuario, y también permitirá editarlos. En caso de ser administrador se mostrarán todos.

- **Listar tipos:** Mostrará de forma paginada todos los tipos creados por el usuario.
- **Listar ediciones:** Mostrará de forma paginada todas las ediciones creadas por el usuario.
- **Listar registros:** Mostrará de forma paginada todos los registros creados por el usuario.
- **Editar tipo:** Un botón que permitirá editar el tipo seleccionado en el listado.
- **Editar edición:** Un botón que permitirá editar la edición seleccionada en el listado.
- **Editar registro:** Un botón que permitirá editar el registro seleccionado en el listado.
- **Nuevo tipo:** Un botón que permitirá abrir el módulo de nuevo tipo.
- **Nueva edición:** Un botón que permitirá abrir el módulo de nueva edición.
- **Nuevo registro:** Un botón que permitirá abrir el módulo de nuevo registro de forma manual.

2.3.2 Requisitos funcionales del módulo Tipos

Este módulo se encargará de la gestión de los tipos. Los tipos son sobre los que cuelgan las ediciones que a la vez serán sobre los que cuelguen todos los datos que sean introducidos al sistema, por lo que su creación ha de ser un paso importante porque determinará un elemento sobre el que otros usuarios podrán trabajar.

- **Listar tipos existentes:** Se mostrará un listado con todos los tipos actuales existentes en el sistema, para poder identificar si se va a crear uno ya existente.
- **Formulario nuevo tipo:** Habrá un formulario en el que se podrán indicar los datos necesarios para crear un nuevo tipo.
- **Crear tipo:** Un botón que en base a los datos rellenados en el formulario creará el nuevo tipo.

2.3.3 Requisitos funcionales del módulo Entidades Participantes

Las entidades participantes deberán ser generadas antes de ser asignadas a alguna edición. De cara a facilitar que sean reconocidos rápidamente al introducir datos se les podrá adjuntar una imagen que será mostrada en otras pantallas. Estarán ligadas a un tipo, y siempre que se genere una nueva edición todas las entidades participantes ligadas a ese tipo serán mostradas para poder ser seleccionadas.

- **Listar entidades participantes:** Al seleccionar un tipo se listarán las entidades participantes existentes para el tipo seleccionado.
- **Asignar imagen:** Se podrán asignar una imagen a las entidades participantes existentes para que actué de icono identificativo de dicha entidad participante.

- **Formulario nueva entidad participante:** El formulario permitirá introducir un nombre a la entidad participante y otros datos como el color y el tipo sobre el que depende.
- **Guardar entidad participante:** Pulsando un botón se guardará la entidad participante tal como haya sido configurada rellenando el formulario.

2.3.4 Requisitos funcionales del módulo ediciones

La pantalla de ediciones permitirá crear nuevas ediciones de los distintos tipos ya existentes en el sistema. Se podrá configurar medidas y entidades participantes de la edición además de otros datos, como el nombre de la edición y rangos numéricos de las medidas.

- **Seleccionar Tipo:** Permite seleccionar el tipo sobre el que se desea crear la nueva edición.
- **Listar Ediciones:** Una vez seleccionado el tipo se mostrará el listado de ediciones actuales presentes en el tipo seleccionado.
- **Insertar Medidas:** Al pulsar insertar medidas se abrirá la opción se insertar medidas manualmente.
- **Importar Medidas:** Al pulsar importar medidas se abrirá la opción se importar medidas desde otra edición.
- **Relacionar Medidas:** Se podrán relacionar medidas para que al introducir una la otra se introduzca automáticamente.
- **Insertar Entidades Participantes:** Permitirá seleccionar qué entidades participantes formarán parte de la edición y a las cuales habrá que asignar valores cuando se guarden registros.
- **Importar Entidades Participantes:** Si se selecciona importar se podrá importar desde otras ediciones la configuración de entidades participantes que formarán parte de la nueva edición que se está creando.
- **Formulario nueva edición:** El formulario contendrá datos básicos de la edición que se está creando.
- **Guardar nueva edición:** Al pulsar guardar se guardará la nueva edición con los datos del formulario, medidas y entidades participantes.

2.3.5 Requisitos funcionales del módulo Usuarios

Esta será la pantalla de admisión de usuarios que se han registrado y de cambio de roles. Sólo será accesible por administradores.

- **Listar usuarios pendientes de admitir:** Mostrará el listado de usuarios que se han registrado.
- **Aceptar usuarios pendientes de admitir:** Se podrá dar acceso a los usuarios que no pueden acceder.
- **Listar usuarios registrados:** Mostrará el listado de usuarios que ya tienen acceso a la aplicación web.
- **Cambiar rol usuarios registrados:** Se podrá cambiar a los usuarios registrados a cualquiera de los roles disponibles.

2.3.6 Requisitos funcionales del módulo Carga Registros Manual

Para añadir datos se deberá poder seleccionar los datos que identifiquen de forma exacta a qué tipo, edición y fuente pertenecen los datos. Una vez seleccionado se mostrarán todos los campos de medidas y entidades participantes de esa edición para ser rellenados. La herramienta podrá guardar opcionalmente una URL o algunas notas para complementar el origen de los datos, pero hay un dato que sí será obligatorio y que funcionará como otro dato identificativo, y se trata de la fuente. Es un dato que permitirá agrupar todos los datos introducidos para finalmente en el visualizador poder ver las estadísticas y fiabilidad de los datos ofrecidos por las distintas fuentes de cara a determinar la que más se ha acercado a los datos finales.

- **Formulario selección edición y fuente:** Seleccionando edición y fuente se cargarán los datos para que puedan ser rellenados. Al seleccionar los datos el sistema identificará qué entidades participantes han de tener datos y qué medidas.
- **Mostrar histórico:** Al seleccionar una edición y fuente en los filtros del formulario se cargarán todos los registros actuales en dicha fuente, para poder identificar si se va a introducir un registro ya existente.
- **Permitir sincronización medidas:** Si algunas medidas están relacionadas se podrá seleccionar que esos valores se autocomplementen automáticamente o no lo hagan. Como ejemplo serían medidas porcentuales de otras medidas numéricas.
- **Guardar registro:** Se podrán guardar todos los registros que se quieran rellenando el formulario de entidades participantes y medidas.
- **Crear fuente:** Independientemente del campo de la URL como de notas, es necesario crear una fuente que será almacenada para poder ser seleccionada por otros usuarios. Se deberá poder hacer desde esta misma pantalla, ya sea con una ventana modal o un formulario.

2.3.7 Requisitos funcionales del módulo Carga Registros desde web

Aunque la funcionalidad principal para introducir datos es manual, opcionalmente se tendrá la posibilidad de usar este módulo que facilitará la inserción de datos facilitando la lectura de páginas web y automatizando buena parte del trabajo.

- **Formulario selección edición y fuente:** Seleccionando edición y fuente identificará qué datos han de permitirse rellenar cuando al indicar una URL se detecten datos. Tras esta selección se mostrarán los elementos para proceder a indicar la URL.
- **Leer estructura tablas web:** Al pulsar el botón se analizará el DOM de la página HTML referenciada en busca de tablas para poder. Una web detectadas se mostrarán los datos y se podrá configurar a qué medida y entidad pertenece cada dato detectado.
- **Ayuda expresión regular:** Se podrá indicar expresiones regulares para facilitar la lectura en caso de que los campos no sean totalmente numéricos.
- **Marcar medidas y entidades participantes:** Se podrá seleccionar entidades participantes y medidas para configurar una lectura rápida de los datos.
- **Retirar filas y columnas:** Se podrán retirar filas y columnas detectadas para descartar datos que no son necesarios.

- **Procesar datos:** Al procesar los datos se validará que todos los datos son correctos, y si lo son se guardará el registro identificado con todos los datos.

- **Crear fuente:** Independientemente del campo de la URL como de notas, es necesario crear una fuente que será almacenada para poder ser seleccionada por otros usuarios. Se deberá poder hacer desde esta misma pantalla, ya sea con una ventana modal o un formulario.

2.3.8 Requisitos funcionales del módulo Resultados

En el módulo de resultados se podrá consultar y modificar los datos de los resultados finales. En el caso de los resultados finales sólo se guardará un dato, pues independientemente de la fuente todos los datos han de referenciar a un único dato real. Este dato será el que las distintas fuentes intentarán predecir o estimar según sus criterios o capacidades para ofrecer sus distintas previsiones.

- **Formulario selección edición:** Se podrá elegir la edición que se desea consultar, añadir o modificar. Una vez seleccionada la edición se cargará el formulario con las distintas fuentes y entidades participantes de la edición seleccionada. Si ya existía se cargará con los datos presentes y si no existía se cargará un formulario con los campos vacíos para ser rellenados.

- **Ver resultados finales:** Si se selecciona una edición que ya incluye datos se mostrarán y podrán ser consultados.

- **Añadir resultados finales:** Si no hay datos se podrán añadir los datos.

- **Modificar resultados finales:** Si hay datos se mostrarán y se permitirá modificarlos.

2.3.9 Requisitos funcionales del módulo Visualizador

Es el módulo en el que todos los datos son procesados para mostrar el objetivo final de la herramienta, y es que el trabajo colaborativo de todos los usuarios se mezcle para dar unos resultados finales.

- **Filtro con todos los desplegables:** Mostrará los distintos desplegables con tipos, ediciones, entidades participantes, medias y registros para poder ir mostrando distintos datos en la pantalla.

- **Comparativa de fuentes de una entidad participante:** Compara una media y una entidad participante a lo largo de todos los registros disponibles en la edición seleccionada.

- **Comparativa de todas las entidades participantes en una misma fuente:** Mostrará un gráfico en la que se vea la progresión de todas las entidades participantes en una cierta fuente.

- **Registro seleccionado:** Mostrará un gráfico con los resultados del registro seleccionado.

- **Resultados finales:** Mostrará un gráfico con los resultados finales de la edición.

- **Fiabilidad fuentes:** Mostrará estadísticas mostrando la fiabilidad de las fuentes de sus estimaciones respecto a los datos finales.

2.3.10 Requisitos funcionales del módulo Validación Datos

Todos los usuarios podrán introducir datos, pero para ser mostrados en listados y gráficas deberán de ser aprobados por administradores, para llevar un control de los datos que consideran válidos así para retirar registros repetidos.

- **Listar registros sin validar:** Se mostrará todo el listado de registros que aún no han sido validados o rechazados.
- **Validar registro:** Al pulsar el botón todos los registros seleccionados se marcarán como aptos.
- **Rechazar registro:** Al pulsar el botón todos los registros seleccionados se marcarán como no aptos.

2.4 Requisitos no funcionales

Son los requisitos que no tienen que ver con las funcionalidad de la herramienta, pero son necesarios para que la experiencia no se vea perjudicada por causas ajenas a la propia herramienta.

- **Accesibilidad instantánea:** Los datos han de estar siempre disponibles y accesibles.
- **Datos accesibles desde cualquier lugar:** El acceso a la herramienta ha de ser posible sin importar desde dónde se está accediendo..
- **Misma experiencia:** El acceso deberá ser idéntica independientemente del sistema operativo que se esté usando, o del navegador web en caso de que fuera necesario.
- **Siempre actualizado:** Los datos que puedan ser consultados por un usuario deberán ser idénticos a los que consulte otro usuario.

2.5 Tipo de aplicación

Llegado a este punto ya tenemos la herramienta estructurada de la forma que sabemos cómo se va a ordenar la información y qué funcionalidades va a tener la herramienta, de cara que se puedan introducir los datos y mostrarse. El problema planteado va a resolverse a través de una herramienta informática colaborativa para añadir datos, mostrarlos y compararlos. Una herramienta así puede plantearse de muchas formas. Esto hace que se puedan presentar distintas alternativas para ejecutarlo, como un software de escritorio o una aplicación web.

Aunque la herramienta pueda ser desarrollada perfectamente tanto como una herramienta instalada localmente o una herramienta web online, se profundizará en cada caso para ver qué alternativa se adecúa más de cara al objetivo final. [2] [3]

2.5.1 Aplicación de escritorio

En una aplicación de escritorio la herramienta se instala localmente en el ordenador de cada usuario, y desde ese software instalado podrá realizar todas las acciones detalladas en los métodos de uso.

Sus principales puntos, positivos y negativos, son los siguientes:

- **Metodología muy sólida y asentada:** Una herramienta de escritorio para realizar el proyecto sería la forma más sencilla de primeras, ya que desde hace décadas nos encontramos lenguajes y librerías que nos proporcionarían muchas posibilidades de cara a plasmar la idea inicial con un resultado muy satisfactorio. Tanto C# [4], con sus clásicos y extendidos `WinForms` [4] o los más recientes WPF [5]; pero también usando java y sus numerosas librerías gráficas.

- **Base de datos:** En este punto tener una base de datos local sería imposible, porque todos los usuarios han de acceder a los mismos datos y unos poder ver los datos introducidos por otros. Es posible tener una base de datos externa y un software local, pero exigiría un constante intercambio de datos, además de poder generar problemas si cada usuario usa distintas versiones del software.

- **Librerías visuales:** Se puede realizar pantallas con todas las funcionalidades posibles, pero suelen ofrecer un resultado bastante austero visualmente salvo que se pague por librerías privativas. Funcionalmente no es un punto importante, salvo por el detalle de que queremos ofrecer datos con resultados visuales en gráficos, por lo que supondría cierta limitación.

- **Actualización:** Una aplicación de escritorio puede controlarse al iniciarse para que se actualice realizando una comprobación a un servidor remoto para comprobar la diferencia de versiones. Esto ya supondrá tener un sistema remoto que gestione la comprobación de la versión y la descarga de instalador con la actualización. Incluso suponiendo esta opción, podría darse el caso de usuarios que nunca cierren la aplicación, lo que supondría distintos usuarios con distintas versiones. Una situación que es necesariamente incompatible con una herramienta como la planteada.

2.5.2 Aplicación web

La otra opción principal es una aplicación web, en la que en vez de instalarse en cada equipo, cada usuario accederá desde una URL y la aplicación web se cargará en el navegador web.

Sus principales puntos, positivos y negativos, son los siguientes:

- **Multiplataforma:** El punto más importante de todos, y es que una aplicación instalada localmente tiene que desarrollarse de forma concreta para cada sistema, como puede ser windows, android, ios, etc. Y salvo tecnologías multiplataforma algo experimentales como MAUI [6], todo desarrollo con tecnologías de escritorio o nativas de dispositivos móviles tienen un desarrollo independiente. En cambio, un desarrollo web sólo requiere un único desarrollo, y luego puede ser accedido con un navegador desde cualquier dispositivo. Esto hace que sea accesible prácticamente por todo el mundo.

- **Librerías visuales:** El desarrollo web tiene disponibles muchísimas librerías para que de forma rápida y sencilla se consigan resultados muy aceptables, mientras que las librerías para desarrollo en escritorio tiene una curva de aprendizaje algo más elevada para no alcanzar el mismo resultado..

- **Siempre actualizado:** Todos los usuarios tendrán siempre la misma y más actualizada versión. Si fuera una aplicación de escritorio cada usuario tendría que actualizar parches para tener al día la aplicación, teniendo que tener controles extra para identificar que se esté usando la versión más reciente. Al ser online los cambios se aplican sobre el servidor y cualquier usuario que acceda ya tendrá visibles dichos cambios.

- **Sólo necesario un navegador:** En caso de que sean necesarios más recursos para realizar distintos cálculos o procedimientos no se verán repercutidos en los requisitos exigidos al usuario para acceder a la aplicación. Seguirá necesitando únicamente un navegador.

- **Interactividad:** .

2.5.3 Conclusiones Escritorio-WEB

Con todos estos puntos llegamos a la conclusión de que aunque ambas metodologías serían válidas para afrontar el problema, la importancia de tener que usar usar una fuente unificada de datos y querer que todos puedan acceder simultáneamente a la misma información hace que la opción de la aplicación web sea la mejor por los siguiente puntos:

- **Velocidad en la comunicación:** No es lo mismo mandar una solicitud con unos pocos datos al servidor y que te devuelva unos resultados finales a enviar datos, te devuelva información, se procese localmente y se vuelva a enviar. Al tener los datos remotamente esto ralentizaría bastante la comunicación y complicaría la consistencia de los datos.
- **Siempre actualizado:** Proporciona la seguridad de que cuando se detecte un fallo y sea corregida el 100 % de los usuarios lo tendrán visible en cuanto cambien de ventana.
- **Multiplataforma:** El tener la seguridad de que haciendo una aplicación web facilitará el acceso a la misma desde cualquier dispositivo, tanto móvil como de sobremesa, es un punto más que suficiente para decantarse por esta opción.

En resumidas cuentas al requerir una fuente única ya se requieren servicios remotos, y para agilizar la comunicación y consistencia de los datos la mejor opción es desarrollar una aplicación web y en la medida de lo posible que sea (*responsive*) [7], para así hacer que tanto en un navegador de ordenador o de teléfono inteligente sea adaptable y proporcione una experiencia satisfactoria.

2.5.4 Tipo de aplicación web: MVC-SPA

Una vez decidido que la mejor opción para desarrollar la herramienta online es realizando una aplicación web, se tiene que elegir cuál es la mejor forma de afrontar el desarrollo de tal aplicación web.

El método más asentado es realizar una aplicación empleando **Modelo-Vista-Controlador** [8], en la que se estructura el proyecto en vistas, con las que interacciona el usuario, que mandará solicitudes al controlador, y que realizará toda la lógica necesaria, y cuando necesite consultar datos de la base de datos lo hará solicitando tal información al modelo. Es un plan de desarrollo perfectamente válido y proporcionaría un resultado perfectamente óptimo y satisfactorio, habiendo miles y miles de proyectos, tanto pequeños como de grandes empresas, que hacen uso del MVC con distintos lenguajes y distintos frameworks sobre los mismos para facilitar el trabajo. Algunas tecnologías que usan MVC son las siguientes:

- **.NET/ASP.NET Core MVC** [9]
- **JAVA/Spring Framework** [10]
- **PHP/Symphony** [11]
- **Ruby/Ruby on Rails** [12]

Pero trabajando con MVC, tanto con frameworks actuales como algo más antiguos, se presentan ciertos problemas que suponen un trabajo extra, tanto en código como en el tiempo necesario para

implementarlo. Y es la parte de la interactividad y la independencia de distintos elementos mostrados en la web. Por ejemplo, cargar un campo sin tener que recargar toda la página. Esa problemática se soluciona empleando AJAX con librerías como JQUERY, pero no deja de ser un añadido, un elemento que potencia el clásico modelo vista controlador (MVC). Muchos frameworks trabajan con layouts para dividir distintos elementos en una vista y poder ordenar mejor esa independencia en el refresco de datos.

Para solucionar toda problemática o desorden en el proyecto y para potenciar esa interactividad existe una metodología de trabajo llamada (*Single Page Application*) o SPA [13], que en vez de trabajar sobre URLs que lleven a distintas funcionalidades, se trabaja siempre bajo la misma pantalla pero dividiendo todo lo que contiene dicha página en mini segmentos que se inyectan sin tener que recargar nada. De esta forma se van añadiendo o quitando componentes tal como sea necesario para la funcionalidad. De esta forma junto con el enlace de datos (*data binding*) tenemos aplicaciones que permite una forma rápida de trabajo y un código mucho más ordenado.

Las dos principales tecnologías que usan SPA son:

- **TypeScript/Angular** [14] Angular emplea TypeScript para la parte cliente, pero la parte servidor puede configurarse con distintas tecnologías.
- **.NET/Blazor** [15] Blazor usa tanto para la parte cliente como servidor el lenguaje C#.

Entre estas dos alternativas la más sencilla de elegir sería Angular, ya que en los últimos años ha gozado de bastante popularidad y eso garantiza que es un movimiento correcto, pero técnicamente las distintas partes del proyecto emplean disientas tecnologías, y de cara al uso de datos supone un trabajo extra, como por ejemplo el modelado de datos, pues una clase tendrá que declararse en el lenguaje usado en la parte front y otra vez para la parte back. En cambio Blazor, que tiene menos popularidad y es bastante más actual, ofrece una visión de trabajo de un único lenguaje para trabajar y luego internamente se encarga de convertir ese código en javascript o HTML. Eso hace que desde el principio, y sin necesidad de grandes conocimientos de desarrollo web ya se pueda empezar a desarrollar. También al elegir Blazor se tiene en mente la apuesta de Microsoft con MAUI y la unificación de todos los desarrollos bajo un mismo lenguaje y metodología y fácil adaptación a cada entorno. Entonces la elección más acertada sería Blazor, ya que la curva de aprendizaje es extremadamente más corta y menos exigente al sólo ser necesario un lenguaje con décadas de experiencia y con actualizaciones continuas.

2.6 Interfaz

La interacción del usuario con la herramienta, siendo cada caso de uso un componente de Blazor

2.6.1 Pantalla Sin Login

La pantalla inicial de la figura 2.2 sin iniciar sesión deberá mostrar información, pero ser imposible que el usuario añada datos.

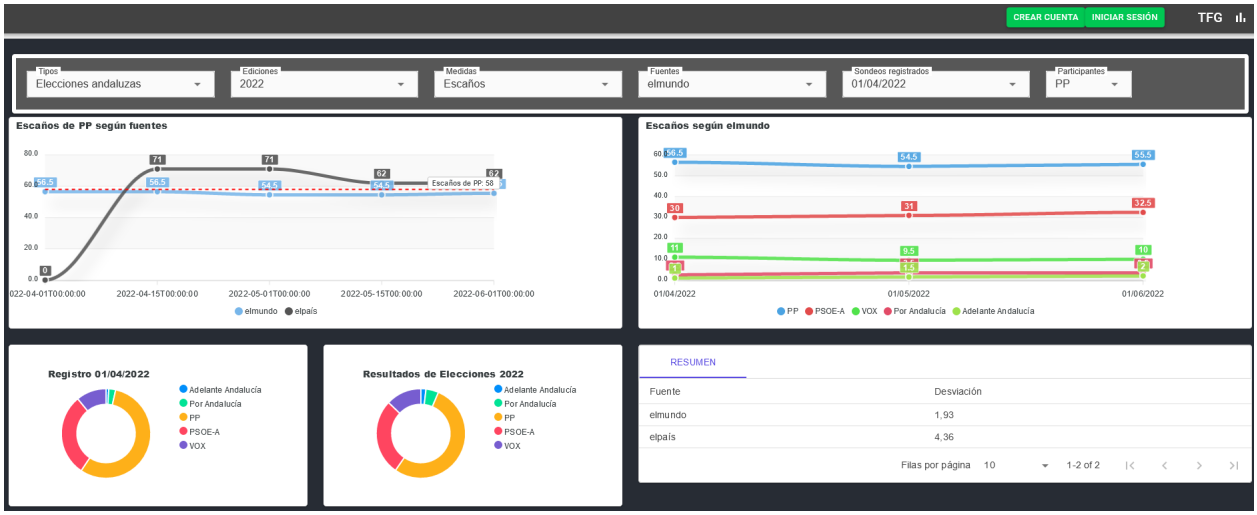

Figura 2.2: Idea de pantalla inicial sin sesión

2.6.2 Pantalla Mi Menú

Una vez iniciada la sesión, la pantalla de la figura 2.3 que se cargará es un dashboard con información que ha añadido el usuario conectado.

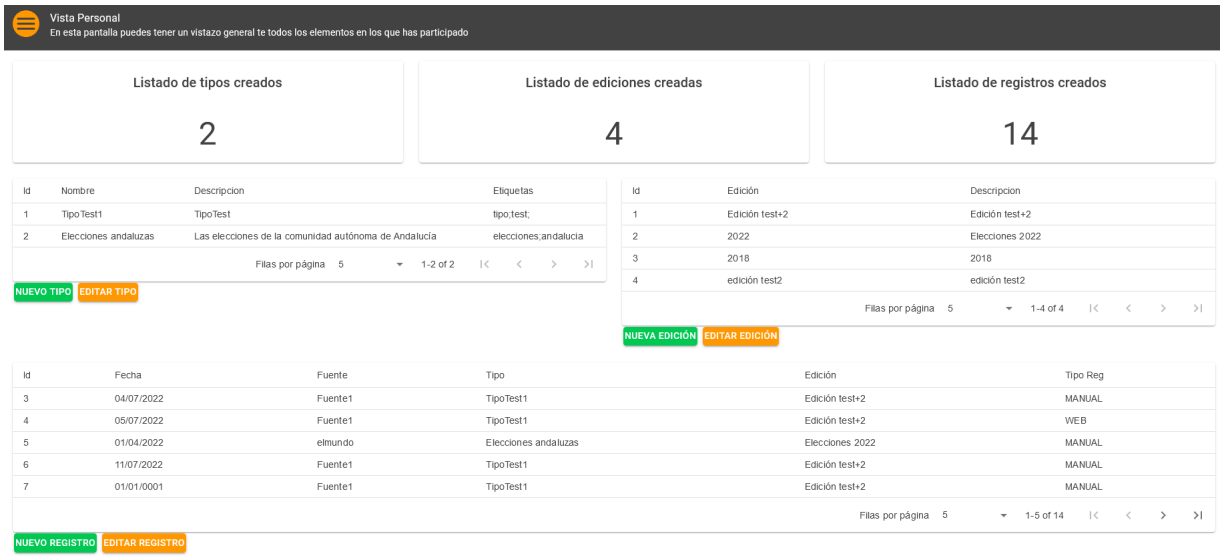

Figura 2.3: Idea de pantalla para gestionar los elementos creados por el usuario

2.6.3 Pantalla Tipos

La pantalla de la figura 2.4 (tipos) permitirá crear tipos.

Figura 2.4: Idea de pantalla para gestionar tipos

2.6.4 Pantalla Entidades Participantes

La pantalla de la figura 2.5 permitirá crear entidades participantes y ser asignadas a un tipo concreto.

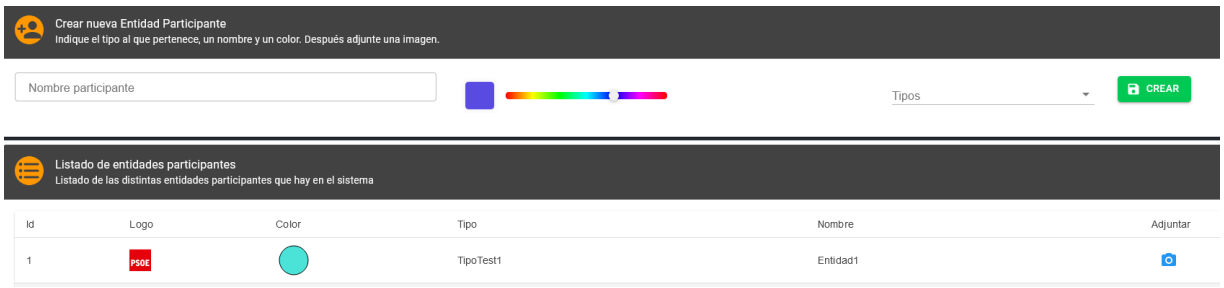

Figura 2.5: Idea de pantalla para gestionar entidades participantes

2.6.5 Pantalla Ediciones

La pantalla de la figura 2.6 permitirá crear ediciones asignadas a un tipo concreto, indicando entidades participantes y distintas medidas dentro de la edición.

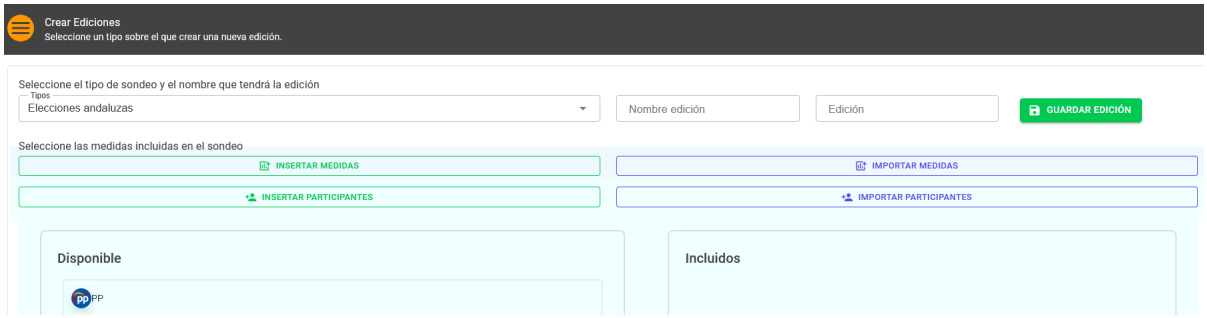

Figura 2.6: Idea de la pantalla para gestionar las ediciones

2.6.6 Pantalla Resultados

La pantalla de la figura 2.7 permitirá asignar resultados finales a las medidas de las ediciones.

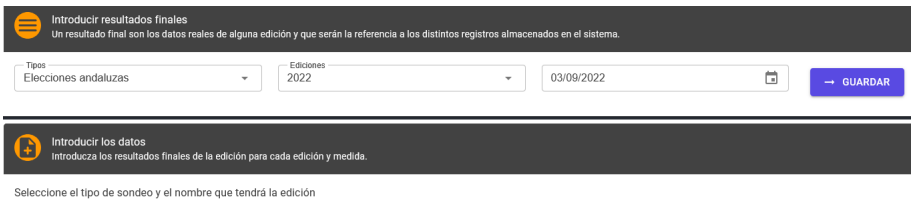

Figura 2.7: Idea de la pantalla para gestionar los resultados de ediciones

2.6.7 Pantalla Validación

La pantalla de la figura 2.8 permitirá a los administradores validar qué registros son aceptados e introducidos.

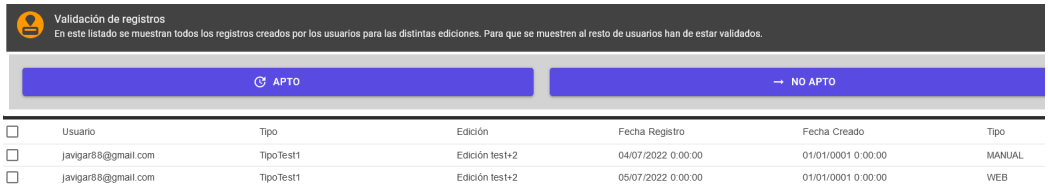

Figura 2.8: Idea de la pantalla para validación de datos

2.6.8 Pantalla Registros Web

La página permitirá introducir una dirección web (figura 2.9) y en base al tipo y edición seleccionada se cargarán los datos capturados en la web. Opcionalmente se aportarán opciones para limpiar los datos y evitar que el usuario tenga que editar muchos datos manualmente.

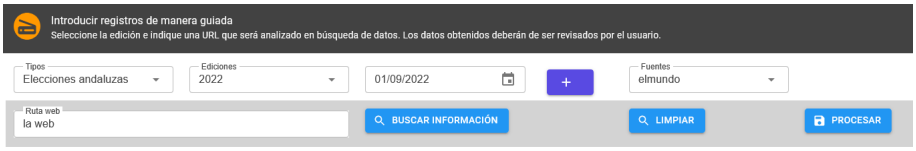

Figura 2.9: Idea de la pantalla para introducir información de fuentes de forma más automática

2.6.9 Pantalla Registros Manual

Habrá una forma principal para introducir sondeos de distintas fuentes y luego complementos a la misma que ayudarán a introducir datos de una forma más automática. La forma manual de la figura 2.10 permitirá seleccionar sobre qué edición vamos a introducir datos y una vez seleccionado se nos pedirá que indiquemos la fuente de procedencia de la información y que introduzcamos cada valor de las medidas esperadas. No será obligatorio introducir todos los datos si la fuente origen no los proporciona, pero esos datos al generarse el registro se guardarán con valores a 0. Eso es así porque aunque la fuente no proporcione información, los resultados finales sí tendrán esa información y es necesario que existan en cada registro almacenado para comparar con otras fuentes y así mismo para poder ver la evolución a lo largo de distintos sondeos.

Figura 2.10: Idea de la pantalla para introducir información de fuentes

2.6.10 Pantalla Visualizador

La idea principal con la pantalla de la figura 2.11 es que en base a los datos que se vayan seleccionando en el formulario superior se vayan mostrando los datos que sean posible en base a los datos seleccionados, y finalmente cuando se hay realizado la combinación completa se muestren todos los gráficos y resultados calculados mostrando comparativa de las fuentes

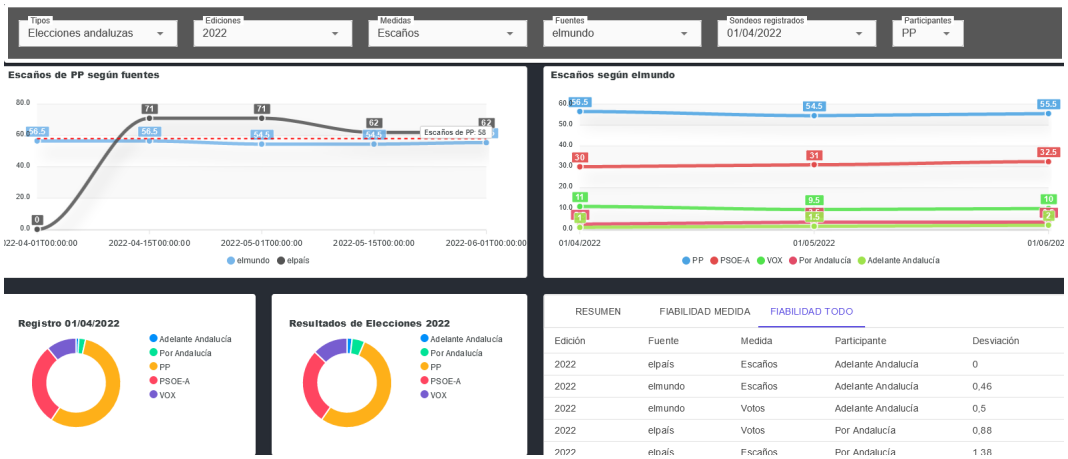

Figura 2.11: Idea de la pantalla para visualizar información

2.7 Conclusiones

En el presente capítulo se ha analizado el problema identificando cómo se va a estructurar los datos, qué datos van a ser necesarios y cómo se van a modificar o consultar en la herramienta por parte de los usuarios. Con todas esas interacciones ya definidas se ha mostrado una pequeña idea visual de cómo serán las pantallas principales y finalmente con todos estos datos ya decididos se ha concluido que una aplicación web SPA usando Blazor es la mejor forma de implementarlo.

Con estos datos ya se puede diseñar el modelo de base de datos y diseñar la descomposición del proyecto en componentes, así como la relación de datos con el proyecto.

Capítulo 3

Diseño del Sistema

La imagen tiende hacia lo infinito y conduce hacia lo absoluto[1].

Andrei Tarkovski

3.1 Introducción

Tras haber analizado el problema de los objetivos, tenemos ahora mismo una estructura base de cómo se va a organizar la información, cómo va a interactuar con el usuario y qué tecnologías son las más apropiadas para coger todo lo planteado y comenzar a diseñar la aplicación web que resolverá el problema.

Con la estructura base de datos y los casos requisitos funcionales ya definidos ya podemos diseñar modelado completo de datos y detectar al completo los datos necesarios, con lo que definiremos la organización física que tendrán los datos en el sistema gestor de base de datos. También teniendo los requisitos funcionales y la división en módulos/pantallas/grupo de funcionalidades se añadirán el diseño y estructuración de la aplicación en componentes

3.2 Roles

La herramienta ofrecerá personalización total de los datos. Eso significa que no sólo se podrán añadir datos, sino también indicar sobre qué se van a añadir esos datos y que características tiene cada uno de esos datos que ha de introducirse así el como se relaciona con otros datos. Es por esto que es necesario jerarquizar las distintas opciones de la aplicación en roles, y de esta forma cada usuario registrado tendrá que pertenecer a un único rol. De esa forma al iniciar sesión la herramienta podrá mostrar u ocultar información, pantallas, botones, etc.

A continuación (tabla 3.1) se detallan los distintos roles y sus características

3.2.1 Rol Anónimo

No es estrictamente un rol existente, pero considerará para mostrar ciertos datos que pueden ser accesibles sin necesidad de registrarse en la herramienta e iniciar sesión.

[1] Andrei Tarkovski, Esculpir en el tiempo, Rialp, 2020.

Rol	Componente	Detalle
Anónimo	Página inicial	El usuario anónimo no es un rol como tal. Son los usuarios que no tienen aún acceso a la herramienta.
Administrador	Validación, Usuarios y todos los del rol Creador de tipos	El rol tiene los niveles del rol creador de tipos y además puede ver todos los elementos creados por otros usuarios.
Creador de tipos	Tipos, Entidades y todos los del rol Creador de ediciones	El rol creador de tipos puede crear tipos y hacer lo mismo que los roles creados de ediciones y registros.
Creador de ediciones	Ediciones, Resultados, Medidas y todos los del rol Creador de registros	El rol creador de ediciones puede crear ediciones y hacer todo lo que un creador de registros.
Creador de registros	Manual, Guiada, Visualizador	Este rol es el básico de un usuario con acceso. Puede añadir registros a las distintas ediciones existentes.

Tabla 3.1: Resumen de componentes por rol

Un usuario anónimo podrá ver la pantalla inicial en el que podrá ver los gráficos es una versión simplificada para poder hacerse una idea de todo lo es accesible creándose una cuenta.

3.2.2 Rol Administrador

El rol administrador cumple una función administradora, por lo que podrá entrar al componente de usuarios, para aceptar sus registros o modificar sus rangos; el componente de validaciones, desde el que se validará qué registros introducidos son válidos para que salgan en los gráficos y sus datos aporten a las estadísticas de fiabilidad de las fuentes.

3.2.3 Rol Creador de tipos

El rol creador de tipos podrá crear tipos de sondeos y crear entidades asignadas a esos tipos, para que puedan ser seleccionados al crear ediciones. En el dashboard podrá ver un resumen de los tipos creados.

El creador de tipos tiene a la vez los permisos del rol creador de ediciones.

3.2.4 Rol Creador de ediciones

El rol creador de ediciones podrá crear ediciones y también medidas. En el dashboard podrá ver un resumen de sus ediciones creadas.

El creador de ediciones tiene a la vez los permisos del rol creador de registros.

3.2.5 Rol Creador de registros

El rol creador de registros tendrá la capacidad de ver los gráficos con datos validados e introducir todos los registros que desee. Todos los registros introducidos por usuarios con este rol deberán ser validados

desde la pantalla de validación, a la que sólo tienen acceso los administradores. En el dashboard podrá ver un resumen de los registros creados.

3.3 Elección del ORM y Code-First con Entity Framework Core y SQL Server

En el caso del diseño y modelado de la base de datos todo se realizará de golpe sin necesidad de dividirlo en varias capas de trabajo. Esto se debe a que se empleará Code-First [16] para el diseño y un ORM [17] para la comunicación entre la aplicación web y la base de datos. El sistema gestor de base de datos elegido será SQL Server [18] por ser de la misma familia (Microsoft) que el resto de tecnologías empleadas y por ser un sistema gestor conocido.
El uso de Code-First y un ORM hace que la elección del sistema gestor de base de datos sea completamente indiferente, pues con configurar correctamente unos pequeños parámetros en el ORM se podrá trabajar perfectamente con otro sistema gestor de base de datos y que eso no suponga un gran impacto en la aplicación web.
Puesto que se va a usar Blazor, el ORM que mejor se adaptar es `Entity Framework Core` [19] [20], pensado para proyectos bajo .NET y SQL Server, pero no es la única razón para emplearse, y es que Blazor se aprovecha de las ventajas que ofrece `Entity Framework Core` con `SignalR` [21] y su sistema de migraciones [22].

La elección de usar un ORM para la comunicación con los datos, Code-First para el modelado está relacionado con el sistema de migraciones y trabajar con objetos sin realizar consultas SQL. Code-First hace que cada entidad de la base de datos sea una clase en C#, y esas clases se configuran para estar relacionadas unas con otras. De esta forma no es necesario emplear ningún software de diseño y que posteriormente genere las consultas SQL para la generación de las tablas y sus relaciones. En el proyecto se usará Toad Data Modeler [23] en paralelo para tener una representación más visual de cara a su explicación, pero no se usará para generar SQL, ya que todo se realizará con clases C# y se explicará en su sección correspondiente.

Con estos puntos con crear una clase con el nombre de la tabla y los campos identificando las distintas columnas ya sería tan sencillo como indicar pequeñas instrucciones para crear y modificar tablas, así como llamar a sencillos métodos del ORM para consultar o modificar datos en la base de datos.

3.4 Diseño y Modelado base de datos

Como se ha explicado, el modelo de base de datos se realizará pensando en la utilización de un ORM para que el intercambio de datos entre software y base de datos se haga sin ejecutar consultas directamente, por eso se hará directamente con código, aunque también se diseñará un gráfico entidad-relación para ilustrar la finalidad de las clases/entidades que será el resultado físico final cuando se realice la migración y que ayudará a tener una idea global de cómo se relacionan y almacenan los datos.

La figura 3.1 muestra el modelo de datos a título ilustrativo, para dar una idea global de su tamaño y complejidad. En cada sección de este apartado se detallará una parte del gráfico completo y su equivalente en código, con un nivel de ampliación adecuado. Estas clases en C# al generarse la migración crearán las tablas y relaciones en la base de datos, y una vez configurado el contexto, permitirán emplearse para consultar y modificar datos.

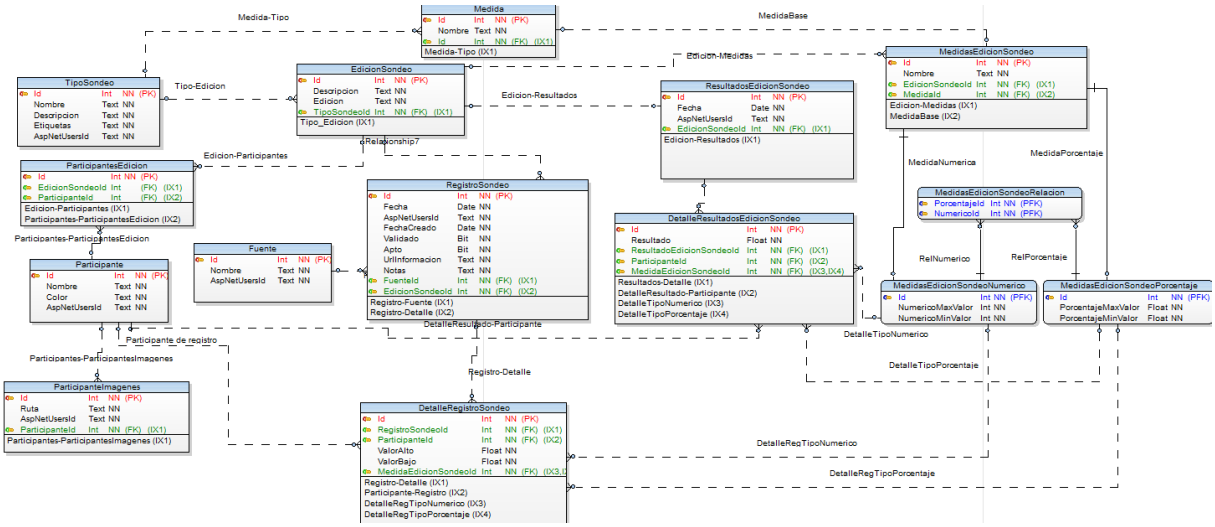

Figura 3.1: Imagen principal del diseño de bases de datos

3.4.1 Diseño Usuarios y roles

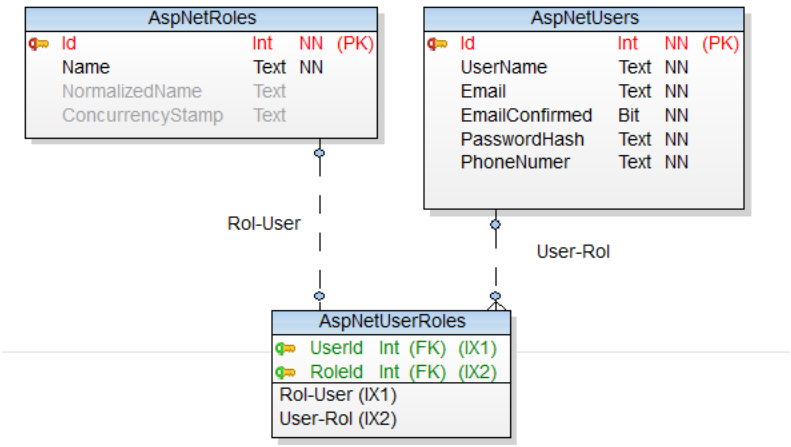

Figura 3.2: Detalla diseño roles y usuarios

En el caso de los usuarios y roles (figura 3.2), además de todas las tablas necesarias para la gestión de las sesiones, se empleará el sistema que proporciona .NET y que se selecciona con un simple check al generar el proyecto. También es posible introducirse posteriormente además de configurarse y alterarse tal como sea necesario. Para esta herramienta lo que proporciona es más que suficiente, por lo que no es necesario modificar nada respecto al diseño que presenta. También son tablas que no son necesarias declarar y que al realizar la migración se generarán en el sistema gestor de base de datos. Al ser tablas ligeramente independientes no se han introducido en el gráfico general de Entidades-Relaciones, aunque sí se usan campos como el ID de los usuarios para identificar qué usuarios han generado ciertos elementos.

El gráfico que acompaña la actual sección representa las tablas resultantes que se darán cuando se realice la migración, en la que una tabla almacenará los usuarios registrados desde la pantalla de registro, otra los roles, que se configurarán manualmente antes del despliegue, y finalmente otra tabla, que relaciona roles con usuarios. Con estas tres tablas y el cómo se relacionan se controlará qué módulos, elementos interaccionables y/o visibles y demás datos puedan ser accedidos o no por unos usuarios u otros.

3.4.2 Diseño Tipos y Ediciones

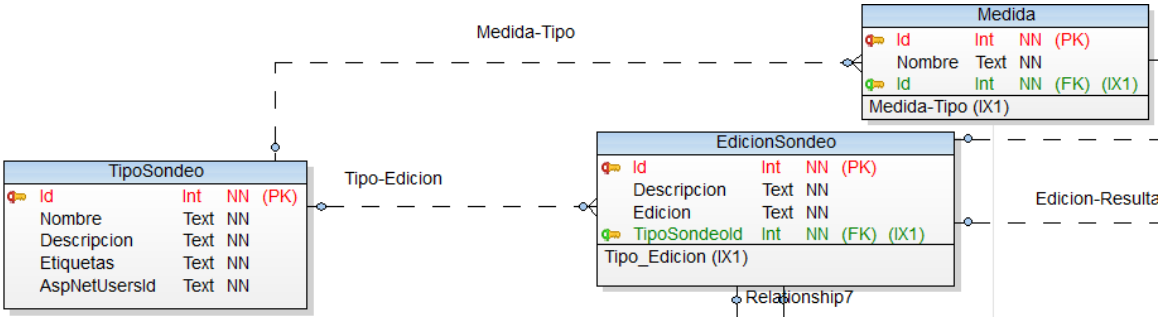

Figura 3.3: Detalle diseño tipos y edición

```
public class TipoSondeo
{
    [Key]
    public int Id { get; set; }
    public string Nombre { get; set; }
    public string Descripcion { get; set; }
    public string Etiquetas { get; set; }
    public string AspNetUsersId { get; set; }

    public List<Participante> Participantes { get; set; }
    public List<Medida> Medidas { get; set; }
    public List<EdicionSondeo> EdicionSondeo { get; set; }
}
```

```
public class EdicionSondeo
{
    [Key]
    public int Id { get; set; }
    public string Descripcion { get; set; }
    public string Edicion { get; set; }
    public string AspNetUsersId { get; set; }

    [ForeignKey(''TipoSondeo'')]
    public int TipoSondeoId { get; set; }
    public TipoSondeo TipoSondeo { get; set; }

    public List<ParticipanteEdicion> ParticipanteEdicion { get; set; }
    public List<RegistroSondeo> RegistroSondeo { get; set; }
    public List<MedidasEdicionSondeo> MedidasEdicionSondeo { get; set; }
    public ResultadosEdicionSondeo ResultadosEdicionSondeo { get; set; }
}
```

3.4.3 Diseño Entidades Participantes y Fuentes

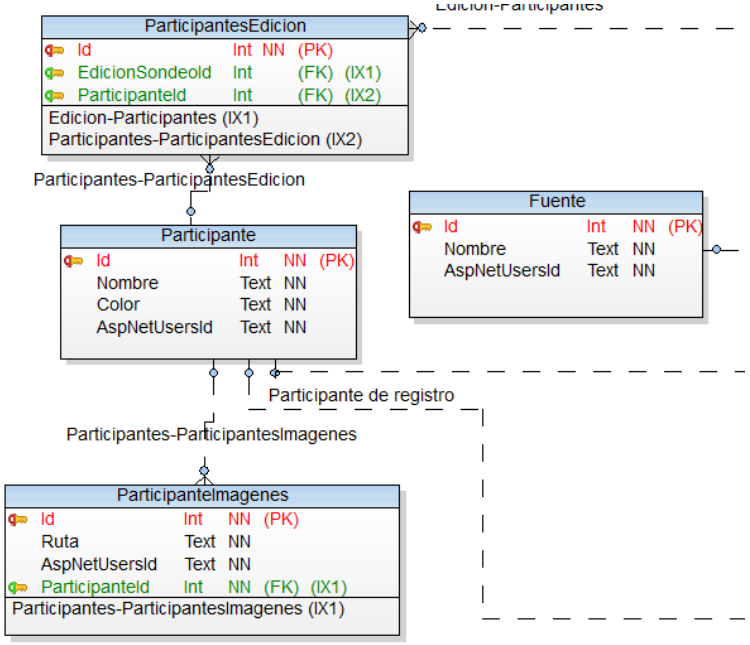

Figura 3.4: Detalle diseño entidades participantes y fuentes

La entidad `Participante` (figura 3.4) hace de nodo principal. Tiene que estar asociado a un tipo 3.3, para que cuando se vayan a crear ediciones puedan seleccionarse para esa edición concreta. Se guardará también un código rgb referenciando un color para que sea usado en las distintas gráfica.

```
1   public class Participante
2   {
3       [Key]
4       public int Id { get; set; }
5       public string Nombre { get; set; }
6       public string Color { get; set; }
7       public string AspNetUsersId { get; set; }
8
9       [ForeignKey(''TipoSondeo'')]
10      public int TipoSondeoId { get; set; }
11      public TipoSondeo TipoSondeo { get; set; }
12
13      public List<ParticipanteEdicion> ParticipanteEdicion { get; set; }
14      public List<DetalleRegistroSondeo> DetalleRegistroSondeo { get; set; }
15      public List<ParticipanteImagenes> ParticipanteImagenes { get; set; }
16      public List<DetalleResultadosEdicionSondeo> DetalleResultadosEdicionSondeo { get; set; }
17  }
```

La entidad `ParticipanteEdicion` se encarga de relacionar a una entidad participante con una edición concreta. Se asociarán a la edición al momento de crearse dicha edición y las claves foráneas son la edición en la que está introducido y la entidad participante a la que hace referencia.

```
public class ParticipanteEdicion
{
    [Key]
    public int Id { get; set; }

    [ForeignKey(''EdicionSondeo'')]
    public int EdicionSondeoId { get; set; }
    public EdicionSondeo EdicionSondeo { get; set; }

    [ForeignKey(''Participante'')]
    public int ParticipanteId { get; set; }
    public Participante Participante { get; set; }
}
```

Las entidades participantes pueden tener imágenes asociadas. Esa imagen será la que se muestre en distintas partes de la herramienta web para mostrar información de las entidades participantes. La clave foránea será la entidad `Participante` la que que hace referencia la imagen.

```
public class ParticipanteImagenes
{
    [Key]
    public int Id { get; set; }
    public string Ruta { get; set; }
    public string AspNetUsersId { get; set; }

    [ForeignKey(''Participante'')]
    public int ParticipanteId { get; set; }
}
```

Los registros de las distintas ediciones tendrán que estar asociados a alguna fuente, ya que en base a los datos de los distintos registros se evaluará la fiabilidad y/o certeza de cada fuente de cara a los distintos ediciones o previsiones en seguimiento.

La fuente estará ligada únicamente a un registro.

```
public class Fuente
{
    [Key]
    public int Id { get; set; }
    public string Nombre { get; set; }
    public string AspNetUsersId { get; set; }

    public List<RegistroSondeo> RegistroSondeo { get; set; }
}
```

3.4.4 Diseño Registros de Ediciones

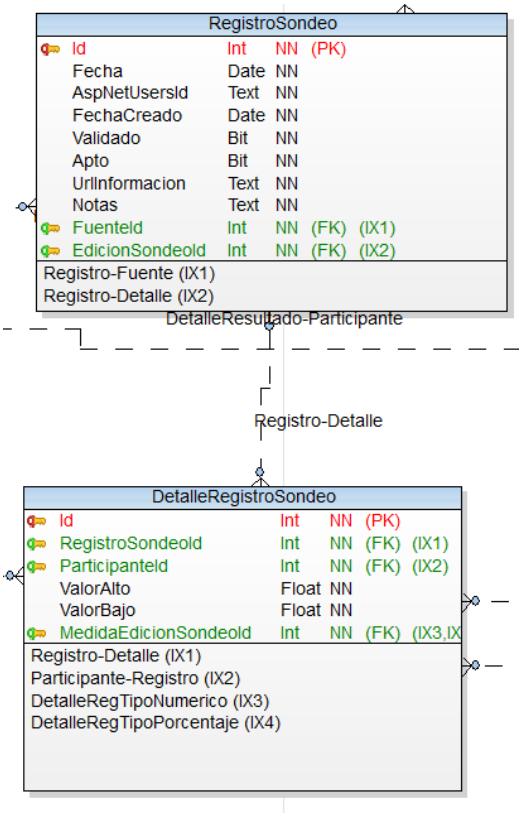

Figura 3.5: Diseño gestión de registros

Los registros (figura 3.5) almacenan la información de los distintos sondeos, encuestas y estimaciones. Asociando estos datos a ediciones, fuentes, entidades participantes y medidas es posible realizar un seguimiento completo de las distintas fuentes y cruzarlos entro otras fuentes así como los los resultados finales que se han dado finalmente.

La entidad `RegistroSondeo` almacenará la cabecera de cada registro. Cada registro podrá ser validado para que sea visible y tenido en cuenta. Las claves foráneas son `Edicion` con el identificador `EdicionSondeoId`, con la edición a la que pertenece el registro y `Fuente` con el identificador `FuenteId`, con la fuente desde la que se han tomado los datos.

```
public class RegistroSondeo
{
    [Key]
    public int Id { get; set; }
    public DateTime Fecha { get; set; }
    public string AspNetUsersId { get; set; }
    public DateTime FechaCreado { get; set; }
    public bool Validado { get; set; }
    public bool Apto { get; set; }
    public string UrlInformacion { get; set; }
    public string Notas { get; set; }

    [ForeignKey(''EdicionSondeo'')]
    public int EdicionSondeoId { get; set; }
    public EdicionSondeo EdicionSondeo { get; set; }

    [ForeignKey(''Fuente'')]
    public int FuenteId { get; set; }
    public Fuente Fuente { get; set; }

    public List<DetalleRegistroSondeo> DetalleRegistroSondeo { get; set; }
    public List<RegistroSondeoModificado> RegistroSondeoModificado { get; set; }
}
```

La entidad `DetalleRegistroSondeo` está asociada a la cabecera por la clave foránea, y guarda el detalle exacto de los datos de cada entidad participante y de cada medida. Además almacenará un valor alto y otro bajo, ya que es muy frecuente que las fuentes no ofrezcan datos exactos, sino rangos de valores en sus estimaciones.

```
public class DetalleRegistroSondeo
{
    [Key]
    public int Id { get; set; }

    public float ValorBajo { get; set; }
    public float ValorAlto { get; set; }

    [ForeignKey(''RegistroSondeo'')]
    public int RegistroSondeoId { get; set; }
    public RegistroSondeo RegistroSondeo { get; set; }

    [ForeignKey(''Participante'')]
    public int ParticipanteId { get; set; }
    public Participante Participante { get; set; }

    [ForeignKey(''MedidasEdicionSondeo'')]
    public int MedidasEdicionSondeoId { get; set; }
    public MedidasEdicionSondeo MedidasEdicionSondeo { get; set; }
}
```

3.4.5 Diseño CodeFirst Resultados

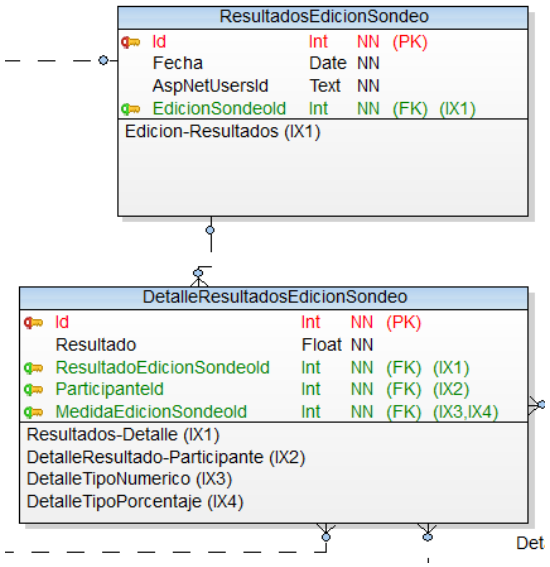

Figura 3.6: Diseño gestión de resultados

Las entidades sobre los resultados (figura 3.6) almacenarán la información referente a resultados finales o datos definitivos sobre las distintas ediciones, y se usarán como referente de cara a calcular lo acertados o desacertados que han estado las distintas fuentes.

La entidad de cabecera almacenará los datos como a qué edición pertenecen los resultados, por lo que la clave foránea será esa edición, y también datos como la fecha, usuario que lo ha creado, etc.

```
1   public class ResultadosEdicionSondeo
2   {
3       [Key]
4       public int Id { get; set; }
5       public DateTime Fecha { get; set; }
6       public string AspNetUsersId { get; set; }
7
8       [ForeignKey(''EdicionSondeo'')]
9       public int EdicionSondeoId { get; set; }
10      public EdicionSondeo EdicionSondeo { get; set; }
11
12      public List<DetalleResultadosEdicionSondeo> DetalleResultadosEdicionSondeo { get; set; }
13  }
```

La entidad de detalle, que tiene como clave foránea la cabecera, guardará todos los datos referentes a esos resultados, por lo que tendrá que haber un registro por cada medida/entidad participante, de ahí que también sean clave foráneas.

```
1     public class DetalleResultadosEdicionSondeo
2     {
3         [Key]
4         public int Id { get; set; }
5         public double Resultado { get; set; }
6
7         [ForeignKey(''ResultadosEdicionSondeo'')]
8         public int ResultadosEdicionSondeoId { get; set; }
9         public ResultadosEdicionSondeo ResultadosEdicionSondeo { get; set; }
10
11        [ForeignKey(''Participante'')]
12        public int ParticipanteId { get; set; }
13        public Participante Participante { get; set; }
14
15        [ForeignKey(''MedidasEdicionSondeo'')]
16        public int MedidasEdicionSondeoId { get; set; }
17        public MedidasEdicionSondeo MedidasEdicionSondeo { get; set; }
18    }
```

3.4.6 Diseño CodeFirst Medidas

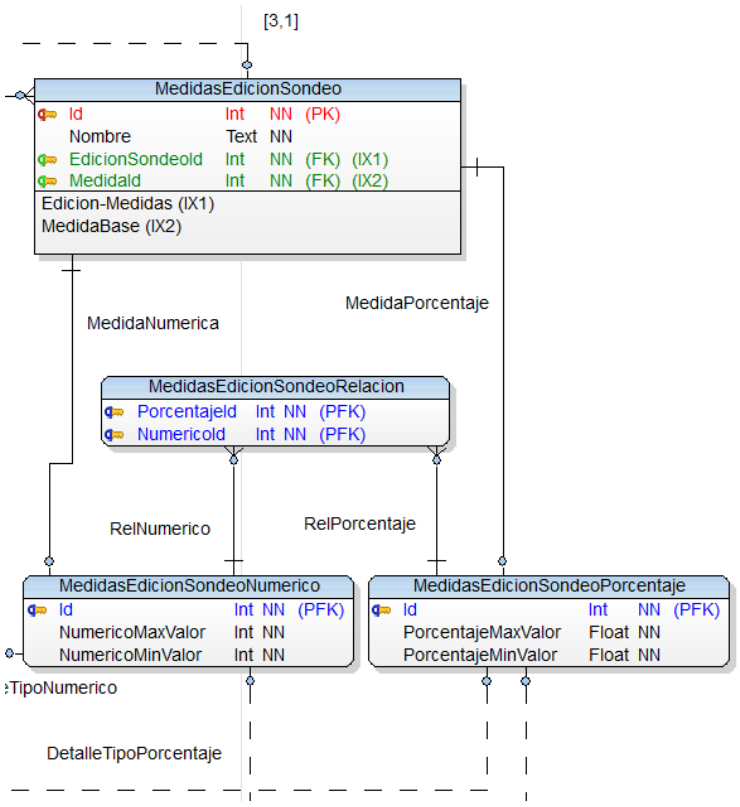

Figura 3.7: Diseño gestión medidas

La medida (figura 3.7) almacenará la relación de distintas medidas para poder relacionarse en algunos gráficos.

```
1   public class Medida
2   {
3       [Key]
4       public int Id { get; set; }
5       public string Nombre { get; set; }
6
7       [ForeignKey(''TipoSondeo'')]
8       public int TipoSondeoId { get; set; }
9       public TipoSondeo TipoSondeo { get; set; }
10
11      public List<MedidasEdicionSondeo> MedidasEdicionSondeo { get; set; }
12  }
```

La `MedidaEdicionSondeo` es la entidad que definirá las medidas de cada edición. Al crearse se podrá asociar a una entidad `Medida`. Si no se indica ninguna se generará una nueva entidad `Medida`. Es una clase abstracta porque como tal esta entidad no será la que se use en el código, ya que actúa como base de las entidades `Numérico` y `Porcentaje`. Las claves foráneas son la entidad `Medida`, tanto la que se relaciones como la que vaya a crearse junto con esta medida; y la entidad `EdicionSondeo`, indicando la relación a la que pertenece.

```
public abstract class MedidasEdicionSondeo
{
    [Key]
    public int Id { get; set; }
    public string Nombre { get; set; }

    [ForeignKey(''Medida'')]
    public int MedidaId { get; set; }
    public Medida Medida { get; set; }

    [ForeignKey(''EdicionSondeo'')]
    public int EdicionSondeoId { get; set; }
}
```

Es la herencia de `MedidasEdicionSondeo` instanciable que gestiona las medidas con valores numéricos.

```
public class MedidasEdicionSondeoNumerico : MedidasEdicionSondeo
{
    public int NumericoMaxValor { get; set; }
    public int NumericoMinValor { get; set; }

    public List<MedidasEdicionSondeoRelacion> MedidasEdicionSondeoRelacion { get; set; }
}
```

Es la herencia de `MedidasEdicionSondeo` instanciable que gestiona las medidas con porcentajes.

```
public class MedidasEdicionSondeoPorcentaje : MedidasEdicionSondeo
{
    public double PorcentajeMaxValor { get; set; }
    public double PorcentajeMinValor { get; set; }

    public List<MedidasEdicionSondeoRelacion> MedidasEdicionSondeoRelacion { get; set; }
}
```

Esta entidad relacionará una entidad `MedidasEdicionSondeoNumerico` con `MedidasEdicionSondeoPorcentaje`. Se utilizará cuando sea necesario actualizar valores de forma automática en base a esta relación. Ambos campos son dos claves foráneas que juntas componen una clave compuesta.

```
public class MedidasEdicionSondeoRelacion
{
    [ForeignKey(''MedidasEdicionSondeoNumerico'')]
    public int MedidasEdicionSondeoNumericoId { get; set; }
    public MedidasEdicionSondeoNumerico MedidasEdicionSondeoNumerico { get; set; }

    [ForeignKey(''MedidasEdicionSondeoPorcentaje'')]
    public int MedidasEdicionSondeoPorcentajeId { get; set; }
    public MedidasEdicionSondeoPorcentaje MedidasEdicionSondeoPorcentaje { get; set; }
}
```

3.5 Estructura del proyecto Blazor

El proyecto está diseñado como una combinación de componentes bajo una misma página, (*Single Page Application*) (SPA) [13], por lo que todo se ejecutará siempre desde la misma página sin recargarse. Sólo las pantallas se login y registro estarán fuera de esa única página al haber usado el sistema de usuarios que proporciona .Net [24] que es muy completo, pero el resto de aplicación web será una única página que irá mostrando componentes sin necesidad de enrutamiento [25] ni peticiones HTTP [26] debido a la bidireccionalidad de SignalR [21].

La idea, que es por lo que se seleccionó una tecnología web bajo SPA es que los componentes se vayan cargando o liberando tal como sean requeridos, sin que nada en la pantalla se recargue. Interamente Blazor alterará el DOM [27] de la página para ir añadiendo tanto el código del componente o la recarga de los datos correspondientes.

De cara a una reutilización del código y posibles modificaciones futuras se intentará que cada pantalla de la aplicación web sea un componente, y a la vez ese componente esté compuesto de otros más pequeños encapsulando micro funcionalidades en componentes y que varios componentes formen una pantalla completa y que los datos entre esos componentes puedan relacionarse gracias a otro componentes bajo el que son llamados.

3.5.1 Organización de componentes [1]

A continuación se mostrará una aproximación del orden de los componentes principales en los que se dividirá deseablemente la aplicación web. Luego dependiendo las necesidades cada uno de los componentes puede reevaluarse en la fase se implementación dependiendo la complejidad y si esa complejidad y tiempo es necesaria o las funciones delegadas a cierto componente pueden incluirse en otros. También hay que tener en cuenta que cada elemento de (*mudblazor*) o la librería de gráficas son un componente en sí, como pueden ser los botones o las tablas. Toman elementos básicos y dentro de un componente los complementan con muchas más funcionalidades. En el casi de los siguientes esquemas se ha optado por sólo mostrar los componentes creados exclusivamente en el proyecto y los que tengan que ver con gráficos.

3.5.2 Componente inicial

Es el componente desde el cual dependen todos los demás componentes (figura 3.8), que actuarán como módulos de funcionalidades. Cada módulo de funcionalidad funcionará de forma totalmente independiente.

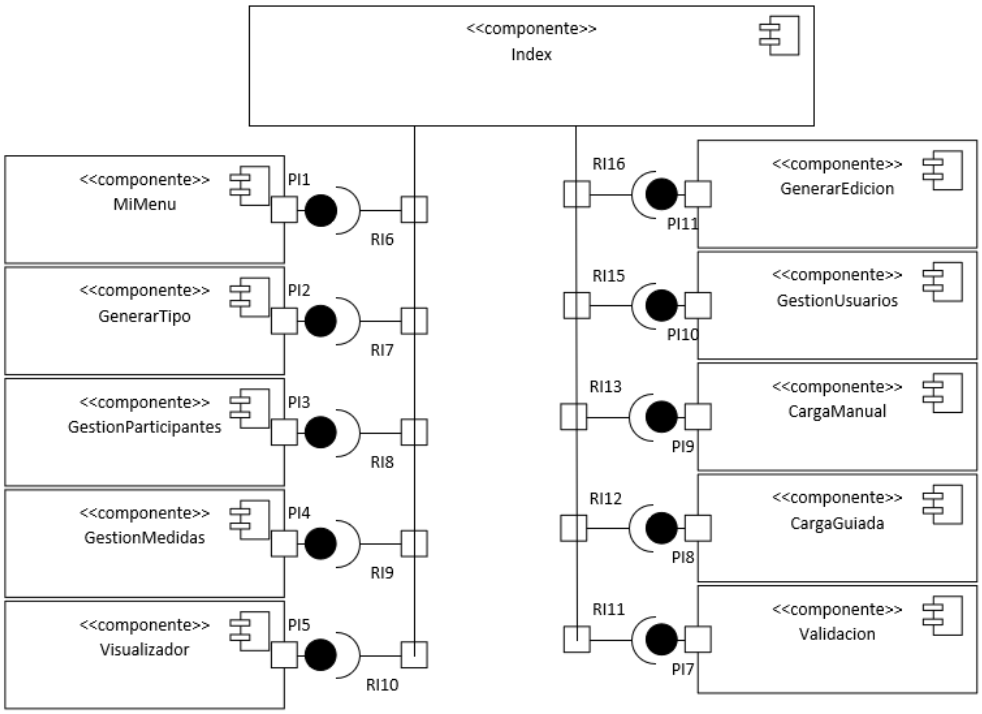

Figura 3.8: Estructura de componentes cargados desde el componente índice

3.5.3 Componente Tipos

Es el componente para gestionar los tipos (figura 3.9). Del componente principal dependen uno para mostrar el formulario de creación y otro para mostrar el listado de tipos.

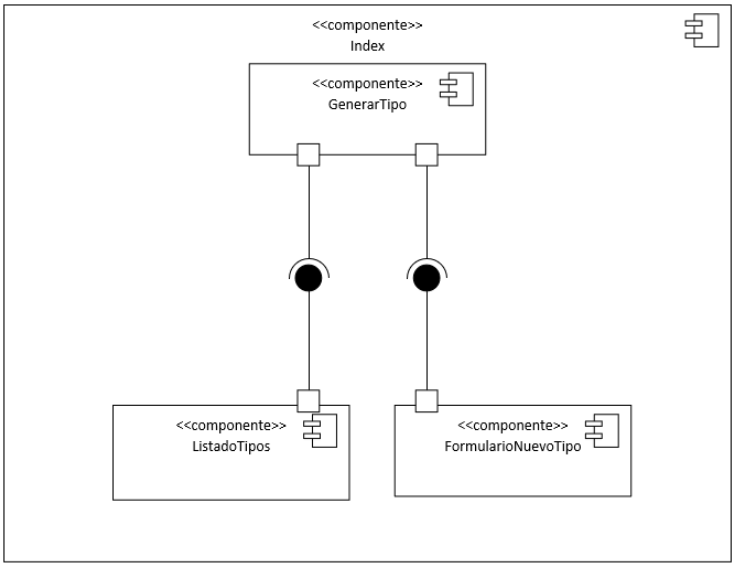

Figura 3.9: Componentes dentro del componente tipos

3.5.4 Componente Ediciones

Es el componente para gestionar las ediciones (figura 3.10). Del principal dependen un componente que gestiona las medidas que tendrá asociada la edición y lo mismo otro componente para las entidades participantes. A la vez cada uno de estos dos componentes tendrán uno para gestionar la importación de esos datos de otras ediciones del mismo tipo.

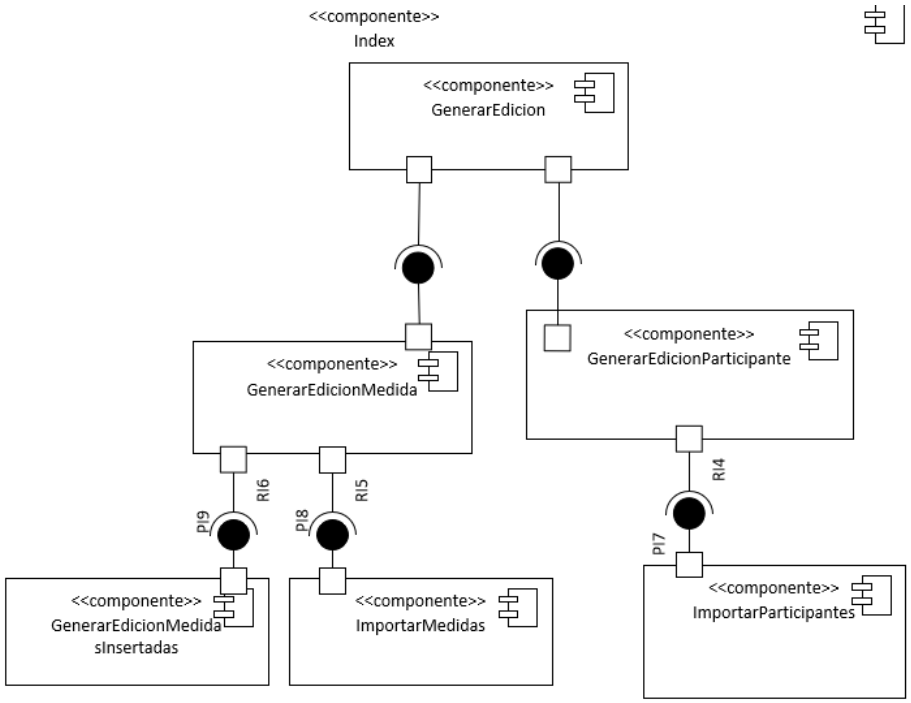

Figura 3.10: Componentes dentro del componente ediciones

3.5.5 Componente Entidades Participantes

Gestiona las entidades participantes (figura 3.11). Se compone de un componente principal y otros dos que dependen del principal. Uno para mostrar el listado y otro para mostrar el formulario para introducir los datos del componente.

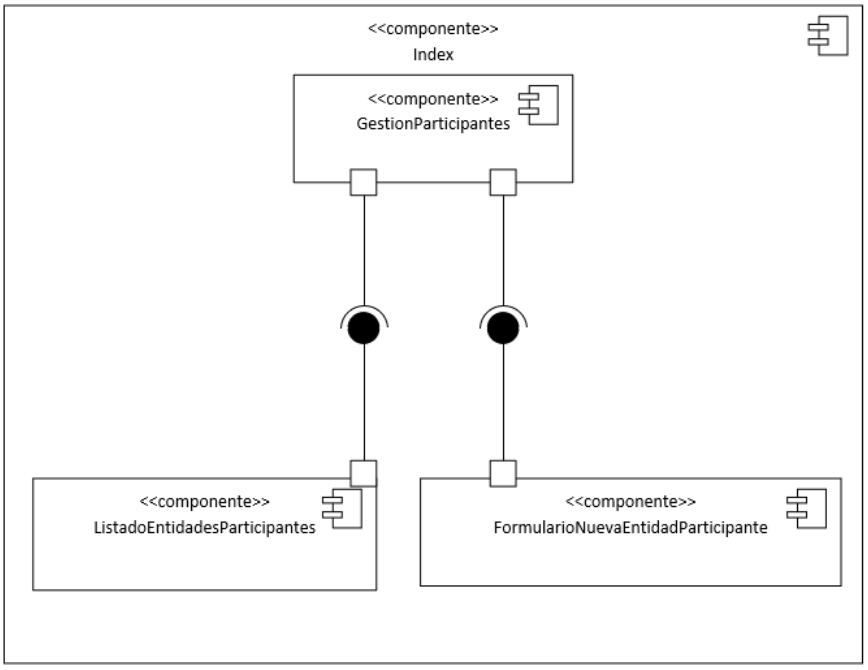

Figura 3.11: Componentes dentro del componente participantes

3.5.6 Componente Visualizador

Es el componente que muestra cruzada toda la información (figura 3.12) de todos los datos que tiene el sistema almacenados. Cada gráfico es un componente de la librería de gráficas a la que se le pasan los dataset correspondientes para que se carguen. Los hay de distintos tipos dependiendo cómo se prefieran mostrar los datos.

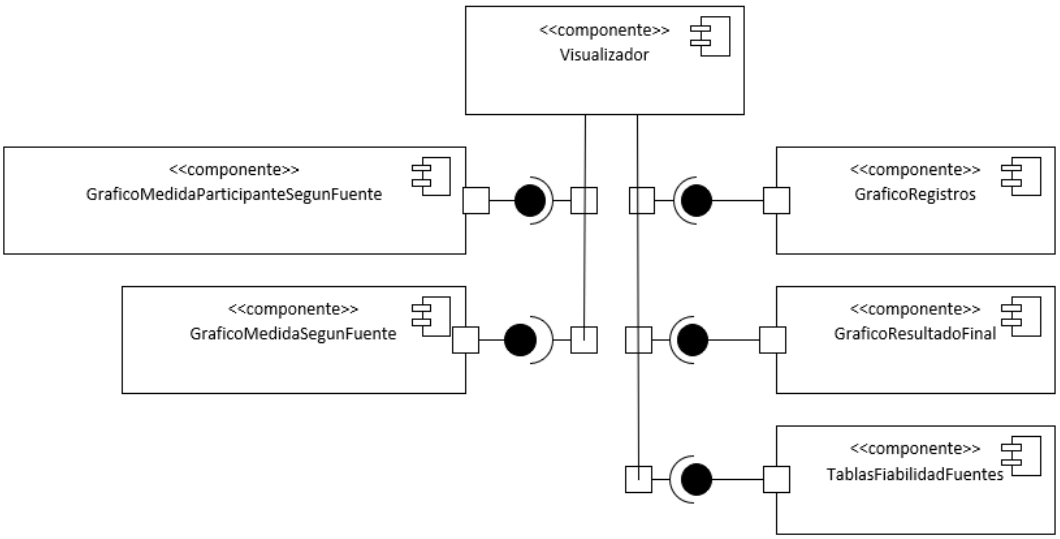

Figura 3.12: Componentes dentro del componente visualizador

3.5.7 Componente Usuarios

Está compuesto por el componente principal de usuarios (figura 3.13) y otro dependiente que muestra los distintos roles disponibles.

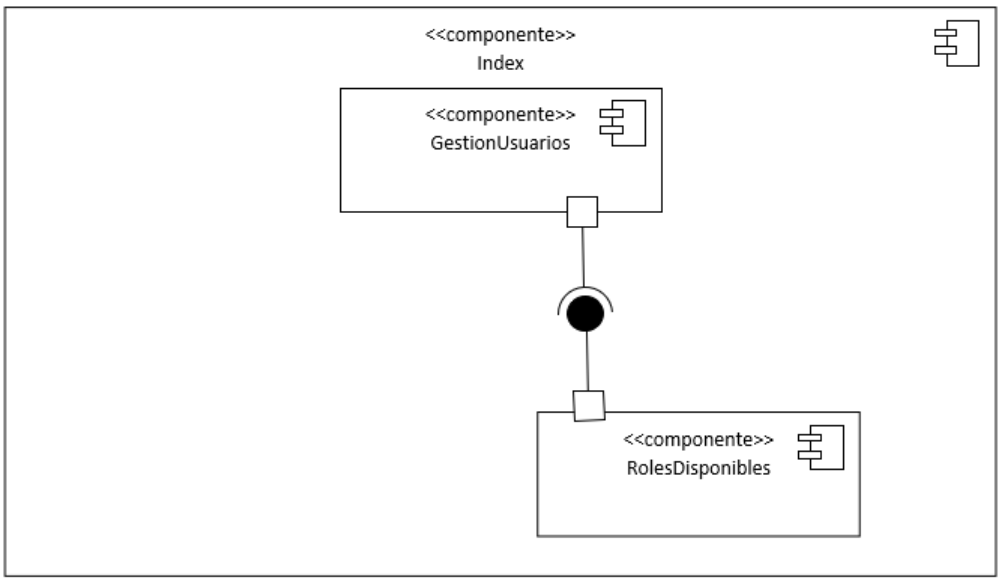

Figura 3.13: Componentes dentro del componente usuarios

3.5.8 Componente Validación

Se compone de un simple componente (figura 3.14) que mostrará dos listados con los pendientes de validar y otro con los ya validados para modificarles su rol.

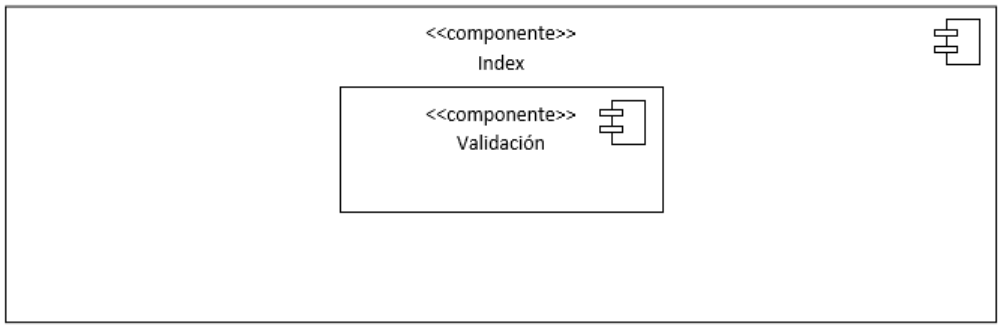

Figura 3.14: Componentes dentro del componente validación

3.5.9 Componente Mi Menú

La figura 3.15 un listado de tipos, ediciones y registros creados por el usuario. Se ha optado por no separar cada listado en un componente ya que en la implementación se tendrá que gestionar la carga de otros componentes principales al pulsar sobre los botones de editar cada uno de esos elementos.

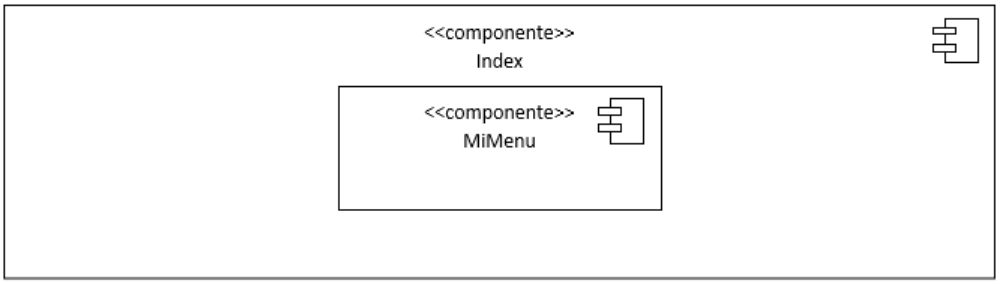

Figura 3.15: Componentes dentro del componente mi menú

3.5.10 Componente Carga Manual

El componente para cargar datos manualmente (figura 3.16) se compondrá de un componente principal, de otro con el formulario con los datos, otro con un histórico de registros actuales en la fuente seleccionada y también un componente sobre un modal-diálogo para poder introducir nuevas fuentes desde la misma pantalla.

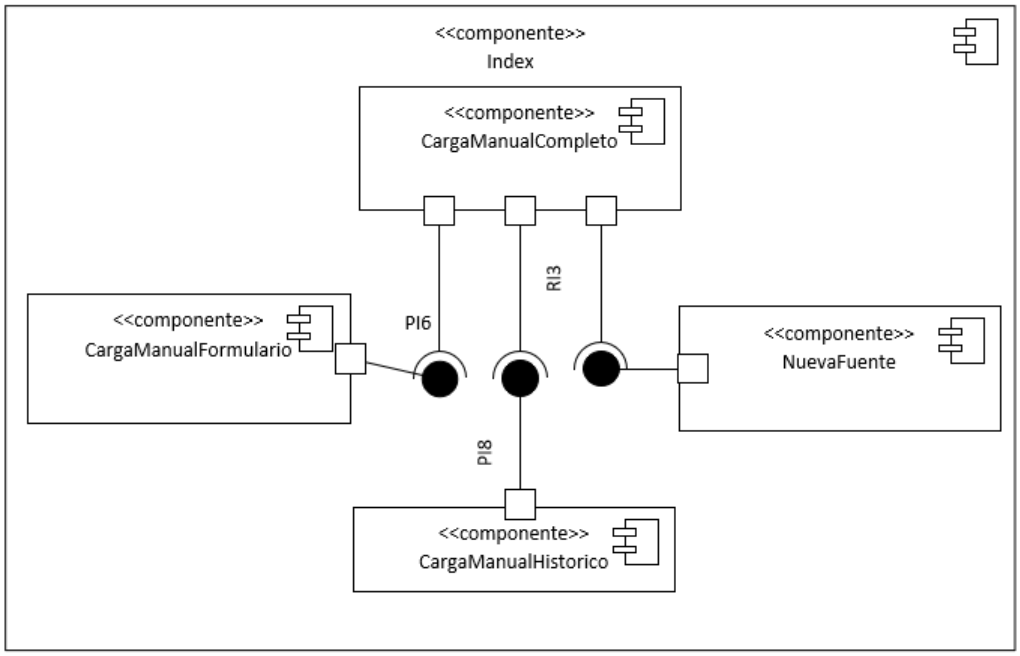

Figura 3.16: Componente dentro del componente de carga manual

3.6 Conclusiones

En este capítulo se han definido los roles que habrá en la aplicación web, con todas las cosas que pueden hacer y las que no; también se ha diseñado toda la estructura de base de datos; y finalmente se han identificado todos los componentes de los que se compondrá la aplicación web.

Con estos puntos ya se puede pasar a la implementación, capítulo en el que se detallará ya con detalle los métodos que hay detrás de cada acción así como los métodos del ORM que accederán a los datos de la base de datos. En la implementación también se detallarán ciertos detalles del desarrollo web, como el uso de las vistas Razor en Blazor, multilenguaje, estructura del proyecto, etc.

Capítulo 4

Implementación

Todo contacto con el bien da lugar a un conocimiento de la distancia entre el mal y el bien[1].

Simone Weil

4.1 Introducción

Una vez tenemos el diseño de datos, cómo se dividirá cada pantalla en componentes y qué funcionalidades se esperan en cada pantalla, toca la implementación de la misma.

Lo primero será detallar los eventos que interactuarán con el cliente y la vista, posteriormente los servicios, se detallará cómo interactúan ambas partes. Finalmente se detallarán puntos clave del desarrollo y se realizarán las pruebas para comprobar su buen funcionamiento.

[1] Simone Wil, La gravedad y la gracia, Editorial Trotta, 2007.

4.2 Casos de uso

En los casos de uso (figura 4.1) se puede observar de una forma general los actores que podrán interaccionar con la aplicación web y con qué pantallas y funcionalidades pueden realizar distintas acciones. Toda la aplicación está construida de una forma de que cada pantalla sea un componente con funcionalidades concretas para cada rol, eso significa que cada usuario que pueda acceder a cierta pantalla tendrá prácticamente total capacidad para hacer uso de todas las funcionalidades que esa pantalla ofrezca. Los roles harán que pueda mostrarse más o menos información, pero no impedirá realizar un uso natural y completo de la pantalla accedida.

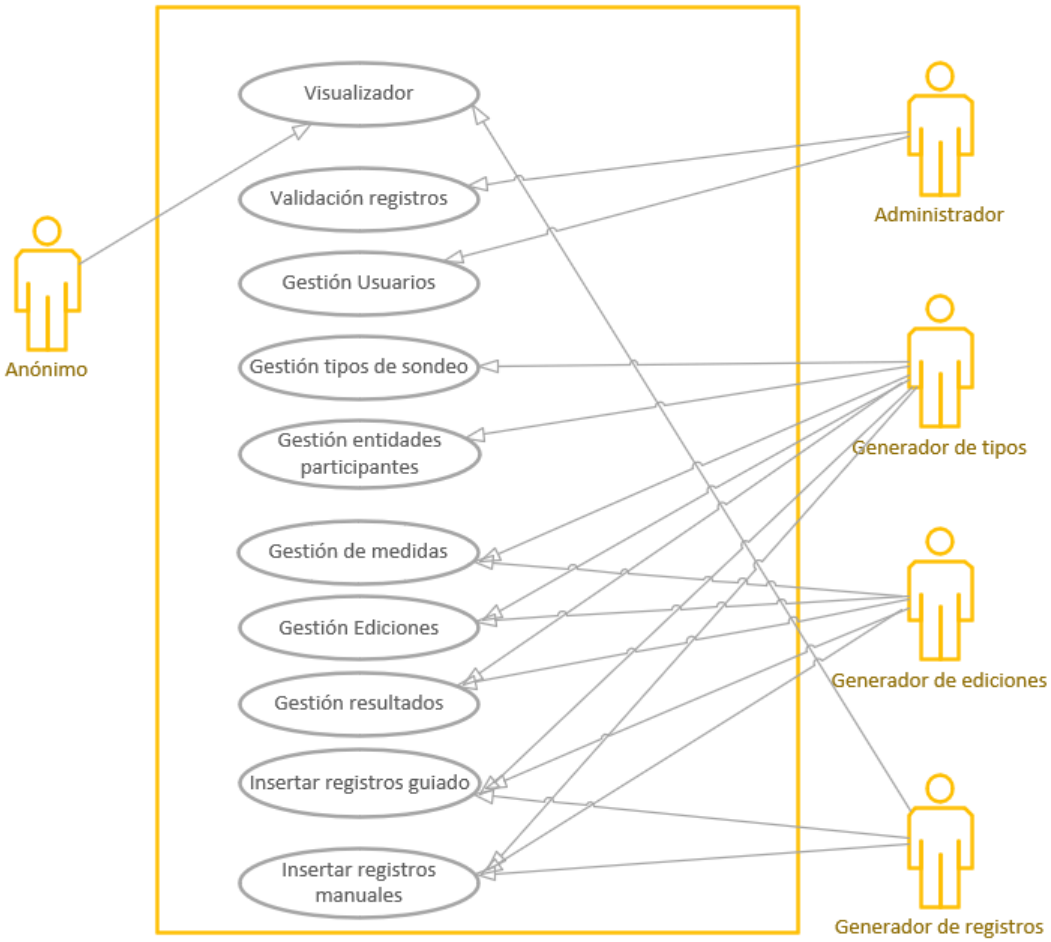

Figura 4.1: Casos de uso

4.3 Explicación del código cliente

Cada componente tendrá una vista y un código asociado, que en el caso de Blazor se escribe en C# que cuando se genera la aplicación web se convertirá en JavaScript, pero ese proceso es transparente de cara al desarrollo y no tiene ningún impacto en el rendimiento. De este modo con sólo conocer un lenguaje, en este caso C#, se podrán programar eventos propios de JavaScript.

Este código que a continuación se detallará es la parte del software que se encargará de todos los cálculos que corresponden a la parte cliente. Se desencadenará al interactuar con la vista, ya sea editando campos o pulsando distintos elementos. En caso de que sea necesario acceder a datos, tanto para obtenerlo, añadirlos o modificarlos, se encargará la parte back de dichas tareas. Ese código estará en clases servicio relacionadas con las entidades del ORM, por lo que siempre que sean necesarios esos servicios el código cliente consultará dicha información a esas clases servicio.

En los esquemas que a continuación se muestran las funciones principales de cada componente, las funciones de sus componentes hijos, así como los servicios que son necesarios. También se detalla lo que ha de realizar cada una de las funciones.

Los servicios serán explicados en su sección correspondiente.

4.3.1 Código cliente en el módulo de gestión de tipos

El módulo de la gestión tipos se descompone en otros dos componentes (figura 4.2), uno para gestionar los datos a generar y otro para gestionar el listado de tipos. En la tabla 4.1 se pueden comprobar los detalles de las funciones.

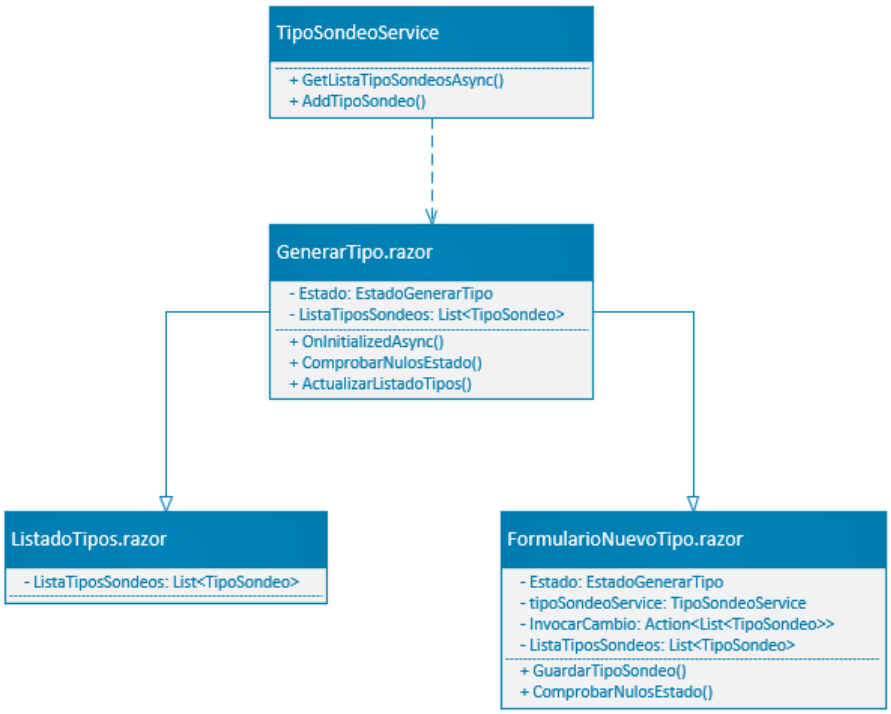

Figura 4.2: Funciones en componentes Gestión Tipos

`OnInitializedAsync`	Carga el listado de tipos disponibles en el sistema.
`ComprobarNulosEstado`	Comprueba posibles campos nulos para marcarlos como cadenas vacías.
`ActualizarListadoTipos`	Actualiza la lista. Es una función llamada desde un Action para actualizar una variable desde otro componente.
`GuardarTipoSondeo`	Guarda el nuevo tipo y actualiza el listado de tipos presentes en el sistema.

Tabla 4.1: Detalle Funciones en componentes Gestión Tipos

4.3.2 Código cliente en el módulo de gestión de ediciones

El siguiente módulo tiene en distintos módulos (figuras 4.3 y 4.4) la gestión de entidades participantes y medidas, así como la gestión del modal que gestiona la importación de esas medidas y entidades participantes. El detalle de las funciones se puede comprobar en la tabla 4.2.

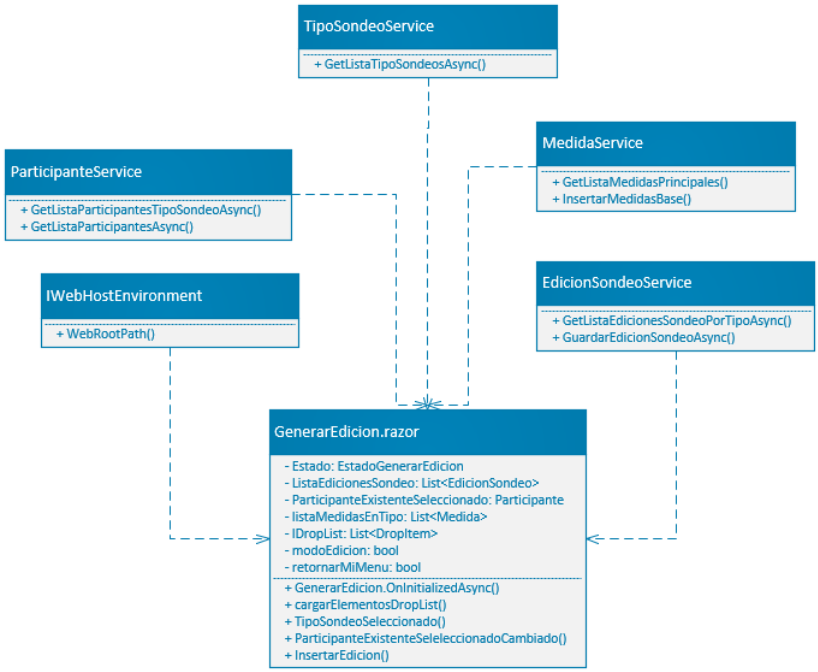

Figura 4.3: Funciones en componentes Gestión Ediciones 1/2

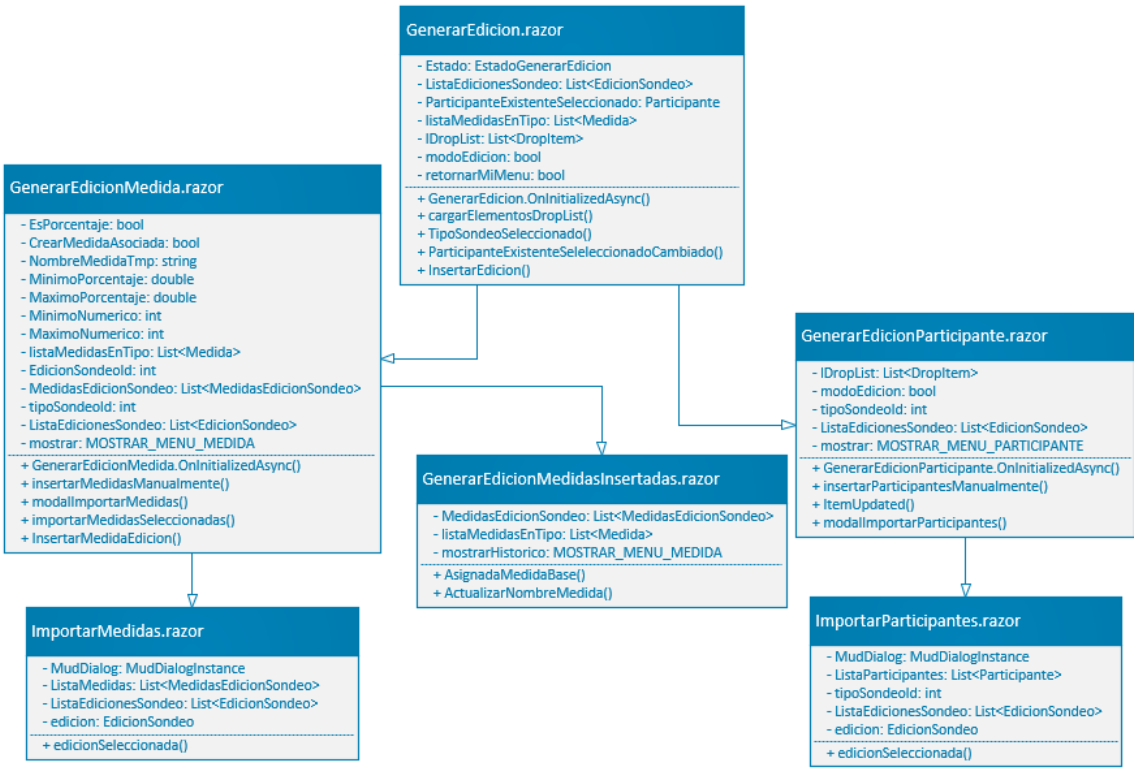

Figura 4.4: Funciones en componentes Gestión Ediciones 2/2

GenerarEdicion.OnInitializedAsync	Configura el componente dependiendo de si se va a realizar una modificación o una insercción de una nueva edición.
cargarElementosDropList	Carga los elementos de entidades participantes que se mostrarán en la pantalla para poder seleccionarlos arrastrando el elemento.
TipoSondeoSeleccionado	Se actualiza el tipo seleccionado. Actualiza entidades participantes, ediciones y medidas.
ParticipanteExistenteSeleleccionadoCambiado	Actualiza el participante seleccionado.
InsertarEdicion	Se guarda la edición introducida.
GenerarEdicionMedida.OnInitializedAsync	Gestiona si se cargará el componente de insercción de medidas.
insertarMedidasManualmente	Actualiza para que las medidas se inserten manualmente.
modalImportarMedidas	Muestra el diálogo cargando el componente correspondiente para importar las medidas desde otra edición del mismo tipo.
importarMedidasSeleccionadas	Si se acepta la importación se actualizan las medidas seleccionadas en la edición que se está creando.
InsertarMedidaEdicion	Se inserta la medida al listado de futuras medidas en la edición, aunque no se guardará hasta que se guarda la edición.
AsignadaMedidaBase	Identifica la medida base a la que pertenece la medida.
ActualizarNombreMedida	Actualiza el nombre de la medida.
GenerarEdicionParticipante.OnInitializedAsync	Determina si se marca como modo de edición o no.
insertarParticipantesManualmente	Se actualiza el modo a mostrar a insertar.
ItemUpdated	Una vez se mueve una entidad participante de seleccionados a no seleccionados se actualiza aquí.
modalImportarParticipantes	Muestra el diálogo cargando el componente correspondiente para importar las entidades participantes desde otra edición del mismo tipo.
ImportarMedidas.edicionSeleccionada	Una vez seleccionada una edición actualiza el listado de medidas pertenecientes a la edición seleccionada.
ImportarParticipantes.edicionSeleccionada	Una vez seleccionada una edición actualiza el listado de participantes pertenecientes a la edición seleccionada.

Tabla 4.2: Detalle Funciones en componentes Gestión Ediciones

4.3.3 Código cliente en el módulo de gestión de entidades participantes

En la figura 4.5 y la tabla 4.3 se detallan todos los eventos que hay en el módulo de entidades participantes.

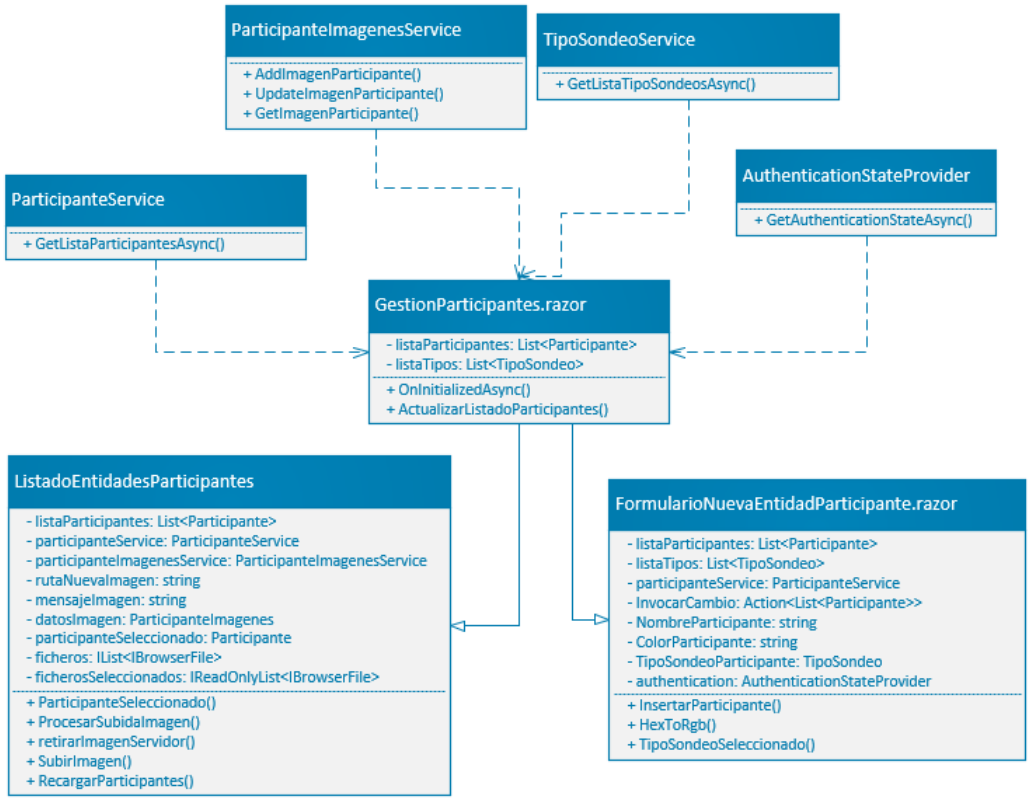

Figura 4.5: Funciones en componente participantes

`OnInitializedAsync`	Hace la carga inicial de participantes y distintos tipos.
`ActualizarListadoParticipantes`	Actualiza el listado de participantes en base a las modificaciones realizadas.
`InsertarParticipante`	Inserta en el sistema la entidad participantes indicada en el formulario.
`HexToRgb`	El color se guarda en hexadecimal, pero necesitamos un formato concreto en rgb para que sea entendido por la librería de gráficos.
`TipoSondeoSeleccionado`	Selecciona el tipo seleccionado en el desplegable.
`ParticipanteSeleccionado`	Carga los datos necesarios de tener en memoria cada vez que se selecciona una entidad participante.
`ProcesarSubidaImagen`	Procesa la subida de la imagen al servidor.
`retirarImagenServidor`	Retira la imagen del servidor.
`SubirImagen`	Gestiona la validación para subir la imagen al servidor.
`RecargarParticipantes`	Recarga el listado de entidades participantes tras una modificación.

Tabla 4.3: Detalle Funciones en componentes Gestión Ediciones

4.3.4 Código cliente en el módulo de guardar manual

En la figura 4.6 y la tabla 4.4 se detalla en módulo de carga manual de registros. Además del componente con campos y el histórico se detalla el componente para generar nuevas fuentes, que será mostrado como un modal al pulsar su botón correspondiente.

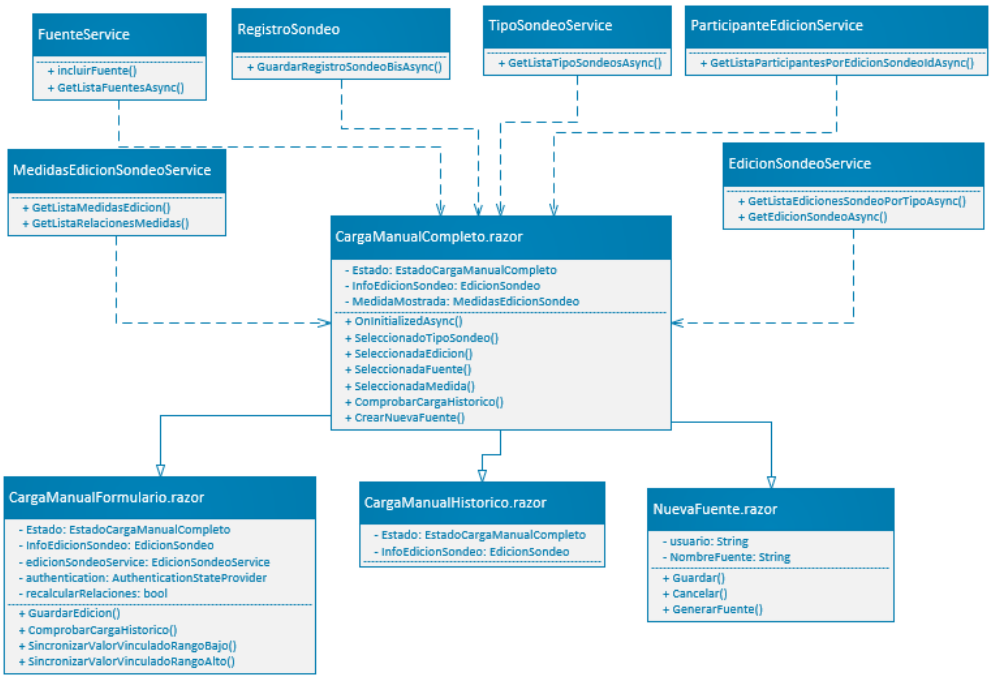

Figura 4.6: Funciones en componente carga manual

OnInitializedAsync	Identifica si la carga del componente es para generación o modificación.
SeleccionadoTipoSondeo	Marca el tipo seleccionado y carga listado de ediciones.
SeleccionadaEdicion	Selecciona la edición y carga el listado de participantes, medidas, relaciones de medidas y fuentes.
SeleccionadaFuente	Selecciona fuente y configura la pantalla para introducir los datos del registro a introducir.
SeleccionadaMedida	Selecciona la medida.
ComprobarCargaHistorico	Actualiza el histórico en base a los datos seleccionados en el filtro.
CrearNuevaFuente	Se encarga de cargar el modal para generar nueva fuente.
GuardarEdicion	Guarda la edición con los datos introducidos.
ComprobarCargaHistorico	Se invoca tras insertar un nuevo registro para actualizar histórico de registros.
SincronizarValorVinculadoRangoBajo	Actualiza los valores bajos para automatizar medidas relacionadas.
SincronizarValorVinculadoRangoAlto	Actualiza los valores altos para automatizar medidas relacionadas.
Guardar	Hace la validación de los datos y guarda los datos en el sistema.
Cancelar	Cierra el modal de añadir nueva fuente.
GenerarFuente	Inserta la nueva fuente en el sistema.

Tabla 4.4: Detalle Funciones en componentes Carga Manual

4.3.5 Código cliente en el módulo de carga guiada

En este módulo se muestran las funciones que analizará el DOM de una URL facilitada en búsqueda de tablas (figura 4.7 y tabla 4.5). Una vez detectadas esas tablas se cargarán en memoria en forma de formularios para que el usuario pueda identificar qué es una entidad participante, una medida, qué datos sobran además de poder modificar los datos manualmente o automatizando dicha tarea.

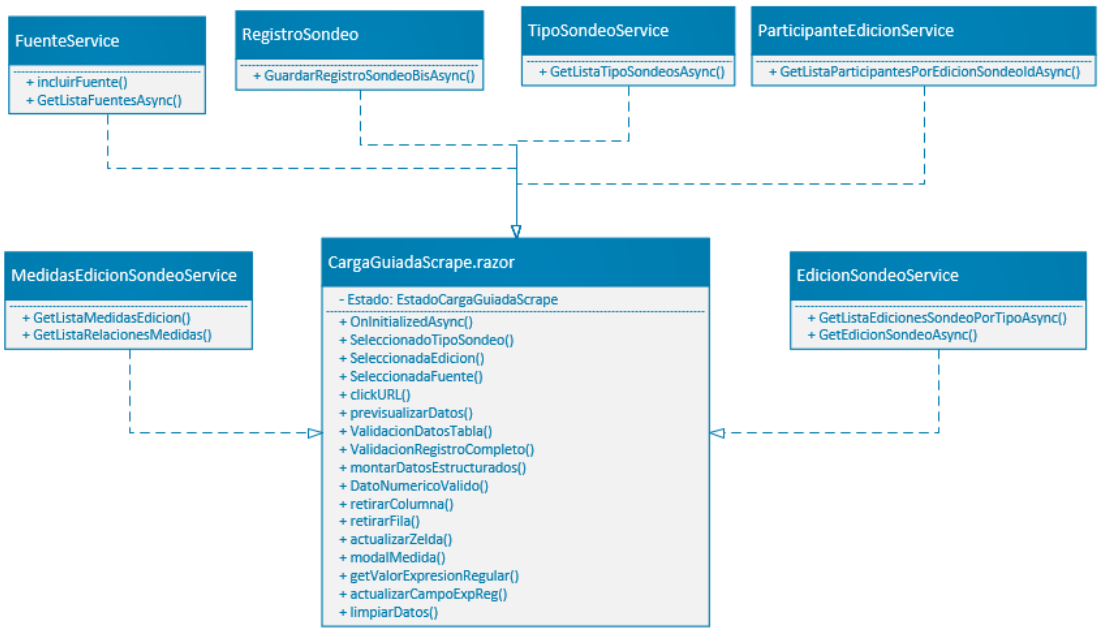

Figura 4.7: Funciones en componente carga guiada

clickURL	Buscará información en la URL facilitada buscando tablas para montar el formulario de caga.
limpiarDatos	Limpiará las filas de datos cargados en base al campo de formato de limpia.
previsualizarDatos	El el botón que guarda los datos, pero antes de guardarlos realizará las validaciones de que todo sea correcto.
ValidacionDatosTabla	Valida que todo sea correcto. En caso de no serlo no avanza para no guardar un registro erróneo.
montarDatosEstructurados	Si los datos son válidos montará la estructura para comunicarse con el ORM.
ValidacionRegistroCompleto	Valida que estén todos los participantes y medidas.
DatoNumericoValido	Evalúa si el campo pasado es un dato flotante.
retirarColumna	Se retira la columna seleccionada.
retirarFila	Se retira la fila seleccionada.
actualizarCelda	Actualiza el valor de la celda cuando el usuario modifica el dato.
getValorExpresionRegular	Lee el texto para limpiar los datos.
actualizarCampoExpReg	Altera el valor enlazado con el texto para limpiar los datos.

Tabla 4.5: Detalle Funciones en componentes Carga guiada

4.3.6 Código cliente en el módulo visualizador

En el siguiente módulo (figura 4.8 y tabla 4.6) se explican todas las funciones en la parte cliente del componente visualizador, que a la vez se compone de un componente distinto por cada tipo de gráfico.

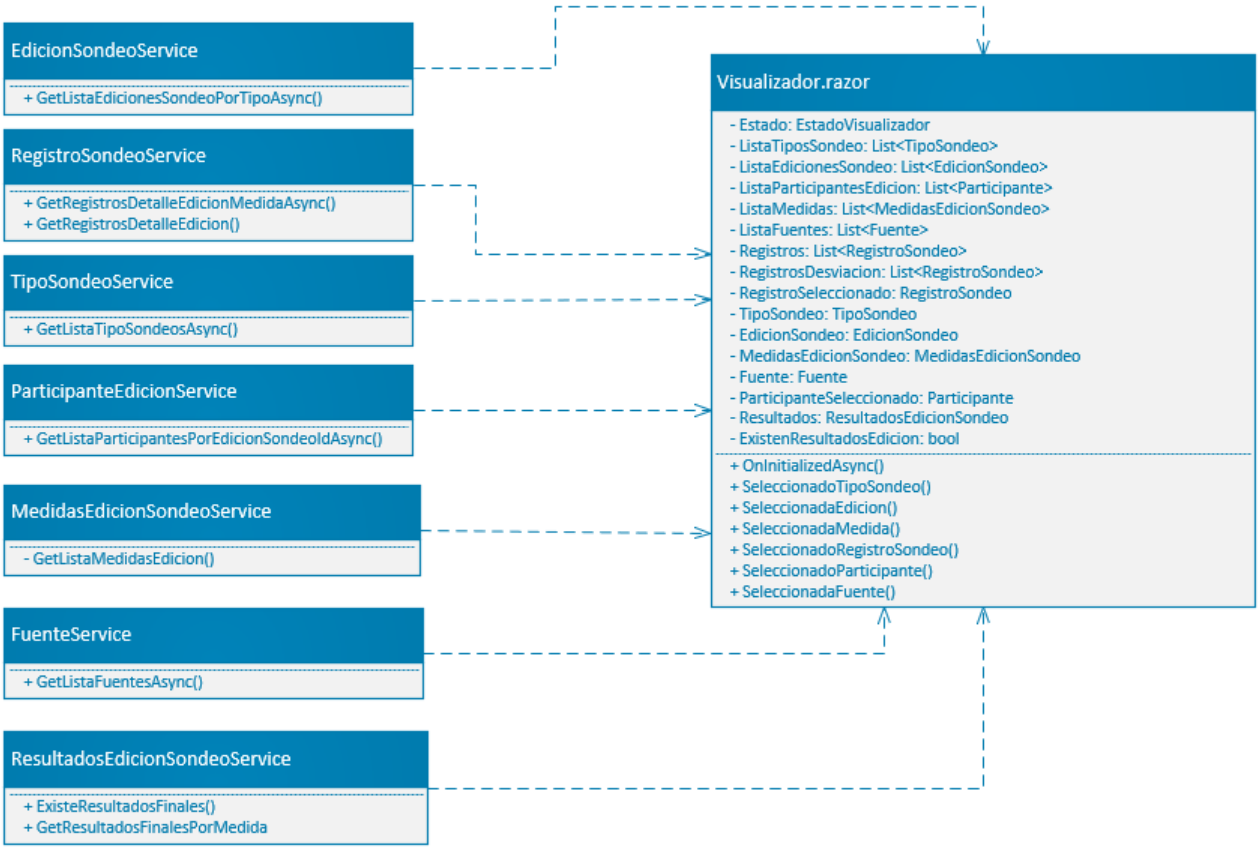

Figura 4.8: Funciones en componente visualizador

`OnInitializedAsync`	Se ejecuta al cargar el componente. Carga el listado de tipos.
`SeleccionadoTipoSondeo`	Se desencadena al seleccionar un tipo. Carga ciertos gráficos.
`SeleccionadaEdicion`	Se desencadena al seleccionar una edición. Carga entidades participantes, fuentes, medidas y resultados finales.
`SeleccionadaMedida`	Se desencadena al seleccionar una medida. Carga resultados y ciertos gráficos.
`SeleccionadoRegistroSondeo`	Se desencadena al seleccionar un registro concreto. Carga ciertos gráficos.
`SeleccionadoParticipante`	Se desencadena al seleccionar una entidad participante. Carga ciertos gráficos.
`SeleccionadaFuente`	Se desencadena al seleccionar una fuente. Carga registros, estadísticas de acierto y carga ciertos gráficos.

Tabla 4.6: Detalle Funciones en componentes visualizador

4.3.7 Código cliente en el módulo de menú personal

El módulo mi menú cargará un resumen de los elementos creados por el usuario. Las funciones han de permitir seleccionar los elementos y pulsar para editarlos o crear nuevos (figura 4.9 y tabla 4.7).

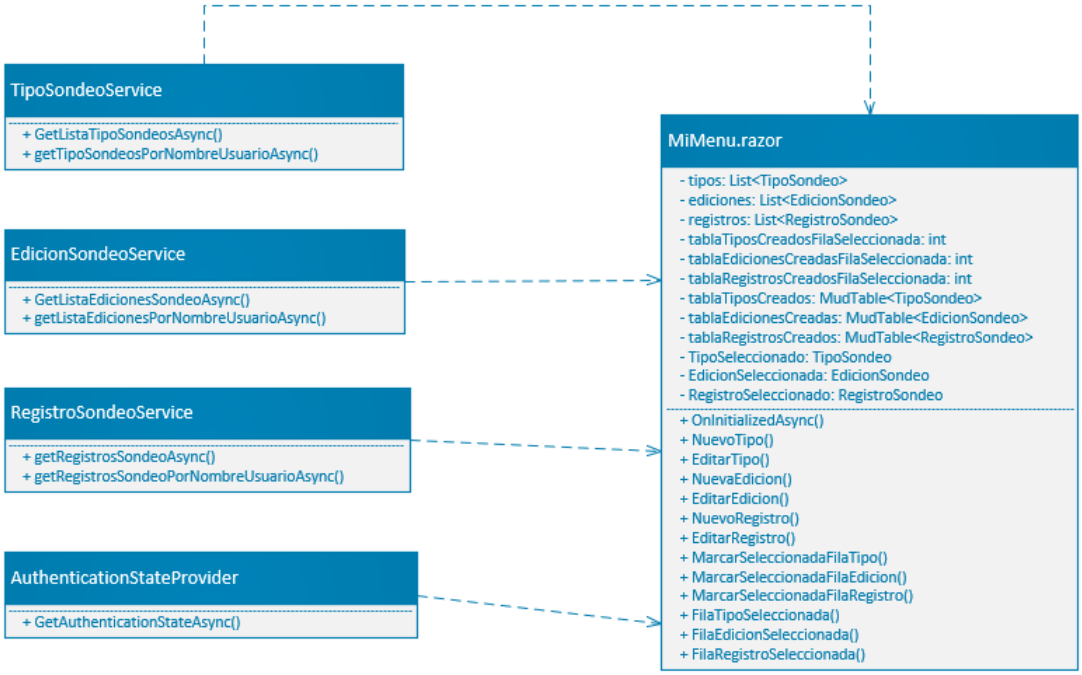

Figura 4.9: Funciones en componente mi menú

`OnInitializedAsync`	Hace la carga inicial de datos. Varía si el usuario es administrador u otro rol.
`NuevoTipo`	Cargará el componente de tipos sin tipos para que sea tomado como un tipo nuevo.
`EditarTipo`	Cargará el componente de tipos con el tipo seleccionado cargado.
`NuevaEdicion`	Cargará el componente para generar una nueva edición.
`EditarEdicion`	Cargará el componente para editar la edición seleccionada.
`NuevoRegistro`	Carga el componente para cargar registro.
`EditarRegistro`	Cargará el componente para editar el registro seleccionado.
`MarcarSeleccionadaFilaTipo`	Indica visualmente la fila de tipo seleccionada.
`MarcarSeleccionadaFilaEdicion`	Indica visualmente la fila de edición seleccionada.
`MarcarSeleccionadaFilaRegistro`	Indica visualmente la fila de registro seleccionada.
`FilaTipoSeleccionada`	Selecciona Tipo pulsado.
`FilaEdicionSeleccionada`	Selecciona Edición pulsada.
`FilaRegistroSeleccionada`	Selecciona registro pulsado.

Tabla 4.7: Detalle funciones en componente mi menú

4.3.8 Código cliente en el módulo de gestión de usuarios

El módulo de gestionar usuarios permitirá dar acceso a usuarios además de cambiar sus roles (figura 4.10 y tabla 4.8).

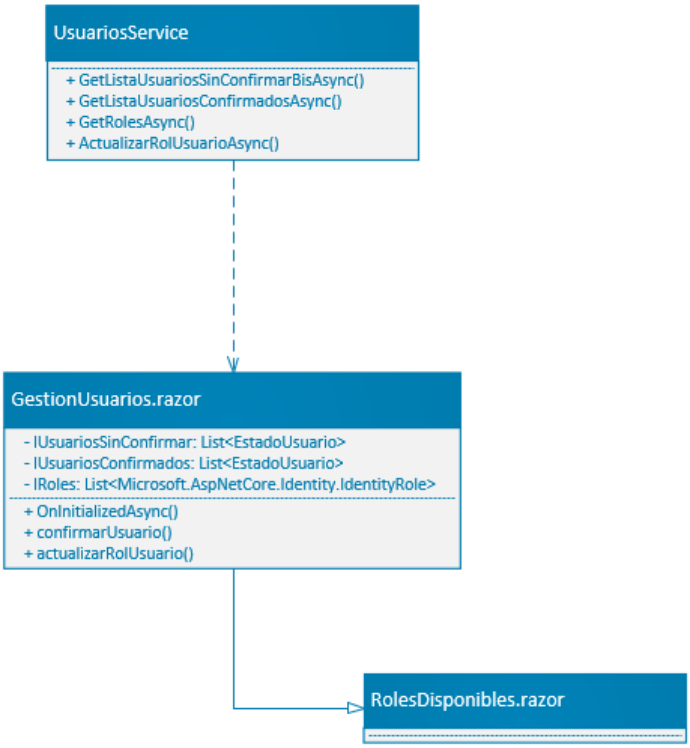

Figura 4.10: Funciones en componente Usuarios

`OnInitializedAsync`	Carga los usuarios del sistema, tanto confirmados como no confirmados, así como los distintos roles para mostrar en lo seleccionable.
`confirmarUsuario`	Al pulsar sobre confirmar usuario lo que se hace es poner el correo electrónico como confirmado para que pueda acceder.
`actualizarRolUsuario`	Modifica el rol del usuario por el rol seleccionado.

Tabla 4.8: Detalle funciones en componente usuarios

4.3.9 Código cliente en el módulo de validación

En este módulo se validarán los registros de distintas fuentes introducidos por los usuarios. Los considerados aptos se mostrarán a los demás en los gráficos (figura 4.11 y tabla 4.9).

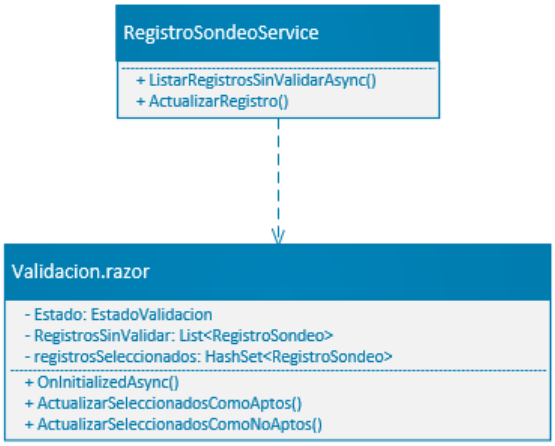

Figura 4.11: Funciones en componente Validación

`OnInitializedAsync`	Carga listado con registros sin validar.
`ActualizarSeleccionadosComoAptos`	Actualiza como aptos los registros seleccionados.
`ActualizarSeleccionadosComoNoAptos`	Actualiza como no aptos los registros seleccionados.

Tabla 4.9: Detalle funciones en componente validación

4.3.10 Código cliente en el módulo inicial

Gestionará dependiendo de su hay sesión y del rol registrado qué se puede mostrar (figura 4.12 y tabla 4.10).

Figura 4.12: Funciones en componente índice

`OnInitializedAsync`	Al iniciarse el menú índice, que a la vez es el primer componente que se carga en el proyecto, se cargará el módulo de Mi menú, para mostrar un dashboard resumen del usuario, y también se cargarán los estilos de los botones superiores.
`InicializarEstilos`	Inicializa los colores de todos los botones superiores.
`ActualizarEstilos`	Actualiza los estilos de los botones para que el módulo pulsado y posteriormente en pantalla aparezca diferenciado del resto para que sea fácilmente reconocer en qué módulo se encuentra el usuario.
`CargarComponente`	Actualiza el componente actual activo. Es empleado cuando se pulsan los botones superiores para abrir un nuevo módulo.

Tabla 4.10: Detalle funciones en componente índice

4.4 Explicación del código servidor (Servicios)

Mientras que las funciones en la parte cliente se encargan de toda interacción con la vista y proceso con toda la información disponible en memoria, serán los servicios, en la parte back en el servidor, los encargados de interactuar con el ORM y en definitiva de crear datos así como introducirlos y alterarlos. En la medida de lo posible los servicios deberán limitarse a esas acciones de consulta ya sea con procesos de `Entity Framework Core` o consultas `LINQ` sobre las entidades, pero siempre evitando cálculos innecesarios ya que ese trabajo corresponde, antes y después de cualquier acceso a datos, a la parte cliente.

4.4.1 Funcionamiento de los servicios

No se detallará en estos capítulos y secciones el código concreto de cada servicio, pero sí que se hará una introducción en este apartado de cómo están programados en general todas las funciones presentes en cada uno de los servicios. Posteriormente en cada servicio sólo de indicará qué datos devuelve en base a la información proporcionada como parámetro.

Como ya se explicó en su correspondiente apartado, cada tabla y relación física tiene su correspondencia en el proyecto en forma de clase, de forma que se puedan hacer migraciones, pero también consultas de distinto tipo. De esta forma, con indicar en el contexto la relación de tabla-clase ya es posible hacer consultas directamente sobre esa clase.

Por ejemplo, la tabla `Fuente` está asignada en el contexto a `Fuentes`. `Fuente` es una clase que replica el contenido de la tabla `Fuente`, por lo tanto si indicamos el siguiente código para introducirlo al contexto:

```
1   public DbSet<Fuente> Fuentes { get; set; }
```

y en Startup inyectamos el servicio asociado:

```
1   services.AddScoped<FuenteService>();
```

Con estas dos instrucciones ya tendremos la entidad `Fuente` en el proyecto para poder ser consultada, y el servicio `FuenteService` preparado para poder ser inyectado y llamado en cualquier componente del proyecto.

Entonces, para realizar consultar sobre ese contexto se realizará empleando instrucciones propias del ORM `Entity Framework Core` o haciendo uso de consultas `LINQ`.

Por ejemplo, el método `GetListaFuentesAsync` de `FuenteService` es el siguiente:

```
1   public async Task<List<Fuente>> GetListaFuentesAsync()
2   {
3     return await _applicationDbContext.Fuentes.ToListAsync();
4   }
```

Fuentes es el acceso directo a los datos de la tabla `Fuente` a través del contexto, por lo que sobre `Fuentes` podremos invocar a métodos propios del Framework para interaccionar sobre la tabla con código C# y otros métodos que proporcione el Framework.

Por ejemplo, para realizar un insert, sería tan sencillo como invocar a `Add` con un objeto del tipo `Fuente`, y `Entity Framework Core` se encargaría de realizar el INSERT o el UPDATE si identifica que el Id ya existe en la tabla.

En el siguiente código de ejemplo no se indica `Fuentes` junto a `applicationDbContext` porque al estar insertando un objeto de tipo `Fuente`, `Entity Framework Core` ya es capaz de identificar a qué entidad y tabla hace referencia. Ambas formas son perfectamente válidas.

```
1   public async Task<Fuente> incluirFuente(Fuente fuente)
2   {
3       _applicationDbContext.Add(fuente);
4       await _applicationDbContext.SaveChangesAsync();
5
6       return fuente;
7   }
```

Otra de las ventajas de acceder directamente a los datos a través de `Entity Framework Core`, que también es posible empleando `LINQ`, es el uso de expresiones lambda. Generando un código limpio, directo y eficiente. Por ejemplo, si se desea consultar por un registro concreto en base a su Id, se hace uso de una expresión lambda junto al método proporcionado por `Entity Framework Core` Include:

```
1   List<Fuente> lFuentes = _applicationDbContext.Fuentes
2                                                .Include(x => x.Id.Equals(fuenteId)).ToList();
```

Esta consulta, en vez de Include podría ser un Where o cualquier otro método disponible, que varía según las necesidades.

Finalmente un ejemplo de uso de `LINQ` sería el siguiente, que perfectamente sería posible refactorizar empleando métodos propios de `Entity Framework Core`.

```
1   public async Task<List<Participante>> GetListaParticipantesPorEdicionSondeoAsync
2                                                    (EdicionSondeo edicionSondeo)
3   {
4       return await (from pe in _applicationDbContext.ParticipanteEdiciones
5                     join p in _applicationDbContext
6                                   .Participantes on pe.ParticipanteId equals p.Id
7                     where pe.EdicionSondeoId == edicionSondeo.Id
8                     select p).ToListAsync();
9   }
```

4.4.2 Funciones en el servicio EdicionSondeoService

El servicio de las ediciones (tabla 4.11) de los distintos tipos devolverá los datos encesarios para guardar actualizaciones o retornar información de las mismas dependiendo de las necesidades.

GetEdicionSondeoAsync	Retorna la edición con las medidas y fuente indicadas.
GetListaEdicionesSondeoPorTipoAsync	Devuelve listado de ediciones del tipo de sondeo seleccionado.
GetListaEdicionesSondeoAsync	Devuelve listado de todas las ediciones de sondeos registradas.
getListaEdicionesPorNombreUsuarioAsync	Devuelve todas las ediciones creadas por un usuario concreto.
GuardarEdicionSondeoAsync	Guarda o actualiza una nueva edición.

Tabla 4.11: Funciones en el servicio EdicionSondeoService

4.4.3 Funciones en el servicio FuenteService

El servicio de la tabla Fuente (tabla 4.12) recuperará e insertará fuentes.

GetListaFuentesAsync	Retorna el listado de todas las fuentes disponibles.
incluirFuente	Añade o actualiza la fuente indicada.

Tabla 4.12: Funciones en servicio FuenteService

4.4.4 Funciones en el servicio MedidaService

El servicio de la tabla Medida (tabla 4.13) recuperará e insertará medidas.

GetListaMedidasPrincipales	Recupera el listado de medidas base.
InsertarMedidasBase	Inserta la medida base en el tipo de sondeo indicado.

Tabla 4.13: Funciones en servicio MedidaService

4.4.5 Funciones en el servicio MedidasEdicionSondeoService

Una medida está pensado como un dato superior, mientras que la MedidaEdicionSondeo es un dato con una configuración concreta para cada edición, y su servicio (tabla 4.14) tiene los siguientes métodos:

GetListaMedidasEdicion	Devuelve las medidas de una edición concreta.
GetListaRelacionesMedidas	Devuelve la relación de medidas en una edición concreta, lo que se empleará para la actualización de esos valores de forma automática en base a la medida relacionada.

Tabla 4.14: Funciones en servicio MedidasEdicionSondeoService

4.4.6 Funciones en el servicio ParticipanteService

Este servicio (tabla 4.15) se encargará de interactuar en la tabla Participante, que almacena información de las distintas entidades participantes.

GetListaParticipantesAsync	Devuelve el listado completo de todos los participantes.
GetListaParticipantesTipoSondeoAsync	Devuelve el listado completo de todos los participantes en un tipo concreto.
IncluirParticipanteAsync	Inserta el participante indicado.

Tabla 4.15: Funciones en servicio ParticipanteService

4.4.7 Funciones en el servicio ParticipanteEdicionService

Este servicio (tabla 4.16) se encarga de interactuar con la tabla ParticipanteEdicion, que relaciona entidades participantes y ediciones.

GetListaParticipantesPorEdicionSondeoIdAsync	Retorna todos los participantes en una edición concreta.

Tabla 4.16: Funciones en servicio ParticipanteEdicionService

4.4.8 Funciones en el servicio ParticipanteImagenesService

Este servicio (tabla 4.17) interactúa con la tabla que gestiona las imágenes de las entidades participantes.

GetImagenParticipante	Retorna la imagen asociada a la entidad participante.
AddImagenParticipante	Inserta la imagen indicada a la entidad participante.
UpdateImagenParticipante	Actualiza la imagen indicada a la entidad participante pasada por parámetro.

Tabla 4.17: Funciones en servicio ParticipanteImagenesService

4.4.9 Funciones en el servicio RegistroSondeoService

Este servicio (tabla 4.18) interactúa con la tabla que almacena los registros de las distintas ediciones.

GuardarRegistroSondeoAsync	Inserta o actualiza el registro indicado.
GuardarRegistroSondeoBisAsync	Inserta o actualiza el registro indicado sin devolver el registro insertado.
ListarRegistrosSinValidarAsync	Retorna el listado, independientemente de tipos y ediciones, de los registros pendientes de ser validados, tanto para ser aptos como no aptos.
ActualizarRegistro	Se actualiza el registro.
GetRegistrosDetalleEdicion	Retorna el listado de registros de una edición con datos del detalle de los mismos.
GetRegistrosDetalleEdicionMedidaAsync	Retorna el listado de registros de una edición con datos del detalle de los mismos filtrando por medida y fuente.
getRegistrosSondeoAsync	Retorna todos los registros disponibles en el sistema.
GetFechasDisintosRegistrosEnEdicion	Retorna el listado de distintas fechas con registros en la edición indicada. Se emplea para mostrar los gráficos estadísticos.
getRegistrosSondeoPorNombreUsuarioAsync	Retorna el listado de registros creados por el usuario indicado.

Tabla 4.18: Funciones en servicio RegistroSondeoService

4.4.10 Funciones en el servicio ResultadosEdicionSondeoService

Este servicio (tabla 4.19) interactúa con la tabla que almacena los resultados finales de las distintas ediciones registradas.

ExisteResultadosFinales	Identifica si hay resultados en la edición indicada.
GetResultadosFinales	Retorna la información de los resultados finales de la edición indicada.
GetResultadosFinalesPorMedida	Retorna la información de los resultados finales de la edición indicada filtrando por medida.
GuardarActualizarFinales	Guarda o actualiza los datos finales indicados.

Tabla 4.19: Funciones en servicio ResultadosEdicionSondeoService

4.4.11 Funciones en el servicio TipoSondeoService

Este servicio (tabla 4.20) interactúa con la tabla que se encarga de almacenar la información de los distintos tipos.

GetListaTipoSondeosAsync	Devuelve la lista de todos los tipos de sondeo.
getTipoSondeosPorNombreUsuarioAsync	Devuelve la lista de todos los tipos de sondeo creados por el usuario indicado.
AddTipoSondeo	Añade o actualiza el tipo indicado.

Tabla 4.20: Funciones en servicio TipoSondeoService

4.4.12 Funciones en el servicio UsuariosService

Este servicio (tabla 4.21) interactúa con las tablas generadas por .NET para gestionar los roles y usuarios, que son Users, UserRoles y Roles.

GetListaUsuariosConfirmadosAsync	Retorna el listado de usuarios que están aceptados en el sistema.
GetListaUsuariosSinConfirmarBisAsync	Retorna el listado de usuarios que están pendientes de ser aceptados o rechazados en el sistema.
GetRolesAsync	Retorna el listado de roles disponibles en el sistema.
ConfirmarUsuario	Confirma como un usuario válido en el sistema el usuario indicado..
ActualizarRolUsuarioAsync	Actualiza el actual rol del usuario indicado al rol que se indica en el parámetro..

Tabla 4.21: Funciones en servicio UsuariosService

4.5 Diagramas de interacción

En los diagramas de interacción se detalla el comportamiento interno que desencadena cada acción posible sobre la interfaz.

Se mostrarán los diagramas de interacción de las principales tareas para entender cómo que comunican las distintas partes de la aplicación.

4.5.1 Diagrama de interacción al iniciar componente

El primer diagrama de interacción (figura 4.13) muestra lo que desencadena cargar un componente Inicio es lo que se desencadena cuando entramos en la pantalla y que ya por defecto cargará información para ser mostrada.

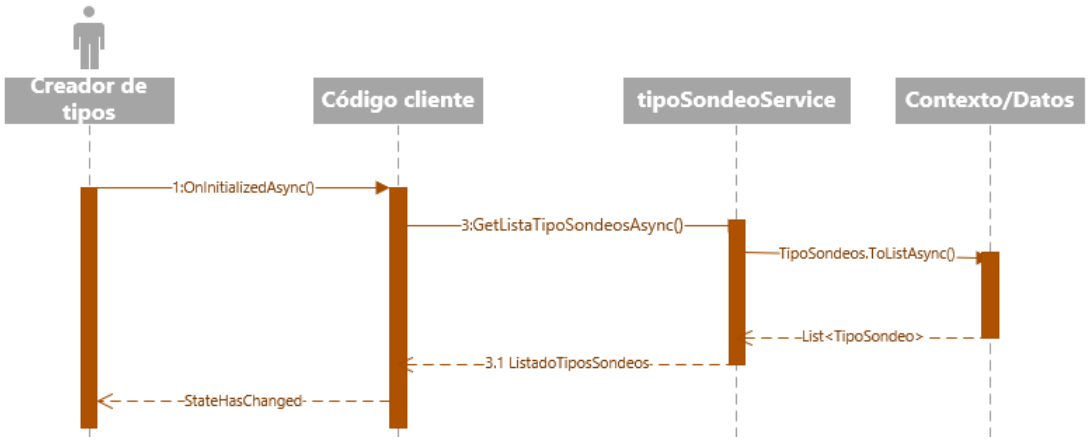

Figura 4.13: Interacciones que se desencadenan al iniciar el componente Tipos

4.5.2 DI - Insertar o actualizar información

La siguiente sucesión de llamadas (figura 4.14) se desencadena cuando se ha pulsado el botón guardar, que lo que hace es coger los valores introducidos y generar un nuevo tipo para insertar ediciones y posteriormente registros.

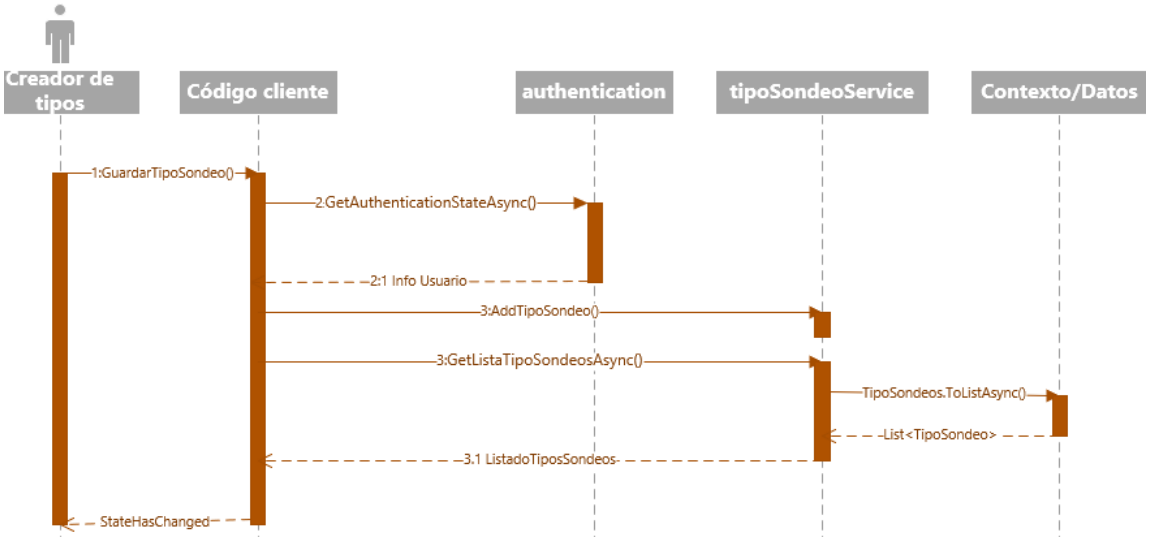

Figura 4.14: Interacciones que se desencadenan al pulsar sobre guardar tipo

4.5.3 Diagrama de interacción al asociar imagen

En cuanto se selecciona una imagen para vincular a una entidad participante (figura 4.15) se hace una primera comprobación, y si el fichero de la imagen ocupa más de 15Kb no se realizará ninguna tarea pues sobrepasa el máximo permitido. En caso de que la imagen sea válida se comprobará si la entidad participante ya tenía imagen. En caso de tener se retira del servidor. Se comprueba si hay directorio para la entidad participante, pues cada una se almacena en su propio directorio. Si no existía se crea. A continuación se crea-copia el fichero en el servidor. Luego dependiendo de si es un añadido o una actualización se actualiza el registro de la base de datos de la entidad participante con la información necesaria para que se muestre en la aplicación web. Finalmente se recarga la información para que se muestre.

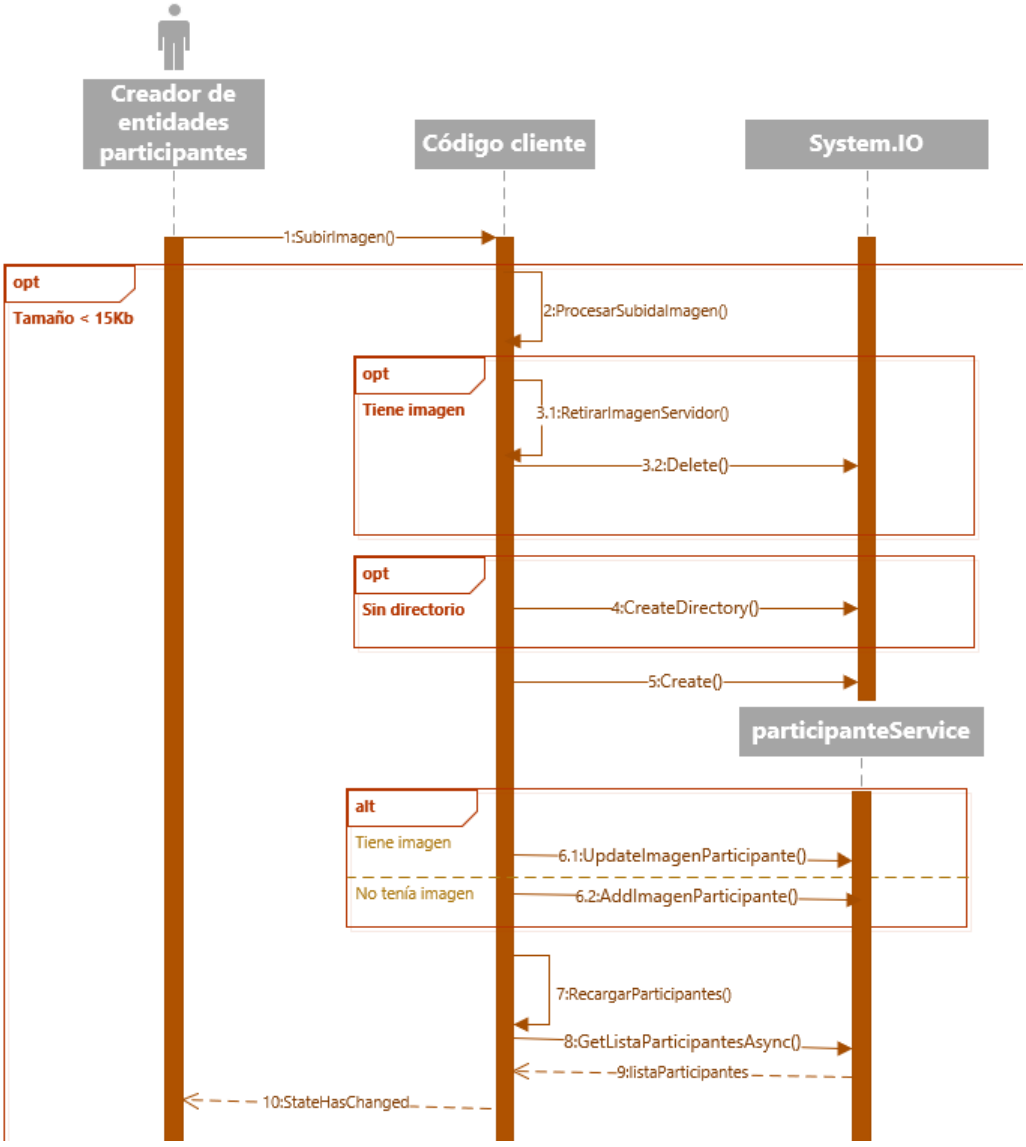

Figura 4.15: Interacciones que se desencadenan al asociar una imagen a una entidad participante

Toda interacción será siempre igual. Un usuario interaccionará con la pantalla y esos eventos serán gestionados por las funciones cliente. En caso de necesitarse comunicarse con la base de datos se solicitará a los servicios, en el servidor, y el servicio interaccionará con el contexto del ORM que realizará la comunicación física con la base de datos. Para devolver la información se realizará de idéntica forma pero pero de forma inversa hasta devolver la información a la vista.

4.6 Implementación del proyecto en Blazor

Detallar minuciosamente cada componente y aspecto relacionado con la programación haría tener un capítulo de implementación demasiado largo y repetitivo, por eso en esta sección lo se que hará es enumerar los distintos detalles, elementos o capacidades tecnológicas para tener una idea completa de qué cosas se han empleado para lograr generar una aplicación más interactiva. No sólo se pretende mostrar cómo es trabajar con componentes, pues esa particularidad ya está mostrada en el diseño y qué beneficios aporta, pero es necesario mostrar cómo llevarlo al proyecto y qué distintas posibilidad se pueden aplicar para resolver distintos problemas. Este apartado es en el que más investigación se ha realizado para implementar todo el diseño del TFG, pues un reto a afrontar tras analizar qué tipo de herramienta realizar fue la de poder hacer uso de Blazor y que todo problema encontrado pudiese ser solventado de la forma más limpia posible y que no lleva a realizar código sucio e ineficiente. Por eso es importante detallar, mostrando ejemplos aplicados en el proyecto, cómo se han ejecutado y qué dificultades se han podido encontrar, pues tecnologías como angular o react tienen una gran base de usuarios detrás y es más sencillo de encontrar respuestas a esas cuestiones, mientras que en el caso de Blazor eso no siempre ha sido posible.

4.6.1 Componentes en Blazor

Como ya se ha comentado, una de las principales razones para emplear Blazor ha sido la simplicidad para generar y configurar componentes, así como la posibilidad de hacer un uso total de SPA, que ha sido empleado en la totalidad de la aplicación salvo para la gestión de usuarios de cara a su creación y su inicio de sesión. De esta forma toda la aplicación funciona de una forma en la que se van mostrando o retirando distintos componentes, desde componentes principales que actúan de módulos principales sobre la página principal, así como componentes más concretos dentro de esos componentes que actúan como módulos. Uno de los principales puntos para hacer uso de componentes y SPA era evitar el routing, es decir, indicar qué páginas se deberían de cargar al introducir cierta URL en el navegador. Por ejemplo, MVC siempre hace uso del routing, y por cada botón se consulta a ese listado de routing, y dependiendo de la URL se carga una página u otro haciendo uso de solicitudes HTTP constantes. Eso hace que el intercambio de datos, tanto con métodos GET o POST se vuelva algo tedioso y complique mucho el poder depurar la herramienta, detectar problemas así como tener que duplicar estructuras. Por ejemplo JavaScript por la parte front, y php por la back. Al tener la aplicación montada como SPA eso deja de ser necesario, no hace que identificar URLs ni qué vistas cargar y luego ver si esa vista está compuesta por componentes o otros elementos para mostrar información. La herramienta cargará pequeñas secciones de la pantalla cargando y actualizando componentes, y cada componentes se encargará de sus propios datos, pudiendo relacionarse, como ya veremos. Otro punto, ya mencionado al detallar qué tecnologías se iban a emplear, es el uso de SignaIR, que con su comunicación bidireccional no hará contantes peticiones HTTP para recuperar información.

A la hora de programar un componente se puede dividir en otros ficheros para tener vista (HTML con código incrustado), lógica (JavaScript o equivalente) y diseño (CSS) separados, pero en la aplicación

desarrollada el uso de CSS ha sido prácticamente nulo, sólo pequeños detalles en colores, y la parte de lógica está totalmente relacionada con la vista, pero estructuralmente bien separada, por lo que se ha decidido mantenerlo junto.

Para crear un componente lo haremos generando un fichero con extensión `.razor`. La parte de arriba será el código HTML, la parte @con será el código en la parte cliente que tendrá parte de lógica relacionada con la vista y solicitará datos a los servicios. Para actuar con código sobre el código HTML se hará uso de las etiquetas `Razor`.

Así es el cuerpo de un componente llamado `MiMenu` que sólo mostrará el texto hora.

```
<p>hola</p>
@con {

}
```

Y para que su contenido sea mostrar en cualquier parte de la aplicación bastará con introducirlo así en la parte HTML. Se mostrará el texto hola. La llamada al componente se puede condicionar a distintos datos, a variables, etc.

```
  <GES_TFG.Pages.MiMenu.MiMenu></GES_TFG.Pages.MiMenu.MiMenu>
```

4.6.2 Enlace de datos en Blazor

Para llevar el control de datos modificados se hará uso del binder, que hace que un dato esté vinculado a una variable, haciendo que si un campo de texto está vinculado a esa variable no será necesario realizar ninguna otra acción, pues automáticamente esa variable ya tendrá enlazado el texto introducido. Es una forma de evitar tener que repetir constante código para asignar valores, permitiendo también que en el momento que se introduzca un dato se desencadenen otros eventos condicionados por el valor de esa variable. De esta forma con simplemente tener configurados los enlaces no es necesario programar lógico que llamen a funciones en base a datos introducidos, ahorrando bastante el código y en eficiencia del mismo.

En caso de que al modificar un dato en un campo de texto se se desee tener más control sobre las acciones desencadenadas se puede ejecutar el clásico onclick, pero en ese caso el bindeo automático en dos direcciones no funcionará pues Blazor tendría que gestionar en paralelo dos acciones sobre el datos, la actualización del enlace y el método desencadenado. Como esta particularidad puede producir incongruencias es necesario decidir si usar el bindeo o asignar el valor introducido con una asignación manual.

Los componentes en Blazor funcionan como pequeños sistemas independientes, pero dependiendo de la complejidad de la aplicación web desarrollada es necesario la comunicación de datos, tanto para pasar información como para evitar que existan dos contextos a la vez accediendo a la misma fuente de datos. Por ejemplo, un contexto accediendo a una tabla y el componente hijo accediendo en paralelo a esos mismos datos.

La forma de pasar información a un componente hijo se realiza de la siguiente forma:

En el componente padre se indica el componente hijo, indicando los datos que se pasará como parámetro. En el ejemplo `Estado`, `authentication`, etc., son parámetros. Y se indica el parámetro y el nombre de la variable que se pasará.

```
<FormularioNuevoTipo Estado="@Estado"
                     authentication="@authentication"
                     tipoSondeoService="@tipoSondeoService"
                     InvocarCambio="@ActualizarListadoTipos"></FormularioNuevoTipo>
```

Para que el componente hijo los pueda procesar tendrán que declararse indicando la etiqueta de que es un parámetro:

```
[Parameter]
public EstadoGenerarTipo Estado { get; set; }

[Parameter]
public AuthenticationStateProvider authentication { get; set; }

[Parameter]
public TipoSondeoService tipoSondeoService { get; set; }

[Parameter]
public Action<List<TipoSondeo>> InvocarCambio { get; set; }
```

Con esta configuración las variables indicadas en el padre se pasarán el componente hijo. Otra forma de pasar datos es pasando los parámetros en cascada.

Pasar parámetros en cascada se usa para compartir un dato directamente entre varios componentes, haciendo que en todos estén referenciados el mismo dato y facilitando el autorefresco en todos los componentes padres si el dato es modificado en el componente padre.

En el ejemplo a continuación se está pasando como parámetro en cascada un listado de entidades participantes:

```
<CascadingValue Value="@this.listaParticipantes">
    <FormularioNuevaEntidadParticipante authentication="@authentication"
                                        listaTipos="@this.listaTipos"
                                        participanteService="@participanteService"
                                        InvocarCambio="@ActualizarListadoParticipantes">
    </FormularioNuevaEntidadParticipante>
    <ListadoEntidadesParticipantes participanteService="@participanteService"
                                   participanteImagenesService="@participanteImagenesService">
    </ListadoEntidadesParticipantes>
</CascadingValue>
```

el dato será declarado en el componente hijo de la siguiente forma:

```
[CascadingParameter]
public List<Participante> listaParticipantes { get; set; }
```

Una vez se tienen los datos, es necesario tener la garantía que la descomposición en componentes de un módulo no haga que la modificación de un dato en un componente no se vea reflejado en otro.

Blazor es capaz de actualizar en ambos sentidos los datos entre componentes al usarse el enlazado de datos, y modificar un dato en un componente debería actualizarse en los demás. Pero la realidad es que no siempre se actualiza correctamente dependiendo la complejidad de los datos y encapsulación de los mismos en los componentes. Por eso es recomendable forzar ese refresco de información para que

la información se actualice en todos los componentes y sea el usuario el que indique cuándo y qué se actualiza y no dejar a Blazor esa decisión.

Un dato tendrá dos formas de actualizarse. Cuando es modificado en un componente padre y ha de informarse a un hijo, o cuando es modificado en un hijo y ha de actualizarse tanto en el componente padre como en otros componentes hijo.

Cuándo y cómo actualizar un componente lo decidirá el programador dependiendo cuándo considera más oportuno realizar tal acción. Para permitir tales acciones se han de configurar así los componentes.

Una variable que se pasa como parámetro al hijo y que tendrá que informarse a otros componentes ha de declararse con una línea de parámetro extra como se muestra en el ejemplo del parámetro `Estado`.

```
[Parameter]
public EstadoGenerarTipo Estado { get; set; }

[Parameter]
public EventCallback<ChangeEventArgs> EstadoChanged { get; set; }
```

Para forzar la comunicación de esa variable modificada al componente padre se realizará con la instrucción `StateHasChanged`, que pondremos en una función concreta:

```
public async Task refrescar()
{
    await EstadoChanged.InvokeAsync();

    //otra opcion es invocar un refresco generalizado es usar:
    await InvokeAsync(StateHasChanged);

}
```

Invocando esta función en el componente hijo se estaría informando a los componentes padre y otros componentes hijo de que esa variable ha sido alterada, por lo que todos los componentes se actualizarían. Realmente en todo momento las variables de datos estarán actualizadas, pero si no se invoca el StateHasChanged las vistas no se refrescarán, lo que generará dudas al usuario.

La forma de forzar un refresco a componentes hijos desde un componente padre es llamar a esa función refrescar que invoca el `StateHasChanged`, y para invocar una función del hijo desde el componente padre es necesario referenciar el componente en el padre:

```
GraficoResultadoFinal RefComponenteGraficoResultadoFinal;
```

```
<GraficoResultadoFinal @ref="RefComponenteGraficoResultadoFinal"
                       EdicionSondeo="@this.EdicionSondeo"
                       MedidasEdicionSondeo="@this.MedidasEdicionSondeo">
</GraficoResultadoFinal>
```

Con el componente referenciado ya se pueden invocar funciones que estén presentes en el componente hijo, por lo que invocando la función que incluye el `StateHasChanged` se estará forzando el refresco.

```
RefComponenteGraficoResultadoFinal.NombreDeLaFuncion();
```

4.7 Uso de estados en el proyecto

Un componente tendrá distintos elementos, y los datos mostrados estarán enlazados a variables con distintos datos. A la vez se estarán editando campos que tendrán distintas acciones sobre la base de datos, como añadir registros o modificarlos. La idea de trabajar con estados es tener un objeto que encapsule todos los elementos de la pantalla, tanto los visibles como los no visibles. De esta forma si se pasa de un componente a otro, se podrá pasar con un único parámetro. Si se desea guardar información con el ORM no habrá que montar nada, se pasará el objeto que ya contendrá entidad que referencia a esas tablas, por lo que el único trabajo será realizar acciones ADD o UPATE con el ORM.

4.8 Uso de estados en las vistas

Cada componente principal, lo que hemos llamado módulos y que agrupa funcionalidades principales, tendrá un estado asociado que se generará al cargar el componente. Desde ese momento todo elemento alterado en la vista, como puede ser un campo de texto, quedará enlazado a dicho estado.

Por ejemplo, al cargar el componente de añadir registros de forma manual se inicializará el siguiente estado, que es instanciar la clase `EstadoCargaManualCompleto`:

```
1     public class EstadoCargaManualCompleto : Estado
2     {
3         public TipoSondeo TipoSondeo { get; set; }
4         public EdicionSondeo EdicionSondeo { get; set; }
5         public MedidasEdicionSondeo MedidaMostrada { get; set; }
6         public Fuente Fuente { get; set; }
7
8
9
10        public DateTime? Fecha { get; set; }
11        public bool ModoModificar { get; set; }
12
13        public RegistroSondeo RegistroSondeo { get; set; }
14
15        public List<TipoSondeo> ListaTiposSondeo;
16        public List<EdicionSondeo> ListaEdicionesSondeo;
17        public List<Participante> ListaParticipantesEdicion;
18        public List<MedidasEdicionSondeo> ListaMedidas;
19        public List<MedidasEdicionSondeoRelacion> ListaRelaciones;
20        public List<Fuente> ListaFuentes;
21    }
```

Los campos superiores son elementos de la vista, los de abajo corresponden a listados que se cargan o al inicializar el componente o tal como se van seleccionando elementos, y `RegistroSondeo` representa el objeto de la entidad `RegistroSondeo`, que es lo que se guardará en base de datos cuando el usuario pulse Guardar.

De esta forma aunque se cierre el componente es posible, si se desea, poder mantener estos datos en memoria y que no dependan de tener abierto o no la vista y componente. Es muy útil de cata al intercambio de información entre componentes y tener un objeto idéntico al que se pasará al ORM para modificar la base de datos.

4.9 Uso de estados para guardar información

Para guardar los datos empleando estados es muy sencillo, ya que no hace falta coger todos los datos, montar el objeto, y pasarlo al servicio para que realice la acción deseada. Es tan sencillo como pasar del estado el campo que corresponde a la tabla para que el ORM lo procese. En el ejemplo es `RegistroSondeo`. Si los distintos elementos de la vista tienen acciones para validar que los datos son correctos o no, se puede habilitar o deshabilitar el botón de guardar. De esta forma se tendría la garantía que lo que se envía al ORM es el objeto completamente correcto listo para actuar sobre la base de datos.

Al pulsar el botón guardar se llama al servicio con el campo `RegistroSondeo`:

```
1 await Task.Run(() => registroSondeoService.GuardarRegistroSondeoBisAsync(Estado.RegistroSondeo));
```

Y el servicio lo único que hace es invocar `Update`, que el ORM identificará para crear o actualizar un registro en base a su clave primaria

```
1 public async Task GuardarRegistroSondeoBisAsync(RegistroSondeo rs)
2 {
3   _applicationDbContext.RegistroSondeos.Update(rs);
4   await _applicationDbContext.SaveChangesAsync();
5 }
```

4.10 Uso de estados para editar información

Otra de las ventajas de trabajar con estados es que si se desea editar en vez de crear, sería tan sencillo como cargar la información del registro a modificar en el estado, y al cargar el componente el enlazado automático mostraría en pantalla todos los datos del registro existente. Y al pulsar sobre guardar, como el ID asignado correspondería a un registro existente lo que haría es actualizar los datos en vez de duplicarlos.

También teniendo en cuenta el ID es posible alterar la vista para mostrar o editar ciertos elementos dependiendo de si es una nueva creación o una edición.

4.11 Detalle desarrollo de los módulos

En la siguiente sección se detallarán detalles concretos de desarrollo de algunas funciones concretas que para entender bien el funcionamiento han de explicarse de una forma más detallada.

4.11.1 Detalle de los módulos de carga manual y gestión de formularios

Una de las principales complejidades al afrontar el proyecto era que los datos que se tenían que introducir no iban a ser algo estático, sino que se configuraría por los usuarios, y esos datos configurados deberían almacenarse de una forma que pudieran posteriormente solicitarse y controlarse por el sistema. Por ejemplo, cargando cierta edición de un tipo concreto se nos muestra la siguiente pantalla (figura 4.16).

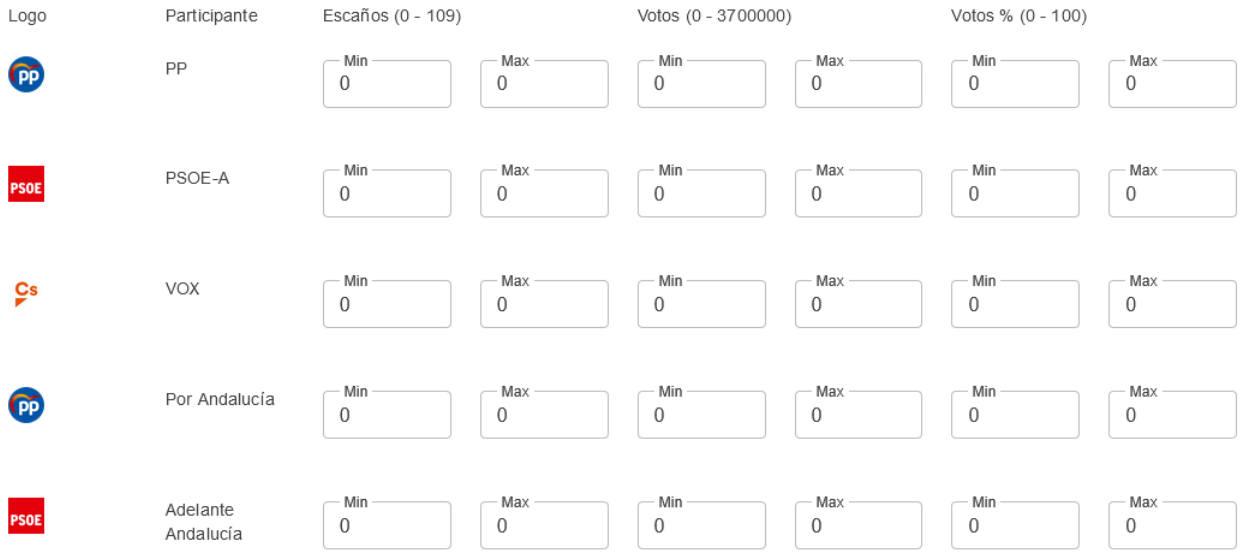

Figura 4.16: Ejemplo de pantalla para introducir información desde alguna fuente

Para cargar este formulario se hará con dos bucles anidados, uno recorriendo las medidas y otro las entidades participantes, de forma que cada campo tenga asignados, en tiempo de ejecución, un ID de la medida, y un ID de la entidad participante.

No es posible hacer un enlace directo entre campo y dato en variable porque no se trabajan con datos fijos, pero teniendo ambos ID y un campo pulsado o modificado, se declaran eventos que llaman a funciones y que modifican las estructuras buscando las posiciones en base a esos id. Con la estructura actualizada sólo será necesario invocar un StateHasChanged para que todos los componentes implicados refresquen con el dato. Cualquier dato modificado en el formulario lo que hace internamente es modificar las estructuras que guardan los datos de medidas y entidades participantes. De esta forma no se necesitan identificar con nombres concretos los textfield, pues dada la interactividad necesaria sería inviable.

4.11.2 Detalle de la carga desde páginas web y gestión de formularios

En esta pantalla se cargará de forma automática los datos desde una URL. Se indicará una URL y el sistema procesará el contenido del DOM en búsqueda de tablas. Las tablas en HTML se componen de las siguientes etiquetas:

```
tabla - tr - th - td
```

Se buscarán las tablas y todas las que tengan menos de 20 filas se cargarán en memoria y se mostrarán los campos leídos de la siguiente forma (figura 4.17)

Figura 4.17: Ejemplo de pantalla tras analizar una página web buscando información

Al tener en memoria las tablas leídas, el usuario al pulsar los iconos de la papelera lo que hará es retirar esas filas de la memoria, por lo que el enlazado de datos automáticamente también las retirará de la vista.

Se tendrá que seleccionar manualmente a qué medidas y entidades participantes pertenece cada fila y columna, que al procesar se identificará si están todos además de no estar repetidos.

Aunque se pueden editar manualmente los datos, se podrá usar una funcionalidad que limpiará los datos

En cada fila el usuario podrá indicar una pseudo-expresión regular para limpiarlos. Indicará con A y B para identificar en qué lado, derecha o izquierda, se encuentra el valor Alto y Bajo del rango, y entre medias los caracteres que separan los datos. El resto de datos no son necesarios de indicar porque no son numéricos y se ignoran. En valores negativos sí será necesaria la participación del usuario ya que puede que el símbolo - no siempre se procese correctamente.

En el siguiente ejemplo tenemos tres medidas (figura 4.18). La primera no tiene rango, por lo que el sistema la podrá limpiar sin ayuda del usuario. La segunda está separada por un guión, por lo que se indicará B-A. Si fuera necesario se podría procesar invirtiendo los valores indicando que la parte alta está a la izquierda y la baja a la derecha. En ese caso el sistema invertiría los valores. En el tercer caso se indica sólo el separador, la coma, porque los paréntesis al no ser numéricos el sistema los ignora.

Figura 4.18: Ejemplo de datos pendientes de procesarse para ser limpiados

Una vez escritos, se pulsa sobre LIMPIAR (figura 4.19) y el sistema los limpia.

Figura 4.19: Botón para desencadenar el proceso de limpia

También dependerá del usuario decidir si el valor necesita ser procesado o no. Por ejemplo, (-1,1) puede tratarse de un rango -1 a 1 o ser un valor negativo de -1,1. Eso queda a elección del usuario y el sistema, si los datos en los campos son correctos, lo procesará (figura 4.20).

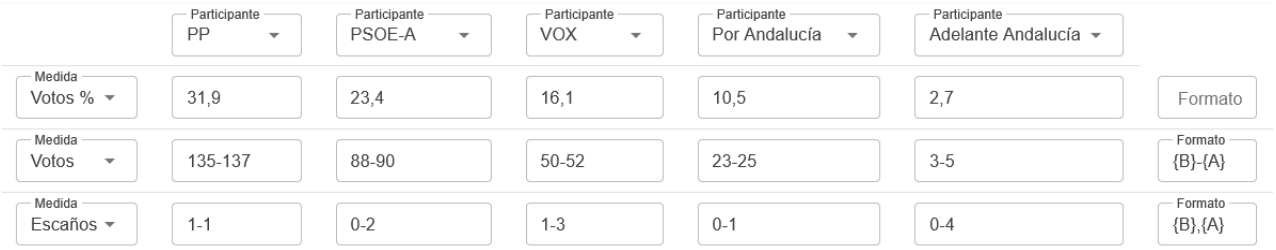

Figura 4.20: Ejemplo de datos limpios

4.11.3 Sincronización de medidas relacionadas

Al generar ediciones se indican entidades participantes y medidas. Estas medidas muchas veces tienen dos valores, una que es un dato numérico y otra que es un dato relacionado a ese dato numérico que representa un porcentaje. Por eso al agregar una medida se puede agregar otra que esté relacionada, y esta relación tiene sentido de cara a auto-complementar dos datos introduciendo sólo uno.

En la pantalla de generar edición, en el apartado de medidas, aparece el siguiente elemento (figura 4.21):

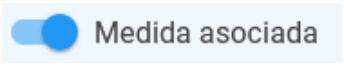

Figura 4.21: Botón para asociar medidas

Cuando es seleccionado, la medida numérica introducida generará otra simultáneamente con un valor porcentual de 0 a 100 y ambas medidas quedarán relacionadas añadiendo un registro a la tabla `MedidasEdicionSondeoRelacion`.

En la pantalla de introducir registro de estimaciones, sondeos, etc, de forma manual aparecerá el siguiente elemento (figura 4.22) en pantalla:

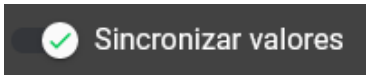

Figura 4.22: Botón para permitir actualizar los valores relacionados

Mientras que este elemento esté seleccionada siempre que se introduzca un valor en una medida se comprobará si aparece en la tabla de relaciones, y en caso de aparecer actualizará en tiempo real el campo de la medida con la que está relacionada. De forma que con sólo modificar un campo se irán rellenando dos, pues un valor tendrá una relación directa y se evita la repetición de trabajo.

Las vistas Razor tienen ciertos elementos controlados con el siguiente condicional:

```
@if (recalcularRelaciones)
{...} else {...}
```

La variable `recalcularRelaciones` está enlazada con el elemento de `Sincronizar Valores` para determinar si el condicional se cumple o no, si se cumple se cargan en el DOM unas etiquetas que gestionan las relaciones, y si no se cumple se cargan etiquetas que no gestionan el DOM.

En el caso de gestionar las relaciones entre medidas están las siguientes etiquetas, que son componentes Mudblazor que enriquecen los `TextField` básicos de Blazor:

```
    <MudTextField T="float"
                  Value="@d.ValorBajo"
                  @onblur="(() => this.refrecar())"
                  ValueChanged="@((float valorBajo)
                                  => SincronizarValorVinculadoRangoBajo(valorBajo, d, p.Id))"
                  Label="Min"
                  HelperText="Inserte"
                  HelperTextOnFocus="true"
                  Variant="Variant.Outlined"
                  Margin="Margin.Dense" />
```

`ValueChanged` detectará cuando el valor del campo ha sido modificado, y en ese caso llama a la función `SincronizarValorVinculadoRangoBajo` para que detecte la medida relacionada, calcule el valor proporcional y actualice el campo de la medida relacionada.

`Onblur` es un evento que se desencadena cuando el textfield pierde el foco, y lo que hace en ese momento es forzar un refresco de la vista, para que se muestren en la vista los datos que han sido editados en las variables que gestionan los datos.

```
    <MudTextField T="float"
                  @bind-Value="d.ValorBajo"
                  Label="Min"
                  HelperText="Inserte"
                  HelperTextOnFocus="true"
                  Variant="Variant.Outlined"
                  Margin="Margin.Dense" />
```

Por el contrario si no hay relación de medidas seleccionada el dato introducido en el textfield está directamente enlazado con @bind-value, por lo que no es necesario refrescar nada porque el dato ya ha sido modificado por el usuario.

No se pueden emplear a la vez `@bind-Value` y `ValueChanged` porque ambos son enlaces directos con datos, y cada elemento sólo puede gestionar un único hilo para evitar problema de inconsistencia de datos. De ahí que se modifiquen los datos de las variables, y una vez esa tarea ha terminado se llame a refrescar para forzar que los datos sean mostrados y el enlace se mantenga correcto.

4.11.4 Gráficos y tablas del Visualizador

En el visualizador de cruzan todos los datos introducidos en el sistema para mostrar información visual en modo de gráficos y en tablas que resumen la fiabilidad de las distintas fuentes.

Los gráficos empleados son usando una librería llamada Blazor `ApexCharts` [28] que adapta a Blazor la librería `ApexCharts` [29]. Se usa esta librería para evitar en lo máximo posible usar `JavaScript` y que usando los conocimientos de la sintaxis de .NET y las vistas Razor se puedan generar atractivos gráficos y que sea la librería la encargada de gestionar la conversión a JavaScript.

4.11.4.1 Generación de gráficos tipo Line

Los gráficos tipo Line (figura 4.23) con la librería Blazor `ApexCharts` se crean usando el componente `ApexChart` y añadiendo un componente `ApexPointSeries` por cada línea que se quiera mostrar.

En los gráficos `Line` de la actual herramienta online cada línea representa una entidad participante o fuente, dependiendo del gráfico, y cada una de esas líneas se compone de fechas. El resultado sería como el mostrado en la siguiente imagen:

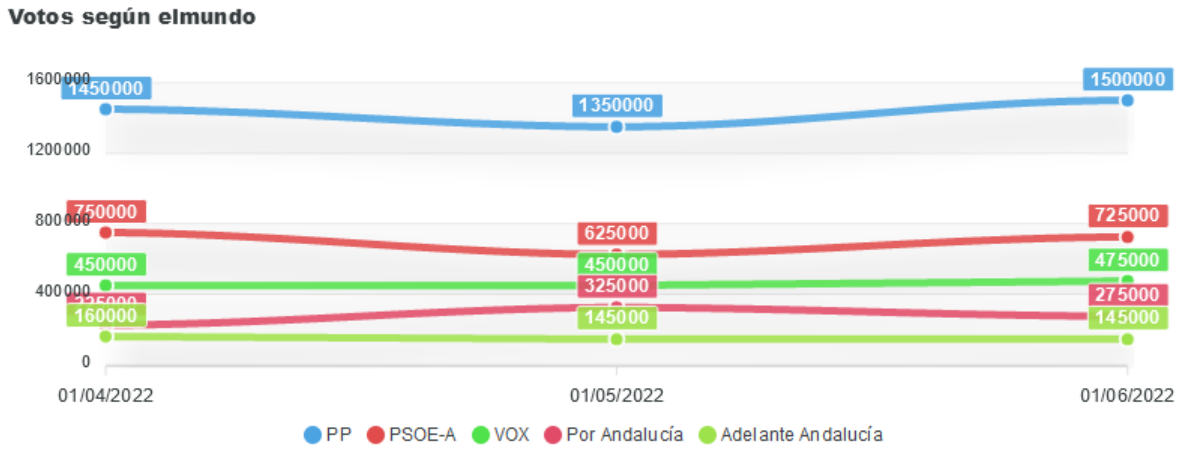

Figura 4.23: Ejemplo de un gráfico tipo line

El gráfico mostrado se genera con el siguiente código:

```
<ApexChart TItem="RegistroLineApex"
    Title=@eltitulo
    @ref=refGrafico
    Width="800"
    Height="300"
    Options="@opciones">

     @foreach (apexLineParticipante p in datasets)
     {
         <ApexPointSeries TItem="RegistroLineApex"
              Items=@p.Registros
              Name=@p.Participante
              SeriesType="SeriesType.Line"
              XValue="@(x => x.Fecha)"
              YValue="@(x => x.Valor)"
              OrderBy="x=>x.X"
              ShowDataLabels />
     }
```

```
</ApexChart>
```

`TItem` identifica los tipos de datos de los que está compuesto el gráfico, los puntos en pantalla. Que los agrupamos con una clase con una fecha y un dato decimal. Por comodidad para trabajar con la librería la fecha se guarda como cadena de texto, pero para ciertas tareas, como ordenar los datos, se convertirá a un tipo `DateTime`.

```
public class RegistroLineApex
{
    public string Fecha { get; set; }
    public decimal Valor { get; set; }
}
```

Cada línea se representará con un `ApexPointSeries`, que es un listado de puntos `RegistroLineApex`, por lo que por cada línea se generará un componente `ApexPointSeries` que está compuesto de diferentes `RegistroLineApex`.

```
public class apexLineParticipante
{
    public string Participante { get; set; }
    public int ParticipanteId { get; set; }
    public List<RegistroLineApex> Registros { get; set; }
}
```

Con el parámetro `XValue` identificamos la línea horizontal, que son las fechas; y con `YValue` los valores numéricos, que son los valores de los registros correspondientes a entidades participantes.

Configurando correctamente dentro de estas estructuras todos los datos almacenados en el sistema pueden realizarse todo tipo de gráficos para mostrar cualquier tipo de información que sea posible calcularse con la información disponible.

4.11.4.2 Ajuste valores en gráficos tipo Line

Al comparar datos de distintas fuentes y cruzarlos en un mismo gráfico (figura 4.24) uno de los ejes no coincidirá, y todos los data set han de tener el mismo número de parámetros para una correcta configuración y funcionamiento. Un ejemplo sería una fuente A con 5 registros (5 fechas) y otra fuente B con 3, en las que sólo coinciden 2 fechas. Si vemos los distintos registros/fechas tendríamos lo siguiente:

Fuente A	Fecha 1	Fecha 2	Fecha 4	Fecha 5	Fecha 6
Dato	4	5	4	5	3

Fuente B	Fecha 2	Fecha 3	Fecha 5
Dato	7	6	7

Figura 4.24: Datos disponibles en cada fuente

Dato que la librería de gráficos requiere el mismo número de elementos pondremos ordenadas las fechas para ver cómo se podría solventar el problema y poder mostrar distintas fuentes en un mismo gráfico para que puedan ser comparadas.

Fuente A	Fecha 1	Fecha 2	*	Fecha 4	Fecha 5	Fecha 6
Dato	4	5	*	4	5	3

Fuente B	*	Fecha 2	Fecha 3	*	Fecha 5	*
Dato	*	7	6	*	7	*

Figura 4.25: Los datos ordenados por fecha

Como se puede ve en la figura 4.25, sólo dos de los registros son del mismo día, mientas que el resto son de fechas distintas. Es la situación más habitual, pues las distintas fuentes no publicarán los datos el mismo día, sino cuando vean conveniente.

La forma planteada de solventar el problema para que todo cuadre es ajustar todas las fuentes para que tengan la misma cantidad de registros en sus dataset y así todas puedan coincidir. Por lo tanto, todas las fuentes deberán tener la misma cantidad de registros y al generar el dataset, explicado anteriormente, generando registros en esas fechas en las que no hay datos en los que mostrar información. Tras esto se plantea la pregunta de cómo rellenar esos datos que no tienen valores, pues modificarlos arbitrariamente haría que el valor informativo de la gráfica desapareciese.

La forma más eficiente es ajustar los valor en los registros sin datos para que extiendan los más próximos, generando continuidad y una nula alteración de los datos. Para el ajuste primero se generarán todos los registros para todas las fechas, asignando un valor NULL a los que no tengan valores existentes. Una vez se tienen las estructuras se realizará una doble pasada, primero ascendente, y después descendente. Lo que se hará es detectar los valores nulos, y al detectarlo se asignará el valor del registro anterior. Al realizar una doble pasada en ambos sentidos se consigue que con la existencia de un único registro, que siempre va a existir, se pueda ajustar completamente todos los datos. La tabla de ejemplo quedaría así ajustada.

Fuente A	Fecha 1	Fecha 2	*	Fecha 4	Fecha 5	Fecha 6
Dato	4	5	5	4	5	3

Fuente B	*	Fecha 2	Fecha 3	*	Fecha 5	*
Dato	7	7	6	6	7	7

Figura 4.26: Los datos ordenados por fechan ajustados

En el ejemplo anterior (figura 4.26) en la pasada ascendente, en la fuente A se pasa el 5 de la fecha 2 a la fecha 3; y en la fuente B se pasa el 6 de la fecha 3 a la fecha 4 y el 7 de la fecha 5 a la fecha 6. En la pasada descendente en la fuente B se pasa el 7 de la fecha 2 a la fecha 7. Con este ajuste todos los datos están corregidos para poder ser mostrados sin que se generen huevos o grandes picos.

En la siguiente se puede ver en funcionamiento. La fuente seleccionada tiene 3 registros en 3 fechas (figura 4.27), pero entre todas las fuentes hay 5 fechas en total. Por lo tanto ha de ajustarse.

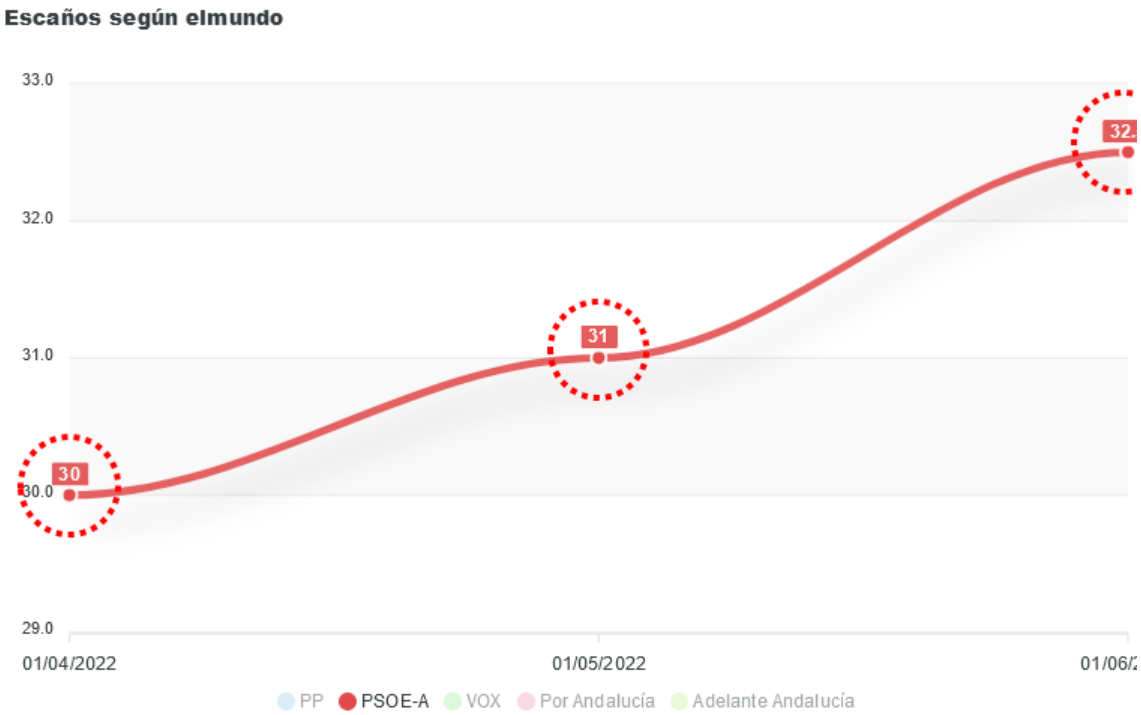

Figura 4.27: Gráfico line con datos que se ajustarán cuando se comparen con otras fuentes

Los círculos rojos (figura 4.28) marcan las fechas existentes en esa fuente y las fechas rojas el ajuste realizado. Sin esos registros "simulados" la librería fallaría al no coincidir los valores en ambos ejes.

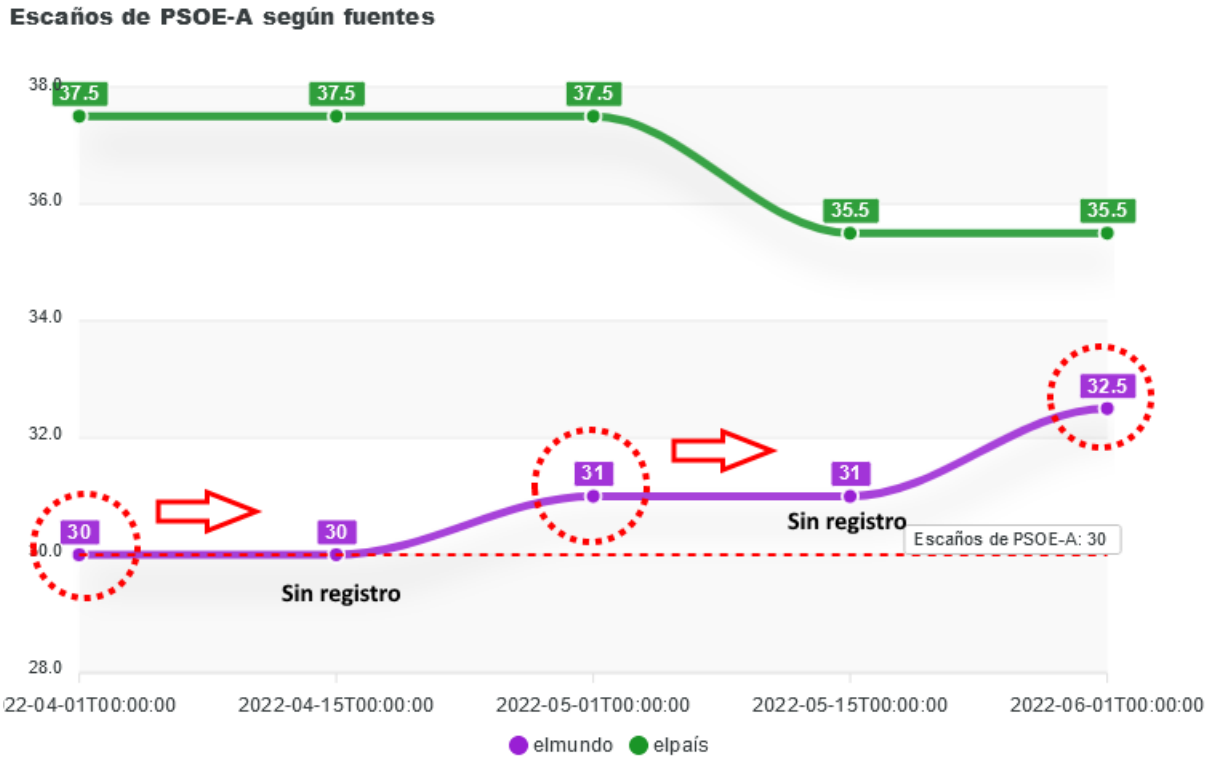

Figura 4.28: Gráfico line ajustado

4.11.4.3 Generación de gráficos tipo Donut

Los gráficos tipo Donut (figura 4.29) generarán círculos con información. Su generación es bastante simular a los tipo Line. Un primer componente para el gráfico y luego otro por cada circunferencia. En los empleados en aplicación online serán todos con un único dataset de datos representando registros concretos de datos que complementen a los datos mostrados en los gráficos Line.

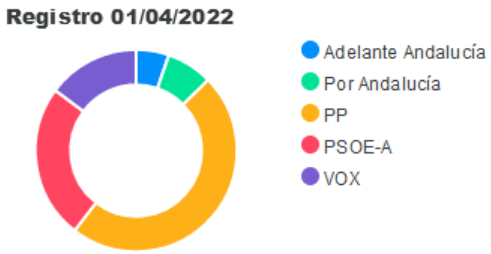

Figura 4.29: Muestra de un gráfico tipo donut

`ApexChart` es el componente principal y `ApexPointSeries` indicará cada circunferencia, que se compondrá de datos tipo `ValorRegistroDonutApex`, que agrupa cada dato de una entidad participante.

```
<ApexChart TItem="ValorRegistroDonutApex"
   Title="@this.graficoRegistro.NombreRegistro"
   Width="350"
   Height="350"
   @ref=refGrafico>
        <ApexPointSeries TItem="ValorRegistroDonutApex"
             Items="this.graficoRegistro.ValoresParticipantes"
             SeriesType="SeriesType.Donut"
             Name="Gross Value"
             XValue="@(e => e.NombreParticipante)"
             YValue="@(x => x.Valor)"
             OrderBy="x=>x.X" />
</ApexChart>
```

```
public class ValorRegistroDonutApex
{
    public string NombreParticipante { get; set; }
    public decimal Valor { get; set; }
}
```

4.11.4.4 Cálculos generación fiabilidad fuentes

Con todos los datos del sistema se mostrarán unas tablas (ejemplo en la tabla 4.22) con información que determinarán la fiabilidad de cada fuente y cómo sus predicciones y estimaciones se acercan o alejan de los datos finales resultantes.

La fiabilidad se calculará en lo cerca que está cada dato propuesto por las fuentes respecto al dato final resultante. Y esa diferencia entre el dato proporcionado y el resultado final se normalizará sobre 100. De esta forma que a cada dato se asignará un valor entre 0 y 100, siendo el más cercano a 0 el que más cerca del resultado final haya estado.

Se generará este valor para cada registro basándose en los rangos altos y bajos que las medidas, para generar correctamente el valor en base a 100, y dependiendo el resultado que se quiera mostrar se harán medias entre fuentes y ediciones.

Estos datos no se guardarán en memoria, pues pueden variar con cada nuevo registro introducido, por lo que siempre se tendrán que calcular en tiempo de ejecución cuando se acceda al módulo visualizador.

`Fiabilidad Medida`	Porcentaje de acierto de fuente/entidad participante en la medida seleccionada.
`Fiabilidad Todo`	Porcentaje de acierto de fuente/entidad participante en todas las medidas.
`Resumen`	Media total de fiabilidad por fuente.

Tabla 4.22: Tablas que mostrarán la fiabilidad de las fuentes

Para generar estos datos se calculará la fiabilidad en cada registro siguiendo los siguientes pasos

1. El rango de la medida se normaliza a 0. Si el rango es -10 a 10, se sumará la parte negativa para dejarlo como 0 a 20.
2. Se calcula la fiabilidad.
   (a) Si el valor de la fuente es igual al resultado final la desviación es 0
   (b) Si no es igual se calcula la diferencia como valor absoluto entre el dato de la fuente y el resultado
   (c) La diferencia no indica nada, pues variará dependiendo de los rangos de la fuente.
   (d) Con una sencilla regla de 3 se adapta el rango de la medida a 0-100. El rango bajo es el 0, y el rango alto el 100. En base a esta relación se saca el porcentaje de fiabilidad de la diferencia entre el dato proporcionado por la fuente y el resultado final.
3. Los datos calculados de fiabilidad se guardan en una estructura con más información relacionada con tipo, edición, etc.
4. Estos cálculos de proximidad de acierto se realizan tanto en el valor alto como en el bajo.
5. Con estos datos ya calculados se realizarán medias para sacar las conclusiones de la tabla anterior. También los datos altos y bajo se tomarán como una media y no se mostrarán por separado.

El dato de la pestaña `Resumen` es, junto a los datos de los distintos gráficos, el objetivo final buscado al realizar la aplicación, ya que ofrece el resultado de fiabilidad en base a todos los registros proporcionados por los usuarios.

4.12 Localización de los textos del proyecto

En plena implementación de la aplicación web surgió la necesidad de internacionalizar la aplicación web, que fuera posible ser mostrada en distintos idiomas y que aunque no se añadan otras localizaciones, sí que la aplicación esté preparada para que de forma sencilla se puedan añadir distintas localizaciones.

4.12.1 Configuración Localización en proyecto

La localización en un proyecto .NET se configura en la clase `Startup`, en concreto en los métodos `ConfigureServices` y `Configure`.

En `ConfigureServices` indicaremos la ruta del proyecto en el que se incluirán los ficheros de recursos, que será `Resources`. Con la clase `CultureInfo` se indicarán las localizaciones que se añadirán, en el caso del proyecto se añadirá castellano e inglés. También se tiene que indicare la localización por defecto, por si el navegador del usuario que accede a la aplicación web no está configurada y sea la que se le muestre.

```
public void ConfigureServices(IServiceCollection services)
{
  ...
  services.AddLocalization(options => options.ResourcesPath = "Resources");
  var supportedCultures = new List<CultureInfo> { new CultureInfo("es"), new CultureInfo("en") };
  services.Configure<RequestLocalizationOptions>(options =>
  {
      options.DefaultRequestCulture = new Microsoft.AspNetCore.Localization.RequestCulture("es");
      options.SupportedUICultures = supportedCultures;
  });
}
```

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
  app.UseRequestLocalization();
}
```

4.12.2 Configuración Localización Recursos

Un recurso es un fichero .NET que se empleará como almacenes de valores. En la localización de la aplicación web se usará para almacenar los textos en cada idioma, de esta forma deberá haber un fichero de recursos por cada idioma disponible en la aplicación. Dado que el proyecto está organizado en componentes, para hacer que todo esté más organizado y que sea fácil de localizar cada texto, los recursos se dividirán por componente. Por lo que será necesario que exista un fichero de recursos por componente y por lengua localizada.

4.12.2.1 Configuración Localización en proyecto

Una vez se tienen configuradas qué localizaciones se usarán en el proyecto y están creadas todos los recursos, así como ordenadas respetando el orden de los componentes (figura 4.31), las variables de los recursos se usan inyectando IStringLocalizer de la siguiente manera:

Se inyecta usando IStringLocalizer pasando como tipo el componente del que se quieren tomar los recursos

```
@inject IStringLocalizer<Index> loc
```

y una vez inyectado se tendrá que hacer referencia a la variable del recurso a mostrar, que dependiendo de nuestro idioma configurado en el navegador se usará una localización u otra

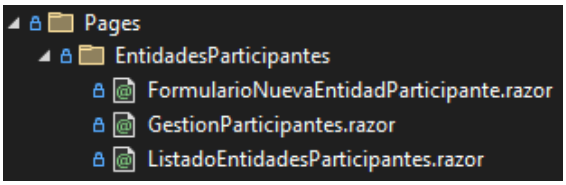

Figura 4.30: Ejemplo componentes Entidades Participantes

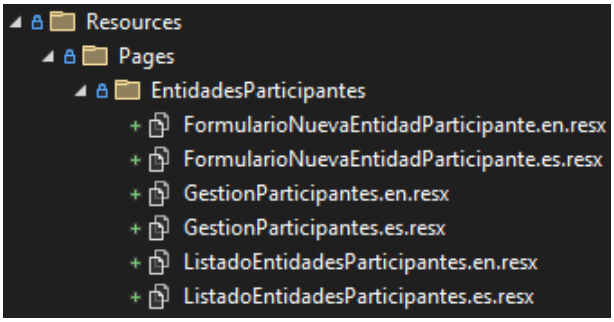

Figura 4.31: Ejemplo Recursos Entidades Participantes

```
1    <p>@loc["RECURSO1"]</p>
```

Por ejemplo, dependiendo del valor de RECURSO1 (figura 4.35) en el recurso de la lengua de nuestro navegador se mostrará un texto u otro.

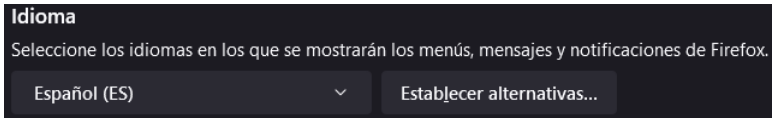

Figura 4.32: Firefox configurado en castellano. La aplicación se mostrará en castellano

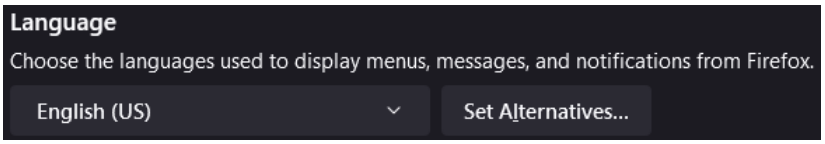

Figura 4.33: Firefox configurado en inglés. La aplicación se mostrará en inglés

El idioma mostrado se basa en el idioma que se tenga configurado en el navegador de internet (figura 4.33). Por ejemplo, en firefox se puede configurar desde la configuración

Name	Value
RECURSO1	TEXTO_RECURSO

Figura 4.34: Recurso del componente en castellano

Name	Value
RECURSO1	RESOURCE_TEXT

Figura 4.35: Recurso del componente en inglés

4.12.2.2 Configuración Nuevo Idioma

Tal como se ha planteado la localización de forma ampliable, si en el futuro se quisiera añadir otra localización, por ejemplo francés, sería algo tan rápido y sencillo como indicar francés en la clase inicial

`Startup`, y añadir los recursos de los componentes que se desean mostrar traducidos. Todos los recursos de componentes no creados se mostrarán con el recurso por defecto, que en el caso de este aplicación web es castellano.

```
1   public void ConfigureServices(IServiceCollection services)
2   {
3     ...
4     var supportedCultures = new List<CultureInfo> { new CultureInfo("es"),
5                                                     new CultureInfo("en"),
6                                                     new CultureInfo("fr") };
7     ...
8   }
```

4.13 Pruebas de integración

Las pruebas de integración [30] son las pruebas que se centran en el funcionamiento de bloques compuestos de varias partes de código o ya funcionalidades concreta ya completas, mientras que las pruebas unitarias [31] son las que se centran en probar cada pequeña función independientemente del uso que se le vaya a dar. Mientras que una prueba de integración puede contener muchas funciones que van pasándose los datos, las unitarias probarán cada una de esas funcionas de forma independiente, así primero se testea el buen funcionamiento de cada pequeña pieza del software y luego se prueban combinadas.

Para este proyecto sólo se mostrarán las pruebas unitarias, ya que cada prueba unitaria no tiene detrás gran combinación de funciones, por lo que en caso de detectar algún error en las pruebas de integración el programador realizará pruebas unitarias para identificar el problema y solucionarlo.

En todas las pruebas que aparecen en la memoria se mostrarán como superadas satisfactoriamente puesto que en caso de haberse detectado algún error se ha procedido a su corrección.

4.13.1 Pruebas de integración en el módulo Mi Menú

Las pruebas referentes al módulo mi menú están presentes en la tabla 4.23.

Acción	Esperado	Resultado
Se pulsa nuevo tipo	Se carga el módulo de generar tipos	El módulo se carga correctamente
Se pulsa nueva edición	Se carga el módulo de generar ediciones	El módulo se carga correctamente
Se pulsa nuevo registro	Se carga el módulo de generar registros manualmente	El módulo se carga correctamente
Se pulsa Editar tipo sin seleccionar nada	Debe advertir de que se ha de seleccionar un tipo	Aparece un aviso como es esperado
Se pulsa Editar tipo seleccionando un tipo	Deberá cargarse el módulo de tipos con los datos para poder actualizarse	Se carga el componente de crear tipo configurado para editar el tipo seleccionado
Se pulsa Editar edición sin seleccionar nada	Debe advertir de que se ha de seleccionar una edición	Aparece un mensaje al darse tal situación
Se pulsa Editar edición seleccionando una edición	Deberá cargarse el módulo de ediciones con la edición seleccionada para poder modificarse	Se carga correctamente el módulo de ediciones con los datos de la edición seleccionada
Se pulsa Nuevo registro sin seleccionar nada	Debe advertir de que se ha de seleccionar un registro	Aparece un mensaje al darse tal situación
Se pulsa Nuevo registro seleccionando un registro	Debe cargarse el módulo de registros manuales con el registro actual seleccionado, con sus medidas y entidades participantes configuradas	Se carga correctamente el módulo de registros manuales con los datos del registro seleccionado

Tabla 4.23: Pruebas de integración del módulo Mi menú

4.13.2 Pruebas de integración en el módulo Tipos

En estas pruebas se probará la creación de tipos y algunas situaciones en las que podrían guardarse datos generando problemas si no son controlados correctamente. Las pruebas referentes al módulo tipos están presentes en la tabla 4.24.

Acción	Esperado	Resultado
Crear tipo sin indicar ningún dato	No debería hacer nada	No hace nada, pero tampoco retira cualquier dato que esté en los campos no obligatorios
Crear tipo indicando correctamente los campos	Se debería crear el tipo, aparecer en el listado y empezar a poder ser usado en otras partes de la aplicación web	Se crea el tipo, se refresca el listado con el nuevo tipo y si se entra en el módulo de ediciones puede seleccionarse para crear ediciones sobre el nuevo tipo creado

Tabla 4.24: Pruebas de integración del módulo Tipos

4.13.3 Pruebas de integración en el módulo Entidades Participantes

En estas pruebas se probará la creación de entidades participantes y se combinarán distintas situaciones para comprobar que están controladas y evitar un mal funcionamiento de la aplicación web. Las pruebas referentes al módulo entidades participantes están presentes en la tabla 4.25.

4.13.4 Pruebas de integración en el módulo Ediciones

Estas pruebas son importantes ya que el módulo ofrece distintas formas de configuración por lo que es necesario que las distintas modalidades, inserciones, importaciones e interacciones con los elementos de

Acción	Esperado	Resultado
Pulsar crear sin seleccionar tipo ni indicar nombre	No debería hacer nada	No hace nada y la entidad participante no es creada
Pulsar crear seleccionando tipo pero sin poner nombre	No debería hacer nada	No hace nada y la entidad participante no es creada
Pulsar crear seleccionando tipo, indicando nombre pero no color	No debería afectar a la creación	La entidad participante se crea correctamente
Asociar imagen	La imagen seleccionada debería mostrarse junto al participante en el listado y en otras partes de la aplicación web	La imagen se adjunto correctamente siempre que esté por debajo de ciertas dimensiones y tamaño

Tabla 4.25: Pruebas de integración del módulo Entidades Participantes

la pantalla funcionen correctamente. Las pruebas referentes al módulo ediciones están presentes en la tabla 4.26.

Acción	Esperado	Resultado
Selecciona un tipo	Se deben cargar los componentes para configurar medidas y entidades participantes	Se cargan correctamente
Seleccionar un tipo habiendo seleccionado otro previamente	Se deben actualizar los componentes para que por ejemplo las entidades participantes sean las del tipo seleccionado	Se resetea la información y se carga con los datos correspondientes al nuevo tipo seleccionado
Arrastrar entidades participantes	Mover entidades participantes en cualquier dirección para configurar manualmente las entidades participantes	Se pueden seleccionar correctamente arrastrándolos con el puntero de un cajón a otro para realizar la selección de una forma interactiva
Importar entidades participantes	Al seleccionar una edición del tipo actual se importará la configuración presente en esa edición de entidades participantes	Se carga perfectamente la configuración de entidades participantes al seleccionar edición
Insertar medida sin nombre	No debería realizarse la acción	No se inserta y se mantienen los datos provisionales
Insertar medida con nombre	Se debería insertar correctamente	Se inserta correctamente
Insertar medida porcentaje	El comportamiento debería ser igual que con la numérica pero con la configuración de los porcentajes	Se inserta correctamente.
Insertar medida asociada	Al insertar una medida con otra asociada debería generar dos medidas insertadas, una numérica y otra de porcentaje	Se insertan ambas correctamente
Importar medidas	Se selecciona otra edición del mismo tipo y se precargan las medidas importadas	Las medidas de la edición seleccionada se cargan perfectamente
Modificar rangos de medidas importadas	Permitir que se modifiquen los rangos de las medidas importadas	Se pueden modificar correctamente
Pulsar en guardar edición	La edición se guarda y ya puede empezar a introducirse registros por parte del resto de usuarios	Tras pulsar a guardar, si se han configurado medidas y entidades participantes, se carga perfectamente la nueva edición en el sistema

Tabla 4.26: Pruebas de integración del módulo Ediciones

4.13.5 Pruebas de integración en el módulo Usuarios

Se probará la interacción con usuarios pendientes de registrar, su validación como usuarios aptos y la posterior gestión de rangos a los usuarios aceptados y validados. Las pruebas referentes al módulo usuarios están presentes en la tabla 4.27.

Acción	Esperado	Resultado
Registrar un nuevo usuario	Debe aparecer en el listado de usuarios pendientes de confirmar	Aparece en dicho listado con un botón para confirmar el usuario
Confirmar usuario sin acceso	Debe desaparecer del listado de usuarios pendientes y aparecer en el listado de usuarios activos	Así sucede. Por defecto se le asigna el rol más bajo
Cambiar rol de un usuario activo	Al cambiarle el rango el usuario pasará a tener la visibilidad que marca dicho rol	Se selecciona un rol en el seleccionable y se pulsa en CAMBIAR RANGO. El usuario ahora pasa a tener el rol asignado

Tabla 4.27: Pruebas de integración del módulo Usuarios

4.13.6 Pruebas de integración en el módulo Carga Manual

Las pruebas con el módulo de carga de datos manuales se basarán en el uso habitual de la pantalla, acciones habituales y errores comunes para detectar si hay algo que no esté controlado. Las pruebas referentes al módulo de carga manual están presentes en la tabla 4.28.

Acción	Esperado	Resultado
Seleccionar tipo y edición	Se carga el formulario de la edición	El formulario con las medidas y entidades participantes de la edición se carga correctamente
Cambiar tipo	Desaparece el formulario	Se quitan los componentes de formularios y de registros al no hacer datos cargados
Seleccionar nuevamente otra edición	carga formulario de la nueva edición	Carga el formulario con medidas y participantes y registros de la edición
Guardar sin todos los datos	Se guarda pues por defecto todos están a 0	Se guarda correctamente el registro con datos de la fuente
Sincronizar valores	Al escribir el valor numérico el valor porcentaje relacionado se actualiza solo	Se actualiza correctamente
No sincronizar valores	Al escribir el valor numérico el valor porcentaje relacionado no se actualiza	Sólo se rellena el dato introducido, no ninguno relacionado
Introducido un valor por encima del máximo	Se modifica para que el valor introducido sea el máximo	Se actualiza indicando como que se ha introducido el dato más alto
Introducido un valor por debajo del mínimo	Se modifica para que el valor introducido sea el mínimo	Se actualiza indicando como que se ha introducido el dato más bajo

Tabla 4.28: Pruebas de integración del módulo de carga manual

4.13.7 Pruebas de integración en el módulo Carga WEB

Estas pruebas comprobarán el buen funcionamiento de este módulo, por lo que se tendrá que emplear alguna web que contenga tablas a ser analizadas y que todas las funcionalidades ofrecidas por la pantallas actúen dentro de su comportamiento esperado. Las pruebas referentes al módulo de carga web están presentes en la tabla 4.29.

Acción	**Esperado**	**Resultado**
Seleccionar edición	Debe cargarse el apartado para introducir URL	Se carga correctamente
Indicar una URL y ejecutar análisis	Debe carga los datos resultantes, tanto encuentre tablas o no. Si encuentra tablas deben mostrarse los datos capturados y los seleccionables de medidas y entidades participantes para poder configurarlo	Lo hace correctamente. Suelen detectarse tablas sin datos por las estructuras internas de algunas páginas web
Seleccionar otro tipo y edición	Debe resetear los datos	Retira todo formulario cargado
Seleccionar otra tabla si se han detectado más de una	Se debe mostrar el formulario configurable de la nueva tabla, pero manteniendo todos los cambios realizados en otras tablas	Todos los datos se mantienen en pantalla salvo que se recargue forzosamente el navegador
Limpiar datos	Escribir una expresión regular que automáticamente identifique el valor alto y bajo en el texto	Escribiendo correctamente el formato para separar valores altos y bajos los limpia por fila e identifica ambos valores
Eliminar una fila	Se retirará del DOM	Se retira correctamente la fila
Eliminar una columna	Se retirará del DOM	Se retira correctamente la columna
Eliminar todas las filas y columnas	Deberá dejar siempre un dato en fila y columnas	Al intentar eliminar una última fina o columna aparece un aviso indicando que no se puede retirar
Pulsar procesar con campos rellenados con letras	Debe avisar del error	Aparece un mensaje alertando de que no son valores numéricos y no guarda nada en el sistema
Pulsar procesar con algunos participantes faltantes	El sistema no hará nada y podrá mostrar un aviso	Detecta que faltan entidades participantes y muestra un mensaje de alerta y no guarda nada en el sistema
Pulsar procesar con todos los datos configurados	El registro se guardará y pasará al estado de validación	Lo hace correctamente

Tabla 4.29: Pruebas de integración del módulo de carga web

4.13.8 Pruebas de integración en el módulo Resultados

En el módulo se probará que los resultados se carguen y actualicen correctamente. Las pruebas referentes al módulo resultados están presentes en la tabla 4.30.

Acción	Esperado	Resultado
Seleccionar una edición	Se cargan los formularios con las medidas y entidades participantes	El formulario para modificar o actualizar los resultados se carga correctamente
Cambiar de edición	Se debe modificar con los formularios y datos correspondientes a la edición seleccionada	Al cambiar el tipo se actualiza el listado de ediciones y al pulsar sobre la nueva edición es cuando los datos se refrescan
Se ha seleccionado una edición sin datos	Deberán cargarse los campos sin datos	Se cargan sin datos, con 0 por defecto
Se ha cargado una edición que ya tenía datos	Se cargan los campos rellenados con los valores actuales	Se cargan correctamente los campos con los datos presentes en el sistema
Se pulsa en guardar en una edición sin resultados	Se guardan los datos introducidos	Los datos se guardan correctamente
Se pulsa en guardar en una edición con resultados	Se actualizan los datos introducidos	Los datos se actualizan correctamente

Tabla 4.30: Pruebas de integración del módulo de Resultados

4.13.9 Pruebas de integración en el módulo Validación

En esta pantalla aparecen todos los registros, independientemente de su procedencia, para ser considerados aptos o no aptos. Las pruebas referentes al módulo validación están presentes en la tabla 4.31.

Acción	Esperado	Resultado
Se marca apto un registro	Desaparece del listado y se muestra visible para todos los usuarios	Comportamiento esperado
Se marca no apto un registro	Desaparece del listado y no se muestra visible para los usuarios	Comportamiento esperado
Se marcan aptos varios registros a la vez	Desaparecen del listado y se muestran visibles para todos los usuarios	Comportamiento esperado
Se marcan no aptos varios registros a la vez	Desaparecen del listado y no se muestran visibles para los usuarios	Comportamiento esperado

Tabla 4.31: Pruebas de integración del módulo de Validación

4.13.10 Pruebas de integración en el módulo Visualizador

Las pruebas referentes al módulo visualizador están presentes en la tabla 4.32.

Acción	Esperado	Resultado
Seleccionar datos en los seleccionables	En base a los datos seleccionados se irán mostrando los distintos gráficos	Se van mostrando y actualizando los distintos gráficos con toda la información
Seleccionar una edición sin resultados finales	Se deberán mostrar sólo los gráficos que son posibles sin resultados finales	Se muestran los gráficos con los datos pero sin resultados e fiabilidad

Tabla 4.32: Pruebas de integración del módulo visualizador

4.14 Conclusiones

Con todo esto ya se tiene desarrollada la aplicación principal con todos los módulos y sus funcionalidades desarrolladas y distintos desarrollos de comodidad como el multilenguaje. Y con las pruebas de integración se han detectado errores y corregido para que el funcionamiento sea el esperado. Con el software la aplicación ya es totalmente usable a nivel local, pero como se trata de una herramienta online el último paso será el despliegue, en el que se publicará a la red la aplicación y podrá ser accedida por todo el mundo para que empiece a usarse del modo en el que fue concebida.

Capítulo 5

Despliegue

5.1 Introducción

Una de las razones que nos llevó a usar tecnologías de Microsoft al desarrollar la aplicación web era que no se perdieran excesivos recursos en ámbitos en los que no se era experto y tampoco eran el punto general del proyecto, pero que a la vez eran de gran importancia para su buen funcionamiento.

Al poder tener todo agrupado en una única plataforma y poder actualizar los recursos de forma escalable según las necesidades con unas pequeñas configuraciones sin requerir grandes conocimientos de mantenimiento hace que este apartado sea rápido, sencillo y mucho más seguro que si lo hubiéramos contratado y montado de forma independiente.

Al ser un proyecto pequeño es una opción perfecta, pero en caso de proyectos grandes con mucho movimiento y mucha carga de trabajo podría no ser tan provechosos según el gasto que suponga y su precio.

5.2 Migración de la base de datos

Desde un principio se planteó el desarrollo y diseño de la base de datos partiendo desde el código, generando entidades en base a la división pensada para particionar todo el sistema en distintos datos relacionados. Por eso se diseñó para que cada tabla tuviese su equivalente en código C# (Code First) para que toda la lógica código C# - consulta SQL fuera trabajo interno del ORM empleado, en este caso `Entity Framework Core`. De esta forma al tener las clases C# bien definidas en nuestro proyecto la generación y despliegue de la base de datos sería algo sumamente sencillo como ejecutar un par de instrucciones.

Es decir, con tener las clases equivalentes a las entidades y sus etiquetas para indicar claves y restricciones el sistema de migraciones de `Entity Framework Core` se encargará de crear las tablas y columnas en la base de datos.

Para realizar la migración del diseño de base de datos al servidor de base de datos deseado se tiene que hacer las configuraciones previas:

5.2.1 Servicio SQL Database Azure

Dadas las facilidades de escalabilidad y cómputo que ofrece Azure [32] tanto el hosting como la base de datos estarán en la plataforma de Microsoft. La configuración que se realizará ahora para la base de datos se compartirá en el siguiente apartado para publicar la aplicación web y se que sea accesible públicamente con acceder a un enlace web.

El servicio que se usará para la base de datos es SQL Database. Al crear la base de datos en Azure es necesario indicar la suscripción y el grupo de recursos.

- **Suscripción** Es la suscripción que tenemos contratada en Azure
- **Grupo de recursos** Azure permite agrupar distintos recursos en un grupo, para así poder interaccionar de forma sencilla

Una vez indicados y/o creados estos dos puntos se ha de configurar el servidor sql database con los siguientes campos

- **Nombre del servidor** Es el nombre que tendrá el servidor al que podremos añadir distintas bases de datos
- **Ubicación** Es la ubicación física de los servidores de Azure que queremos usar. No todos ofrecen las mismas capacidades de cómputo
- **Método de autentificación** Cómo vamos a iniciar sesión en el servidor. Se usará usuario y contraseña
- **Inicio de sesión** El usuario con el que nos conectaremos con el servidor
- **Contraseña** La contraseña que permitirá el acceso al usuario que hemos indicado

Finalmente sólo queda configurar los datos finales de la base de datos que estamos creando, tras haber indicado y/o creado el grupo de recursos y el servidor de base de datos:

- **Nombre de la base de datos** El nombre que le queremos dar
- **Proceso y almacenamiento** Es la opción escalable de Azure. Podemos elegir desde una modalidad gratuita hasta configuraciones de alto rendimiento para grandes sistemas.

Una vez tenemos creado el servidor y una base de datos en ese servidor ya no es necesario realizar más acciones relacionadas con la base de datos, pues el resto se hará desde el proyecto con migraciones o mediante el ORM de `Entity Framework Core`.

5.2.2 Configuración del proyecto web

Lo primero que se debe configurar es la clase contexto que será la que empleará el ORM como enlace entre el código y los datos. Todas las clases configuradas como entidades que se añadan en esta clase serán migradas, se creará (actualizará o eliminará) su tabla, columnas y relaciones, y podrá ser consultado su contenido desde el código tanto con consultas de `Entity Framework Core` o con `LINQ`, siempre que se haya instanciado la referencia en el componente desde el que se desea consultar la información contenida en las tablas.

Además de las entidades, también se pueden confirmar claves compuestas, como en el case de `MedidasEdicionSondeoRelacion`.

```
public class ApplicationDbContext : IdentityDbContext
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }

    protected override void OnModelCreating(ModelBuilder builder)
    {
        base.OnModelCreating(builder);
        builder.Entity<MedidasEdicionSondeoRelacion>()
            .HasKey(x =>
              new { x.MedidasEdicionSondeoNumericoId, x.MedidasEdicionSondeoPorcentajeId });
    }

    public DbSet<TipoSondeo> TipoSondeos { get; set; }
    public DbSet<EdicionSondeo> EdicionSondeos { get; set; }
  ...
}
```

En el fichero de configuración inicial, `Startup.cs`, en la función `ConfigureServices`, se ha de indicar la siguiente línea, que se encargará de indicar sobre qué base de datos se trabajará y el nombre de la cadena de conexión empleada.

```
services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(
        Configuration.GetConnectionString(''DefaultConnection'')));
```

En el fichero `appsettings.json` se pueden configurar las distintas conexiones a las bases de datos. Aquí se indicará la cadena de nos proporcione azure para conectarse al servidor empleando el usuario, contraseña y nombre de la base de datos.

```
''ConnectionStrings'': {
  ''DefaultConnection'': ''la cadena de conexion'"
},
```

5.2.3 Despliegue de la base de datos

Teniendo las clases tal como se definieron en la implementación, en la que cada clase actúa como una entidad, la migración para crear las tablas en el servidor de base de datos es tan sencillo como ejecutar dos instrucciones.

EF Core, el ORM de Microsoft ofrece varios modos para realizar la migración, desde todo automatizado hasta generar el script SQL. El modo automatizado no es realmente automatizado del todo, pero sí un modo muy guiado. Es decir, al generar la migración coge las clases-entidades indicadas en el apartado anterior y en base al diseño que hemos indicado genera un código que muestra todas las modificaciones que va a realizar sobre la base de datos. Ese código lo que hace es traducir lo que hemos puesto con las etiquetas, pero también nos permite indicar ciertas acciones previas a la migración. Por ejemplo, en el caso de `Entity Framework Core` es muy importante indicar algunas relaciones como NO ACTION dónde indica CASCADE. De esta forma tendremos control total para que cuando añadamos información

sólo se añada el dato introducido y no posibles objetos relacionados que sólo cumplen función informativa en el código. También se evitan bucles.

Las migraciones, y el código intermedio que se genera, no es algo que se haga una única vez, sino que puede realizarse en pleno desarrollo como un modo de pruebas, o en producción como una forma rápida de añadir nuevas funcionalidades sin modificar directamente la base de datos. Así todo cambio en el código se verá reflejado de forma idéntica físicamente en la base de datos.

Desde visual studio 2022 las instrucciones para las migraciones se ejecutan desde `Package Manager Console`.

```
Add-Migration migracion1
```

En cuando se ejecuta esta instrucción se genera en el proyecto la ruta `Migrations` y dentro todas las modificaciones que se van a reflejar en la base de datos.

Para que se reflejen hay que ejecutar lo siguiente:

```
Update-Database
```

Si en el futuro es necesario hacer nuevas modificaciones el proyecto, una vez programados los cambios, con realizar una nueva migración y posterior update los cambios se verán reflejados en la base de datos. Es posible que si ya existen datos y se añaden nuevas columnas que son claves foráneas es posible indicar el valor por defecto, o hacer dos migraciones. Primero la que crea la tabla, actualizar los datos, y hacer una nueva migración con los cambios de las relaciones. Al hacer update si hay problemas con ciclos y campos que no pueden ser nulos, etc, se indicará en ese paso y podrán ser corregidos tanto realizando una nueva migración o modificando el fichero que se genera en el directorio de `Migrations`.

5.3 Despliegue de la aplicación web

El despliegue de la aplicación web, una vez se tiene configurada toda la base de datos, es aún más sencillo ya que con tener vinculada una cuenta de Azure en Visual Studio 2022 todos los posibles pasos necesarios son ejecutados directamente desde el IDE en distintas pantallas que nos va marcando qué opciones hay así el qué posibles errores nos pueden salir. Una vez configurado dónde y cómo publicar la aplicación web sólo será necesario volver a pulsar un botón para publicar los cambios en el código del proyecto.

En la versión en inglés de `Visual Studio 2022` la opción para publicar está en

```
  Build -> Publish...
```

Si es la primera vez se tendrán que configurar algunos datos.

Lo primero es dónde se va a publicar la aplicación web (figura 5.1), por lo que seleccionaremos de las distintas opciones Azure.

- **Azure** Desplegarlo en Azure
- **Docker** Desplegarlo en un contenedor
- **Directorio local o en red** Desplegarlo en una ruta local
- **FTP** Desplegarlo en un servidor FTP
- **IIS** Desplegar en IIS

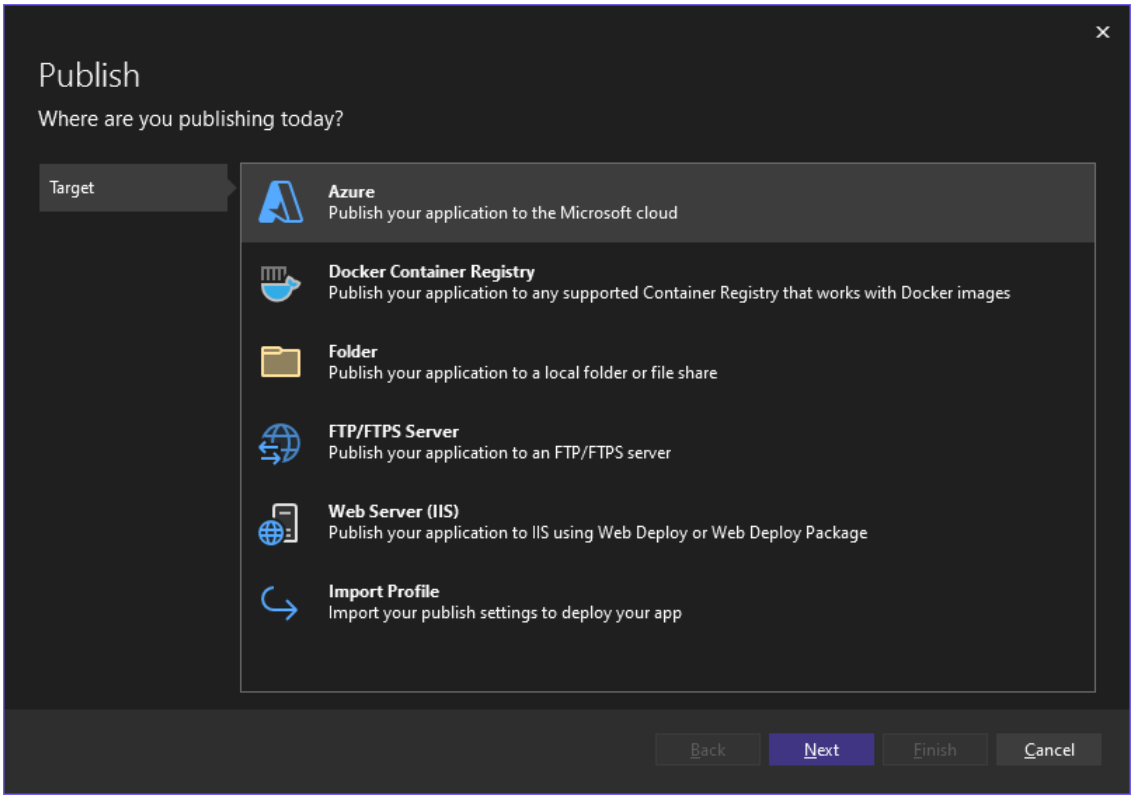

Figura 5.1: Listado de opciones para desplegar la aplicación

Una vez se ha seleccionado Azure es necesario seleccionar el tipo de Servicio Azure a elegir (figura 5.2). Elegimos para que sea Azure App Service bajo windows.

- **Azure App Service Windows**
- **Azure App Service Linux**
- **Azure Container Apps Linux**
- **Azure App Service Container**
- **Azure Container Registry**
- **Azure Virtual Machine**

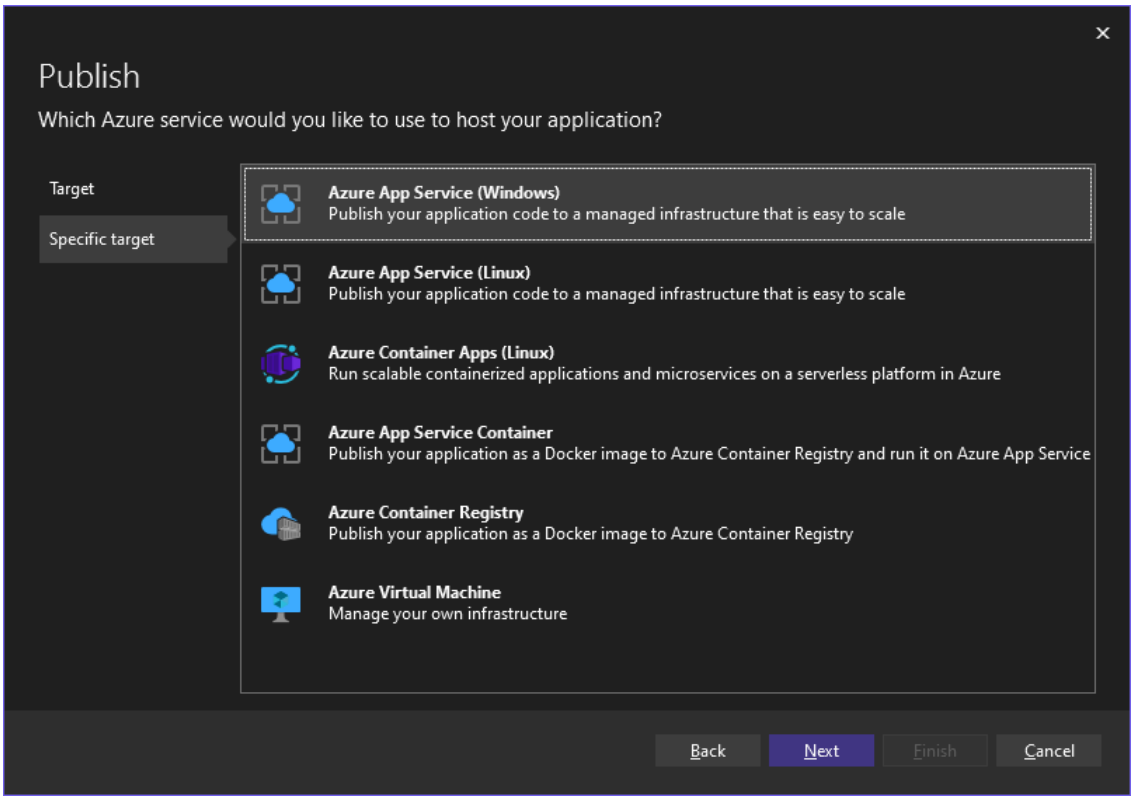

Figura 5.2: Listado de despliegues dentro de Azure

Ahora falta finalizar el servicio contratado sobre el que se hará el despliegue seleccionando un Hosting Plan adecuado. Para esta aplicación web el más bajo es suficiente, y dependiendo del uso de datos y carga de usuarios se valoraría aumentar la capacidad de almacenamiento o la capacidad de cómputo.

En este punto ya está configurado el despliegue, pero antes de poder pulsar en `Publish` se han de configurar dos puntos más.

5.3.1 Cadena de conexión a los datos

Aunque el proyecto Blazor está configurado con una cadena de conexión, para facilitar ese trabajo se nos permite desde la propia pantalla de publicación configurar la cadena de conexión o servicio SQL Database de Azure se va a ser necesaria una vez la aplicación está pública, de esa forma Azure ya sabe que los

datos están dentro de la propia plataforma. Es una forma de garantizar que la conexión datos-proyecto una vez publicado es correcta. Configuramos para que la base de datos enlazada sea la que se ha creado antes en Azure y con migraciones.

5.3.2 SignaIR

Blazor hace uso de `SignaIR` para la comunicación entre la parte cliente y la parte servidor. SignaIR es una tecnología que facilita la comunicación bidimensional de datos y aunque no se implemente deliberadamente en algún componente nativo se usa para el intercambio de datos entre la vista y la lógica tras esa vista. De esa forma el refresco al cambiar el valor de las variables es tan rápido. Se crea el servicio `SignaIR` de Azure y se enlaza al proyecto, igual que la base de datos.

5.3.3 Publicación del proyecto

Con todos los servicios de Azure configurados y enlazados con `Visual Studio 2022` sólo es necesario pulsar el botón Publish y si todo ha ido correcto se publicará en pocos segundos en Azure y será accesible desde una url proporcionada por Azure.

Capítulo 6

Conclusiones y líneas futuras

6.1 Conclusiones

El objetivo principal era desarrollar una herramienta colaborativa para el análisis temporal de la fiabilidad de encuestas, sondeos y estimaciónn de indicadores generadas por distintas fuentes (medios de comunicación, organismos nacionales o internacionales, asociaciones, etc.). El resultado más interesante radica en que, con los datos facilitados por distintos usuarios, se pudiera llegar a conclusiones sobre la fiabilidad y acierto de las distintas fuentes, e identificar cómo unas ofrecen datos más fiables o certeros que otras.

Partiendo de esa idea y usando algunos casos prácticos para identifica el problema se planificó la herramienta dividiendo el problema en pequeños elementos relacionados, y a la vez esta división se usó como base para dividir el objetivo en pequeñas herramientas relacionadas, por lo que poco a poco se fue estructurando esa división como módulos de la herramienta en los que unos complementaban a otros, para en conjunto todos esos datos crear esa idea inicial.

Data la particularidad de tener que ser colaborativa también se valoraron las distintas formas de desarrollarlo y con qué tecnologías, apostando por nuevas metodologías y tecnologías y con la intención de ser desplegadas en una plataforma como azure con todos sus servicios unidos y escalables en caso de que fuera necesario.

La herramienta al tener todos los datos bien estructurados, divididos y relacionados es capaz de ofrecer el dato que se buscaba, pero ofrece infinitas oportunidades dada la facilidad se acceder a la información almacenada y la rapidez para procesarla y consultarla.

En conclusión, se ha desarrollo una aplicación web accesible desde internet empleando una implementación Single Page Application usando Blazor para la parte visual, el lenguaje C# para toda la lógica cliente y servidor y el ORM `Entity Framework Core` para el intercambio de información con la base de datos, que en conjunto y el trato de la información almacenada en la base de datos se cumplen todos los requisitos planteados.

6.2 Líneas futuras

La aplicación web ha sido desarrollada de forma que sea escalable incrementando funcionalidades así como que no sea muy complicado mejorar las ya disponibles.

Son varias las líneas futuras a afrontar en la aplicación:

- **Implementar otras formas de introducir datos** Igual que en la actual versión se pueden introducir datos de forma manual o con una ayuda que escanea los datos, afrontar nuevos módulos que analicen imágenes en busca de los datos, o que sean capaces de autorellenar las columnas de entidades participantes y medidas, serían una gran ayuda. Una amplia gama de formas de introducir que facilite la velocidad y ahorre el trabajo tedioso hace que más gente se acerque a colaborar, y la herramienta se convierte en algo mucho más útil cuanta más gente colabore y la complemente.

- **Añadir más gráficos e información estadística** Los cálculos se han centrado en determinar qué fuentes proporcionan los datos más precisos, pero la gran cantidad de datos disponible puede emplearse para determinar muchos más datos, así como complementarlos con ayudas externas para que esos datos se conviertan en mucha más información que sólo el de fiabilidad.

- **Implementar un sistema de reputación para gestionar roles** El sistema actual para validar la información está basado en la supervisión de administradores de la información introducida, y es un sistema que quedaría obsoleto o generaría controversia al crecer el número de usuarios, al poder generarse dudas sobre las figuras de esos administradores y hasta qué punto su labor otorgando roles similares a otros usuarios estaría siendo el deseado. Con ese futuro problema se tendría que implementar un sistema en el que se tuviera en cuenta puntos como la antiguedad, su actividad en la plataforma así como las valoraciones de otros usuarios, y que sus cambios de rol estuvieran basados en esas variables y no en un acto manual realizado por otro usuario.

- **Perfeccionar funcionalidades presentes** Algunas funcionalidades que pueden ser consideradas habituales como eliminar datos no han sido tenidas en cuenta pues el objetivo no buscaba realizar una plataforma con cientos de opciones, sino que se centraba en el objetivo del dato final. Si se cuenta con tiempo y personal es posible añadir más funcionalidades a las pantallas, mejor gestión de cada dato almacenado, rehacer ciertas secciones con unas pantallas más sencillas de usar, etc.

En resumen, con la base diseñada y desarrollada actualmente las opciones son infinitas pues todo lo que se añada se basará en un diseño que funciona y que se mantendría intacto y funcional con el tiempo.

Capítulo 7

Presupuesto

Pues oculto y silencioso es el camino por el que la gracia se adentra en los corazones [1].

Simone Weil

En este capítulo se incluye una estimación del coste total para la realización del proyecto. En las siguientes secciones los costes son divididos en función de su origen, exponiendo el subtotal en cada uno de los apartados y al final se añade el total de estas cifras.

7.1 Recursos Hardware

Debido a tratarse de un desarrollo software no será necesario realizar una gran inversión económica en recursos hardware. Dichos recursos son enumerados en el siguiente listado.

- Ordenador personal (Procesador i7-9700K, 16GB RAM, Windows 10, Tarjeta gráfica GeForce GTX 1660 o superior).
- Monitor.

El precio y las unidades son incluidos en la tabla 7.1 junto con el total de los recursos hardware.

Concepto	Precio por Unidad	Cantidad	Subtotal
Ordenador	1500,00 €	1	1500,00 €
Monitor	150,00 €	2	300,00 €
TOTAL			1800,00 €

Tabla 7.1: Recursos hardware usados

7.2 Recursos Software

Para poder implementar el desarrollo software así como pruebas y documentación serán necesarios los siguientes programas. Algunos de estos software tienen licencias gratuitas para estudiantes así como para proyectos no comerciales, pero son necesarias para un uso comercial.

[1] Simone Wil, A la espera de Dios, Editorial Trotta, 2009.

- Visual Studio 2022.
- SQL Server Express.
- SQL Server Management Studio.
- Sublime Text.
- TeXworks.
- Microsoft Visio.

El precio y las unidades son incluidos en la tabla 7.2 junto con el total de los recursos software.

Concepto	Precio por licencia	Cantidad	Subtotal
Visual Studio 2022	534,00 €	1	534,00 €
SQL Server Express	0,00 €	1	0,00 €
SQL Server Management Studio	0,00 €	1	0,00 €
Sublime Text	0,00 €	1	0,00 €
TeXworks	0,00 €	1	0,00 €
Microsoft Visio	889,00 €	1	889,00 €
Librería MudBlazor	0,00 €	1	0,00 €
TOTAL			1423,00 €

Tabla 7.2: Recursos software usados

7.3 Recursos Servicio

Además de software serán necesarios algunos servicios de pago por uso proporcionados por Azure. El precio es imposible de predecir ya que dependiendo de los datos cargados, solicitados o servidos el precio variará. En base al plan seleccionado al contratar los servicios se estimará el precio aproximado que se espera de forma mensual.

- Azure Base de datos SQL.
- Azure App Service.

Puesto que se esperan muchas conexiones simultáneas se ha elegido el plan `Premium Rendimiento y escala mejorados` tal como se muestra en el listado de precios de Azure [33], con un precio de 146,27€ mensuales, que ofrece espacio e instancias para cómputo suficientes y no hay límite de tiempo de CPU activa [34].

Para la base de datos ha elegido el plan de 10,2GB de `Serie Estándar (Gen 5)` indicado en su tabla de precios [35] con un coste de 368,87€ para poder hacer frente a continuas consultas y modificaciones a los datos.

El precio y la estimación mensual son incluidos en la tabla 7.3 junto con el total anual de los recursos servicios.

Concepto	Precio por mes	meses	Subtotal
Azure Base de datos SQL	368,87 €	12	4426,44 €
Azure App Service	146,27 €	12	1755,24 €
TOTAL			6181,68 €

Tabla 7.3: Recursos servicio usados

7.4 Recursos Humanos

El coste proveniente a los recursos humanos del trabajo provienen de uno o varios ingenieros con capacidades para actuar en todos los frentes del proyecto y que desarrollarán el trabajo completo, desde la toma de requisitos, el análisis y finalmente el desarrollo. Los rangos de precios en base a la plataforma malt [36] de agentes libres para ser contratados varía entre 30 y 50 euros, por lo que se tomará 40€ como media de su precio. En caso de no optar por esta opción de agentes libres se tendría que optar por contratos de obra y el coste total supondría terminaría siendo parecido, y si se decide por perfiles con sueldos más bajos se comprometería el desarrollo al estar pensado para un equipo muy reducido.

Se estima una duración aproximada de 6 meses, aproximadamente 1500 horas, al tratarse de un proyecto con distintas fases en las que el comienzo de una fase depende de la finalización de la anterior. La duración podría reducirse dependiendo del número de profesionales, pero eso no variaría la suma de horas total.

A continuación se incluyen dichos costes.

Concepto	Precio por hora	Cantidad de horas	Subtotal
Ingeniero	40,00 €	1500	60000,00 €
TOTAL			60000,00 €

Tabla 7.4: Recursos humanos

7.5 Presupuesto de ejecución material

El presupuesto de ejecución material es la suma de los recursos hardware, software y humanos necesarios para llevar a cabo el trabajo.

Concepto	Subtotal
Recursos hardware	1800,00 €
Recursos software	1423,00 €
Recursos servicio	6181,68 €
Coste mano de obra	60000,00 €
TOTAL	132672,68 €

Tabla 7.5: Presupuesto de ejecución material

7.6 Importe de la ejecución por contrata

Los costes de la ejecución por contrata deben incluir los gastos derivados del uso de las instalaciones donde se ha llevado a cabo el trabajo, las cargas fiscales, los gastos financiero, las tasas administrativas y las obligaciones de control del proyecto.

Dicho gasto se asume estableciendo un recargo sobre el coste del importe del presupuesto de ejecución material. Dicho recargo equivale al 22 % de dicho importe

Concepto	Subtotal
22 % del coste total de ejecución de material	5648,21 €

Tabla 7.6: Importe de ejecución por contrata

7.7 Honorarios facultativos

Se fija en este proyecto un porcentaje del 7 % sobre el coste total de ejecución por contrata.

Concepto	Subtotal
7 % del coste de ejecución por contrata	395,38 €

Tabla 7.7: Importe de los honorarios facultativos

7.8 Presupuesto Total

A continuación, en la tabla 7.8 se realiza la suma de todos los conceptos del presupuesto tenidos en cuenta en los anteriores apartados.

Concepto Precio	Subtotal
Presupuesto de ejecución material	63268,00 €
Importe de la ejecución contratada	5648,21 €
Horarios facultativos	395,38 €
TOTAL (sin IVA)	31717,29 €
IVA (22 %)	6977,80€
TOTAL	108006,68 €

Tabla 7.8: Importe del presupuesto total del proyecto

Bibliografía

[1] Microsoft, "Implementación de componentes blazor en asp.net core," 2022, en línea en https://docs.microsoft.com/es-es/aspnet/core/blazor/components/?view=aspnetcore-6.0/, ultima consulta 08/2022.

[2] AppMaster, "Artículo con pros y contras de desarrollos sobre escritorio y web." 2022, en línea en https://appmaster.io/es/blog/aplicacion-de-escritorio-o-aplicacion-web-pros-y-contras/, ultima consulta 08/2022.

[3] InternetYa, "Artículo con pros y contras de desarrollos sobre escritorio y web." 2022, en línea en https://www.internetya.co/aplicaciones-web-vs-escritorio-2/, ultima consulta 08/2022.

[4] Microsoft, "Implementación de aplicaciones con windows forms," 2022, en línea en https://docs.microsoft.com/es-es/dotnet/desktop/winforms/?view=netdesktop-6.0/, ultima consulta 08/2022.

[5] ——, "Implementación de aplicaciones con wpf," 2022, en línea en https://docs.microsoft.com/es-es/visualstudio/designers/getting-started-with-wpf?view=vs-2022/, ultima consulta 08/2022.

[6] ——, "Implementación de aplicaciones con maui," 2022, en línea en https://docs.microsoft.com/es-es/dotnet/maui/what-is-maui/, ultima consulta 08/2022.

[7] Wikipedia, "Información sobre responsive web design," 2022, en línea en https://en.wikipedia.org/wiki/Responsive_web_design/, ultima consulta 08/2022.

[8] ——, "Información sobre modelo-vista-controlador," 2022, en línea en https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller/, ultima consulta 08/2022.

[9] Microsoft, "Implementación de aplicaciones con asp.net core mvc," 2022, en línea en https://docs.microsoft.com/es-es/aspnet/core/tutorials/first-mvc-app/start-mvc?view=aspnetcore-6.0&tabs=visual-studio/, ultima consulta 08/2022.

[10] Wikipedia, "Información sobre spring framework," 2022, en línea en https://en.wikipedia.org/wiki/Spring_Framework/, ultima consulta 08/2022.

[11] ——, "Información sobre symfony," 2022, en línea en https://en.wikipedia.org/wiki/Symfony/, ultima consulta 08/2022.

[12] ——, "Información sobre ruby on rails," 2022, en línea en https://en.wikipedia.org/wiki/Ruby_on_Rails/, ultima consulta 08/2022.

[13] ——, "Información sobre spa," 2022, en línea en https://en.wikipedia.org/wiki/Single-page_application/, ultima consulta 08/2022.

[14] Google, "Página web oficial de angular," 2021, en línea en https://angular.io/, ultima consulta 08/2022.

[15] Microsoft, "Página web oficial de blazor," 2021, en línea en https://docs.microsoft.com/es-es/aspnet/core/blazor/?view=aspnetcore-6.0/, ultima consulta 08/2022.

[16] E. F. Tutorial, "Información sobre entity framework tutorial - code first," 2022, en línea en https://www.entityframeworktutorial.net/code-first/what-is-code-first.aspx/, ultima consulta 08/2022.

[17] Wikipedia, "Información sobre orm," 2022, en línea en https://en.wikipedia.org/wiki/Object%E2%80%93relational_mapping/, ultima consulta 08/2022.

[18] Microsoft, "Página principal de sql server y sus distintas versiones." 2022, en línea en https://www.microsoft.com/es-es/sql-server/sql-server-downloads/, ultima consulta 08/2022.

[19] ——, "Página principal de entity framework core," 2022, en línea en https://docs.microsoft.com/es-es/ef/core/, ultima consulta 08/2022.

[20] E. F. Tutorial, "Página principal de entity framework tutorial," 2022, en línea en https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx/, ultima consulta 08/2022.

[21] Microsoft, "Implementación de signalr en asp.net core," 2022, en línea en https://docs.microsoft.com/es-es/aspnet/core/signalr/introduction?view=aspnetcore-6.0/, ultima consulta 08/2022.

[22] E. F. Tutorial, "Información sobre entity framework tutorial - migraciones," 2022, en línea en https://www.entityframeworktutorial.net/efcore/entity-framework-core-migration.aspx/, ultima consulta 08/2022.

[23] T. World, "Página principal del software toad data modeler." 2022, en línea en https://www.toadworld.com/products/toad-data-modeler/, ultima consulta 08/2022.

[24] Microsoft, "Implementación de autenticación y autorización de asp.net core blazor," 2022, en línea en https://docs.microsoft.com/es-es/aspnet/core/blazor/security/?view=aspnetcore-6.0/, ultima consulta 08/2022.

[25] ——, "Implementación de enrutamiento en asp.net mvc," 2022, en línea en https://docs.microsoft.com/es-es/aspnet/mvc/overview/older-versions-1/controllers-and-routing/asp-net-mvc-routing-overview-cs/, ultima consulta 08/2022.

[26] Mozilla, "Información sobre solicitudes http," 2022, en línea en https://developer.mozilla.org/es/docs/Web/HTTP/Methods/, ultima consulta 08/2022.

[27] ——, "Información sobre dom," 2022, en línea en https://developer.mozilla.org/es/docs/Glossary/DOM/, ultima consulta 08/2022.

[28] B. ApexCharts, "Documentación oficial de la librería blazor-apexcharts," 2022, en línea en https://apexcharts.github.io/Blazor-ApexCharts/, ultima consulta 08/2022.

[29] ApexCharts, "Documentación oficial de la librería apexcharts," 2022, en línea en https://apexcharts.com/, ultima consulta 08/2022.

[30] IBM, "Información sobre pruebas de integración." 2022, en línea en https://www.ibm.com/docs/es/rtw/9.0.1?topic=phases-integration-testing/, ultima consulta 08/2022.

[31] ——, "Información sobre pruebas unitarias." 2022, en línea en https://www.ibm.com/docs/es/rtw/9.0.1?topic=phases-unit-testing/, ultima consulta 08/2022.

[32] Microsoft, "Implementación de aplicaciones con azure devops," 2021, en línea en https://docs.microsoft.com/es-es/learn/paths/deploy-applications-with-azure-devops/, ultima consulta 08/2022.

[33] ——, “Detalle de precios de azure app service,” 2022, en línea en https://azure.microsoft.com/es-es/pricing/details/app-service/windows/, ultima consulta 09/2022.

[34] ——, “Detalle de azure app service,” 2022, en línea en https://learn.microsoft.com/es-es/azure/azure-resource-manager/management/azure-subscription-service-limits/, ultima consulta 09/2022.

[35] ——, “Detalle de precios azure sql database,” 2022, en línea en https://azure.microsoft.com/es-es/pricing/details/azure-sql-database/single/, ultima consulta 09/2022.

[36] malt, “Plataforma freelance,” 2022, en línea en https://www.malt.es/, ultima consulta 08/2022.

Apéndice A

Manual de usuario

A.1 Introducción

La aplicación web permite visualizar información agrupada procedente de distintas fuentes que es comparada buscando la fiabilidad de esos datos frente a sus resultados finales. El usuario tendrá la capacidad de acceder a la plataforma para aportar información procedente de nuevas fuentes o añadir registros de fuentes ya presentes aún no disponibles, ayudando a la comundiad y logrando unos resultados mucho más fiables y completos.

A.2 Nomenclatura

A continuación se detallan los principales conceptos presentes en la aplicación web

- **Edición:** Es el concepto sobre el que se van a añadir los datos y sobre el que las distintas fuentes publicarán informes para predecir o estimar su resultado. Un ejemplo serían las elecciones generales de España de 2019 o el crecimiento del PIB de España en 2021.
- **Tipo:** Es el concepto que agrupa las ediciones. Por ejemplo, todas las elecciones generales de España estarán bajo un tipo para poder ver la variación de las fuentes no sólo en un rango de tiempo, sino a lo largo de todo el histórico posible.
- **Medida:** Es el dato sobre e l que se aporta información. Por ejemplo número de escaños o número de votos en unas elecciones, o porcentaje de crecimiento del PIB de un estado.
- **Entidad participante:** Es el concepto al que se le asignan los datos de esas medidas, como puede ser un partido político o un estado.
- **Registro:** Es la información procedente de una fuente e introducida al sistema por un usuarios que incluye los datos respecto a los sondeos, predicciones, etc.
- **Fuente:** Es la fuente que proporciona la información de sondeos, estimaciones, predicciones, etc.
- **Fiabilidad:** Es el valor de 0 a 100 que mide lo cerca o lejos que ha estado una predicción o sondeo de acercarse a los resultados finales, siendo 0 el mejor resultado y 100 el peor.

A.3 Pantalla inicial

Por defecto al entrar en la aplicación web se cargará la vista sin sesión, que mostrará una vista ligeramente reducida de todos los datos que se pueden consultar iniciando sesión.

Para iniciar sesión o crear cuenta se deben pulsar los botones de la imagen A.1. Tenga en cuenta que al crear una cuenta deberá esperar a que un administrador la valide para poder entrar.

Figura A.1: Botones para iniciar sesión y crear cuenta de usuario

Una vez iniciada la sesión podrá cerrarla pulsando el botón de la imagen A.2.

Figura A.2: Botón para cerrar la sesión

Al iniciar sesión se cargará por defecto la pantalla de Mi menú, que muestra un resumen de todos los elementos creados por el usuario. En caso de ser administrador se mostrarán todos.

Para desplazarse por las distintas pantallas principales ha de pulsar sobre los botones que hay en la barra superior tal como indica la imagen A.3.

Figura A.3: Menú para desplazarse a las pantallas principales

A.4 Tipos

En la pantalla tipos puede crear tipos indicando un nombre y una descripción. Se recomienda indicar también etiquetas que lo identifiquen. Pulsando en el botón guardar se creará el tipo y se mostrará en el listado de tipos existentes. Puede identificar los campos y botón en la imagen A.4.

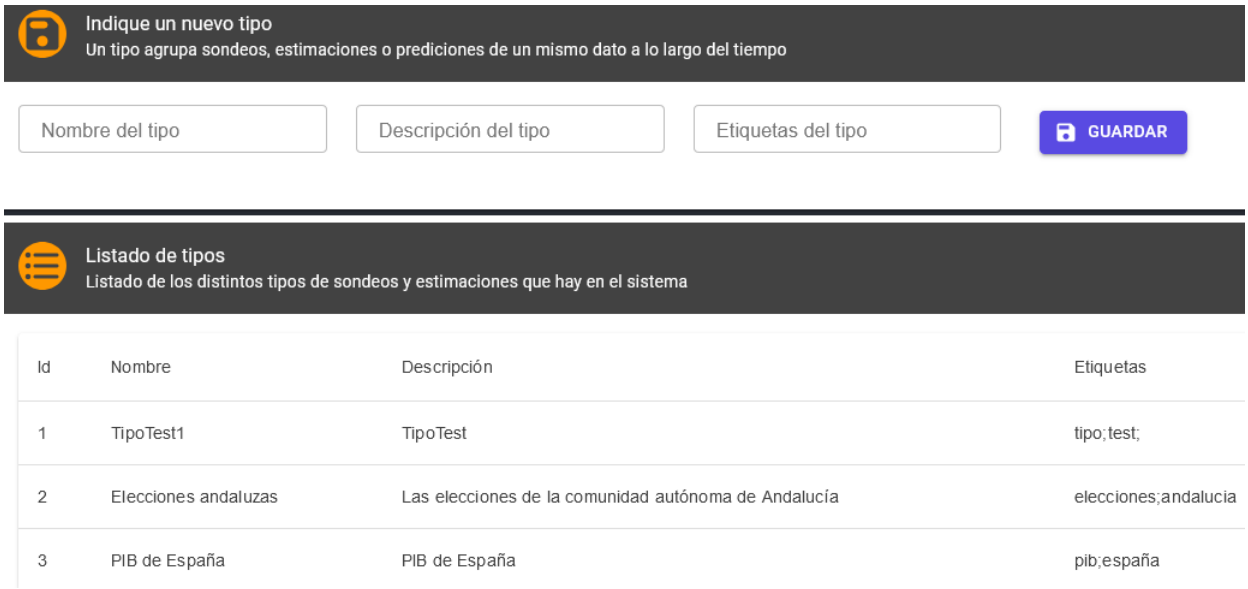

Figura A.4: Pantalla para crear tipos

A.5 Entidades participantes

Para crear una entidad participante deberá indicar a qué tipo pertenece, asignarle un nombre y un color. Al pulsar CREAR la entidad participante se creará y quedará ligada al tipo. También se podrán ver las entidades existentes tal como indica la imagen A.5.

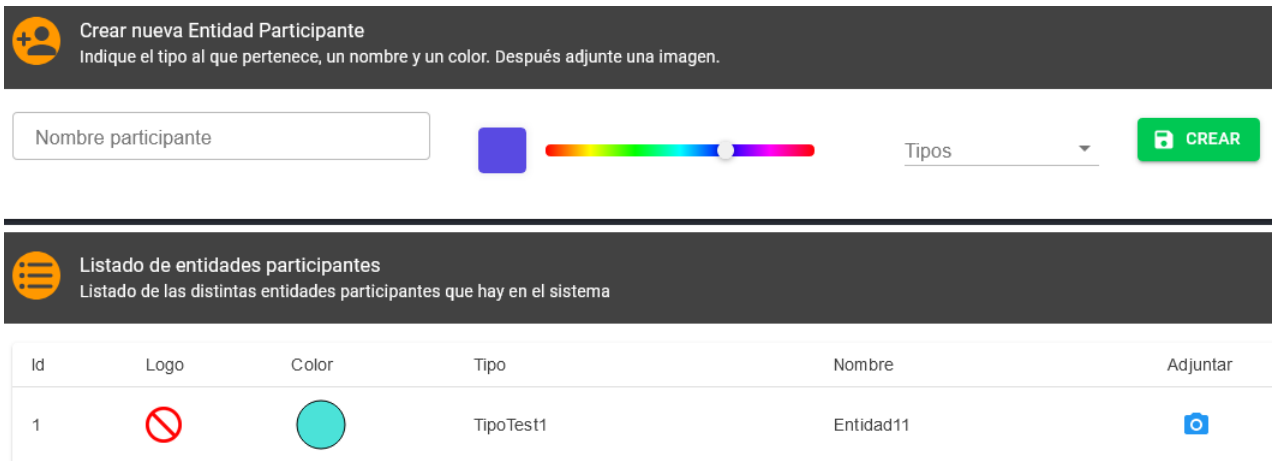

Figura A.5: Pantalla para crear entidades participantes

A.6 Ediciones

Una edición es un contexto concreto de un tipo sobre el que se van a ofrecer sondeos, estimaciones o predicciones. Por ejemplo, mientras un tipo son unas elecciones, las ediciones serán las elecciones concretas de ciertos años.

Para crear una edición se deberá indicar un tipo y un nombre, pero también es necesario indicar entidades participantes, a lo que se le asigna un dato, y medidas, el dato concreto asignado.

Se mostrarán los botones para importar medidas y entidades participantes, mostrado en la figura A.6, y los botones para insertar, mostrados en la figura A.7

Figura A.6: Botones para importar entidades participantes y medidas

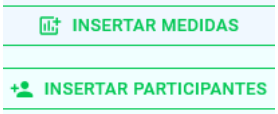

Figura A.7: Botones para insertar entidades participantes y medidas

Para importar medidas desde otra edición al pulsar el botón de importar medidas se abrirá la pantalla de la figura A.8 en la que se tendrá que seleccionar otra edición. Al pulsar IMPORTAR se configurarán automáticamente.

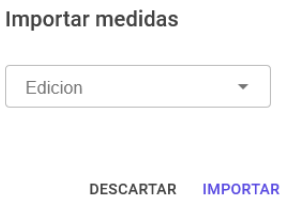

Figura A.8: Modal para importar medidas

Para importar entidades participantes desde otra edición al pulsar el botón de importar entidades participantes se abrirá la pantalla de la figura A.9 en la que se tendrá que seleccionar otra edición. Al pulsar IMPORTAR se configurarán automáticamente.

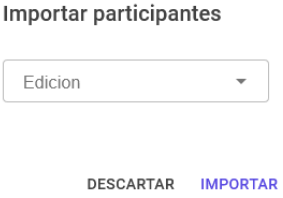

Figura A.9: Modal para importar entidades participantes

Para insertar medidas puede hacerlo con tipo numérico (figura A.10) o tipo porcentaje (figura A.11). Es necesario indicar un rango y si tiene medida relacionada. Si se indica medida relacionada se generará en paralelo otra tipo porcentaje y quedarán enlazadas. Al pulsar insertar se van añadiendo.

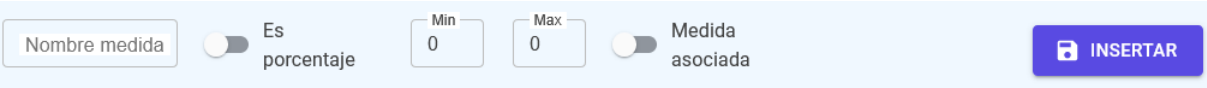

Figura A.10: Insertar medida numérica

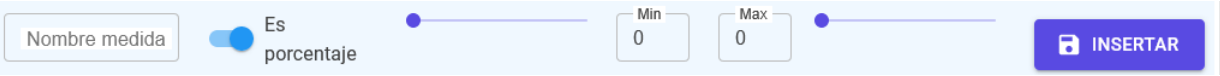

Figura A.11: Insertar medida porcentaje

Para insertar entidades participantes puede seleccionarlos arrastrándolos al panel de la derecha (figura A.12).

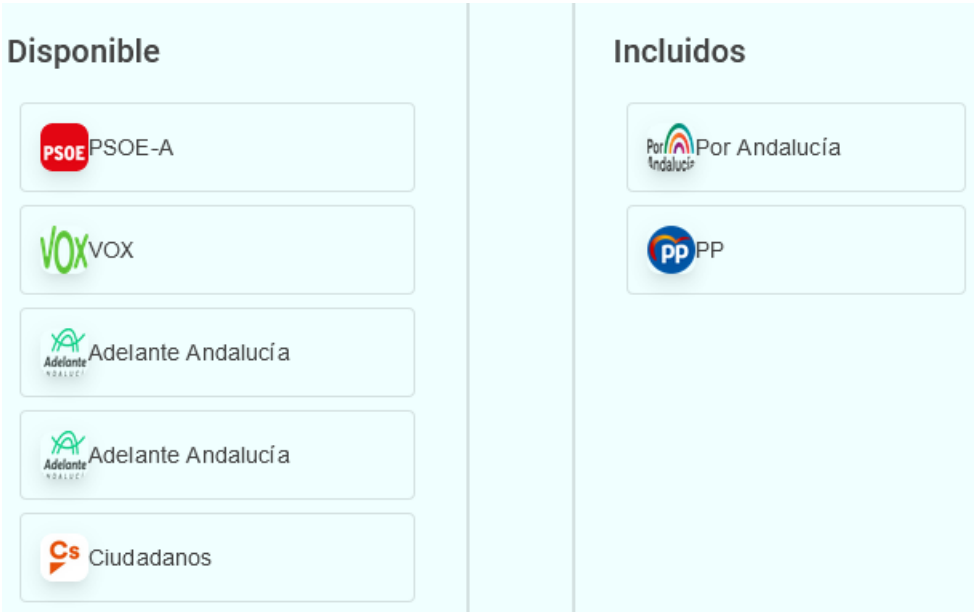

Figura A.12: Panel selección entidades participantes

A.7 Pantalla usuarios

En la pantalla de usuarios se podrán ver los usuarios que se han registrados y que están pendientes de ser validados. Una vez sean validados podrán acceder.

Una vez han accedido se les asigna el rol más bajo, pero seleccionando CAMBIAR RANGO (figura A.13) se puede actualizar el rango del usuario.

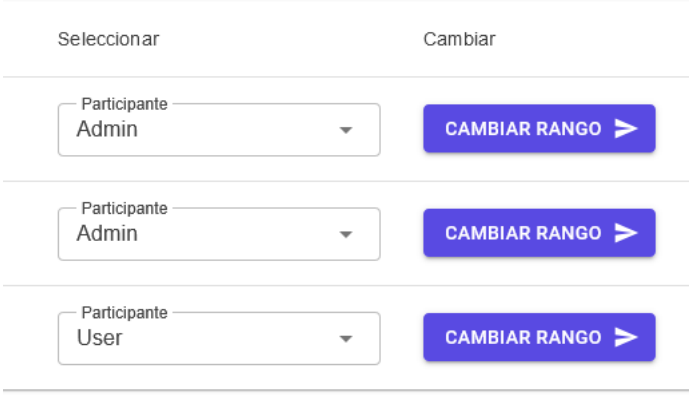

Figura A.13: Botones para cambiar el rol de usuarios

A.8 Pantalla validación

Cada vez que un usuario añada información desde una fuente, lo que entendemos como registros, aparecerán en esta pantalla (figura A.14). Un administrador deberá revisarlos para decidir si los valida. Una vez sean validados aparecerán en los gráficos.

Figura A.14: Botones para validar registros de fuentes

A.9 Pantalla Registros Manuales

Para registrar datos ofrecidos por las fuentes se ha de seleccionar un tipo, edición, fuente y fecha. Una vez se han seleccionado estos datos se cargará el formulario de la imagen A.16 adaptado a cada edición. Por lo que se tendrán que indicar los datos de cada medida por cada entidad participante. Si se selecciona Sincronizar valores los datos de medidas relacionadas se actualizarán en base a sus rangos en la medida relacionada. La fuente no siempre tiene que significa el sitio desde el cual se han recogido los datos, por lo que opcionalmente se pueden indicar esos datos en el campo de la URL así como en notas (figura A.15). Finalmente si se pulsa en guardar se guardarán los datos, que deberán ser validados por un administrador.

Figura A.15: Panel para notas

Figura A.16: Formularios de caga

Para generar fuentes se puede pulsar el botón a la derecha del campo de selección de fuente. Al pulsarlo aparecerá una pantalla modal (figura A.17) para indicar el nombre de la fuente que deseamos crear y poder ser seleccionada.

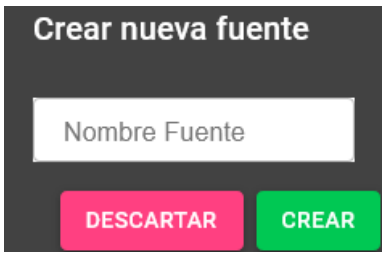

Figura A.17: Modal para crear fuente

A.10 Pantalla Registros Guiados

En esta pantalla la introducción de registros desde fuentes busca facilitar el trabajo en la medida de lo posible. Al indicar una dirección web y pulsar BUSCAR INFORMACIÓN (figura A.18) la aplicación buscará tablas en la estructura de la página web y las mostrará en pantalla en forma de tabla configurable (figura A.19). El objetivo de esta carga guiada es que el usuario sólo relacione correctamente los datos y corrija los que sobran sin tener que añadirlos todos manualmente. Pulsando sobre los iconos con una papelera se retirarán columnas y filas, correspondientes. Los que se dejen deberán asignarse a entidades participantes y medidas.

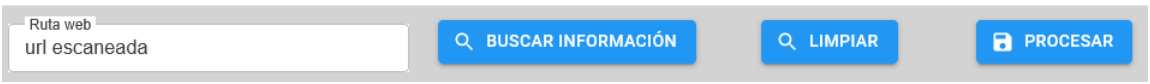

Figura A.18: Formularios de caga

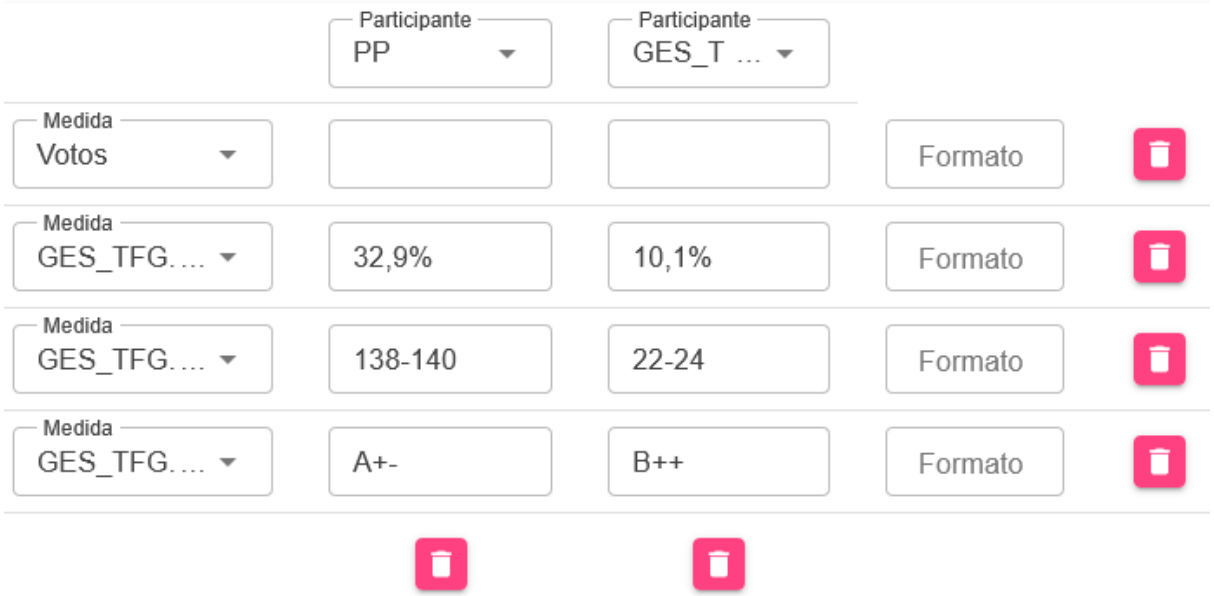

Figura A.19: Formularios de caga

El campo "formato"sirve para adaptar los datos a un formato que pueda ser entendido por el sistema. Es tan simple con indicar en qué lado del texto se encuentra el rango algo y el bajo y los caracteres entre medias. Por ejemplo, si tenemos el texto

```
aaaaa(5)-(7)fgsfdg
```

identificamos que 5 es el rango bajo y 7 el alto, por lo que la parte baja B estará a la izquierda y la alta A a la derecha. Y entre ambos valores están los caracteres ")-(", por lo que en el campo formato hay que indicar

```
{B})-({A}
```

Al pulsar sobre LIMPIAR (figura A.18) el dato se convertirá en 5-7, que es un formato entendible por el sistema. Si el dato no tiene rango no hace falta poner formato, el sistema lo limpiará automáticamente.

Finalmente si se pulsa en GUARDAR (figura A.18) el registro se guardará.

A.11 Visualizador sin sesión

Cuando se entra en la aplicación web sin iniciar sesión por defecto sólo se podrán ver los datos ya introducidos, a falta de algunos datos ocultos para miembros no registrados.

En la imagen A.20 se puede observar una entidad participante comparada a lo largo de los datos ofrecidos por distintas fuentes en una edición.

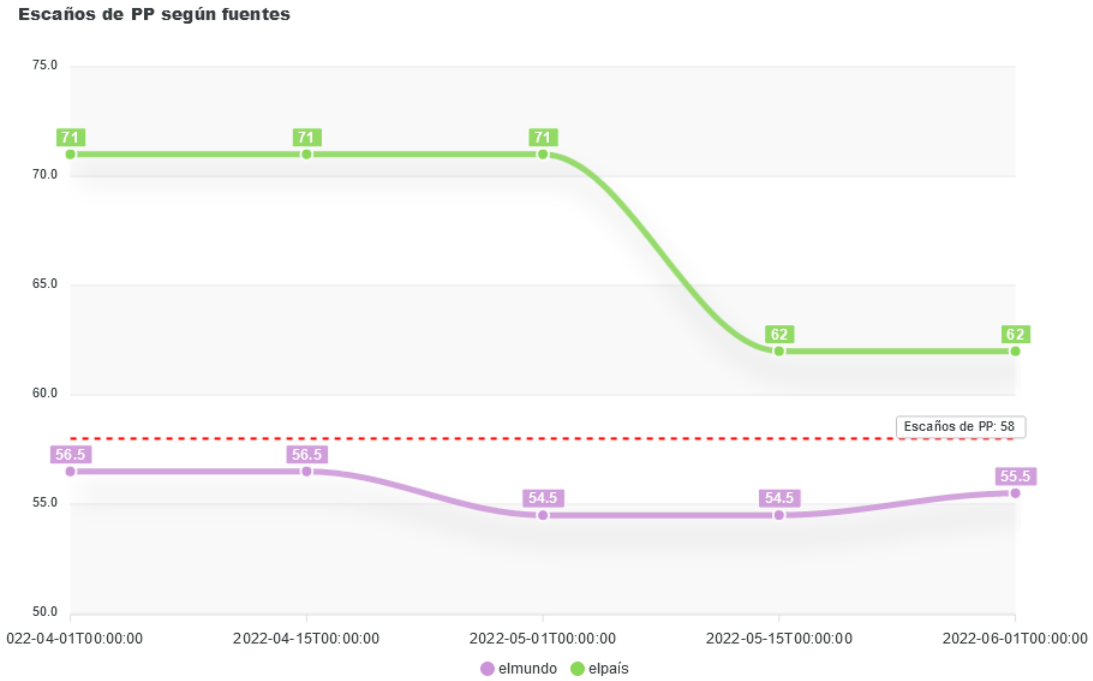

Figura A.20: Entidad participante según distintas fuentes

En la imagen A.21 se pueden ver las distintas entidades participantes de una edición según los datos aportados por una fuente.

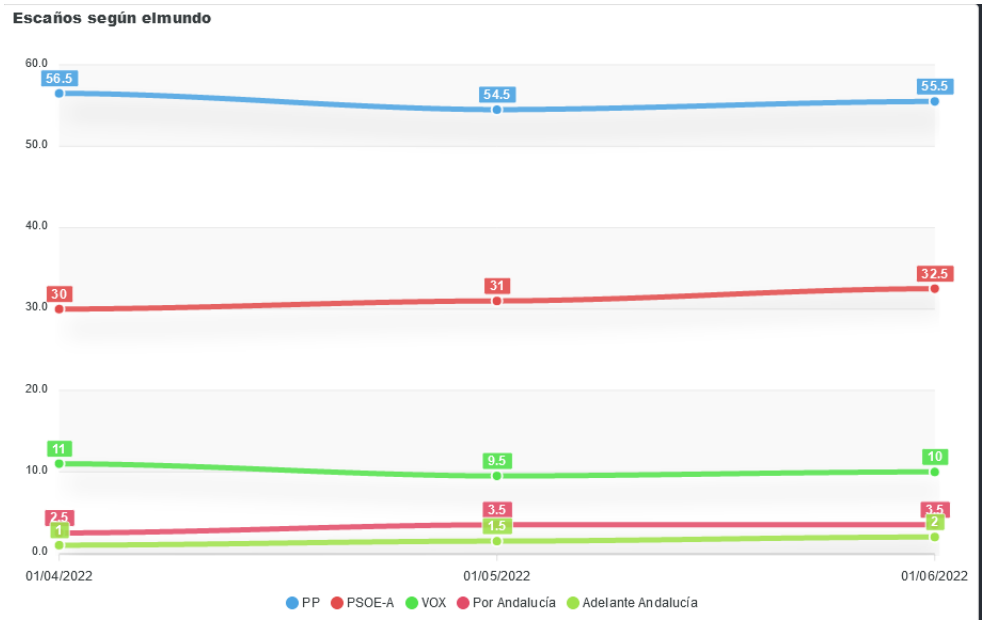

Figura A.21: Entidad participante según una fuente

La imagen A.22 mostrará el detalle del registro seleccionando, indicando los datos de la fuente en una medida concreta.

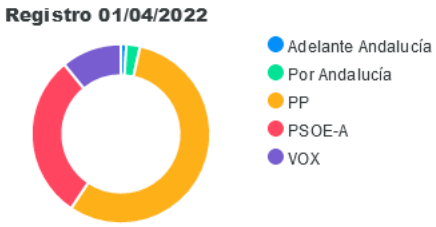

Figura A.22: Detalle del registro seleccionado

Si la edición tiene resultados en la imagen A.23 se mostrarán los relacionados con la medida seleccionada.

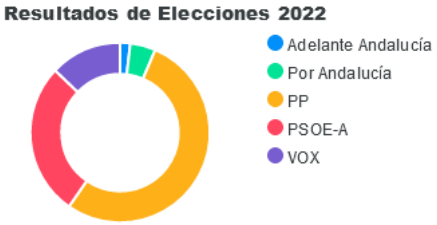

Figura A.23: Detalle de los resultados de la edición

Finalmente en la tabla mostrada en la imagen A.24 se podrá ver qué fuente ofrece menos desviación respecto a los valores finales siendo el valor más cercano a 0 el que mayor acierto suponga. Por lo que cuanto más alto sea el valor menos certeros serán los datos ofrecidos por dicha fuente.

Figura A.24: Fiabilidad de las fuentes

A.12 Visualizador con sesión

Una vez iniciada la sesión los gráficos disponibles en la pantalla serán los ya presentes en la vista sin sesión (figura A.11) además de los siguientes gráficos.

El gráfico de la figura A.25 unifica todas las ediciones de un tipo, mostrando una entidad participante a lo largo de las distintas ediciones. Dependiendo los rangos de las ediciones el dato mostrado podrá ofrecer datos confusos.

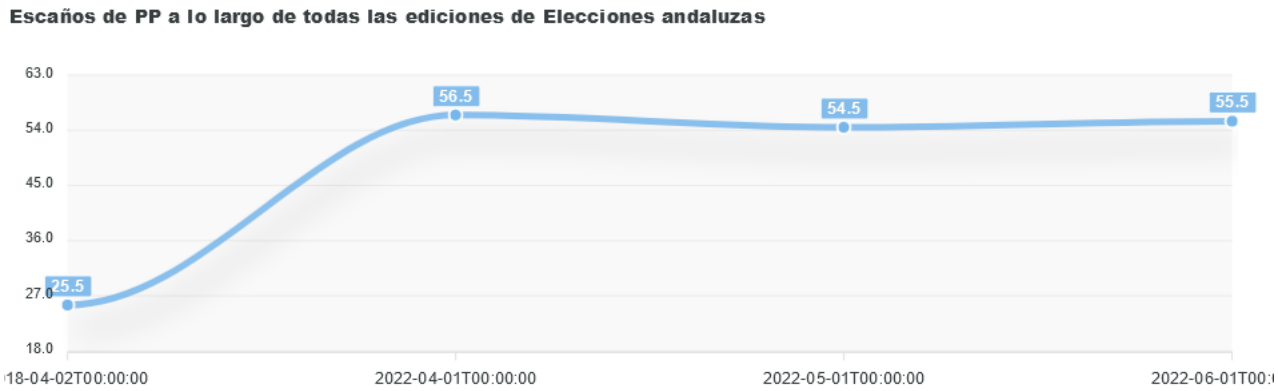

Figura A.25: Gráfico histórico entidad participante

La tabla A.26 muestra un resumen de todo por medida.

RESUMEN | FIABILIDAD MEDIDA | FIABILIDAD TODO

Edición	Fuente	Medida	Participante	Desviación
2022	elpaís	Escaños	Adelante Andalucía	0
2022	elmundo	Escaños	Adelante Andalucía	0,46
2022	elpaís	Escaños	Por Andalucía	1,38
2022	elmundo	Escaños	Por Andalucía	1,68
2022	elmundo	Escaños	PSOE-A	1,68
2022	elmundo	Escaños	PP	2,29

Filas por página 10 1-10 of 10

Figura A.26: Fiabilidad de todos los datos por medida

La figura A.27 muestra una tabla con el detalle de todos los datos.

RESUMEN | FIABILIDAD MEDIDA | FIABILIDAD TODO

Edición	Fuente	Medida	Participante	Desviació
2022	elpaís	Escaños	Adelante Andalucía	0
2022	elmundo	Escaños	Adelante Andalucía	0,46
2022	elmundo	Votos	Adelante Andalucía	0,5
2022	elpaís	Votos	Por Andalucía	0,88
2022	elpaís	Escaños	Por Andalucía	1,38
2022	elpaís	Votos	Adelante Andalucía	1,43
2022	elmundo	Votos %	Por Andalucía	1,5
2022	elmundo	Votos	VOX	1,52
2022	elmundo	Votos	Por Andalucía	1,58
2022	elmundo	Votos %	VOX	1,67

Filas por página 10 1-10 of 30

Figura A.27: Gráfico histórico entidad participante

En conjunto, las tres tablas ofrecen una visión sobre la fiabilidad de todas las fuentes y dónde han estado más acertada o menos.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá