eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Facultad de Informática

Grado de Ingeniería Informática

Trabajo Fin de Grado

Ingeniería de Software

Portfolio: Una plataforma para gestionar las finanzas personales.

Xabier Azabal Gómez

Septiembre 2021

Índice

Lista de Figuras y Tablas

FIGURAS

TABLAS

1. Introducción

En este capítulo se hará una breve introducción al proyecto, poniendo en contexto la situación del proyecto respecto al alumno y la organización que seguirá la memoria.

1.1. Contexto

Portfolio es una aplicación web para gestionar, estudiar o analizar las finanzas personales u oportunidades de inversión. Este capítulo describirá de donde surge la idea principal del proyecto teniendo como objeto esta aplicación e introduciremos los diferentes capítulos que nos encontraremos a lo largo de la memoria sobre cómo ha sido su desarrollo, cuál ha sido su gestión y qué tecnologías se han utilizado.

Esta aplicación web surge tras cuestionarme durante unas semanas cual iba a ser el tema de mi TFG. Desde hace un tiempo tengo como interés ver y estudiar cómo se le puede sacar partido al capital que tienes ahorrado. Es por eso que parte de mi tiempo me lo paso estudiando empresas o formándome en el tema. Además, como ya empecé a invertir mi capital en diferentes activos, tuve la necesidad de tener una herramienta para registrar todos los movimientos hechos y así de paso ver el rendimiento de los mismos.

En este punto observe que ninguna de las plataformas gratuitas que probaba me satisfacía al 100%, ya que algunas disponían de cosas interesantes con las que quería contar, pero no contaban con otras características que tenían las demás. Es por eso, que aprovechando que tenía que hacer el TFG, decidí hacer una plataforma uniendo las características que a mi mejor me venían, y haciendo un plataforma fácil y útil.

1.2. Organización de la memoria

- En los primeros capítulos Antecedentes y Alcance del proyecto se analizarán los conocimientos personales con los que se inició el desarrollo del proyecto, y también cual sería el objetivo a lograr en la plataforma para obtener una experiencia de usuario apropiada y un funcionamiento correcto.
- Después, en los siguientes capítulos Análisis y diseño, Tecnologías e Implementación se analizará que tecnologías se han elegido y por qué, y como se han aplicado en la solución. Concretamente, en el capítulo Análisis y diseño se expondrá el modelo de dominio que se ha definido para la aplicación. También se expondrá el modelo de casos de uso acompañado de todo el funcionamiento. En cuanto al capítulo Tecnologías, se analizarán las diferentes tecnologías (Angular, ASP.NET Core, Bootstrap…) usadas en el proyecto y las razones por las cuales se han elegido, así como las ventajas que nos proporcionan.
- Más adelante, en el capítulo Implementación se describirá la implementación realizada lograda a partir de los resultados de análisis previamente realizados. Finalmente, en el capítulo Pruebas, se llevará a la practica el uso de la plataforma y se analizarán los resultados de usabilidad de personas diferentes.
- Para finalizar, en los últimos capítulos Gestión y seguimiento y Conclusiones, hablaremos de como se ha llevado a cabo el proyecto y cuáles son los resultados, conclusiones y aprendizajes que obtenemos de todo el proceso.

2. Antecedentes

En este capítulo se expondrán los antecedentes del proyecto. Concretamente, se hablará de la descripción del proyecto, el interés previo que había en el tema del proyecto, las motivaciones y una pequeña investigación realizada sobre aplicaciones similares.

2.1. Descripción

Este proyecto tiene como objetivo el diseño y desarrollo de una herramienta de gestión de carteras personales de inversión, en este caso de acciones de empresas. Esta debe almacenar y permitir la correcta gestión de diferentes transacciones que el usuario podrá registrar en sus carteras. Así mismo, debe hacer accesible el control y seguimiento de dichas carteras, mostrando diferentes parámetros como rentabilidades, dinero invertido, diversificación, riesgo… para conseguir como objetivo un control total sobre sus inversiones.

Por otra parte, la herramienta tendrá un apartado donde se podrán filtrar y revisar individualmente las empresas para ver si sus acciones son una inversión adecuada o no. Para esto se le permitirá al usuario acceder a diversa información y múltiples métricas de la economía y el progreso de dichas empresas, así como su precio en tiempo real.

A su vez, el sistema contará con una sección de fiscalidad donde el usuario introduciendo sus datos podrá crear informes anuales de sus movimientos, para facilitar la tributación de las mismas.

El objetivo final es crear una plataforma de gestión y análisis de inversiones completo que permita al usuario llevar de manera más fácil el control de sus inversiones, para así poder optimizar las mismas.

2.2. Interés previo al proyecto

A la hora de la realización del proyecto, incluso antes de decidir el tema principal del mismo, era que la aplicación iba a ser una aplicación WEB y que iba a usar unas tecnologías concretas. Después de toda la carrera y los aprendizajes recibidos, y también del tiempo que llevo trabajando desarrollando aplicaciones WEB entre otras cosas, consideraba que tenía unos conocimientos y unas bases bastante buenas para desarrollar el trabajo con estas tecnologías. También, sabía que, de haber algún problema relacionado con el desarrollo, iba a saber desenvolverme para solucionarlo.

Por otra parte, al estar familiarizado con el desarrollo de aplicaciones WEB, soy sabedor de todas las ventajas que tienen estas frente a otras metodologías. Entre otras, la ventaja principal de una aplicación WEB es la facilidad que tiene el usuario de acceder a ella. El usuario solo necesita un dispositivo conectado a internet y un navegador WEB para poder usar la aplicación. Sin duda esta accesibilidad era algo con lo que quería que mi proyecto contase.

2.3. Aplicación a desarrollar y motivaciones

Para sacarle partido a la accesibilidad multiplataforma anteriormente comentada, empecé a pensar sobre qué idea podría realizarse el trabajo. Fue ahí cuando me di cuenta de que tenía problemas a la hora de gestionar mis inversiones. Por más que probaba aplicaciones para registrar los movimientos realizados en bolsa, ninguna era acorde a mis necesidades. Algunas tenían unos usos muy concretos, otras estaban muy limitadas, otras eran de pago, etc. Además, cuando encontré una que me gustó bastante, resultó que solo había versión de escritorio, y no se podía

acceder desde el teléfono móvil. Fue ahí cuando decidí hacer una plataforma que se pudiese acceder desde cualquier dispositivo.

En cuanto a la estructura de los datos que se iban a usar, aunque por mis conocimientos tenía bastante claro como tenía que conformar la base de datos, ha habido que hacer un estudio mínimo sobre cómo funcionan los diferentes mercados bursátiles. Por otro lado, como la aplicación iba a ser ejecutada en diferentes dispositivos, iba a tener que enfrentarme a un problema bastante común en la realización de estas aplicaciones, y es que la interfaz iba a tener que ser usable en todos los dispositivos, independientemente de la resolución y tamaño de la pantalla.

2.4. Análisis de aplicaciones similares

En el mercado hay infinidad de aplicaciones de las características de la nuestra, con usos muy diferentes las unas de las otras. Es por eso que, para desarrollar una aplicación de este tipo, se precisaba un estudio de aplicaciones similares. Con este estudio se conseguiría un enfoque más amplio del alcance de nuestro proyecto y se podría enfatizar el esfuerzo en implementar funcionalidades que no son comunes y que pueden ser muy útiles para el usuario.

En este caso, se hizo un análisis a la funcionalidad de las aplicaciones Yahoo Finance, Ticker y SigFig. Esta elección es debida a que estas tres aplicaciones son las que yo más había usado y más me habían gustado hasta la fecha. Estas son las características principales de cada una:

- Yahoo Finance: Esta aplicación es probablemente la más conocida y usada en cuanto a información y gestión bursátil se refiere. Tiene un diseño muy fácil de usar, por lo que se pueden rastrear fácilmente sus acciones, materias primas, bonos y divisas. Puedes recibir noticias y alertas personalizadas y seguir los movimientos del mercado en tiempo real. También puedes crear una lista de favoritos y obtener las cotizaciones en vivo mientras monitoreas el desempeño de su cartera. A su vez, es uno de los sitios de noticias empresariales más grandes de EE. UU, con datos, comentarios y comunicados de prensa entre su contenido diario. El punto fuerte de Yahoo Finance es sin duda la cantidad de datos que tiene. Cualquier dato histórico al que quieras acceder de alguna acción y que no esté en las demás plataformas, está en Yahoo Finance, y eso es un punto diferenciador. Sin embargo, se queda atrás en cuanto a diseño y da la sensación de que es una aplicación vieja. Otro punto en contra es que, a la hora de analizar tu cartera, no dispones de casi métricas y gráficos para obtener una información más detallada.
- Ticker: Esta aplicación, a diferencia de la anterior, está más centrada en la gestión de carteras personales, y no tanto en la información de acciones. Ticker te permite administrar múltiples carteras de acciones desde un solo panel. En cuanto a diseño es una aplicación muy en línea con lo que se quiere conseguir en el proyecto: cuadros coloridos, gráficos y análisis detallados, valores en tiempo real, ganancias/pérdidas, etc. Además, también cuenta con noticias relevantes para las diferentes acciones. Ingrese información comercial manualmente para acciones. También puedes estar atento a las acciones con múltiples listas de observación y creando alertas para notificarle si una acción cotiza por encima o por debajo de un nivel de activación establecido, según los

cambios de precio, volumen y porcentaje. Esta aplicación es lo opuesto a Yahoo Finance, y es que, aunque es una aplicación excelente para la gestión de carteras, le faltan infinidad de datos de los mercados.

- SigFig: Esta aplicación es muy similar a la anterior. Un diseño muy cuidado y muy orientada a la gestión de carteras. Un punto diferencial de SigFig frente a Ticker es la posibilidad de configurar resúmenes semanales que te lleguen al correo electrónico. Estos resúmenes se basan en tu elección de sectores y acciones. De las tres, esta es la que más afín es a lo que yo buscaba, pero el problema principal es que para disfrutar de todas las funcionalidades de la aplicación había que pagar una cuota mensual, la versión gratuita se quedaba muy corta.

De este análisis obtuve la conclusión de cómo iba a ser la aplicación a desarrollar. Primero, iba a estar orientada a la gestión de carteras, como Ticker y SigFig, pero incluyendo algún grafico con el que estas dos no contaban y consideraba útil. Por otra parte, también quería que incluyera una gran cantidad de datos y variados como tiene Yahoo Finance y como no tiene Ticker y SigFig. Para conseguir eso, se decidió incluir en la aplicación la API de Yahoo Finance, que es una API accesible para todo el mundo de la cual se pueden obtener todos los datos con los que cuenta Yahoo Finance.

Sobre todo, en la aplicación quería contar con una funcionalidad que en ninguna plataforma la he visto. A menudo, los inversores se encuentran con muchos problemas a la hora de declarar las acciones que tienen a hacienda. Las inversiones son un bien patrimonial del que se pueden sacar beneficios, por consiguiente, hay que declarar todos los movimientos realizados en los diferentes mercados. A la hora de hacer estos informes, resulta muy engorroso obtener y clasificar los movimientos por fecha, precio, mercado, etc. Es por eso, que se buscó una plataforma que hiciera esto automáticamente sobre tus carteras, sin éxito. Por lo tanto, la plataforma a desarrollar tendría que contar con esta característica la cual no es muy común en este tipo de aplicaciones.

3. Objetivos y alcance del proyecto

Este capítulo se centrará en describir en su primera sección los objetivos planteados para la realización del TFG. Se empezará por los primeros objetivos más generales. Después, se repasarán otros objetivos más específicos. Finalmente, se ahondará en los objetivos más concretos con los que se espera que la aplicación finalmente cuente.

3.1. Objetivos

Los objetivos principales para el desarrollo de este proyecto serán los siguientes:

- Desarrollar una aplicación WEB accesible desde diferentes dispositivos.
- Desarrollar una aplicación priorizando la experiencia de usuario (usabilidad, diseño, rapidez…).
- Pruebas y desarrollo de la aplicación de manera local.
- Despliegue de la aplicación WEB en servidor en la nube y pruebas del servidor.
- Despliegue de la API en servidor en la nube y pruebas del servidor

En cuanto a las tecnologías elegidas los objetivos específicos que se necesitarán satisfacer para el proyecto serán los siguientes:

- Desarrollo de la lógica de la aplicación en el framework[1] Angular.
- Creación de un entorno seguro para el usuario y sus datos (autenticación, cifrado de contraseña).
- Desarrollo de una API independiente para la obtención de datos en ASP.NET Core.
- Análisis y diseño de una base de datos orientada a las finanzas en SQL.
- Diseño de la interfaz usando estilos y Bootstrap
- Integración de la API de Yahoo Finance para la obtención de datos.
- Integración de gráficos para el análisis de carteras por medio de Amcharts.

En relación a las tecnologías elegidas los objetivos específicos que se necesitarán satisfacer para el proyecto serán estos:

1. El objetivo principal del proyecto será crear una plataforma WEB orientada a inversores para que registren los movimientos que realicen con sus activos, analicen diferentes acciones y gestionen la fiscalidad.
2. La plataforma contará con una estética y diseño cuidada y llamativa para el usuario.
3. Todo el tratado de datos se hará de forma segura, así como la relación entre las diferentes partes de la arquitectura del proyecto. Deberá asegurar conexiones seguras entre el usuario y el servidor, y del servidor con la base de datos.
4. Todas las llamadas a la API serán de tipo POST para aumentar la seguridad.
5. La WEB contará con un registro y un login para que el usuario pueda acceder a la plataforma con sus datos. La contraseña tendrá que seguir una serie de reglas y se guardaran encriptadas para mantener la integridad de las mismas.
6. La aplicación contara con un menú superior con los diferentes apartados donde el usuario puede acceder. También, tendrá un indicador de usuario con sus iniciales donde podrá cambiar la configuración de la cuenta o cerrar sesión. Este menú también contará con un

[1] Framework: Es un entorno de trabajo que tiene como objetivo facilitar la labor de programación ofreciendo una serie de características y funciones que aceleran el proceso, reducen los errores, favorecen el trabajo colaborativo y consiguen obtener un producto de mayor calidad.

buscador para poder buscar las empresas que quiera y solo será visible una vez se ha producido un login.

7. La información de los usuarios estará compuesta por un nombre de usuario, un nombre de pila, un apellido y una contraseña como campos obligatorios. Una vez registrado podrá añadir más datos para la fiscalidad (identificación, dirección, teléfono…).
8. Los usuarios de la aplicación podrán tener tres roles diferentes. El primero es el administrador que tendrá acceso a todas las funcionalidades. El segundo es el usuario registrado que podrá utilizar todo lo que no tenga que ver con gestión de entidades. El tercero es el usuario sin registrar que no podrá acceder a ninguna funcionalidad hasta que cree una cuenta.
9. El usuario de tipo administrador podrá gestionar las empresas. Podrá eliminar las empresas que se hayan disuelto, podrá editar los datos de las empresas que hayan cambiado algún campo (nombre, ubicación, divisa…) y podrá crear nuevas empresas.
10. La página principal será la página "carteras", ahí cada usuario podrá crear múltiples carteras. Todas estas carteras se podrán editar o eliminar en cualquier momento. Los usuarios también tendrán la posibilidad de hacer públicas las carteras que quieran para compartirlas con otros usuarios.
11. En la página principal se mostrará un tutorial para que el usuario se pueda manejar mejor por la plataforma.
12. La página de "cartera" mostrará todos los datos y estadísticas de una cartera a la que el usuario ha accedido. En esta página será donde el usuario cree/edite/elimine todas las transacciones realizadas.
13. La página "empresas" mostrará la lista de empresas que tenemos en la plataforma. En esta lista de empresas podremos ver, ordenar y filtrar las empresas por diferentes campos (nombre, país, sector, industria, precio en tiempo real…).
14. La página "empresa" mostrará toda la información relacionada a una empresa (localización, número de empleados, métricas de valoración y muchas cosas más). También podremos ver el histórico de precio en dos gráficos distintos (uno de velas y otro de líneas). Aquí el usuario podrá añadir la empresa a favoritos o quitarla si ya la tenía.
15. La página "favoritos" mostrará una lista de empresas que el usuario ya haya añadido a favoritos. Aquí también se podrán filtrar por diferentes campos y se podrán eliminar directamente de favoritos.
16. La página "fiscalidad" aunará todo lo relacionado con la fiscalidad. Contará con dos apartados. El primero será para poder guardar los datos relacionados con la fiscalidad (identificación, dirección, teléfono…). Una vez añadidos los datos fiscales se podrán generar informes de las carteras que queramos en los años que estuvieron activas. Estos informes servirán a la hora de declarar los activos.

3.2. Alcance del proyecto

En este apartado describiremos los diferentes apartados a completar para poder realizar el proyecto. Estos apartados se irán desarrollando de manera simultánea los unos con los otros. Estos son los apartados a realizar:

- **Gestión:** Este apartado contiene el trabajo que se realizará relacionada con la planificación del proyecto, el seguimiento, estudio y la obtención de diferentes opiniones. Con esto se intentará tener un aprendizaje para poder llevar a cabo la realización del proyecto.
- **Aplicación WEB**: Este apartado cubrirá todo lo relacionado con la aplicación. Aquí dentro irá todo el análisis de requisitos, incluyendo el modelo de dominio y el modelo de casos de uso. También contendrá el diseño de la aplicación, la implementación y todo lo relacionado con las pruebas.
- **Memoria del proyecto:** Este apartado de trabajo contendrá todo lo referente a la realización de la memoria del proyecto.
- **Defensa del proyecto:** De cara a defender el proyecto, este paquete cubrirá todo lo relacionado con la documentación del proyecto y la defensa del mismo.

A continuación, se muestra el alcance que tiene nuestro proyecto a partir de un EDT (estructura de desglose de trabajo). Esto esquematizara la estructura de los apartados a realizar en nuestro proyecto:

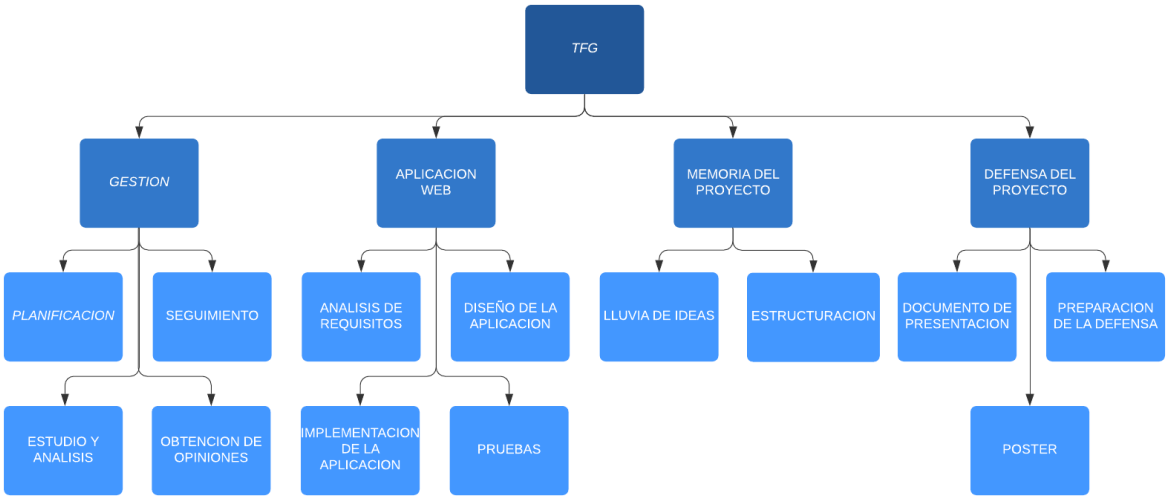


Figura 1. Esquema de Desglose de Trabajo del proyecto

4. Análisis de requisitos

En este capítulo realizaremos una visión global sobre aquellos casos de uso que forman nuestra aplicación entendiendo el flujo de los mismos. Así, se describirá la lógica del sistema, esto es, toda la documentación necesaria para recopilar la información sobre lo que se desarrollará. Para ello, se describirán los siguientes puntos: modelo de casos de uso, descripción de los mismo y dominio E-R.

4.1. Modelo de casos de uso

La aplicación según el análisis de requisitos realizado tendría el siguiente modelo de casos de uso. En este apartado se verán los papeles que desempeñan en la aplicación los tres roles (Invitado, Usuario y Administrador) junto a aquellos casos de uso que los componen.

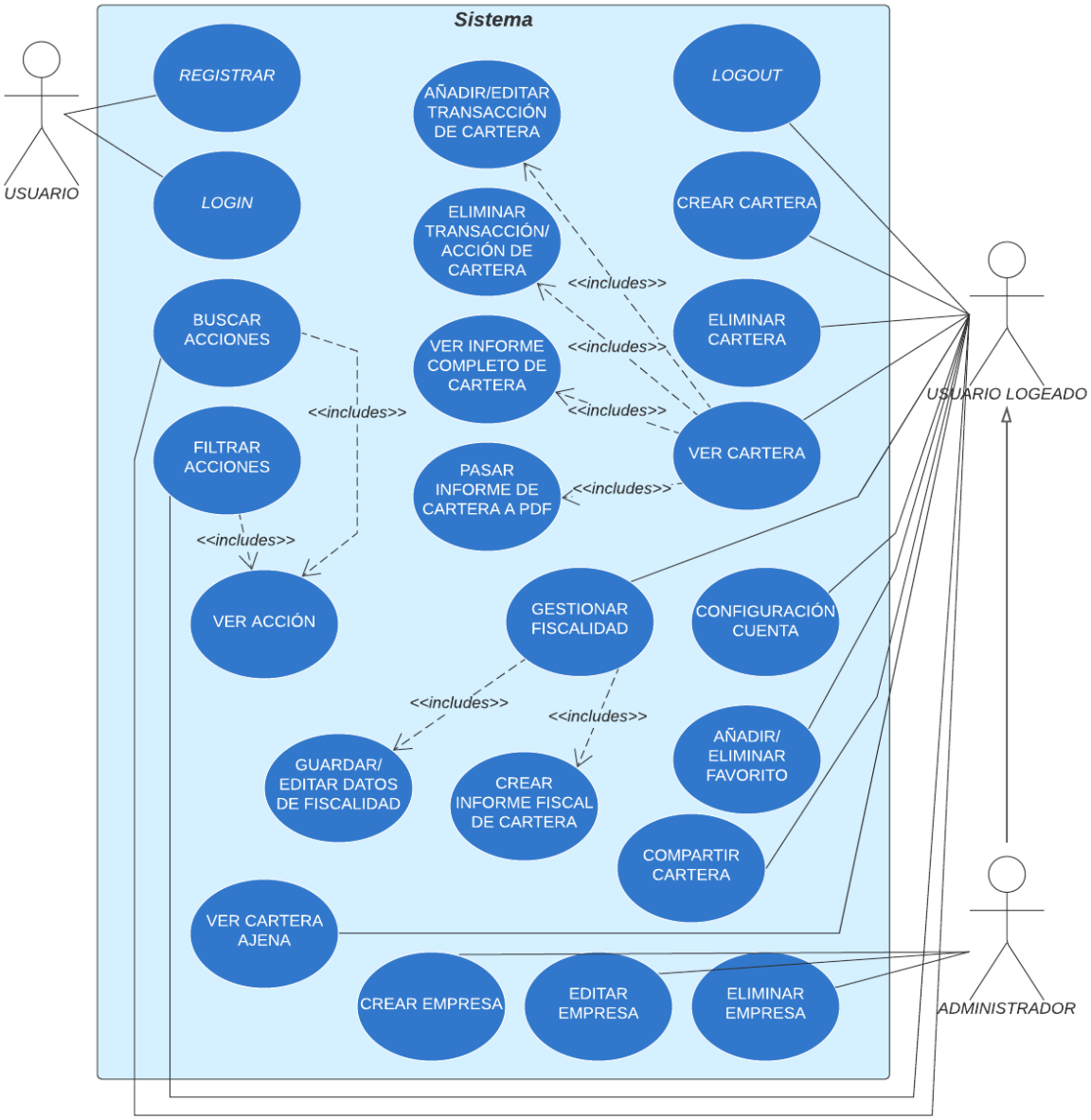


Figura 2. Modelo de casos de uso de la aplicación

4.2. Descripción de los actores

En la aplicación podrán interactuar tres tipos de actores, el usuario sin loguear, el usuario logueado y el administrador.

1. El usuario sin loguear es cualquiera que entra a la web sin haber hecho un login previo. Este actor tendrá la posibilidad de crear una cuenta o de acceder con su cuenta si ya la tuviera creada. No tiene acceso a ninguna funcionalidad más.
2. El usuario logueado es un usuario él cual ya tiene una cuenta previamente creada y ha entrado a la web con sus credenciales. Este usuario tiene acceso a todas las funcionalidades de la web orientadas al cliente.
3. El administrador es un usuario logueado con más permisos, por lo tanto, hereda todas las funcionalidades a las que puede acceder un usuario logueado corriente. Además de eso, un administrador también tendrá acceso a crear, eliminar o editar empresas de todo el sistema.

4.3. Descripción de casos de uso

A continuación, se van a describir los casos de uso, es decir, se va a describir la acción o actividad que lleva a cabo cada caso de uso. Con esto, especificaremos la comunicación y el comportamiento del sistema mediante su interacción con los usuarios. Por cada caso de uso se mostrará la siguiente información:

- Nombre: Es el nombre que le hemos dado al caso de uso.
- Actor: Es el actor que puede llevar a cabo el caso de uso.
- Descripción: Es una breve explicación de lo que hace el caso de uso.
- Precondiciones: Son las condiciones previas que tiene que cumplir el usuario o sistema para poder llevar a cabo el caso de uso.
- Flujo normal: Son los pasos ordenados que sigue el conjunto sistema/actor para llevar a cabo el caso de uso.
- Flujo alternativo: Son las posibles vertientes no comunes del flujo normal.
- Postcondiciones: Son el resultado de haber completado el caso de uso.

4.3.1. Rol de usuario sin loguear

Registro:

Nombre	Registrar
Actor	Usuario sin loguear
Descripción: Permite a un usuario crear una cuenta en la plataforma para acceder a sus funcionalidades.	
Precondiciones: – No puedes estar logueado en el sistema.	
Flujo normal: 1. El actor pulsa sobre el botón para dirigirse a la página de registro. 2. El sistema muestra un formulario para introducir los datos de la cuenta que creará el actor. 3. El actor introduce los datos de la cuenta a crear. 4. El sistema comprueba la validez de los datos y los almacena. 5. El actor recibe un aviso de que la cuenta se ha creado correctamente y ya la puede usar.	
Flujo alternativo: 4. El sistema comprueba la validez de los datos, si los datos no son correctos, se avisa al actor de ello permitiéndole que los corrija.	
Postcondiciones: – El usuario ha sido almacenado en el sistema y puede usarse para acceder a la plataforma.	

Tabla 1. Caso de uso registro

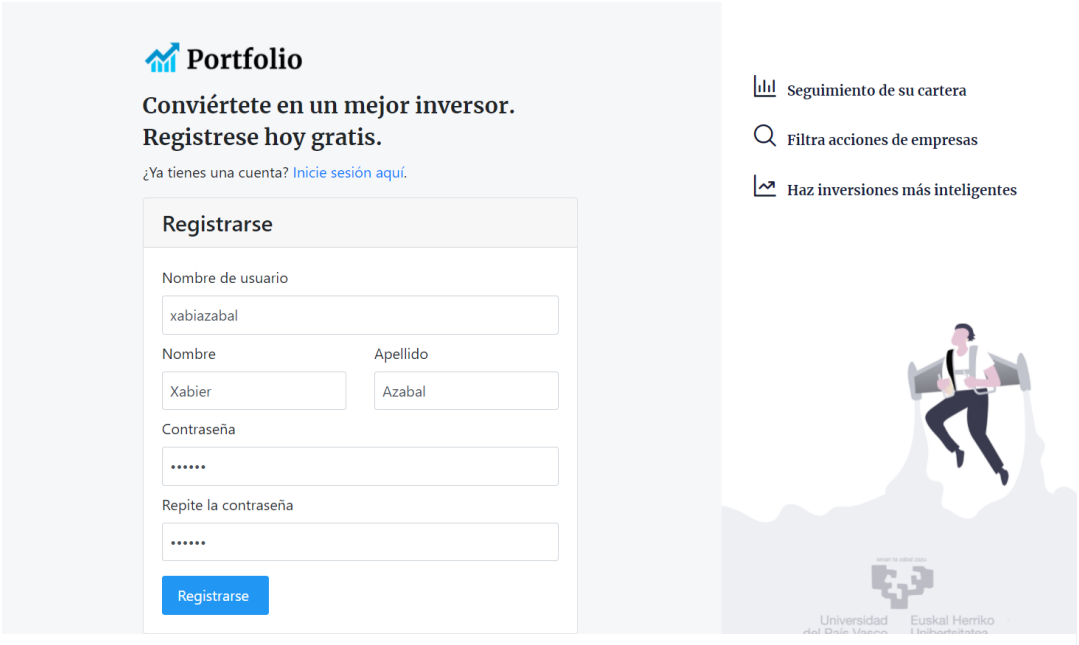


Figura 3. Interfaz de la pantalla de registro

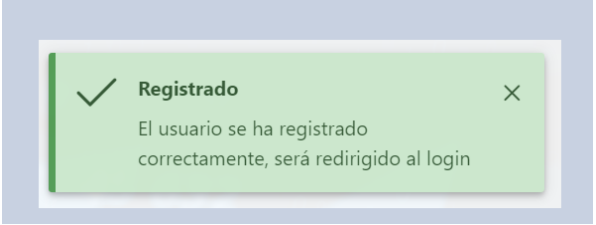


Figura 4. Mensaje que recibe el usuario al registrarse

Login:

Nombre	Login
Actor	Usuario sin loguear
Descripción: Permite a un usuario que ya tiene una cuenta registrada iniciar sesión en la plataforma.	
Precondiciones: – No puedes estar logueado en el sistema.	
Flujo normal: 1. El actor pulsa sobre el botón para dirigirse a la página de logueo. 2. El sistema muestra un formulario para introducir los datos de la cuenta con la que accederá el usuario. 3. El actor introduce los datos de su cuenta. 4. El sistema comprueba la validez de los datos y los valida. 5. El actor recibe un aviso de que la cuenta y las credenciales son correctas y accede a la plataforma. 6. El sistema guarda el usuario localmente para que pueda volver a acceder con su cuenta en otro momento sin necesidad de tener que volver a loguearse.	
Flujo alternativo: 4. El sistema comprueba la validez de los datos, si los datos no son correctos, se avisa al actor de ello permitiéndole que los corrija.	
Postcondiciones: – El usuario ha accedido a la plataforma.	

Tabla 2. Caso de uso login

Figura 5. Interfaz de la pantalla login

4.3.2. Rol de usuario logueado

Logout:

Nombre	Logout
Actor	Usuario logueado
Descripción: Permite a un usuario logueado en el sistema cerrar la sesión iniciada con su cuenta.	
Precondiciones: – El usuario debe de estar autenticado en el sistema.	
Flujo normal: 1. El actor despliega el menú superior haciendo click en las iniciales del nombre y hace click en el botón de cerrar sesión. 2. El sistema borra el objeto de usuario guardado localmente. 3. El sistema manda al actor a la página de logueo.	
Flujo alternativo:	
Postcondiciones: – El usuario ha salido de la plataforma.	

Tabla 3. Caso de uso logout

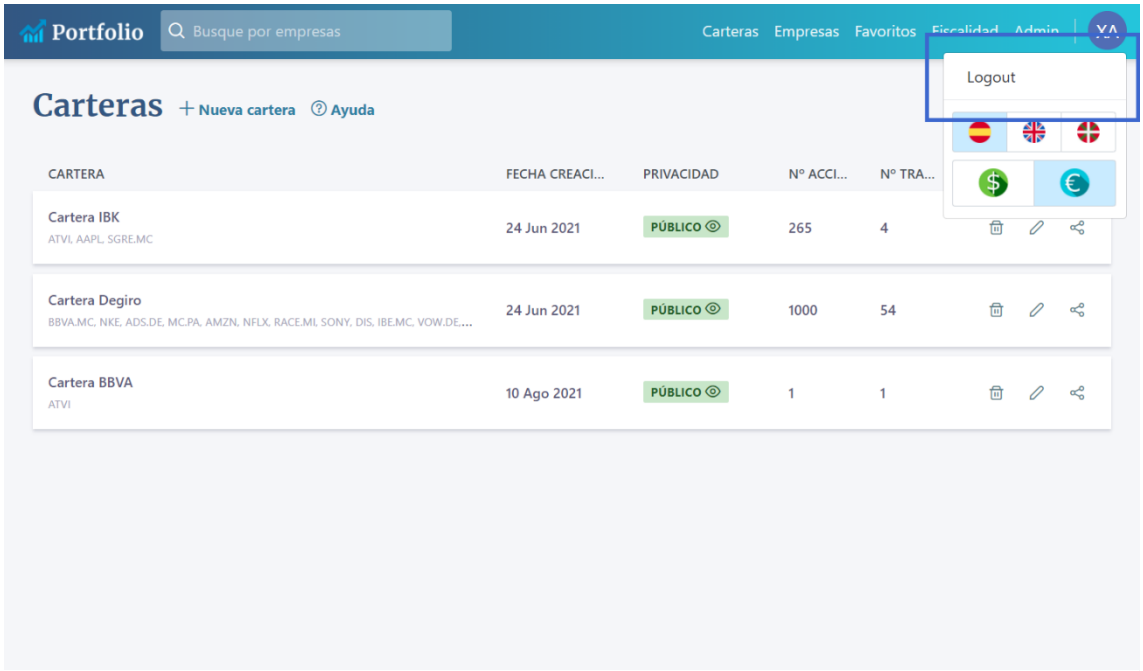


Figura 6. Interfaz de la pantalla principal con indicación de logout

Configurar cuenta:

Nombre	Configuración cuenta (cambiar idioma/cambiar divisa)
Actor	Usuario logueado
Descripción: Permite al usuario cambiar la configuración de su cuenta. En esta configuración puede cambiar el idioma de la plataforma a castellano, euskera o inglés y también puede cambiar la divisa para enseñar la cartera a dólares o euros.	
Precondiciones: – El usuario debe de estar autenticado en el sistema.	
Flujo normal: 1. El actor despliega el menú superior haciendo click en las iniciales del nombre en el menú superior. 2. El actor hace click encima de algún botón de configuración (idiomas: español, euskera o inglés, divisas: dólar o euro). 3. El sistema cambia el objeto local del usuario con la nueva configuración y también registra el cambio en la base de datos. 4. El sistema cambia los datos de la ventana en la que estemos adaptándose a la nueva configuración de idioma o divisa.	
Flujo alternativo:	
Postcondiciones: – El usuario ha cambiado la configuración.	

Tabla 4. Caso de uso configurar cuenta

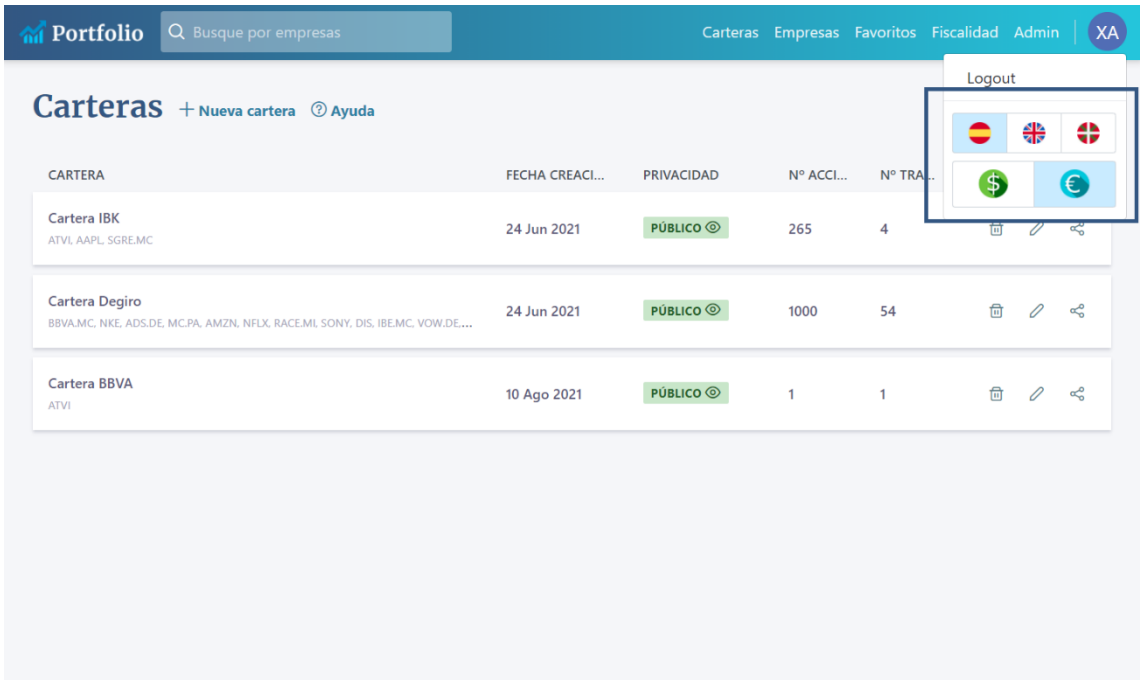


Figura 7. Interfaz de la pantalla principal con indicación de configuración

Crear cartera:

Nombre	Crear cartera
Actor	Usuario logueado
Descripción: Permite al usuario crear una cartera donde luego podrá registrar sus transacciones de acciones.	
Precondiciones: – El usuario debe de estar autenticado en el sistema.	
Flujo normal: 1. El actor entra en el apartado carteras a través del menú superior. 2. El actor hace click en el botón de crear cartera. 3. El sistema muestra un pop-up[2] con un formulario para rellenar con los datos de la cartera. 4. El actor rellena el formulario con los datos y hace click en el botón de guardar cartera. 5. El sistema comprueba la validez de los datos y guarda la cartera en el sistema. 6. El actor recibe un aviso de que la cartera se ha creado y el sistema le manda a la página de la cartera.	
Flujo alternativo: 5. El sistema comprueba la validez de los datos, si los datos no son correctos, se avisa al actor de ello permitiéndole que los corrija.	
Postcondiciones: – El usuario ha creado la cartera.	

Tabla 5. Caso de uso crear cartera

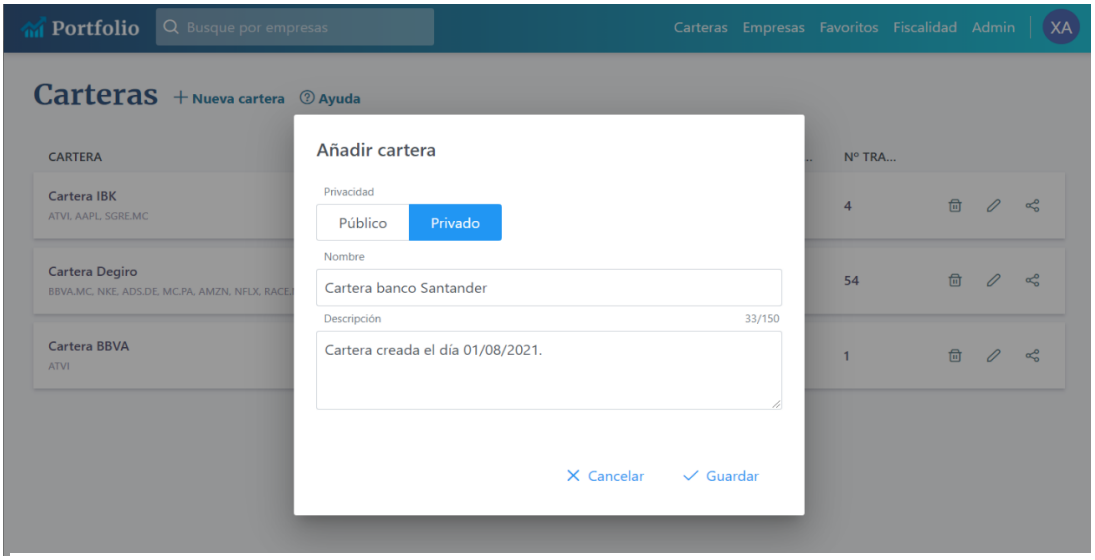


Figura 8. Interfaz de la pantalla principal con el pop-up de añadir cartera

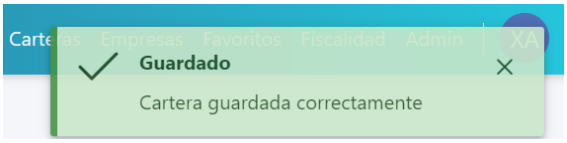


Figura 9. Mensaje que recibe el usuario al crear la cartera

[2] Pop-up: Un pop-up, también conocido como ventana emergente o ventana pop-up, es ese contenido que aparece de forma repentina en un navegador web o en la pantalla de tu ordenador.

Eliminar cartera:

Nombre	Eliminar cartera
Actor	Usuario logueado
Descripción: Permite al usuario eliminar una de las carteras que tiene y todas las transacciones de la misma.	
Precondiciones: – El usuario debe de estar autenticado en el sistema. – El usuario debe tener al menos una cartera.	
Flujo normal: 1. El actor entra en el apartado carteras a través del menú superior. 2. El actor hace click en el botón de eliminar cartera de la cartera que quiera eliminar. 3. El sistema muestra un pop-up con una pregunta para confirmar la acción. 4. El actor confirma la acción haciendo click en el botón de aceptar. 5. El sistema elimina la cartera y todos los registros asociados a ella. 6. El actor recibe un aviso de que la cartera se ha eliminado.	
Flujo alternativo: 4. El actor anula la acción haciendo click en el botón cancelar y el sistema cierra el pop-up.	
Postcondiciones: – El usuario ha eliminado la cartera.	

Tabla 6. Caso de uso eliminar cartera

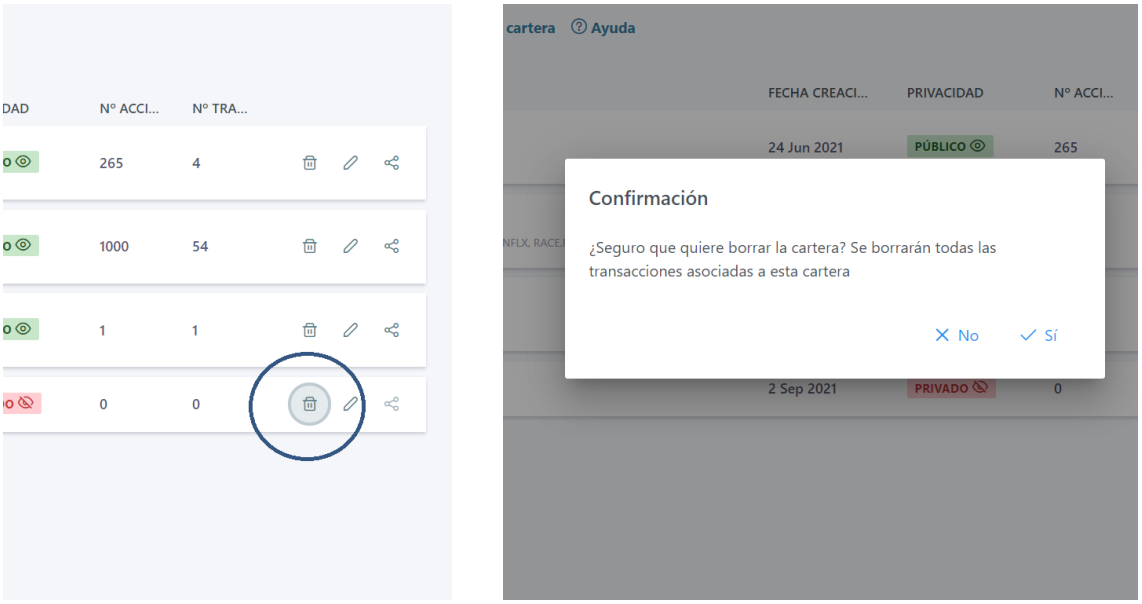


Figura 10. Interfaz de la pantalla principal con el pop-up de borrar cartera

Ver cartera

Nombre	Ver cartera
Actor	Usuario logueado
Descripción: Permite al usuario acceder a la página de información de una de sus carteras.	
Precondiciones: – El usuario debe de estar autenticado en el sistema. – El usuario debe tener al menos una cartera.	
Flujo normal: 1. El actor entra en el apartado carteras a través del menú superior. 2. El actor hace click encima de la cartera que quiera visualizar. 3. El sistema le manda a la página de la cartera. 4. El actor ahora tendrá la posibilidad de añadir/editar transacciones de cartera, eliminar transacción/acciones cartera, ver informe completo de cartera o pasar informe de cartera a PDF.	
Flujo alternativo:	
Postcondiciones: – El usuario ha entrado a la cartera.	

Tabla 7. Caso de uso ver cartera

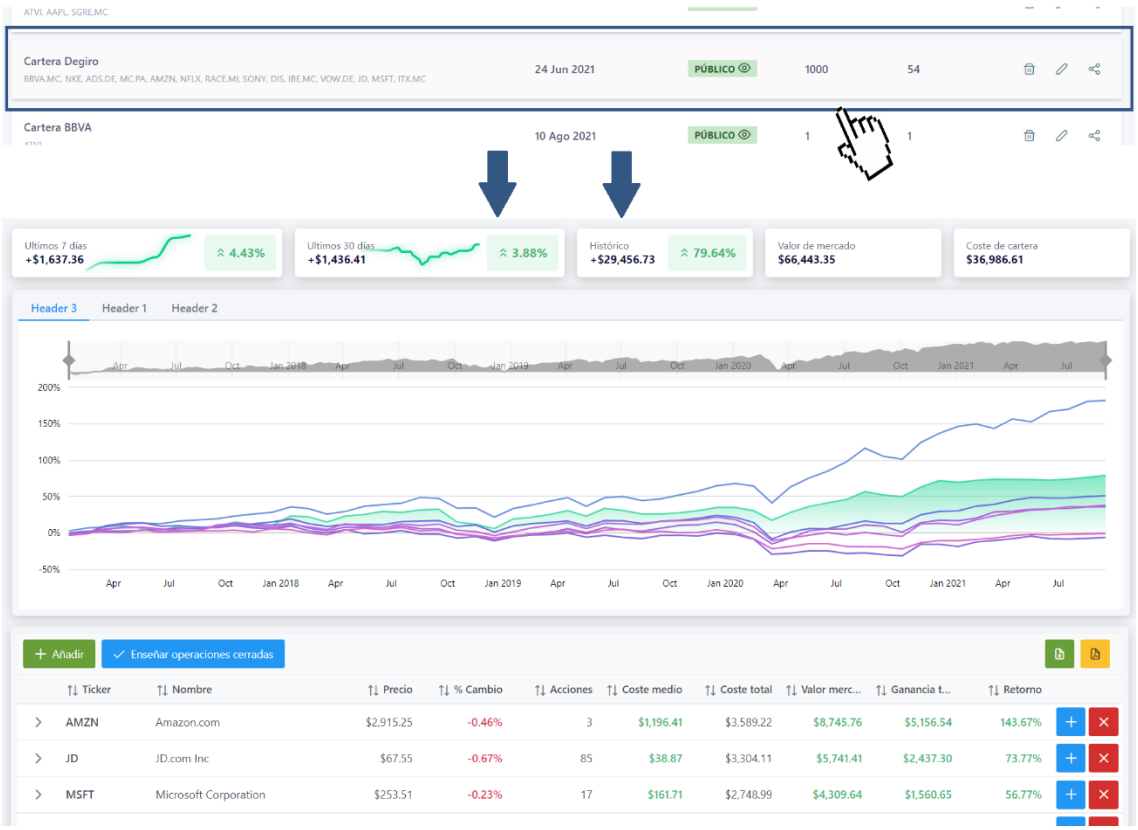


Figura 11. Interfaz de la pantalla principal para pasar a la ventana de cartera

Añadir transacción

Nombre	Añadir transacciones de cartera
Actor	Usuario logueado
Descripción: Permite al usuario incluir en una de sus carteras una transacción que haya realizado.	
Precondiciones: – El usuario debe de estar autenticado en el sistema. – El usuario debe estar en la página de la cartera.	
Flujo normal: 1. El actor hace click encima del botón de añadir o encima del botón (+) de alguna de sus acciones. 2. El sistema muestra un pop-up con un formulario para rellenar con los datos de la transacción. 3. El actor rellena el formulario con los datos y hace click en el botón de guardar transacción. 4. El sistema comprueba la validez de los datos del formulario. 5. Si los datos son correctos, el sistema hace una segunda comprobación de la nueva transacción con las demás transacciones de la cartera para verificar que es una transacción que se puede hacer (P. ej.: Si el actor tiene un total de una acción de una empresa no podrá hacer una transacción de venta de dos acciones de esa empresa, porque no dispone de ellas). 6. El sistema guarda la transacción en el sistema. 7. El actor recibe un aviso de que la transacción se ha creado y el sistema le actualiza la cartera.	
Flujo alternativo: 4. El sistema comprueba la validez de los datos, si los datos no son correctos, se avisa al actor de ello permitiéndole que los corrija. 5. El sistema hace una segunda comprobación de la nueva transacción, si es una transacción que no se puede hacer, se avisa al actor de ello permitiéndole que cambie la transacción.	
Postcondiciones: – El usuario ha creado la transacción.	

Tabla 8. Caso de uso añadir transacción

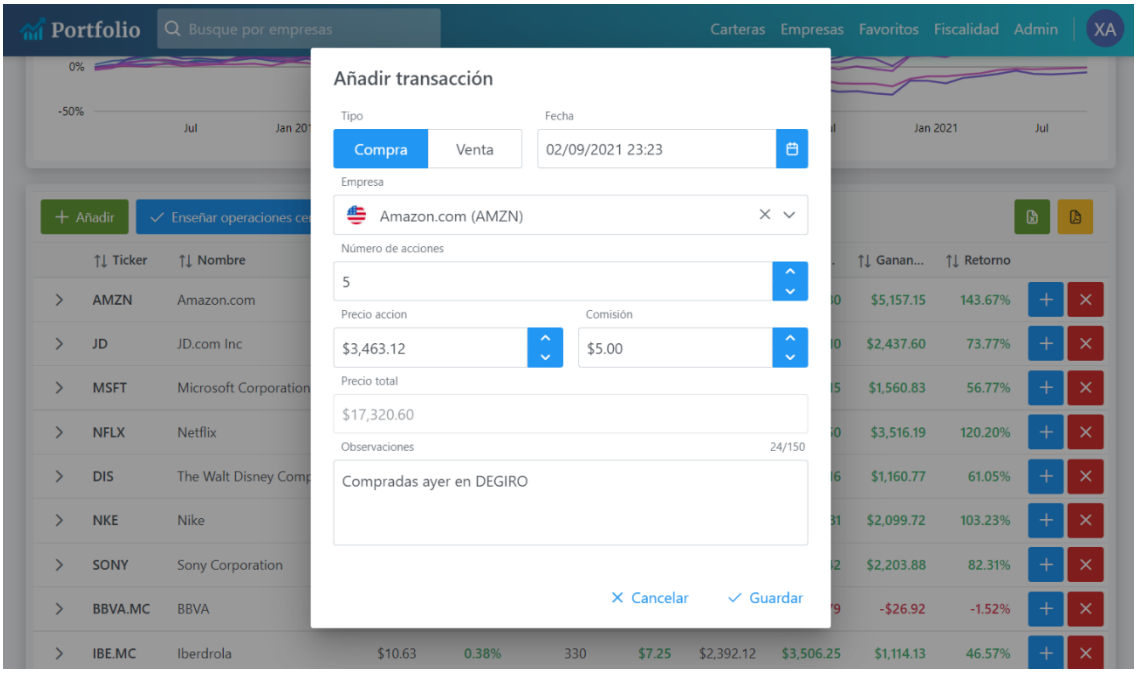


Figura 12. Interfaz de la pantalla cartera con el pop-up de añadir transacción

Editar transacción:

Nombre	Editar transacción de cartera
Actor	Usuario logueado
Descripción: Permite al usuario editar los datos de una de las transacciones ya creadas de alguna de sus carteras.	
Precondiciones: – El usuario debe de estar autenticado en el sistema. – El usuario debe estar en la página de la cartera. – El usuario debe tener alguna transacción en la cartera.	
Flujo normal: 1. El actor hace click encima del botón de editar de alguna de sus transacciones de cartera. 2. El sistema muestra un pop-up con un formulario con los datos de la transacción. 3. El actor edita los datos del formulario y hace click en el botón de guardar transacción. 4. El sistema comprueba la validez de los datos del formulario. 5. Si los datos son correctos, el sistema hace una segunda comprobación de la nueva transacción con las demás transacciones de la cartera para verificar que es una transacción que se puede hacer (P. ej.: Si el actor tiene una acción de una empresa y la transacción era una venta de tres acciones, no podrá editarla a una transacción de venta de cinco acciones, porque no dispone de ellas). 6. El sistema cambia la transacción en el sistema. 7. El actor recibe un aviso de que la transacción se ha editado y el sistema le actualiza la cartera.	
Flujo alternativo: 4. El sistema comprueba la validez de los datos, si los datos no son correctos, se avisa al actor de ello permitiéndole que los corrija. 5. El sistema hace una segunda comprobación de la nueva transacción, si es una transacción que no se puede hacer, se avisa al actor de ello permitiéndole que cambie la transacción.	
Postcondiciones: – El usuario ha editado la transacción.	

Tabla 9. Caso de uso editar transacción

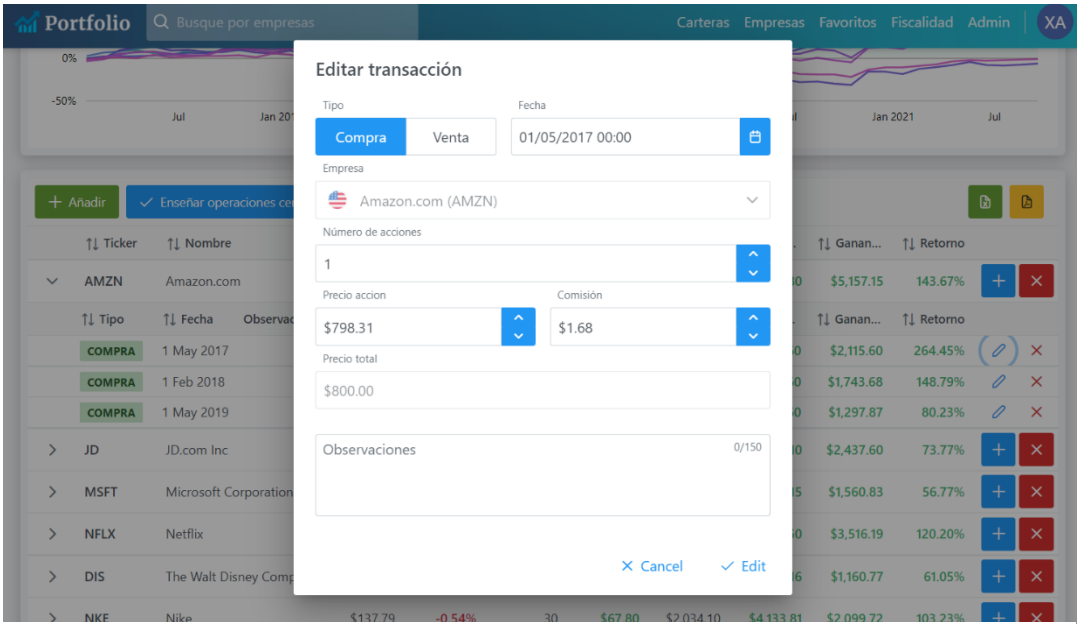


Figura 13. Interfaz de la pantalla cartera con el pop-up de editar transacción

Eliminar transacción y Eliminar acción:

Nombre	Eliminar transacción de cartera
Actor	Usuario logueado
Descripción: Permite al usuario eliminar una de las transacciones ya creadas de alguna de sus carteras.	
Precondiciones: – El usuario debe de estar autenticado en el sistema. – El usuario debe estar en la página de la cartera. – El usuario debe tener alguna transacción en la cartera.	
Flujo normal: 1. El actor hace click en el botón de eliminar de la transacción que quiera eliminar. 2. El sistema muestra un pop-up con una pregunta para confirmar la acción. 3. El actor confirma la acción haciendo click en el botón de aceptar. 4. El sistema hace una comprobación con los nuevos datos de transacción para verificar que es una transacción viable (P. ej.: Si el actor tiene un total de una acción de una empresa y la transacción era una compra de tres acciones, no podrá eliminarla porque el total de acciones sería negativo). 5. El sistema elimina la transacción. 6. El actor recibe un aviso de que la transacción se ha eliminado y el sistema le actualiza la cartera.	
Flujo alternativo: 3. El actor anula la acción haciendo click en el botón cancelar y el sistema cierra el pop-up. 4. El sistema hace una comprobación de los nuevos datos con la transacción a eliminar, si es una transacción que no se puede eliminar, se avisa al actor de ello y no se eliminará la transacción.	
Postcondiciones: – El usuario ha eliminado la transacción.	

Tabla 10. Caso de uso eliminar transacción

Nombre	Eliminar acción de cartera
Actor	Usuario logueado
Descripción: Permite al usuario eliminar todas las transacciones asociadas a la misma empresa de una cartera.	
Precondiciones: – El usuario debe de estar autenticado en el sistema. – El usuario debe estar en la página de la cartera. – El usuario debe tener alguna transacción en la cartera.	
Flujo normal: 1. El actor hace click en el botón de eliminar de la empresa de la que quiera eliminar todas sus transacciones. 2. El sistema muestra un pop-up con una pregunta para confirmar la acción. 3. El actor confirma la acción haciendo click en el botón de aceptar. 4. El sistema elimina todas las transacciones asociadas a esa empresa/acción. 5. El actor recibe un aviso de que la acción se ha eliminado y el sistema le actualiza la cartera.	
Flujo alternativo: 3. El actor anula la acción haciendo click en el botón cancelar y el sistema cierra el pop-up.	
Postcondiciones: – El usuario ha eliminado la transacción.	

Tabla 11. Caso de uso eliminar acción

Ver informe completo de cartera:

Nombre	Ver informe completo de la cartera
Actor	Usuario logueado
Descripción: Permite al usuario entrar y ver toda la información de una de sus carteras. En esta información se incluyen acciones, transacciones de estas acciones, empresas con sus rentabilidades, gráficos de rentabilidades, gráficos de dinero invertido, gráficos de diversificación (por país, mercado, divisa, industria y sector), información por fechas…	
Precondiciones: – El usuario debe de estar autenticado en el sistema. – El usuario debe estar en la página de la cartera.	
Flujo normal: 1. El sistema recibe el identificador de la cartera y la divisa seleccionada en el sistema. 2. El sistema entra en modo de carga. 3. El sistema identifica las transacciones que tiene la cartera. 4. El sistema en base a las transacciones obtiene los datos de los precios desde la API foránea (Yahoo Finance). 5. El sistema hace la conversión de los datos a la divisa seleccionada. 6. El sistema hace los cálculos de las tablas y los gráficos y se los muestra al usuario. 7. El sistema sale del modo de carga.	
Flujo alternativo: 4. El sistema al obtener los datos de la API foránea (Yahoo Finance) recibe un error y se le informa al usuario que ha habido un error.	
Postcondiciones: – El usuario ve el informe completo de la cartera.	

Tabla 12. Caso de uso ver informe completo de cartera

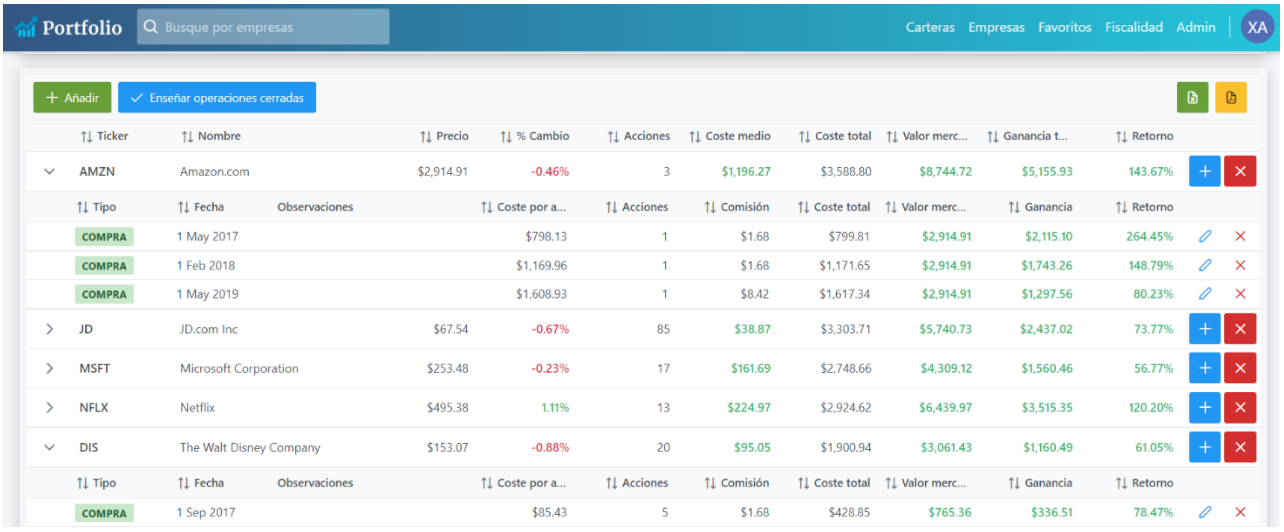


Figura 14. Interfaz del apartado de transacciones dentro de la ventana cartera

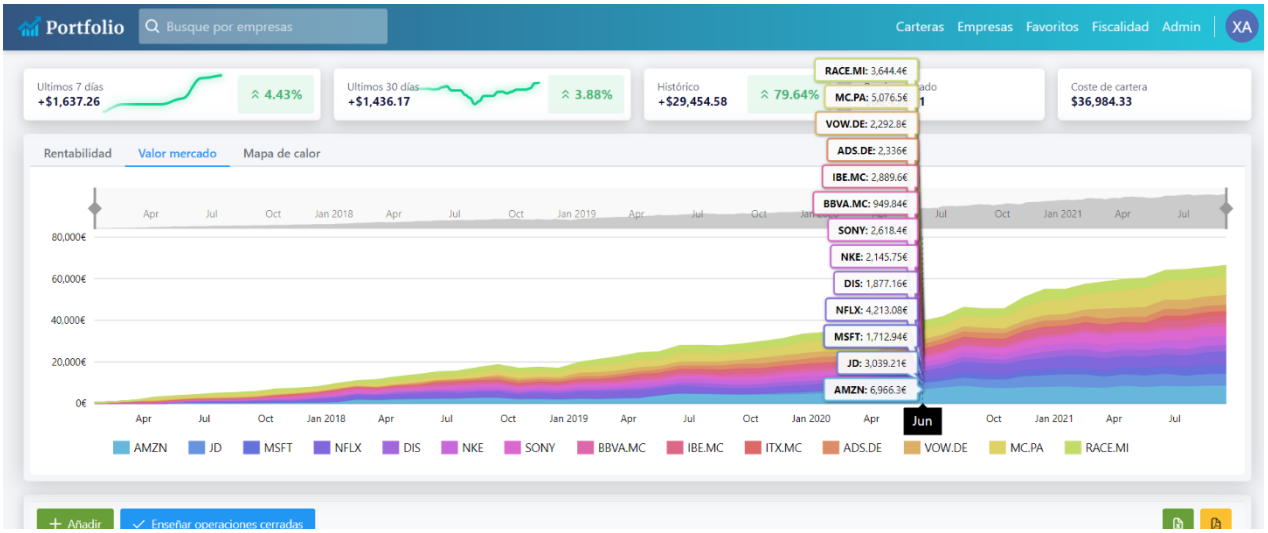


Figura 15. Interfaz del gráfico de valor de mercado dentro de la ventana cartera

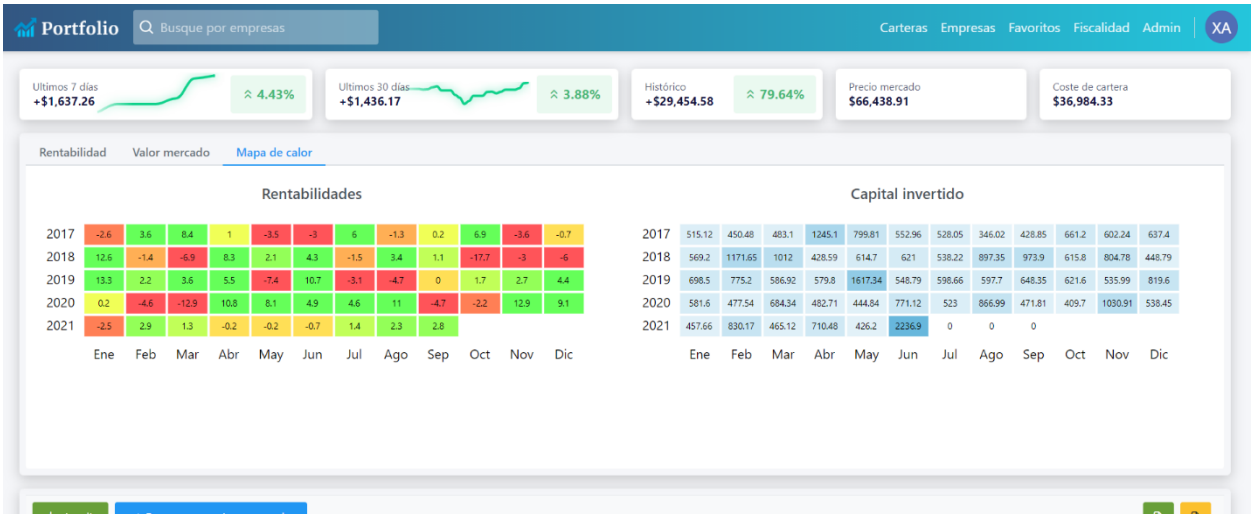


Figura 16. Interfaz de los mapas de calor dentro de la ventana cartera

Pasar informe a PDF:

Nombre	Pasar informe de cartera a PDF
Actor	Usuario logueado
Descripción: Permite al usuario descargar toda la información de una de sus carteras en formato PDF (con transacciones, rentabilidades, todos los gráficos…).	
Precondiciones: – El usuario debe de estar autenticado en el sistema. – El usuario debe estar en la página de la cartera. – El usuario debe tener el informe completo de la cartera cargado.	
Flujo normal: 1. El actor hace click en el botón de PDF. 2. El sistema genera un PDF con los datos previamente cargados de la cartera. 3. El sistema descarga el PDF en el navegador del usuario.	
Flujo alternativo:	
Postcondiciones: – El usuario ha descargado el informe completo de la cartera en formato PDF.	

Tabla 13. Caso de uso pasar informe a PDF

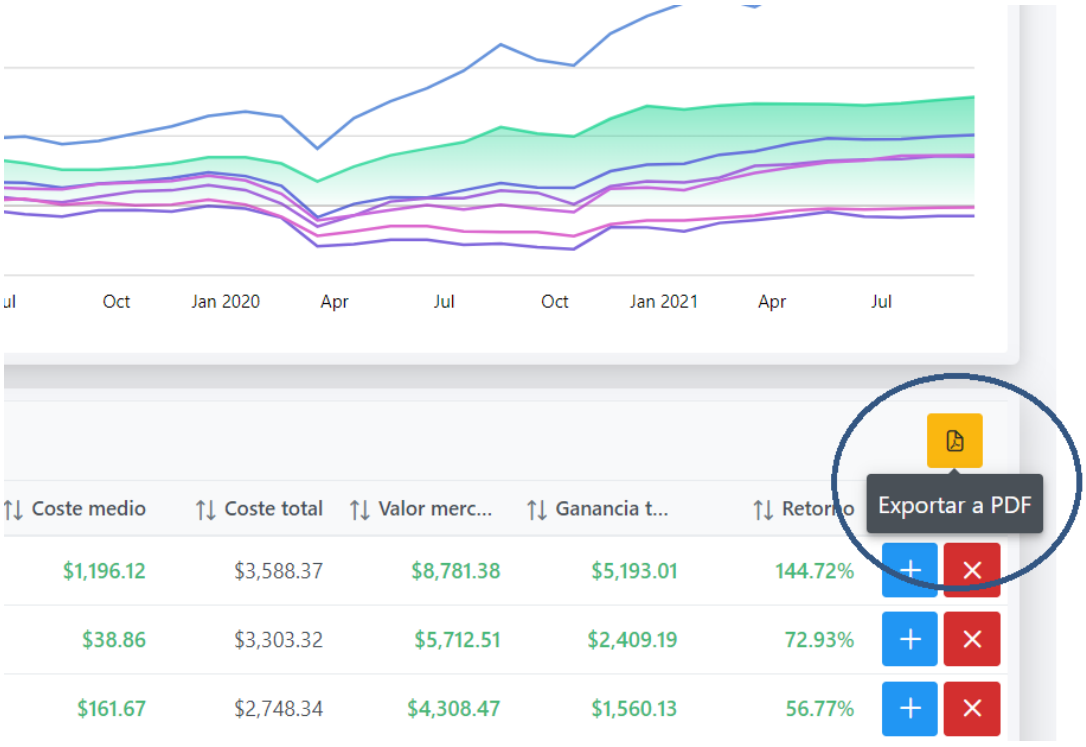


Figura 17. Indicador para exportar a PDF el informe completo de cartera

Buscar acciones:

Nombre	Buscar acciones
Actor	Usuario logueado
Descripción: Permite al usuario buscar empresas y sus acciones a través de la introducción de sus nombres o sus tickers[3].	
Precondiciones: – El usuario debe de estar autenticado en el sistema.	
Flujo normal: 1. El usuario busca la empresa de la que quiera obtener datos en el menú superior (por nombre o ticker). 2. El sistema le muestra un desplegable con las empresas filtradas en base al texto introducido. 3. El usuario hace click encima de la empresa de la cual quiera obtener más información. 4. El sistema manda al usuario a la página de información de la empresa seleccionada.	
Flujo alternativo: 2. El sistema no encuentra ninguna empresa en base al texto introducido y le muestra un mensaje de búsqueda no encontrada al usuario.	
Postcondiciones: – El usuario ha buscado la empresa deseada.	

Tabla 14. Caso de uso buscar acciones

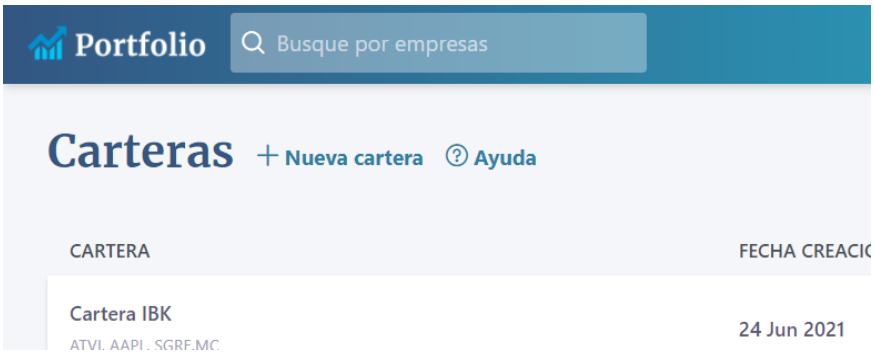


Figura 18. Interfaz de la barra de búsqueda

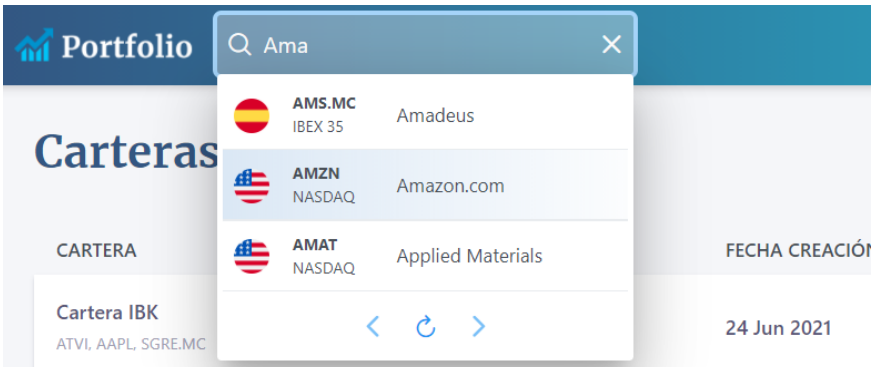


Figura 19. Interfaz de la barra de búsqueda con texto

[3] Ticker: Un símbolo bursátil o código bursátil, también conocido como ticker, es un código alfanumérico que sirve para identificar de forma abreviada las acciones de una determinada empresa que cotiza.

Filtrar acciones:

Nombre	Filtrar acciones
Actor	Usuario logueado
Descripción: Permite al usuario filtrar empresas y sus acciones a través de los criterios introducidos en el menú de filtrado. Se podrán filtrar empresas por país, por divisa, por mercado, por sector, por industria y por diferentes métricas de la acción.	
Precondiciones: – El usuario debe de estar autenticado en el sistema.	
Flujo normal: 1. El usuario hace click encima del botón empresas. 2. El sistema manda al usuario a la página de empresas. 3. El sistema obtiene de la API foránea (Yahoo Finance) los datos en tiempo real resumidos de todas las empresas y se las muestra al usuario en una tabla. 4. El usuario desde el menú de filtrado busca las empresas (con sus acciones) que desee en base a sus criterios. 5. El sistema filtra las empresas (con sus acciones) y le muestra al usuario en la tabla solo las que cumplen los requisitos.	
Flujo alternativo: 3. El sistema no encuentra ninguna empresa en base a los criterios introducidos y le muestra un mensaje de sin resultados al usuario.	
Postcondiciones: – El usuario ha obtenido una lista con las empresas filtradas.	

Tabla 15. Caso de uso filtrar acciones

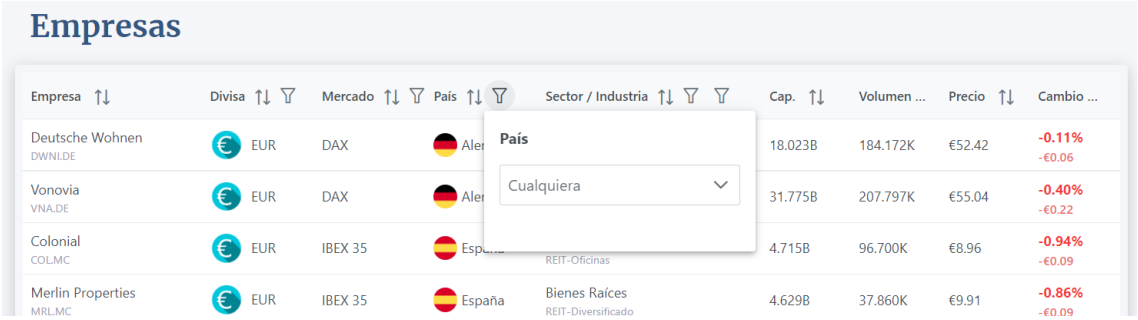


Figura 20. Interfaz de la ventana empresas sin ningún filtro aplicado

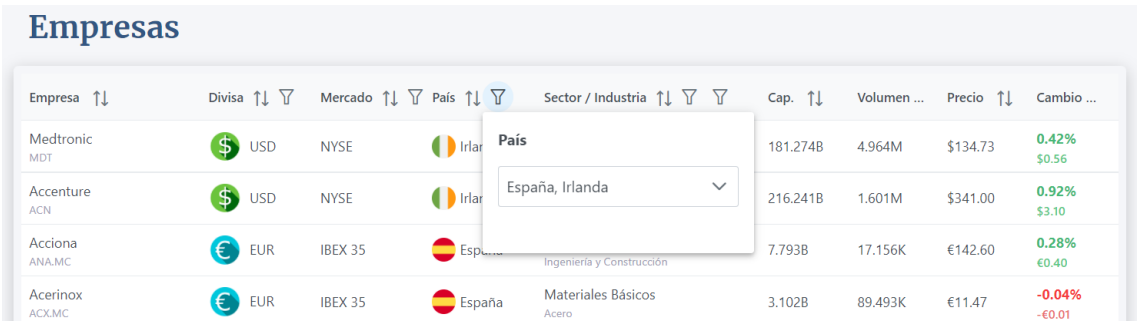


Figura 21. Interfaz de la ventana empresas con el filtro por país

Ver acción:

Nombre	Ver acción
Actor	Usuario logueado
Descripción: Permite al usuario ver toda la información de una empresa y su acción en tiempo real. También podrá ver diferentes métricas y balances de la empresa en cuestión para analizar si es una buena inversión o no. También podrá ver los principales fondos de inversión que invierten en dicha acción de la empresa.	
Precondiciones: – El usuario debe de estar autenticado en el sistema. – El usuario debe haber elegido alguna empresa desde la búsqueda de menú superior o desde la ventana empresas.	
Flujo normal: 1. El sistema obtiene de la API foránea (Yahoo Finance) los datos en tiempo real completos de la empresa que el usuario ha elegido y las métricas de la acción. 2. El sistema crea los gráficos y muestra los datos al usuario.	
Flujo alternativo: 1. El sistema al obtener los datos de la API foránea (Yahoo Finance) recibe un error y se le informa al usuario que ha habido un error.	
Postcondiciones: – El usuario ha obtenido toda la información en tiempo real de la empresa seleccionada.	

Tabla 16. Caso de uso ver acción

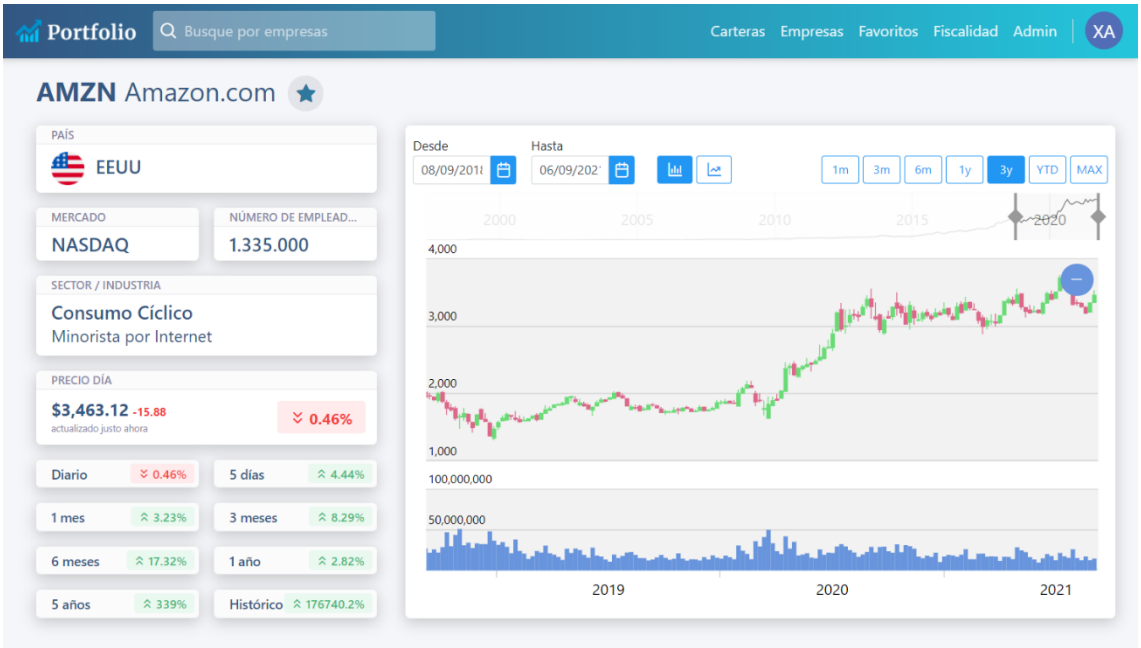


Figura 22. Interfaz de la pantalla empresa con la info. general y el gráfico de velas

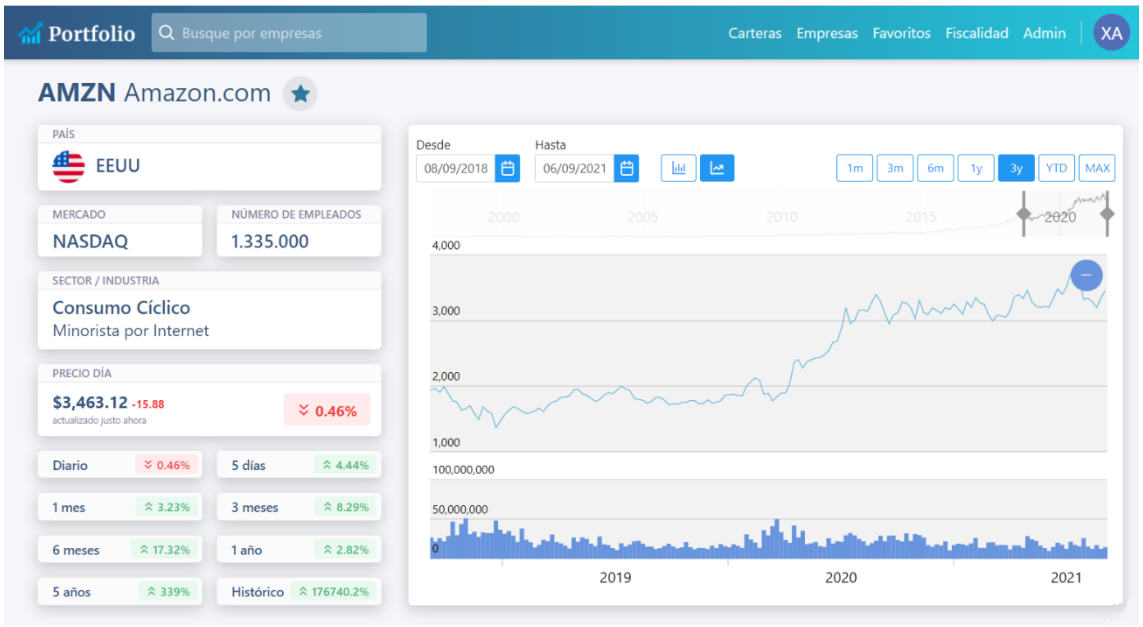


Figura 23. Interfaz de la pantalla empresa con la info. general y el gráfico de líneas

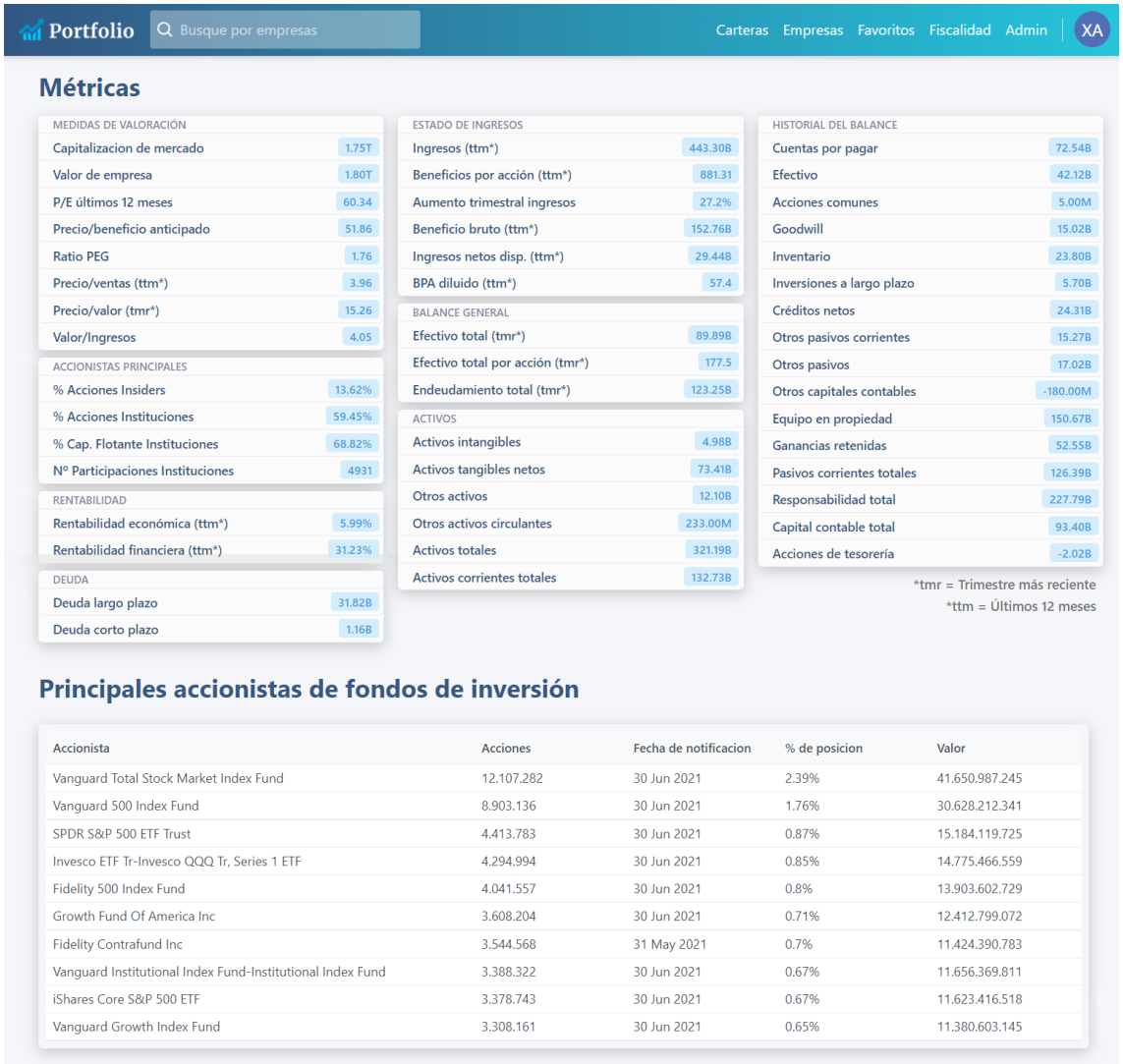


Figura 24. Interfaz de la pantalla empresa con las métricas y datos de accionistas

Guardar/Editar datos de fiscalidad:

Nombre	Guardar/Editar datos de fiscalidad
Actor	Usuario logueado
Descripción: Permite al usuario registrar sus datos fiscales (nombre, identificación, domicilio…) para poder crear informes fiscales de sus transacciones.	
Precondiciones: – El usuario debe de estar autenticado en el sistema. – El usuario debe estar en la página de fiscalidad.	
Flujo normal: 1. Al actor se le muestra un formulario con los campos para introducir todos los datos de fiscalidad (si el usuario ya ha introducido antes sus datos entonces se mostrarán en el formulario). 2. El actor rellena el formulario con los datos que deseé. 3. El actor hace click en guardar datos. 4. El sistema guarda los datos de fiscalidad del usuario.	
Flujo alternativo: 3. Algún dato introducido por el usuario no tiene el formato correcto o está vacío y el sistema no guarda los datos.	
Postcondiciones: – El usuario ha guardado/editado sus datos de fiscalidad.	

Tabla 17. Caso de uso guardar/editar datos de fiscalidad

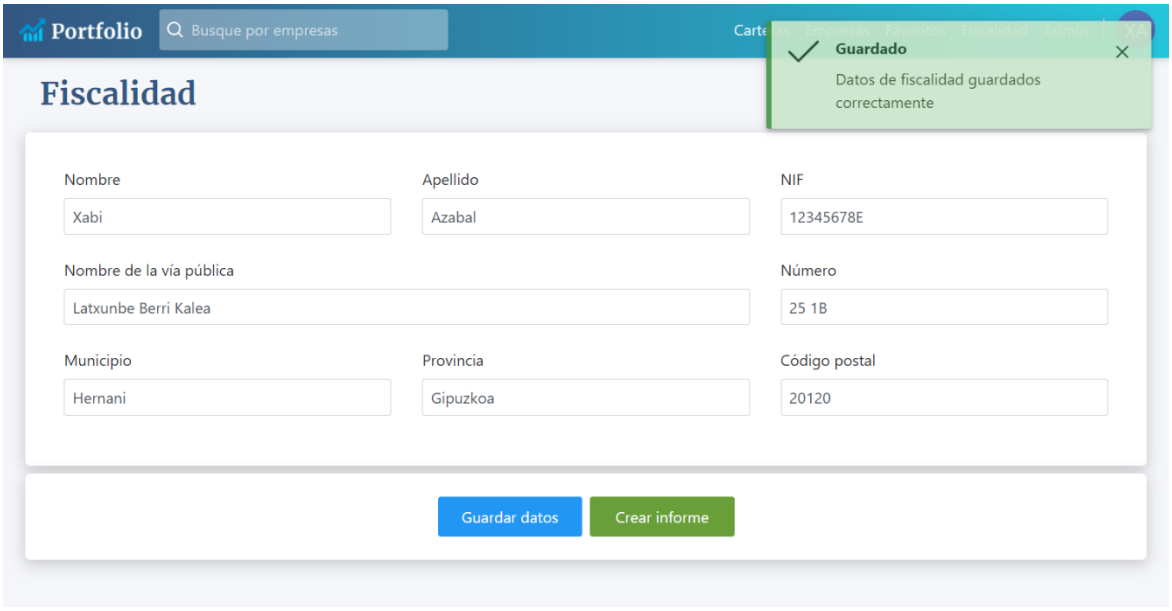


Figura 25. Interfaz de la pantalla fiscalidad después de haber guardado los datos

Crear informe fiscal:

Nombre	Crear informe fiscal de cartera/s
Actor	Usuario logueado
Descripción: Permite al usuario descargar un informe fiscal de las transacciones de la cartera o las carteras que elija, en el año que elija. Este informe fiscal es una declaración anual que se le entrega a hacienda sobre las inversiones españolas en el exterior de los activos financieros que cotizan en las bolsas de valores o en mercados organizados, tanto nacionales como internacionales. Este caso de uso te permite crear el informe automáticamente para entregarlo a la hora de hacer la declaración de la renta.	
Precondiciones: – El usuario debe de estar autenticado en el sistema. – El usuario debe tener alguna cartera creada. – El usuario debe tener registrados unos datos fiscales correctos.	
Flujo normal: 1. El actor hace click en el botón de crear informe. 2. El sistema muestra un pop-up con un desplegable para que el actor elija las carteras de las que quiere hacer el modelo y también el año del que quiere descargar el informe. También le da a elegir en la divisa que quiere mostrar los costes de las transacciones (dólares, euros o mixto). 3. El actor elige en el desplegable las carteras, el año y la divisa y hace click en descargar informe. 4. El sistema genera un PDF con los datos fiscales del usuario y también los datos de las transacciones de las carteras seleccionadas en el año seleccionado y genera el informe. 5. El sistema descarga el PDF en el navegador del usuario.	
Flujo alternativo: 3. El actor anula la acción haciendo click en el botón cancelar y el sistema cierra el pop-up. 3. Las carteras elegidas en el año elegido no tienen transacciones y el no se dejará crear el informe.	
Postcondiciones: – El usuario ha descargado el informe fiscal en PDF.	

Tabla 18. Caso de uso crear informe fiscal

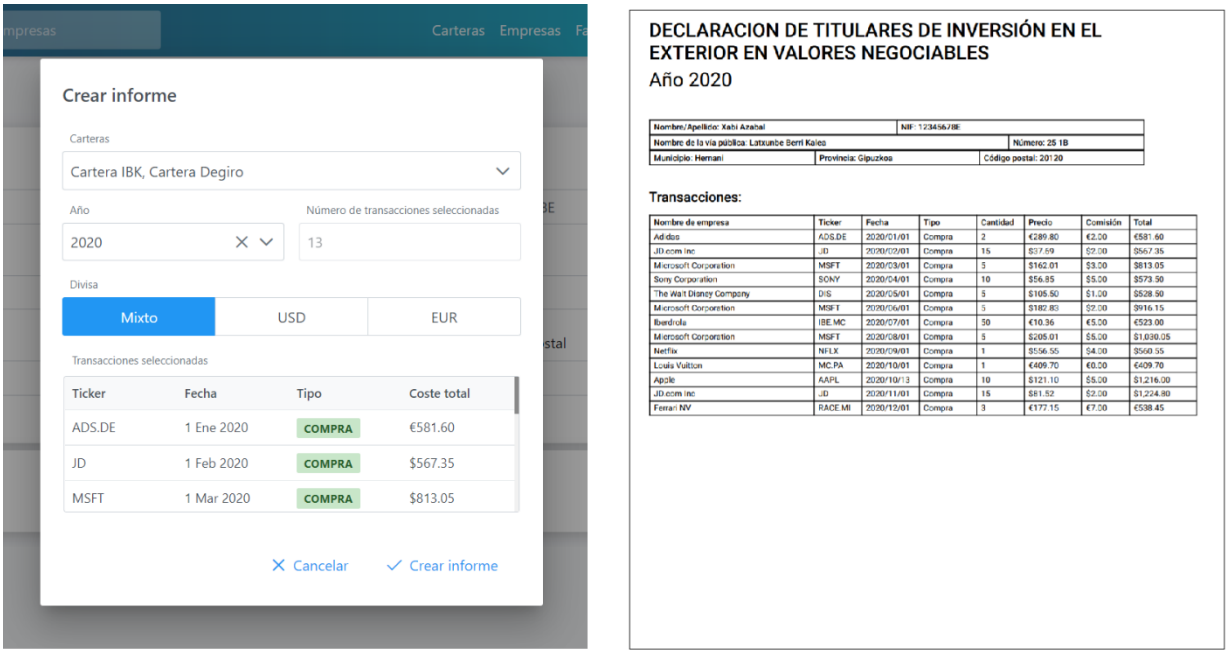


Figura 26. Pop-up de crear informe y un ejemplo de un informe generado

Añadir/Eliminar favorito:

Nombre	Añadir/Eliminar favorito
Actor	Usuario logueado
Descripción: Permite al usuario añadir las empresas del sistema que deseé a sus favoritos, para tenerlas guardadas y poder visualizarlas.	
Precondiciones: – El usuario debe de estar autenticado en el sistema. – El usuario debe estar en la página de la empresa que guardará/eliminará de favoritos.	
Flujo normal: 1. El actor hace click en el botón de favorito de la empresa. 2. Si la empresa ya estaba en favoritos, el sistema la eliminará de favoritos. Si por el contrario la empresa no estaba en favoritos, se añadirá a favoritos.	
Flujo alternativo:	
Postcondiciones: – El usuario ha añadido/eliminado la empresa a sus favoritos.	

Tabla 19. Caso de uso añadir/eliminar favorito

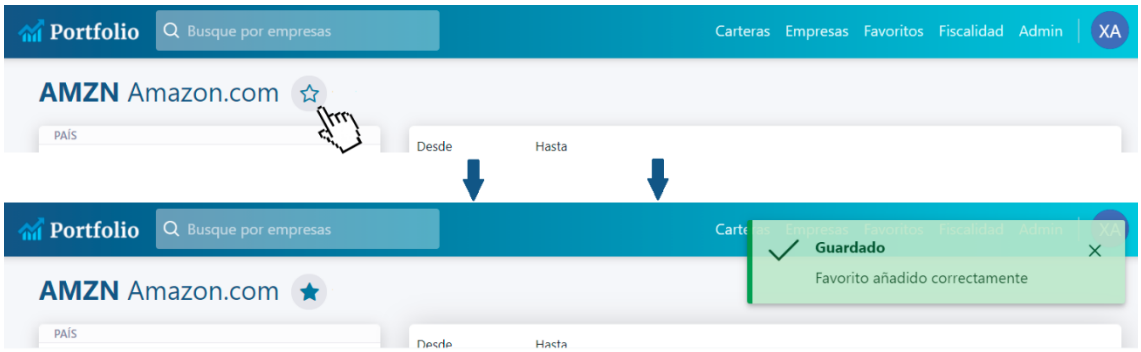


Figura 27. Proceso de añadir una empresa a favoritos

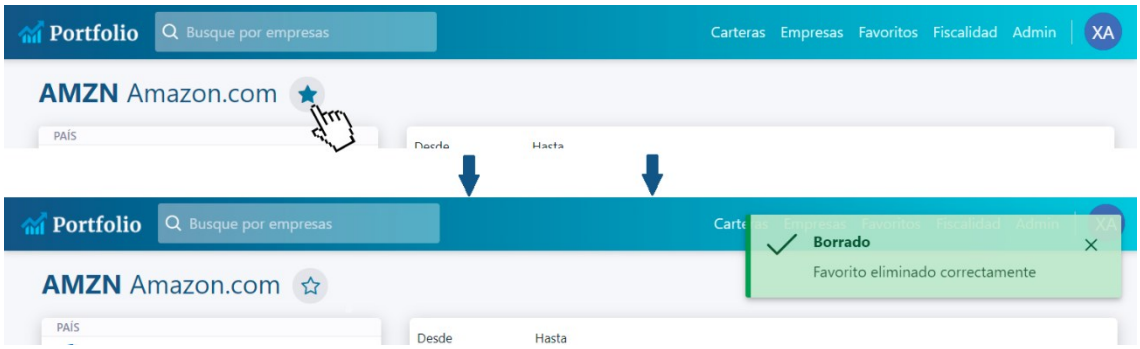


Figura 28. Proceso de eliminar una empresa de favoritos

Compartir cartera:

Nombre	Compartir cartera
Actor	Usuario logueado
Descripción: Permite al usuario compartir alguna de sus carteras con otros usuarios, para poder mostrar el rendimiento de la misma.	
Precondiciones: – El usuario debe de estar autenticado en el sistema. – El usuario debe estar en la página de carteras. – El usuario debe tener alguna cartera en público.	
Flujo normal: 1. El actor hace click en el botón de compartir de la cartera la cual quiera compartir. 2. El sistema muestra un pop-up con el enlace con el que se podrá acceder a la cartera. 3. El actor copia el enlace y lo comparte con la gente que deseé	
Flujo alternativo:	
Postcondiciones: – El usuario ha compartido su cartera.	

Tabla 20. Caso de uso compartir cartera

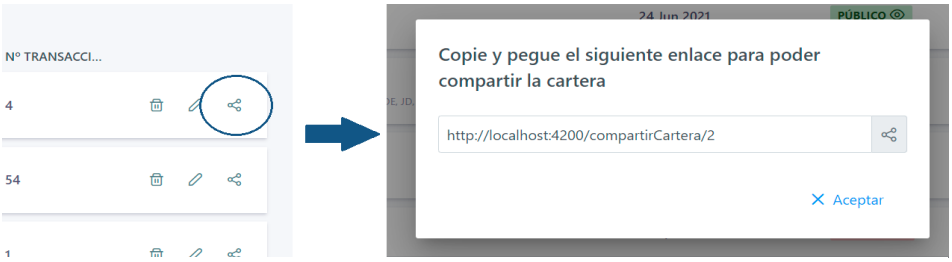


Figura 29. Proceso de compartir una cartera

Ver cartera ajena:

Nombre	Ver cartera ajena
Actor	Usuario logueado
Descripción: Permite al usuario ver el desempeño que ha tenido alguna cartera que otro usuario ha compartido con él. Podrá ver todas las transacciones y estadísticas de la cartera.	
Precondiciones: – El usuario debe de estar autenticado en el sistema. – El usuario debe haber recibido un link de otro usuario para ver su cartera.	
Flujo normal: 1. El actor introduce el link recibido en la barra del navegador y accede a la página. 2. El sistema le manda a la página de la cartera. 3. El actor ahora tendrá la posibilidad de ver las transacciones de cartera, ver informe completo de cartera o pasar informe de cartera a PDF.	
Postcondiciones: – El usuario ha visto la cartear ajena.	

Tabla 21. Caso de uso ver cartera ajena

4.3.3. Rol de administrador

Añadir empresa

Nombre	Añadir empresa
Actor	Administrador
Descripción: Permite al administrador añadir empresas nuevas en el sistema. El ticker de la empresa tendrá que ser uno que no esté usando ninguna otra empresa.	
Precondiciones: – El usuario debe de estar autenticado en el sistema. – El usuario debe estar en la página de administrador	
Flujo normal: 1. El actor hace click encima del botón de añadir empresa 2. El sistema muestra un pop-up con un formulario para rellenar con los datos de la transacción. 3. El actor rellena el formulario con los datos. 4. El sistema comprueba que el ticker es válido y que no está todavía en uso. 5. El sistema guarda la empresa en el sistema. 6. El actor recibe un aviso de que la empresa se ha creado y el sistema le actualiza la lista completa de empresas.	
Flujo alternativo: 4. El sistema hace la comprobación del ticker y este no es válido. Se avisa al actor permitiéndole que cambie el ticker y lo vuelva a comprobar. 4. El sistema hace la comprobación del ticker y este ya está en uso por otra empresa. Se avisa al actor permitiéndole que cambie el ticker y lo vuelva a comprobar.	
Postcondiciones: – El administrador ha creado la empresa.	

Tabla 22. Caso de uso añadir empresa

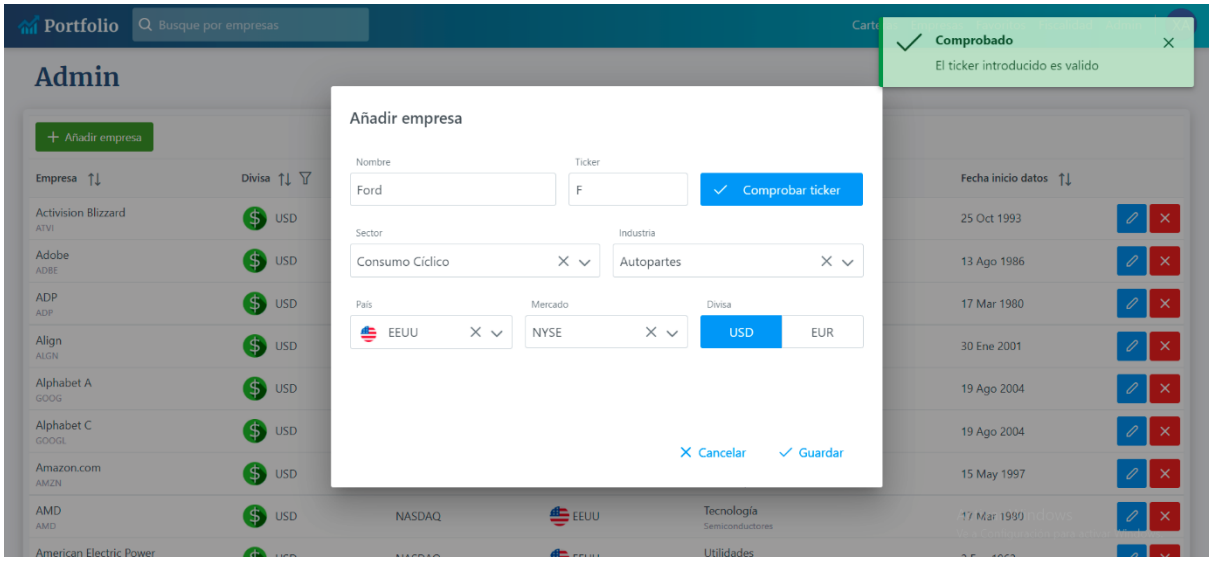


Figura 30. Interfaz del pop-up de añadir una empresa después de validar el ticker

Editar empresa:

Nombre	Editar empresa
Actor	Administrador
Descripción: Permite al administrador editar empresas en el sistema. El ticker de la empresa tendrá que ser uno que no esté usando ninguna otra empresa.	
Precondiciones: – El usuario debe de estar autenticado en el sistema. – El usuario debe estar en la página de administrador	
Flujo normal: 1. El actor hace click encima del botón de editar de alguna de las empresas del sistema. 2. El sistema muestra un pop-up con un formulario con los datos de la empresa. 3. El actor edita el formulario con los nuevos datos. 4. El sistema comprueba que el ticker es válido y que no está todavía en uso. 5. El sistema guarda la empresa en el sistema. 6. El actor recibe un aviso de que la empresa se ha editado y el sistema le actualiza la lista completa de empresas.	
Flujo alternativo: 4. El sistema hace la comprobación del ticker y este no es válido. Se avisa al actor permitiéndole que cambie el ticker y lo vuelva a comprobar. 4. El sistema hace la comprobación del ticker y este ya está en uso por otra empresa. Se avisa al actor permitiéndole que cambie el ticker y lo vuelva a comprobar.	
Postcondiciones: – El usuario ha editado la empresa.	

Tabla 23. Caso de uso editar empresa

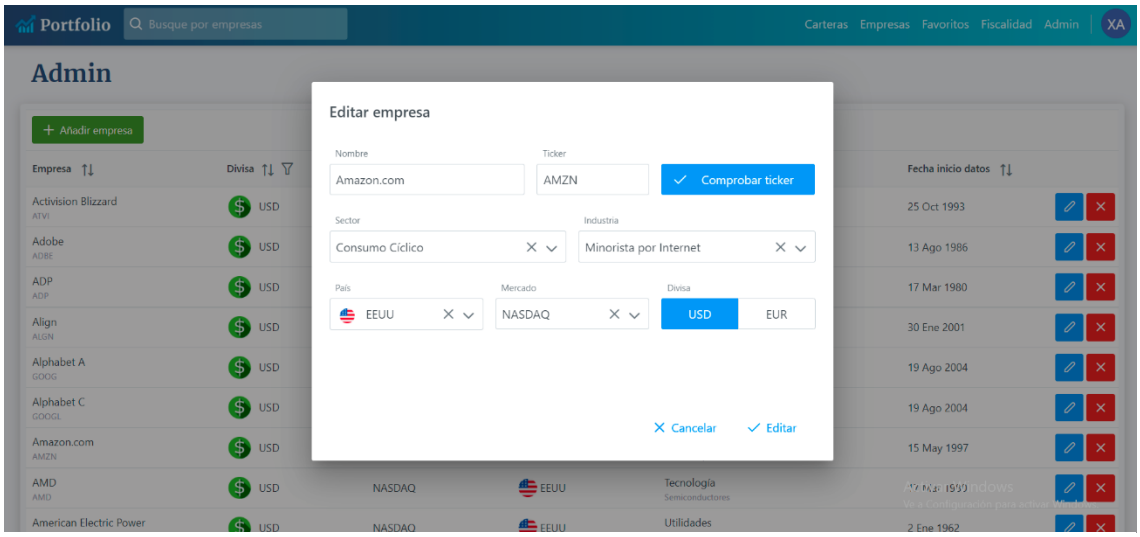


Figura 31. Interfaz del pop-up de editar una empresa

Eliminar empresa:

Nombre	Eliminar empresa
Actor	Administrador
Descripción: Permite al usuario eliminar una de la empresa ya creada del sistema.	
Precondiciones: – El usuario debe de estar autenticado en el sistema. – El usuario debe estar en la página de administrador	
Flujo normal: 1. El actor hace click en el botón de eliminar de la empresa que quiera eliminar. 2. El sistema muestra un pop-up con una pregunta para confirmar la acción. 3. El actor confirma la acción haciendo click en el botón de aceptar. 4. El sistema elimina la empresa y todos los datos asociados a ella (transacciones y favoritos). 5. El actor recibe un aviso de que la empresa se ha eliminado y el sistema le actualiza la lista de empresas.	
Flujo alternativo: 3. El actor anula la acción haciendo click en el botón cancelar y el sistema cierra el pop-up.	
Postcondiciones: – El usuario ha eliminado la empresa.	

Tabla 24. Caso de uso eliminar empresa

4.4. Modelo de dominio Entidad-Relación

Este diagrama o modelo entidad-relación es una herramienta para el modelo de datos, la cual facilita la representación de entidades de la base de datos del proyecto. Con este modelo de datos tendremos una percepción basada en el mundo real de las relaciones que tienen nuestras clases (llamadas entidades) de la base de datos.

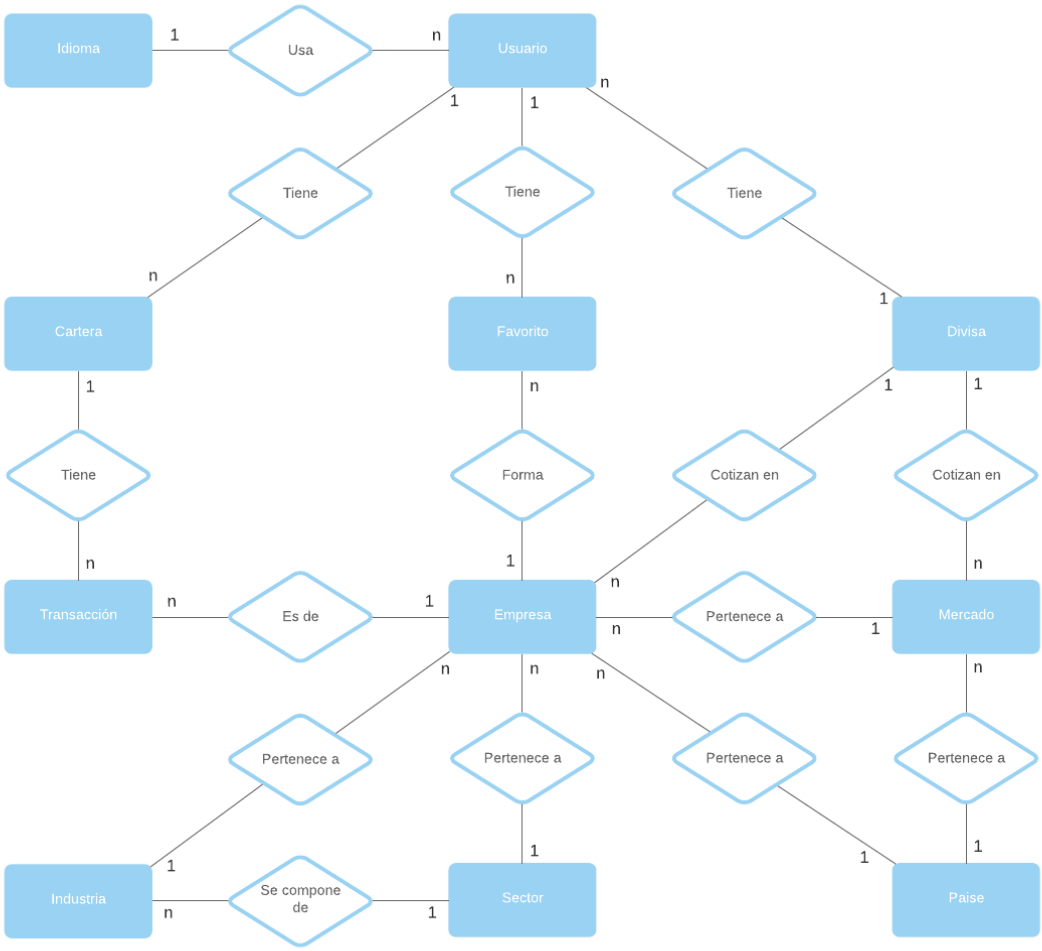


Figura 32. Diagrama de Entidad-Relación

5. Diseño

En este capítulo realizaremos una visión global sobre el diseño que seguirá nuestro proyecto. Se describirá la base de datos más específicamente y también se mostrarán los casos de uso por medio de diagramas de secuencia.

5.1. Diagrama de la base de datos

El modelo de dominio correspondiente al análisis de requisitos que hemos diseñado se muestra a continuación utilizando la notación UML. Dentro de este nos encontraríamos con once clases: Usuarios, Carteras, Transacciones, Idiomas, Empresas, Favoritos, Mercados, Países, Divisas, Sectores e Industrias.

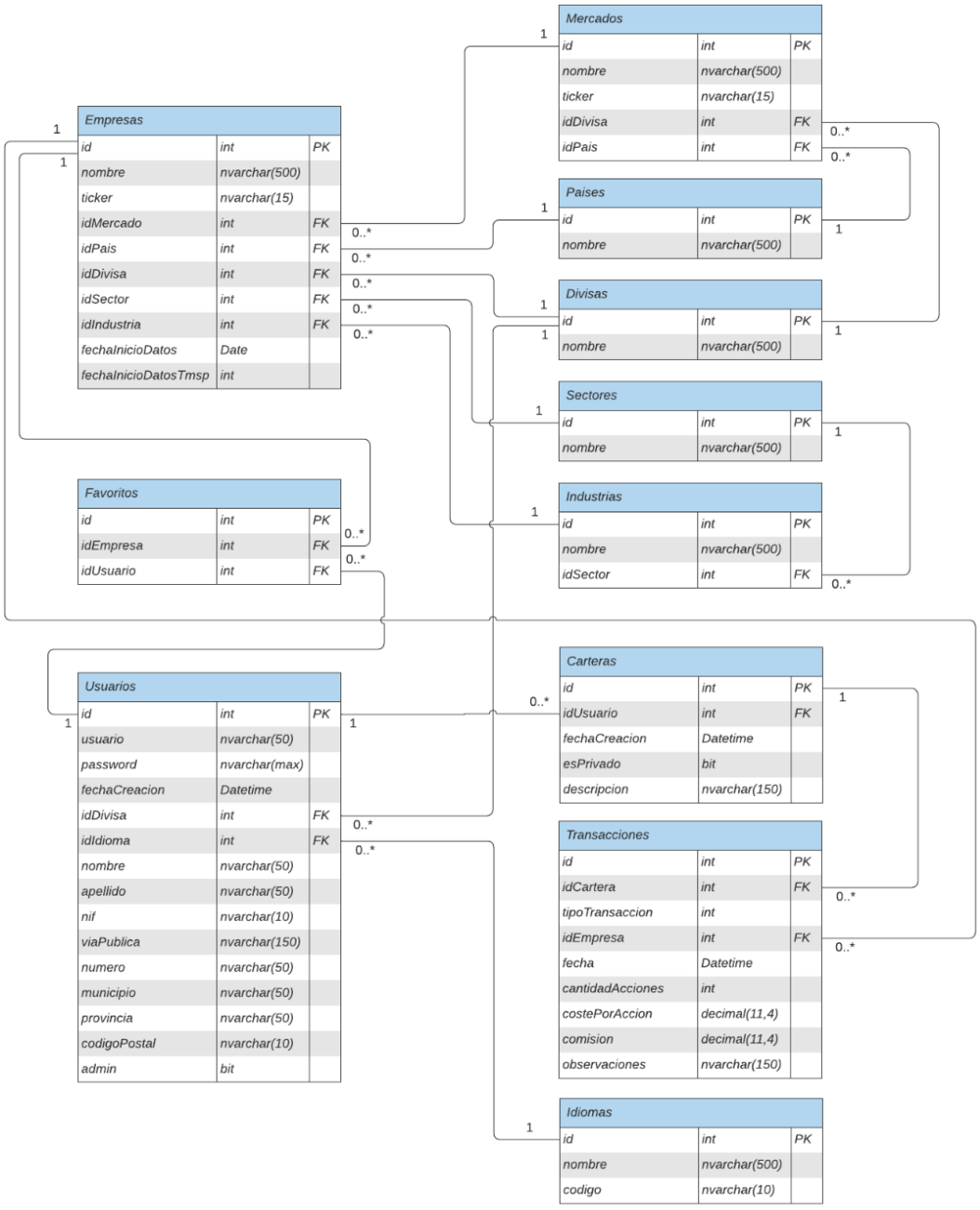


Figura 33. Diagrama de las tablas de la base de datos

5.2. Descripción de las tablas

- Empresas: Esta tabla representa las diferentes empresas que usaremos en el sistema, de las cuales podremos obtener información y también registrar transacciones en nuestras carteras. Los activos financieros tienen un identificador a nivel mundial llamado ticker el cual permite identificarlos, por lo tanto, las empresas que cotizan en bolsa también lo tienen. Usaremos este ticker para poder descargar la información (precios, histórico, rangos…) en tiempo real de las empresas desde la API. Las columnas que forman la tabla:
  - *id*: Es el identificador con el que distinguiremos la empresa dentro de la base de datos.
  - *nombre*: Representa el nombre de la empresa.
  - *ticker*: Identificador alfanumérico para distinguir la acción de la empresa y su cotización a nivel mundial.
  - *idMercado*: Clave foránea para identificar el mercado (tabla Mercados) al que pertenece la empresa. Cada empresa tendrá un mercado.
  - *idPais*: Clave foránea para identificar el país (tabla Países) al que pertenece la empresa. Cada empresa tendrá un país.
  - *idDivisa*: Clave foránea para identificar la divisa (tabla Divisas) con la que se negocian las acciones de la empresa. Cada empresa tendrá una divisa.
  - *idSector*: Clave foránea para identificar el sector (tabla Sectores) al que pertenece la empresa. Cada empresa tendrá un sector.
  - *idIndustria*: Clave foránea para identificar la industria (tabla Industrias) a la que pertenece la empresa. Cada empresa tendrá una industria.
  - *fechaInicioDatos*: Representa la primera fecha de la cual podemos obtener datos de la empresa a través de la API, en formato de fecha.
  - *fechaInicioDatosTmsp:* Representa la primera fecha de la cual podemos obtener datos de la empresa a través de la API, en formato numérico.

- Mercados: Esta tabla representa los diferentes mercados que usaremos en el sistema. Los mercados son un espacio (físico, virtual o ambos) en el que se realizan los intercambios de instrumentos financieros y se definen sus preferencias. Los mercados se dividen por países y tienen en su haber instrumentos financieros que cotizan en dicho país, también se negocian estos instrumentos con la divisa local del país. Es este caso los diferentes mercados tendrán en su haber un conjunto de empresas. Para poder tener una referencia de la economía del mercado y todos sus activos, se hace una ponderación de las empresas que lo forman y sus valores para luego formar un índice el cual tendrá un valor, por lo tanto, cada mercado también tiene un ticker para poder acceder a la cotización de ese mercado a través del mismo. Las columnas que forman la tabla:
  - *id*: Es el identificador con el que distinguiremos el mercado dentro de la base de datos.
  - *nombre*: Representa el nombre del mercado.
  - *ticker*: Identificador alfanumérico para distinguir el mercado y su cotización a nivel mundial.
  - *idDivisa*: Clave foránea para identificar la divisa (tabla Divisas) con la que se negocian las empresas que forman el mercado. Cada mercado tendrá una divisa.
  - *idPais*: Clave foránea para identificar el país (tabla Países) al que pertenece el mercado. Cada mercado tendrá un país.

- Países: Esta tabla representa los diferentes países que usaremos en el sistema. En la base de datos estarán insertados los diferentes países que tengan las empresas y mercados. Las columnas que forman la tabla:
  - *id*: Es el identificador con el que distinguiremos el país dentro de la base de datos.
  - *nombre*: Representa el nombre del país.

- Divisas: Esta tabla representa las diferentes divisas que usaremos en el sistema. En la base de datos estarán insertadas las diferentes divisas que tengan las empresas y mercados. En este caso las divisas podrán ser el dólar y el euro, porque los mercados utilizados serán mercados estadounidenses y europeos. Las columnas que forman la tabla:
  - *id*: Es el identificador con el que distinguiremos la divisa dentro de la base de datos.
  - *nombre*: Representa el nombre de la divisa.

- Sectores: Esta tabla representa los diferentes sectores que usaremos en el sistema. Los sectores nos sirven para clasificar las empresas por el tipo de actividad que llevan a cabo, p. ej.: tecnología, energía, materiales básicos... Cada sector tiene una serie de industrias para especificar aún más el cometido de cada empresa. Por lo tanto, cada empresa pertenece a un sector y también a una industria. Las columnas que forman la tabla:
  - *id*: Es el identificador con el que distinguiremos el sector dentro de la base de datos.
  - *nombre*: Representa el nombre del sector.

- Industrias: Esta tabla representa las diferentes industrias que usaremos en el sistema. Como ya hemos dicho cada industria pertenece a un sector, para poder especificar al máximo el cometido de cada empresa. P.ej.: El sector tecnológico tiene las industrias software, comunicación… Las columnas que forman la tabla:
  - *id*: Es el identificador con el que distinguiremos la industria dentro de la base de datos.
  - *nombre*: Representa el nombre de la industria.
  - *idSector*: Clave foránea para identificar el sector (tabla Sectores) al que pertenece la industria. Cada industria tendrá un sector.

- Usuarios: Esta tabla representa los usuarios que van a poder acceder a la plataforma. Aquí se guardará también las configuraciones que cambie en la web (idioma y divisa). También se guardarán los datos personales a la hora de poder hacer el informe para declarar los movimientos hechos. Las columnas que forman la tabla:
  - *id*: Es el identificador con el que distinguiremos el usuario dentro de la base de datos.
  - *usuario*: Es el nombre de usuario con el que se accederá a la plataforma. Este nombre de usuario también será único y no se podrá repetir entre diferentes usuarios.
  - *password*: Es la contraseña con la que el usuario accederá a la plataforma. En la base de datos se guardará la contraseña con el cifrado aplicado.
  - *fechaCreacion*: Representa la fecha donde el usuario se registró en la plataforma.
  - *idDivisa*: Clave foránea para identificar la divisa (tabla Divisas) que tiene configurada (seleccionada) el usuario en la plataforma. Cada usuario tendrá una divisa seleccionada al mismo tiempo, la cual podrá cambiar.
  - *idIdioma*: Clave foránea para identificar el idioma (tabla Idiomas) que tiene configurado (seleccionado) el usuario en la plataforma. Cada usuario tendrá un idioma seleccionado al mismo tiempo, el cual podrá cambiar.

- *nombre*: Representa el nombre de pila del usuario.
- *apellido*: Representa el apellido del usuario.
- *nif*: Representa el número identificativo nacional del usuario.
- *viaPublica*: Representa la vía pública donde se encuentra el domicilio del usuario. Forma parte de la dirección.
- *numero*: Representa el número en la vía pública donde se encuentra el domicilio del usuario. Forma parte de la dirección.
- *municipio*: Representa el municipio donde reside el usuario.
- *provincia*: Representa la provincia donde reside el usuario.
- *codigoPostal*: Representa el código postal que tiene el municipio donde reside el usuario.
- *admin*: Esta columna es un booleano que indica si el usuario cuenta con los permisos de ser administrador o no.

- Carteras: Esta tabla representa las carteras propiedad de los usuarios que están registradas en la plataforma. Una cartera es un conjunto de acciones de diferentes empresas que el dueño tiene en su propiedad. Cada cartera tendrá un único dueño, los cuales serán los usuarios. Cada cartera estará formada por diferentes transacciones que el dueño haya registrado. Las columnas que forman la tabla:
  - *id*: Es el identificador con el que distinguiremos la cartera dentro de la base de datos.
  - *idUsuario*: Clave foránea para identificar el usuario (tabla Usuarios) el cual es dueño de la cartera. Cada cartera tendrá un único usuario.
  - *fechaCreacion*: Representa la fecha donde la cartera fue creada en la plataforma.
  - *esPrivado*: Booleano que indica si la cartera es privada o pública. Con este campo el usuario dueño de la cartera decidirá si otros usuarios pueden ver el comportamiento de su cartera.
  - *descripción*: Representa la descripción que el usuario quiere darle a la cartera.

- Transacciones: Esta tabla representa las transacciones pertenecientes a una cartera registradas en la plataforma. Como ya hemos dicho una cartera tiene una serie de transacciones. Estas transacciones representan las compras o ventas de acciones de empresas que ha hecho el usuario en esa cartera. Cada transacción es una compra o venta de una o más acciones de una empresa, a un precio por acción y con una comisión. P.Ej.: Una transacción en la que hemos comprado 10 acciones de Amazon a un precio por acción de 3.000$ y con una comisión de 10$ sería una transacción de 30.010$. Las columnas que forman la tabla:
  - *id*: Es el identificador con el que distinguiremos la transacción dentro de la base de datos.
  - *idCartera*: Clave foránea para identificar la cartera (tabla Carteras) a la cual pertenece la transacción. Cada transacción tendrá una única cartera.
  - *tipoTransaccion*: Representa si la transacción es una compra o una venta. Si tipoTransaccion = 1 entonces la transacción es una compra de acciones. Si tipoTransaccion = 2 entonces la transacción es una venta de acciones.
  - *idEmpresa*: Clave foránea para identificar la empresa (tabla Empresas) de la cual se han comprado/vendido las acciones de la transacción. Cada transacción tendrá una única empresa.

- *fecha*: Representa la fecha en la que se ha hecho la transacción. Representa la fecha en la que el usuario ha hecho la compra/venta, no la fecha en la que ha registrado esa compra/venta en la plataforma.
- *cantidadAcciones*: Representa el número de acciones que se han comprado/vendido en la transacción. Este número o podrá ser menor a uno.
- *costePorAccion*: Es el precio por el que se ha vendido/comprado cada acción de la transacción. El número se guarda en formato decimal. El coste por acción será siempre mayor a cero.
- *comision*: Representa el precio de gestión que se le ha cobrado al usuario por la transacción. Esta comisión se la cobra el bróker al comprador, y el precio depende de este bróker, hay algunos brókeres que no cobran comisión, por lo tanto, este valor puede ser cero. El número se guarda en formato decimal. La comisión será mayor o igual a cero.
- *observaciones*: Es un breve comentario que quiera añadir el usuario a la transacción. Puede ser máximo de 150 caracteres.

- Idiomas: Esta tabla representa los diferentes idiomas que usaremos en el sistema. En la base de datos estarán insertados los diferentes idiomas que los usuarios van a poder elegir en la plataforma. Los idiomas registrados son castellano, euskera e inglés. Las columnas que forman la tabla:
  - *id*: Es el identificador con el que distinguiremos el idioma dentro de la base de datos.
  - *nombre*: Representa el nombre del idioma.

- Favoritos: Esta tabla representa las empresas que los usuarios pueden tener en favoritos. En la plataforma los usuarios podrán guardar como favoritas las empresas que más les gusten, esos favoritos quedan registrados en esta tabla. Las columnas que forman la tabla:
  - *id*: Es el identificador con el que distinguiremos el favorito dentro de la base de datos.
  - *idUsuario*: Clave foránea para identificar el usuario (tabla Usuarios) el cual ha guardado el favorito. Cada favorito tendrá un único usuario.
  - *idEmpresa*: Clave foránea para identificar la empresa (tabla Empresas) la cual el usuario ha guardado en favoritos. Cada favorito tendrá una única empresa.

Los datos recogidos e insertados en las tablas de la base de datos para un funcionamiento acorde a una plataforma real han sido recogidos y contrastados a mano. Los datos se han recogido de fuentes como Yahoo Finance o Investing. Dentro de todos estos datos creados se incluyen:

- 300 empresa.
- 88 industrias.
- 12 sectores.
- 20 países.
- 6 mercados
- 2 divisas.

A continuación, se muestra un ejemplo del proceso de creación y alimentación de una de las tablas de la base de datos. En este caso es la tabla Empresas.

Script para la creación de la tabla:

```
CREATE TABLE Empresas (
	id int NOT NULL IDENTITY(1, 1),
	nombre nvarchar(500),
	ticker nvarchar(15),
	idMercado int,
	idPais int,
	idDivisa int,
	idSector int,
	idIndustria int,
	PRIMARY KEY (id),
	FOREIGN KEY (idMercado) REFERENCES Mercados(id),
	FOREIGN KEY (idPais) REFERENCES Paises(id),
	FOREIGN KEY (idDivisa) REFERENCES Divisas(id),
	FOREIGN KEY (idSector) REFERENCES Sectores(id),
	FOREIGN KEY (idIndustria) REFERENCES Industrias(id)
)
GO
```

Script para la inserción de datos (las claves referenciadas tienen que tener algún objeto en la tabla correspondiente):

```
INSERT INTO Empresas (nombre,ticker,idMercado,idSector,idIndustria,idPais,idDivisa)
VALUES ('Activision Blizzard','ATVI',1,3,9,19,1)
INSERT INTO Empresas (nombre,ticker,idMercado,idSector,idIndustria,idPais,idDivisa)
VALUES ('Adobe','ADBE',1,11,83,19,1)
INSERT INTO Empresas (nombre,ticker,idMercado,idSector,idIndustria,idPais,idDivisa)
VALUES ('ADP','ADP',1,9,70,19,1)
INSERT INTO Empresas (nombre,ticker,idMercado,idSector,idIndustria,idPais,idDivisa)
VALUES ('Alexion','ALXN',1,8,45,19,1)
INSERT INTO Empresas (nombre,ticker,idMercado,idSector,idIndustria,idPais,idDivisa)
VALUES ('Align','ALGN',1,8,52,19,1)
```

Muestra de datos:

```
SELECT TOP (5) * FROM Empresas
```

id	nombre	ticker	idMercado	idPais	idDivisa	idSector	idIndustria	fechaInicioDatos	fechaInicioDatosTmsp
1	Activision Blizzard	ATVI	1	19	1	3	9	25/10/1993	751555800
2	Adobe	ADBE	1	19	1	11	83	13/08/1986	524323800
3	ADP	ADP	1	19	1	9	70	17/03/1980	322151400
5	Align	ALGN	1	19	1	8	52	30/01/2001	980865000
6	Alphabet A	GOOG	1	19	1	3	11	19/08/2004	1092922200

Tabla 25. Muestra de datos de la tabla Empresas

5.3. Diseño mediante diagramas de secuencia

Login:

Para que el usuario complete un login, primero tiene que introducir las credenciales en el formulario y enviarlo. Una vez enviado el formulario, el Frontend[4] encripta los datos y se los envía a la API. La API comprueba si los datos son correctos en la BD y le devuelve la respuesta al Frontend. Si los datos no son correctos se cancelará el login y no se le permitirá el acceso al usuario. Por otra parte, si los datos son correctos se cargará el sistema, para eso se obtienen los datos de todas las entidades del sistema (empresas, mercados, sectores...) y se crean en el Frontend objetos locales de esas entidades y del usuario. Una vez hecho esto el usuario ha completado el login.

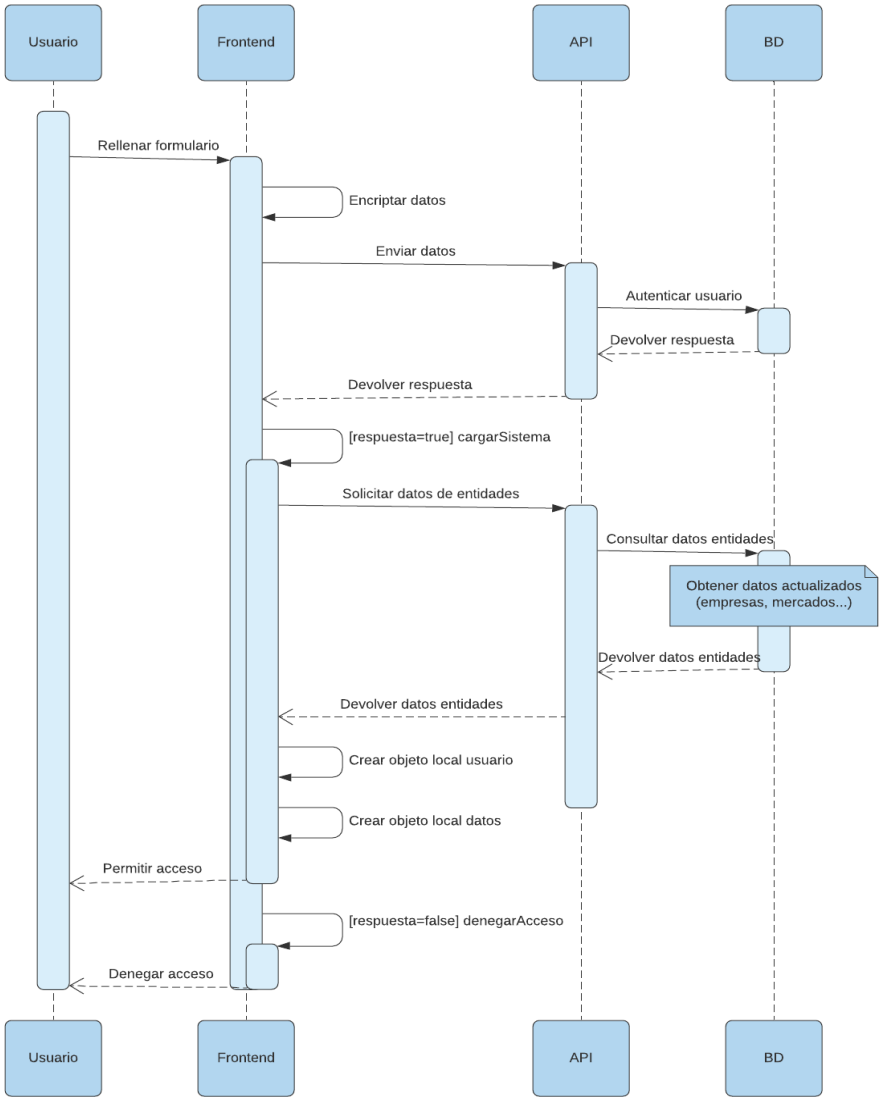


Figura 34. Diagrama de secuencia del caso de uso login

[4] Frontend: Es la parte de un programa o dispositivo a la que un usuario puede acceder directamente. Son todas las tecnologías de diseño y desarrollo web que corren en el navegador y que se encargan de la interactividad con los usuarios.

Registrar:

Para que el usuario complete un registro, primero tiene que introducir las credenciales en el formulario y enviarlo. Una vez enviado el formulario el Frontend encripta los datos y se los envía a la API. La API comprueba en la BD que no exista ningún usuario creado con el mismo nombre de usuario que se ha introducido en el formulario. Si ya existe ese nombre de usuario entonces se deniega el registro, si el nombre de usuario no existe entonces se sigue adelante con el registro. Si se sigue adelante con el registro se creará el objeto usuario en la BD y se le devolverá la respuesta al usuario. Si la respuesta recibida es que el usuario se ha creado correctamente entonces se le redirigirá al usuario al login.

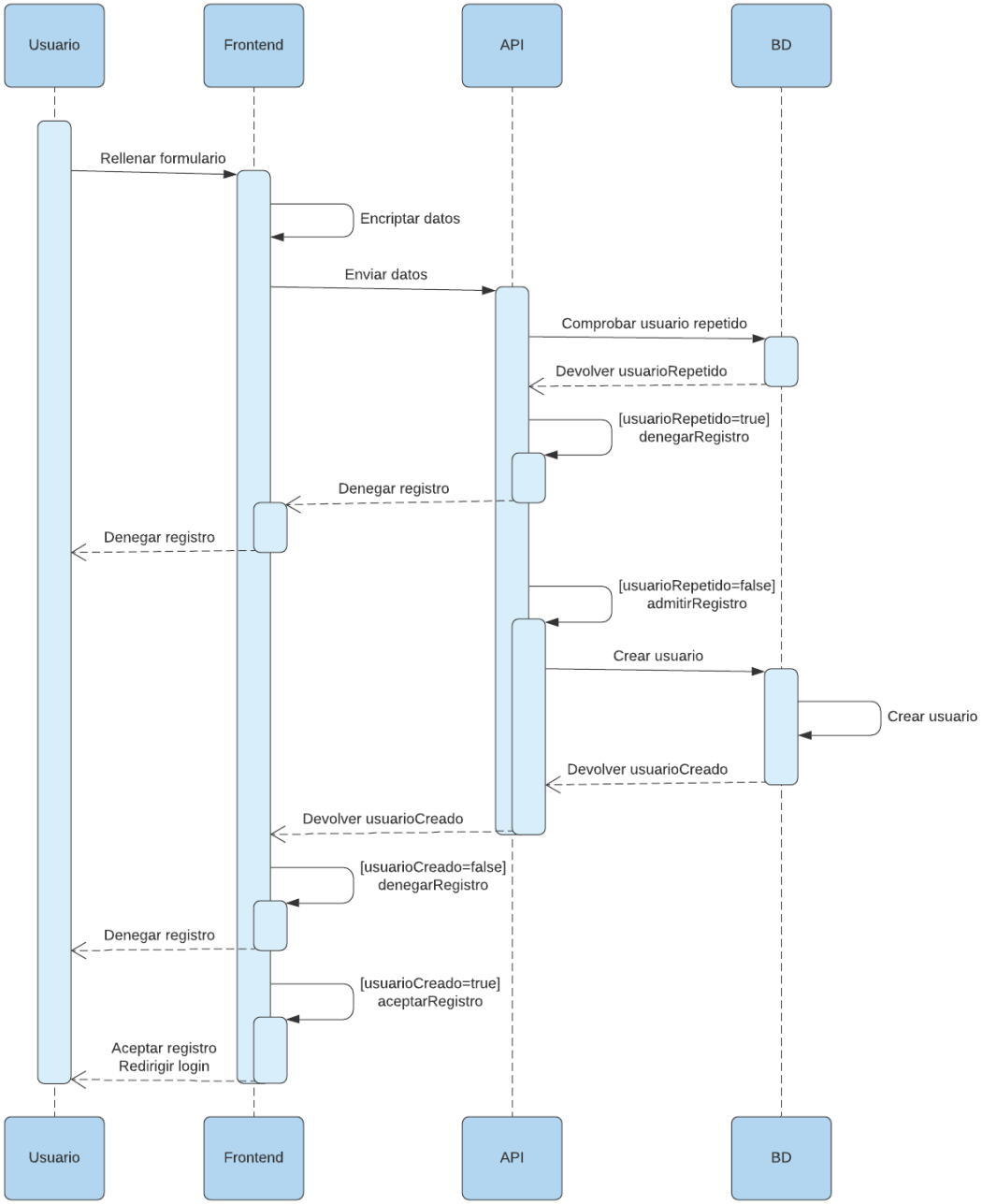


Figura 35. Diagrama de secuencia del caso de uso registrar

Ver carteras:

Para que el usuario pueda ver sus carteras, primero tiene que acceder a la ventana en el Frontend. Luego, el Frontend llama a la API para obtener las carteras pasándole el id del usuario. La API obtiene las carteras y todas las transacciones de estas carteras de la BD y se las devuelve al Frontend. Una vez el Frontend ha recibido las carteras con las transacciones se las muestra al usuario y este ya podrá hacer operaciones sobre ellas (editar, eliminar, compartir, crear más carteras...).

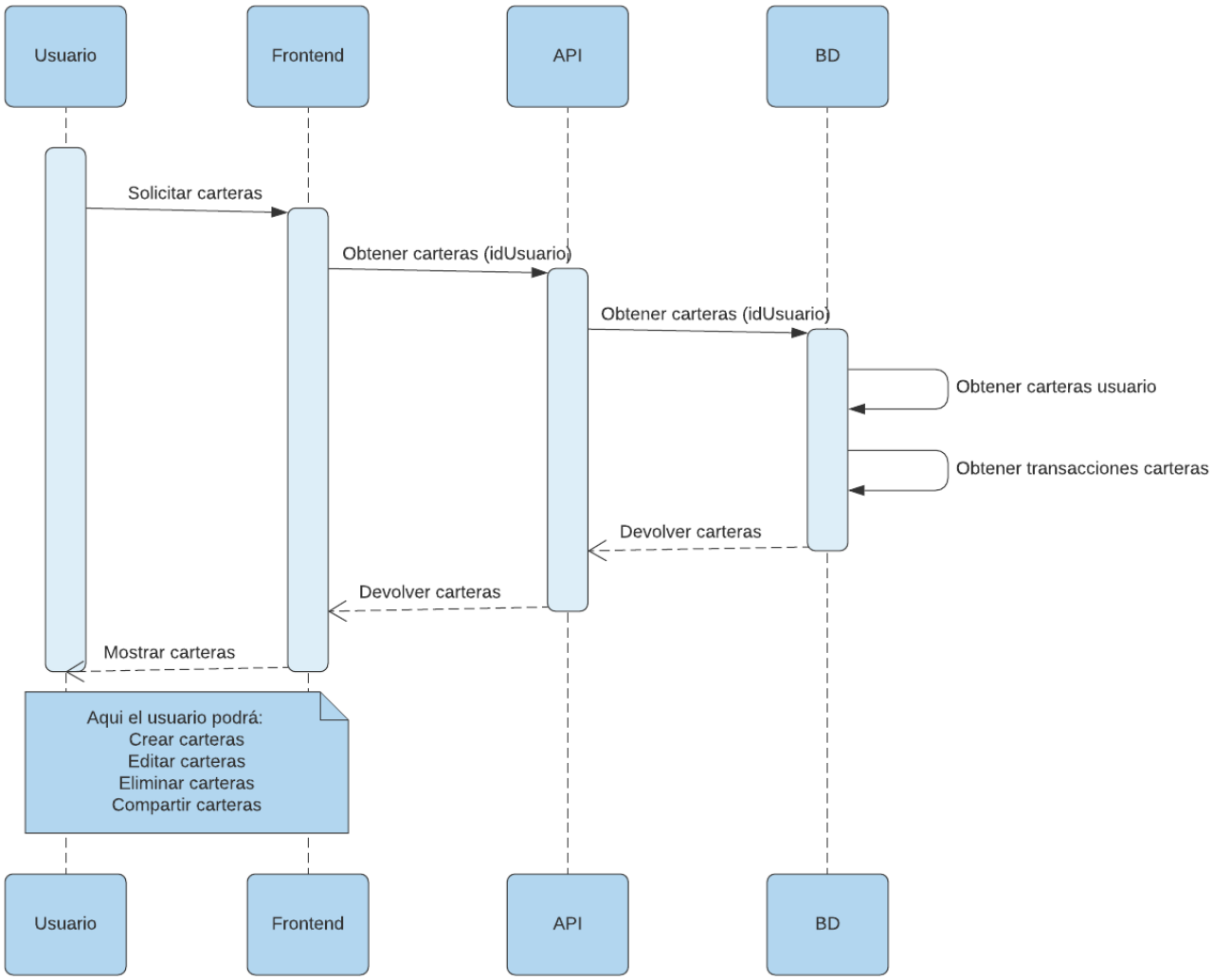


Figura 36. Diagrama de secuencia del caso de uso ver carteras

Crear cartera:

Para que el usuario pueda crear una cartera, primero tiene que solicitar crear una cartera en el Frontend. El Frontend le devolverá al usuario un formulario para introducir los datos de su nueva cartea. Una vez el usuario ha rellenado correctamente el formulario, el Frontend envía los datos para crear la cartera a la API. La API crea en la BD el objeto de la cartera y le devuelve la respuesta al Frontend. Para terminar, el Frontend le muestra al usuario su lista de carteras actualizada.

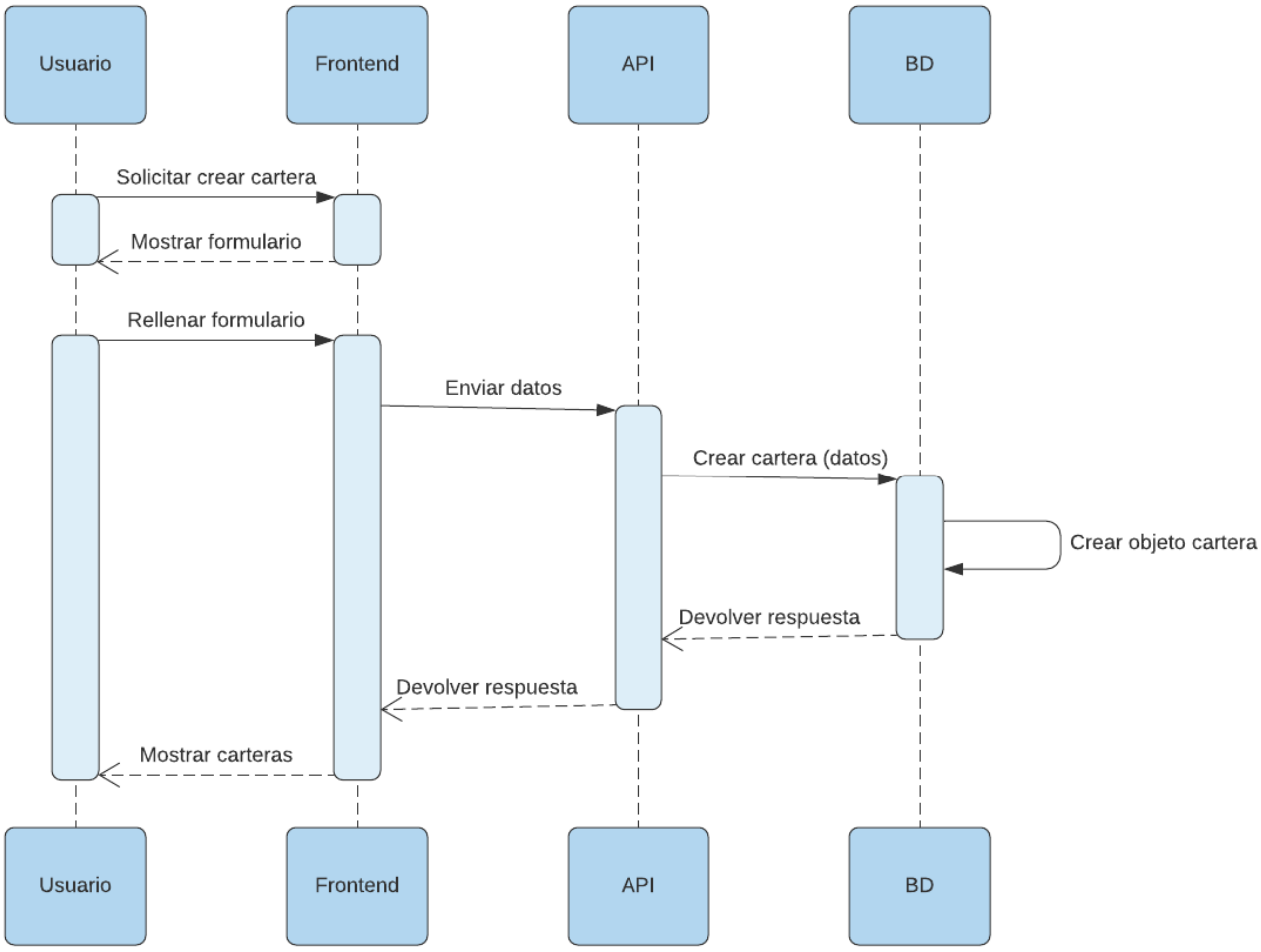


Figura 37. Diagrama de secuencia del caso de uso crear cartera

Editar cartera:

Para que el usuario pueda editar una cartera, primero tiene que solicitar editar una de sus carteras en el Frontend. El Frontend le devolverá al usuario un formulario con los datos de su cartera, para que pueda introducir los datos editados de su cartea. Una vez el usuario ha rellenado correctamente el formulario, el Frontend envía los datos y el id de la cartera para editarla a la API. La API edita en la BD el objeto de la cartera y le devuelve la respuesta al Frontend. Para terminar, el Frontend le muestra al usuario su lista de carteras actualizada.

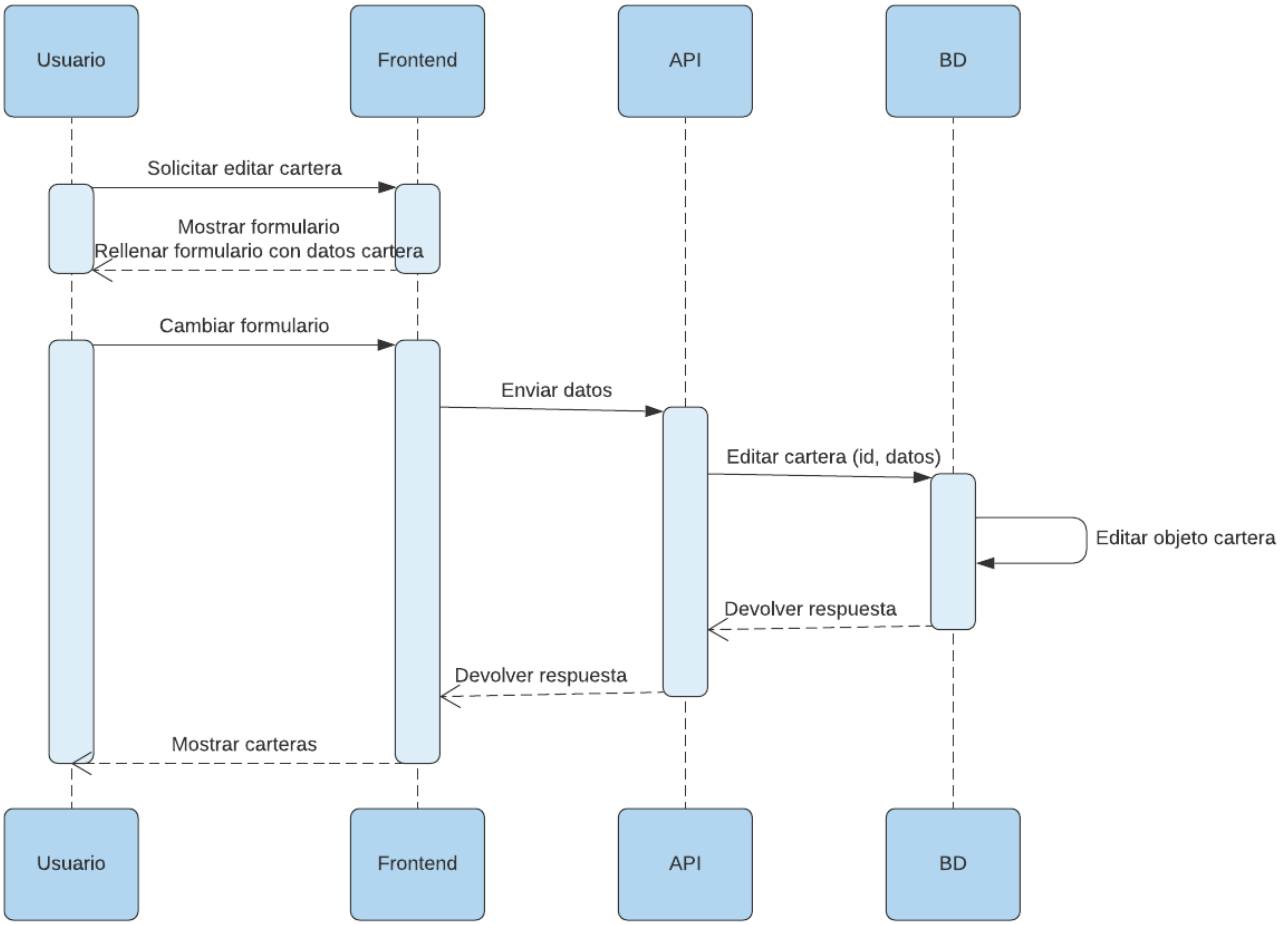


Figura 38. Diagrama de secuencia del caso de uso editar cartera

Eliminar cartera:

Para que el usuario pueda eliminar una cartera, primero solicita en el Frontend la cartera que quiere eliminar. El Frontend envía el id de la cartera que hay que eliminar a la API. La API elimina en la BD el objeto de la cartera y también todas las transacciones asociadas a esa cartera y le devuelve la respuesta al Frontend. Para terminar, el Frontend le muestra al usuario su lista de carteras actualizada.

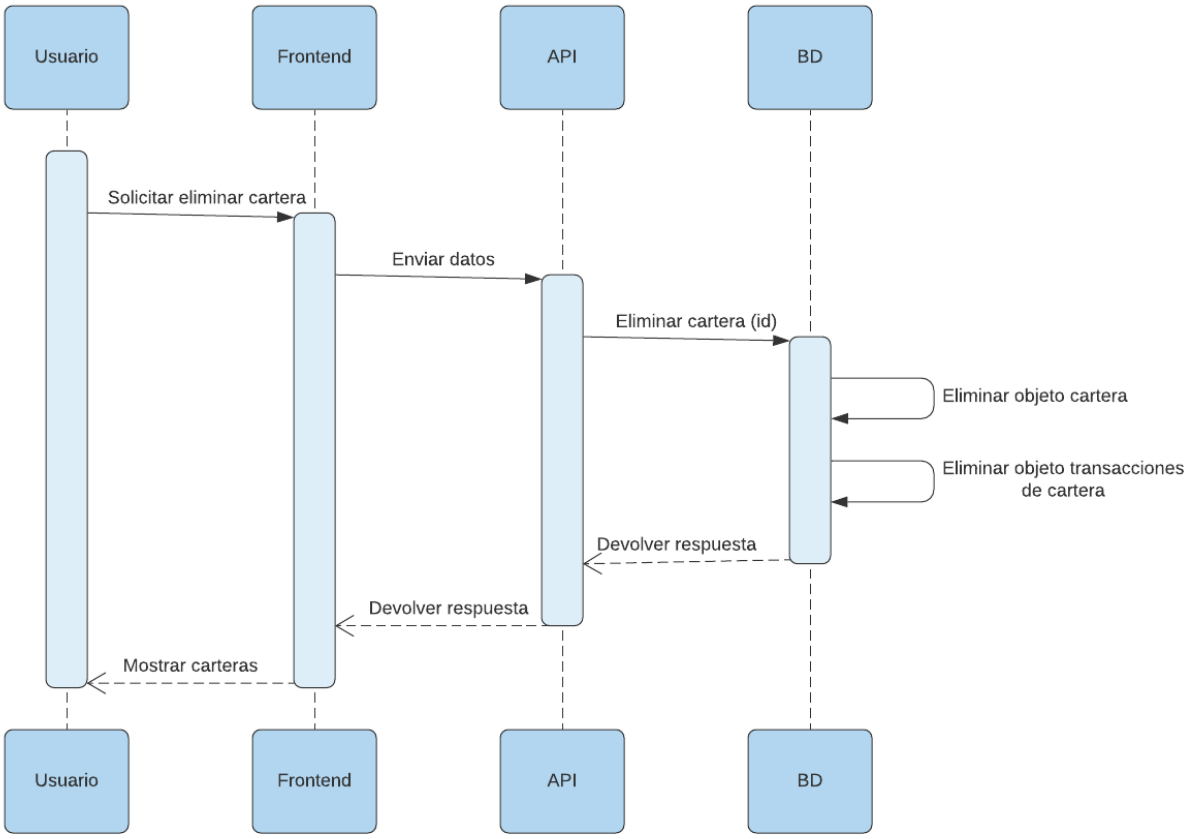


Figura 39. Diagrama de secuencia del caso de uso eliminar cartera

Ver cartera:

Para que el usuario pueda ver una de sus carteras, primero solicita en el Frontend acceder a una cartera. El Frontend controla si la cartera que se quiere ver es propiedad del usuario. Si el usuario no es propietario de la cartera se deniega el ver la cartera y si el usuario si es el propietario se sigue adelante. El Frontend le pide a la API los datos de cartea pasándole el id de la cartera y la divisa seleccionada del usuario. La API obtiene de la BD la lista de transacciones que tiene esa cartera y la lista de empresas de las cuales tiene acciones esa cartera. La API también obtiene de Yahoo Finance la relación en tiempo real del dólar/euro para poder hacer los cambios de divisa de las transacciones de ser necesarios.

Una vez la API tiene esos datos, por cada empresa obtenida se consulta en Yahoo Finance el histórico de precios que ha tenido esa empresa desde que está en cartera, y luego se normalizan los precios (de ser necesario, se cambian de euros a dólares o viceversa). Después, la API le devuelve esos datos ya tratados al Frontend y este calcula las rentabilidades de las transacciones y genera los gráficos necesarios. Finalmente, se le enseña la cartera completa al usuario y este ya podrá hacer operaciones sobre ella (crear transacciones, editar transacciones, eliminar transacciones...).

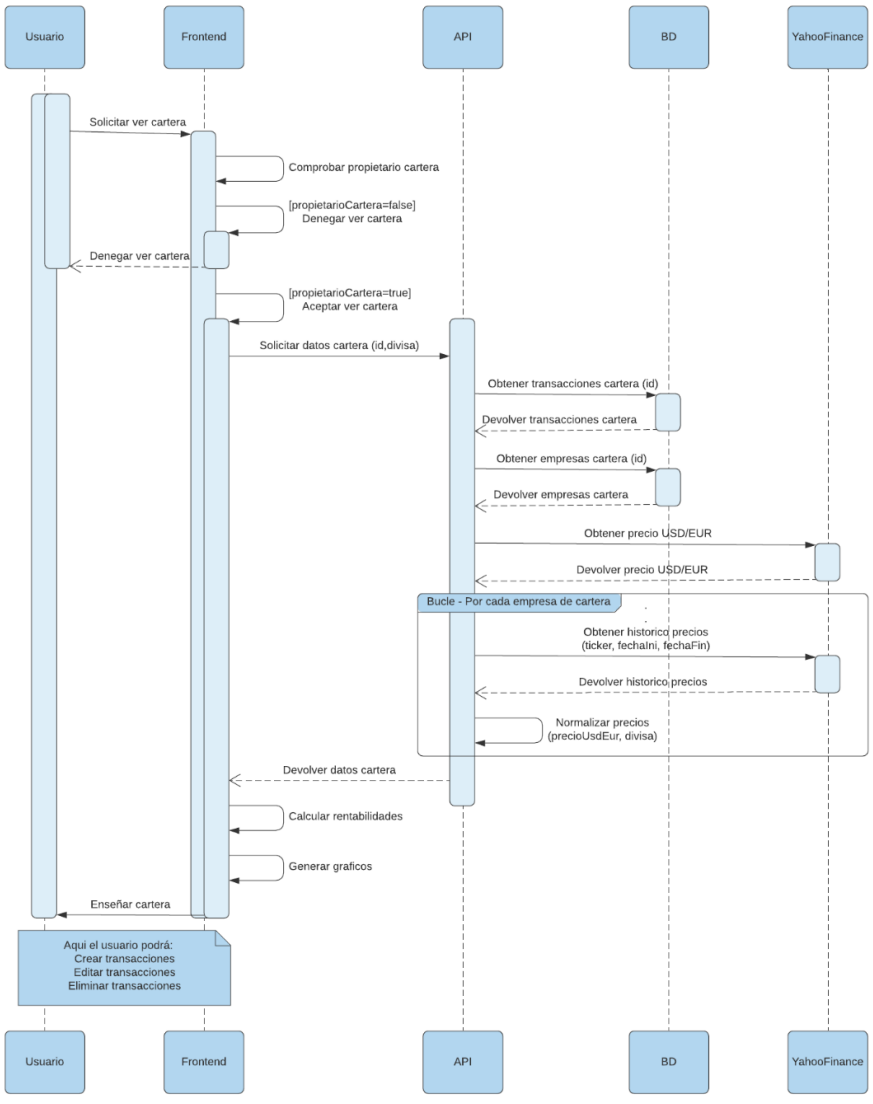


Figura 40. Diagrama de secuencia del caso de uso ver cartera

Crear transacción:

Para que el usuario pueda crear una transacción, primero tiene que solicitar crear una transacción en el Frontend. El Frontend le devolverá al usuario un formulario para introducir los datos de su nueva transacción. Cuando el usuario esté rellenando el formulario, cada vez que cambie la fecha de transacción o la empresa de la que se compran/venden acciones en la transacción, se le sugerirá al usuario el precio de las acciones de la empresa seleccionada en el día seleccionado. Para ello el Frontend le solicitará a la API ese precio pasándole el ticker de la empresa y la fecha. La API obtendrá ese precio de Yahoo Finance y se lo devolverá al Frontend para que este se lo sugiera al usuario.

Una vez el usuario ha rellenado correctamente el formulario y lo valida, el Frontend comprueba la viabilidad de la transacción creada (por ejemplo, no se podrán vender 5 acciones que no tenemos). Si la transacción no es viable se le deniega la creación de la transacción al usuario y puede volver a cambiar los datos en el formulario. Si por el contrario la transacción es viable, se sigue adelante y el Frontend envía los datos para crear la transacción a la API. La API crea en la BD el objeto de la transacción y le devuelve la respuesta al Frontend. Para terminar, el Frontend le muestra al usuario su cartera actualizada con la nueva transacción.

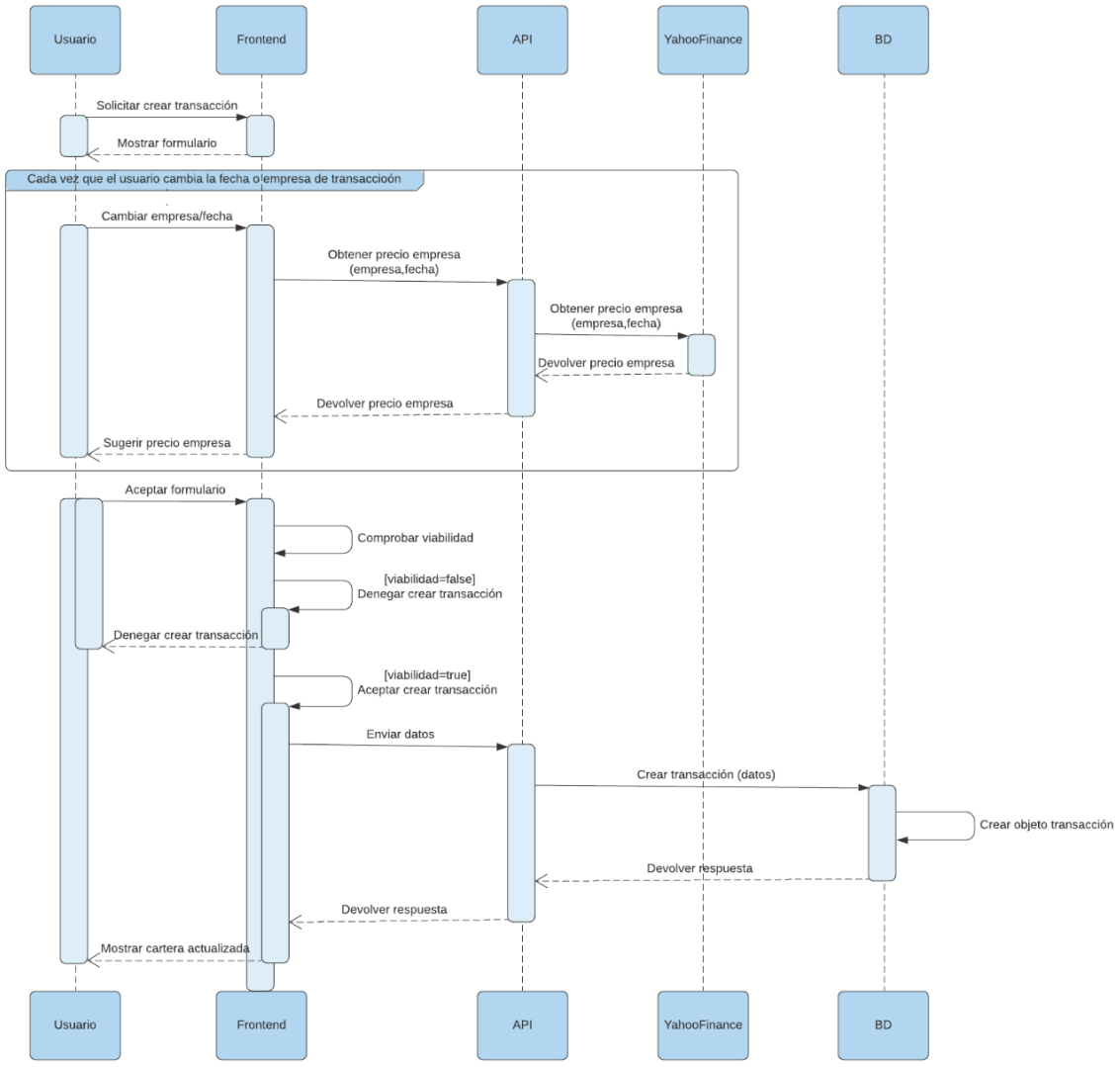


Figura 41. Diagrama de secuencia del caso de uso crear transacción

Editar transacción:

Para que el usuario pueda editar una transacción, primero tiene que solicitar editar una transacción en el Frontend. El Frontend le devolverá al usuario un formulario con los datos de la transacción seleccionada para que pueda editarlos. Cuando el usuario esté editando el formulario, cada vez que cambie la fecha de transacción, se le sugerirá al usuario el precio de las acciones de la empresa seleccionada en el día seleccionado. Para ello el Frontend le solicitará a la API ese precio pasándole el ticker de la empresa y la fecha. La API obtendrá ese precio de Yahoo Finance y se lo devolverá al Frontend para que este se lo sugiera al usuario.

Una vez el usuario ha rellenado correctamente el formulario y lo valida, el Frontend comprueba la viabilidad de la transacción editada (por ejemplo, no se podrán vender 5 acciones que no tenemos). Si la transacción no es viable se le deniega al usuario el editar la transacción y puede volver a cambiar los datos en el formulario. Si por el contrario la transacción es viable, se sigue adelante y el Frontend envía los datos para editar la transacción a la API. La API edita en la BD el objeto de la transacción y le devuelve la respuesta al Frontend. Para terminar, el Frontend le muestra al usuario su cartera actualizada con la transacción editada.

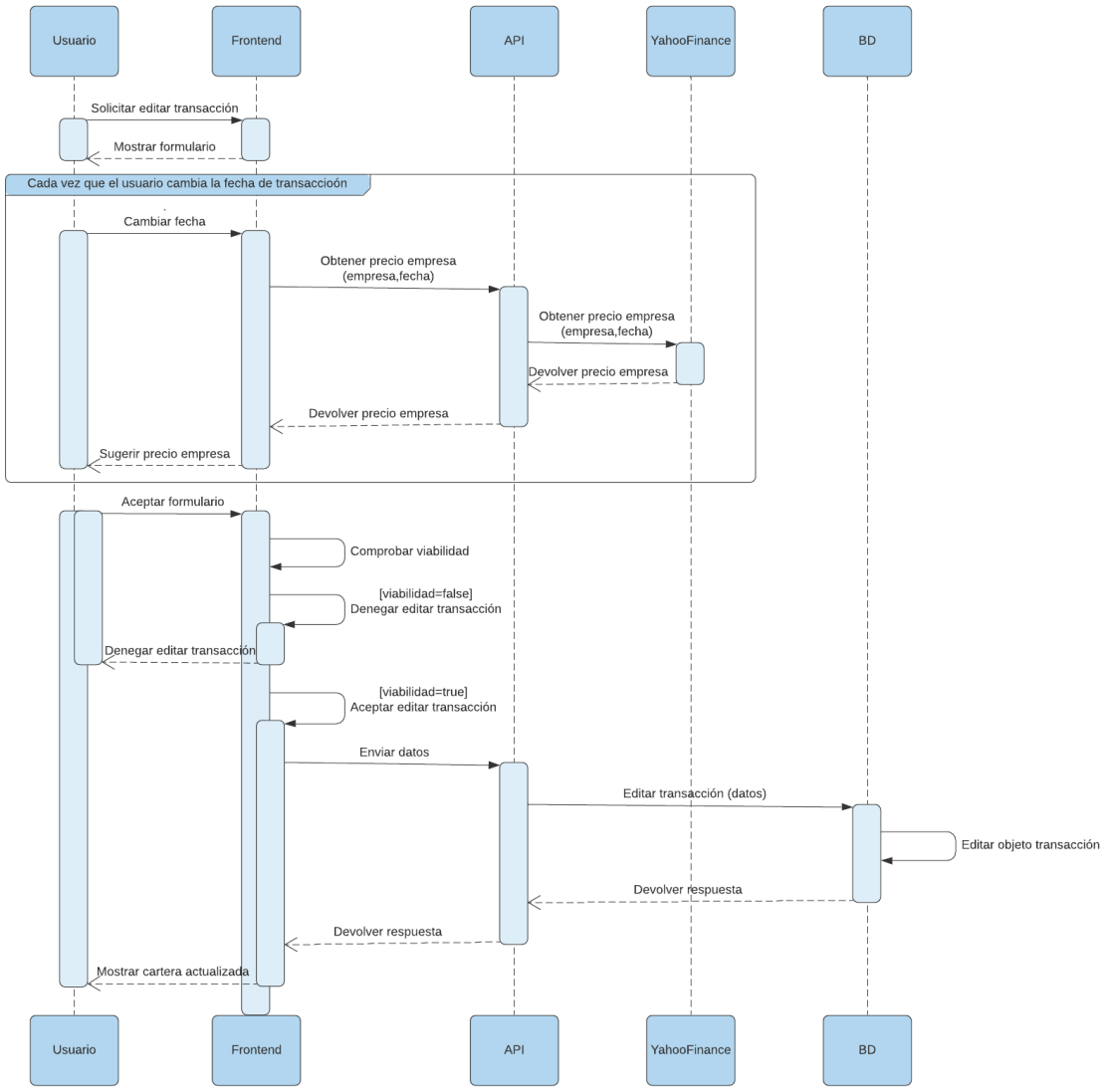


Figura 42. Diagrama de secuencia del caso de uso editar transacción

Eliminar transacción:

Para que el usuario pueda eliminar una transacción, primero tiene que solicitar eliminar esa transacción en el Frontend. El Frontend comprueba la viabilidad de eliminar la transacción (por ejemplo, no se podrá eliminar una compra de 5 acciones si hay una transacción posterior vendiendo esas acciones). Si eliminar la transacción no es viable se le deniega al usuario el eliminarla y se le informa de ello. Si por el contrario, eliminar la transacción es viable, se sigue adelante y el Frontend le solicita eliminar la transacción a la API enviándole el id. La API elimina en la BD el objeto de la transacción y le devuelve la respuesta al Frontend. Para terminar, el Frontend le muestra al usuario su cartera actualizada con la transacción eliminada.

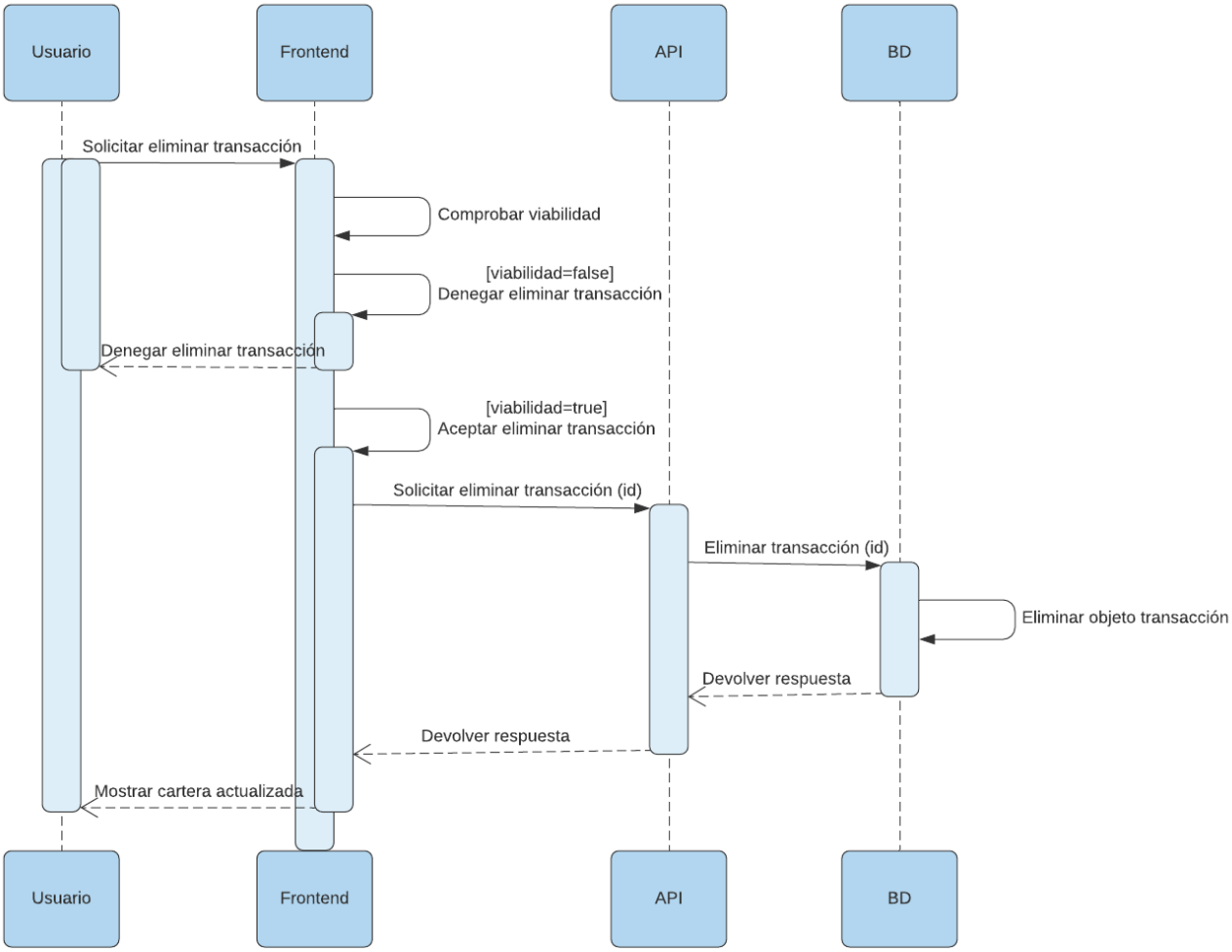


Figura 43. Diagrama de secuencia del caso de uso eliminar transacción

Ver empresas:

Para que el usuario pueda ver las empresas del sistema, primero tiene que acceder a la ventana en el Frontend. Luego, el Frontend llama a la API para obtener las empresas. La API obtiene todas las empresas de la BD, y crea una lista con todos los tickers de las empresas recibidas. Con esta lista de ticker hace una consulta a Yahoo Finance para obtener la información general de dichas empresas. La API corre un bucle en el cual a cada empresa recibida de la BD le asigna la información general obtenida de Yahoo Finance. Una vez hecho esto, le devuelve al Frontend la lista de empresas. Una vez el Frontend ha recibido las empresas, se las muestra al usuario, y el usuario si quiere puede ordenarlas o filtrarlas por campos.

Cada vez que el usuario quiera filtrar u ordenar las empresas, le tendrá que aplicar el filtro u ordenación deseado a la lista de empresas para que el Frontend lo reciba y lo aplique, para luego devolver la lista de empresas filtrada u ordenada al usuario.

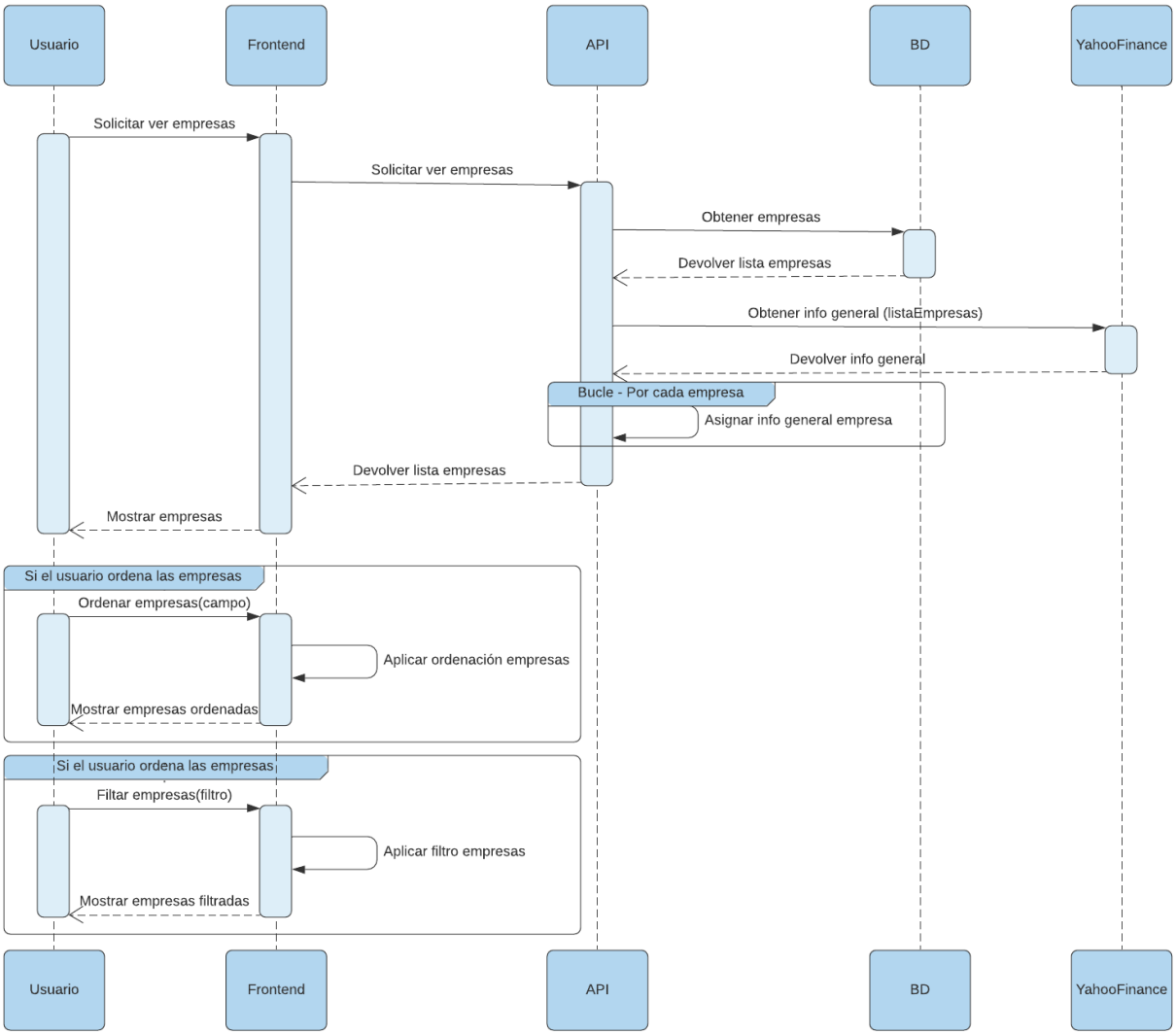


Figura 44. Diagrama de secuencia del caso de uso ver empresas

Ver empresa:

Para que el usuario pueda ver una empresa del sistema, primero tiene que acceder a la ventana de esa empresa en el Frontend (a través del menú superior o a través de la página empresas). Luego, el Frontend llama a la API para obtener los datos de esa empresa pasándole el id. La API obtiene el objeto de la empresa de la BD y usa ese objeto para coger el ticker de la empresa. Después, la API obtiene de Yahoo Finance el histórico de precios y la información de métricas de esa empresa y se los devuelve al Frontend. El Frontend obtiene estos datos y genera las métricas y gráficos necesarios para mostrárselos al usuario. Una vez el usuario ha recibido la empresa, podrá cambiar la vista del gráfico para obtener diferentes visiones de los precios. Cada vez que cambie la vista del gráfico, el Frontend recibirá la configuración que tiene que aplicarle al gráfico y se la aplicará, para finalmente devolvérselo al usuario.

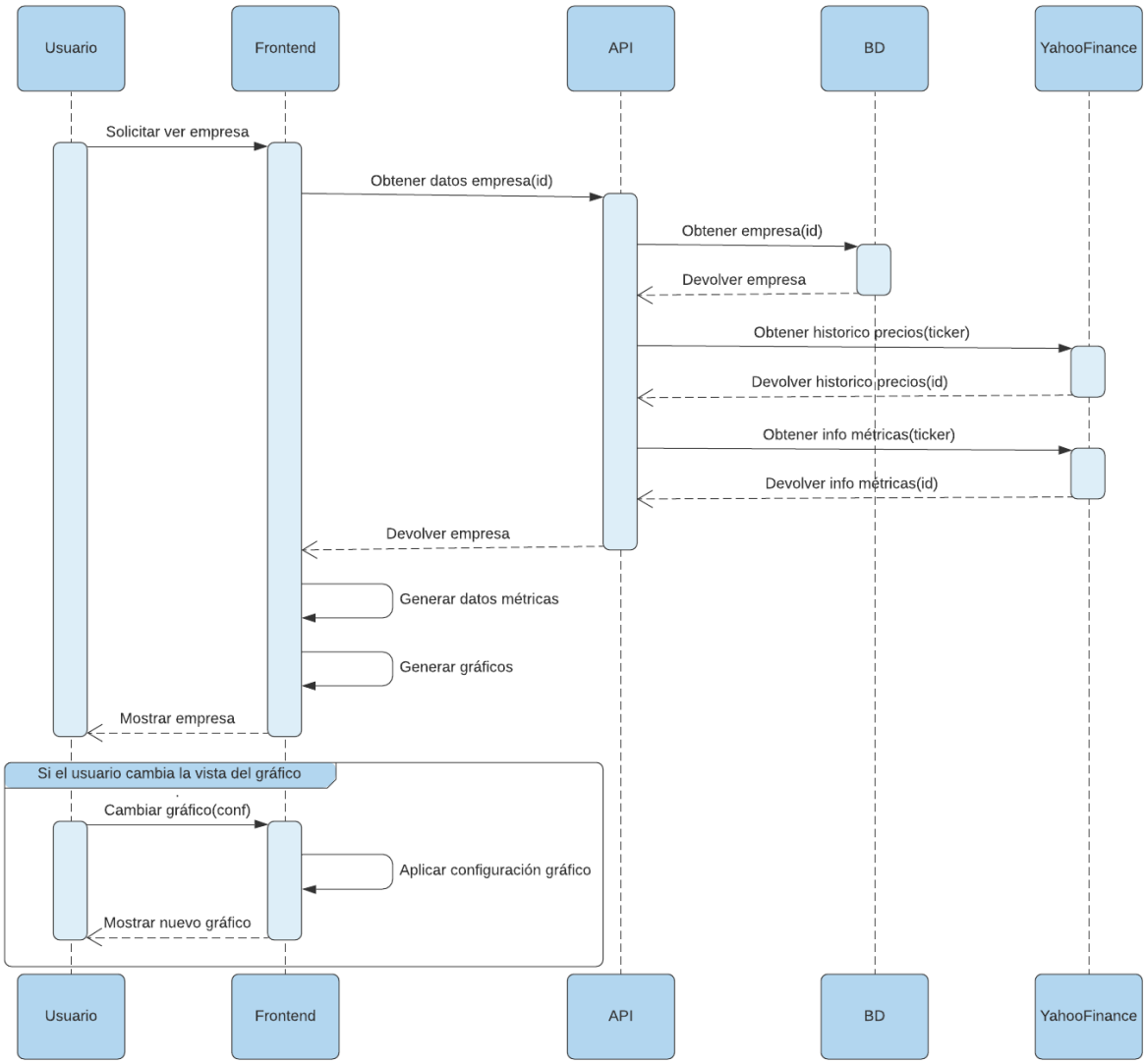


Figura 45. Diagrama de secuencia del caso de uso ver empresa

Añadir/Eliminar favorito:

Para añadir un favorito, el usuario le solicita al Frontend la adición del favorito seleccionando la empresa. El Frontend lo recibe y le manda a la API la orden de añadir el favorito, pasándole el usuario y la empresa. La API recibe la orden y añade el objeto favorito en la BD y le devuelve la respuesta al Frontend. Finalmente, el Frontend le muestra al usuario la respuesta.

Para eliminar un favorito, el usuario le solicita al Frontend el eliminar el favorito seleccionándolo la empresa. El Frontend lo recibe y le manda a la API la orden de eliminar el favorito, pasándole el usuario y la empresa. La API recibe la orden y elimina el objeto favorito en la BD y le devuelve la respuesta al Frontend. Finalmente, el Frontend le muestra al usuario la respuesta.

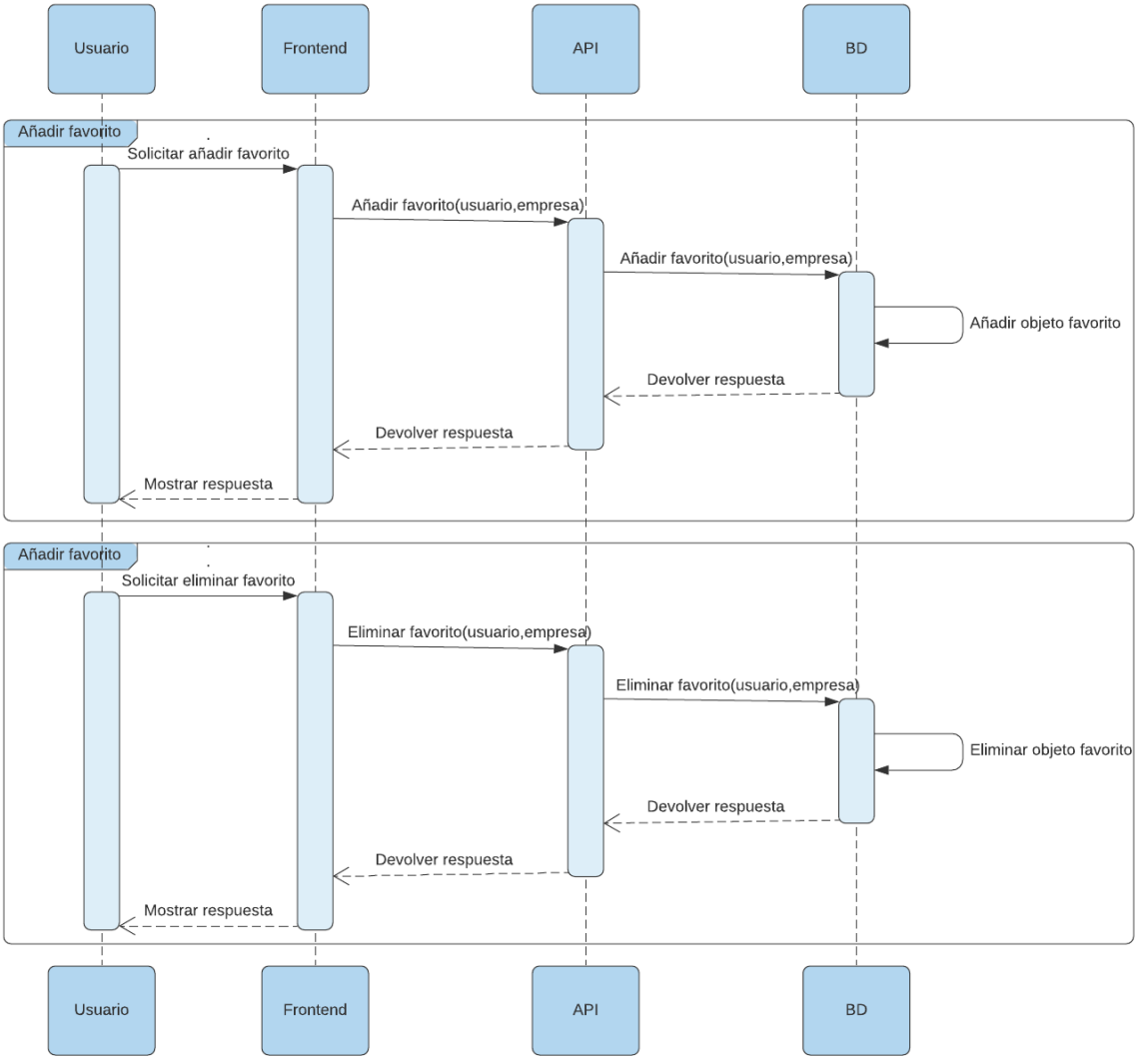


Figura 46. Diagrama de secuencia del caso de uso añadir/eliminar favorito

Guardar datos fiscalidad:

Para que el usuario guarde los datos de fiscalidad, primero tiene que introducir las credenciales en el formulario y enviarlo. Una vez enviado el formulario el Frontend valida los datos para ver si son correctos. Si los datos no son correctos, entonces se le deniega al usuario guardar los datos de fiscalidad. SI los datos son correctos, se procede a guardar los datos enviándolos a la API. Cuando la API recibe los datos, esta edita el objeto usuario con los nuevos datos en la BD y devuelve la respuesta al Frontend. Finalmente, el Frontend le muestra al usuario la respuesta y le actualiza los datos.

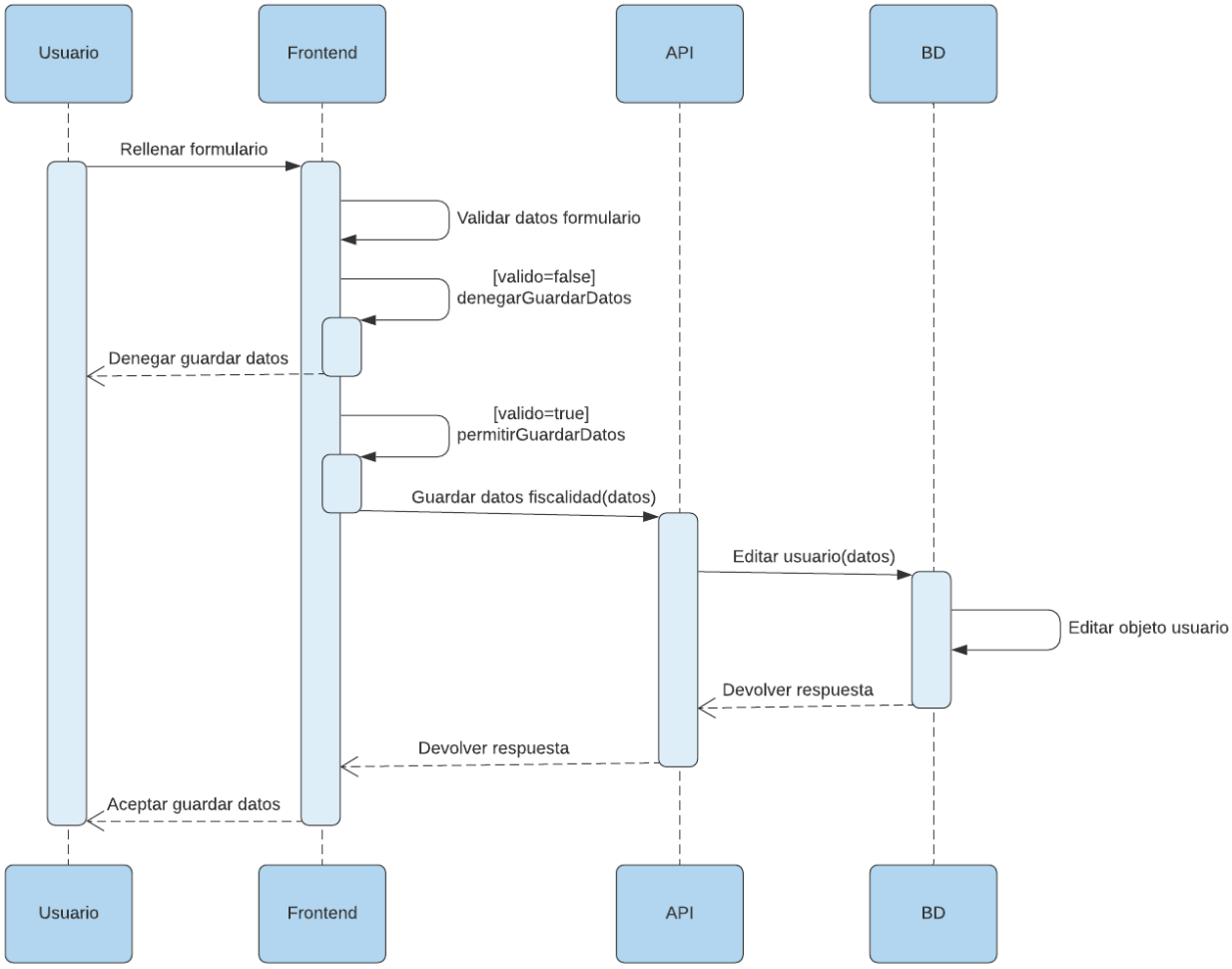


Figura 47. Diagrama de secuencia del caso de uso guardar datos fiscalidad

Crear informe fiscalidad:

Para crear un informe de fiscalidad, primero el usuario tiene que acceder al apartado fiscalidad en el Frontend. Cuando se ejecuta este apartado, el Frontend solicita los datos fiscales del usuario en la API. Cuando la API recibe la llamada, esta obtiene de la BD las carteras del usuario y las transacciones del usuario, para devolvérselas al Frontend. Una vez el Frontend dispone de los datos, se le habilita al usuario la posibilidad de crear un informe.

Para que el usuario cree el informe, primero tiene que solicitarle al Frontend el formulario. Una vez solicitado, el Frontend se lo muestra y el usuario ya puede empezar a seleccionar la configuración de su informe. Cada vez que el usuario cambie la selección de carteras que incluirá el informe, o el año del que se hará el informe, el Frontend lo recibirá y le mostrará una previsualización de todas las transacciones que entrarían en esa configuración de informe. Cuando el usuario haya acabado de configurar el informe, se le solicitará al Frontend la creación del mismo. Finalmente, el Frontend recibe la señal y crea el informe, para así devolvérselo al usuario descargándolo en el navegador.

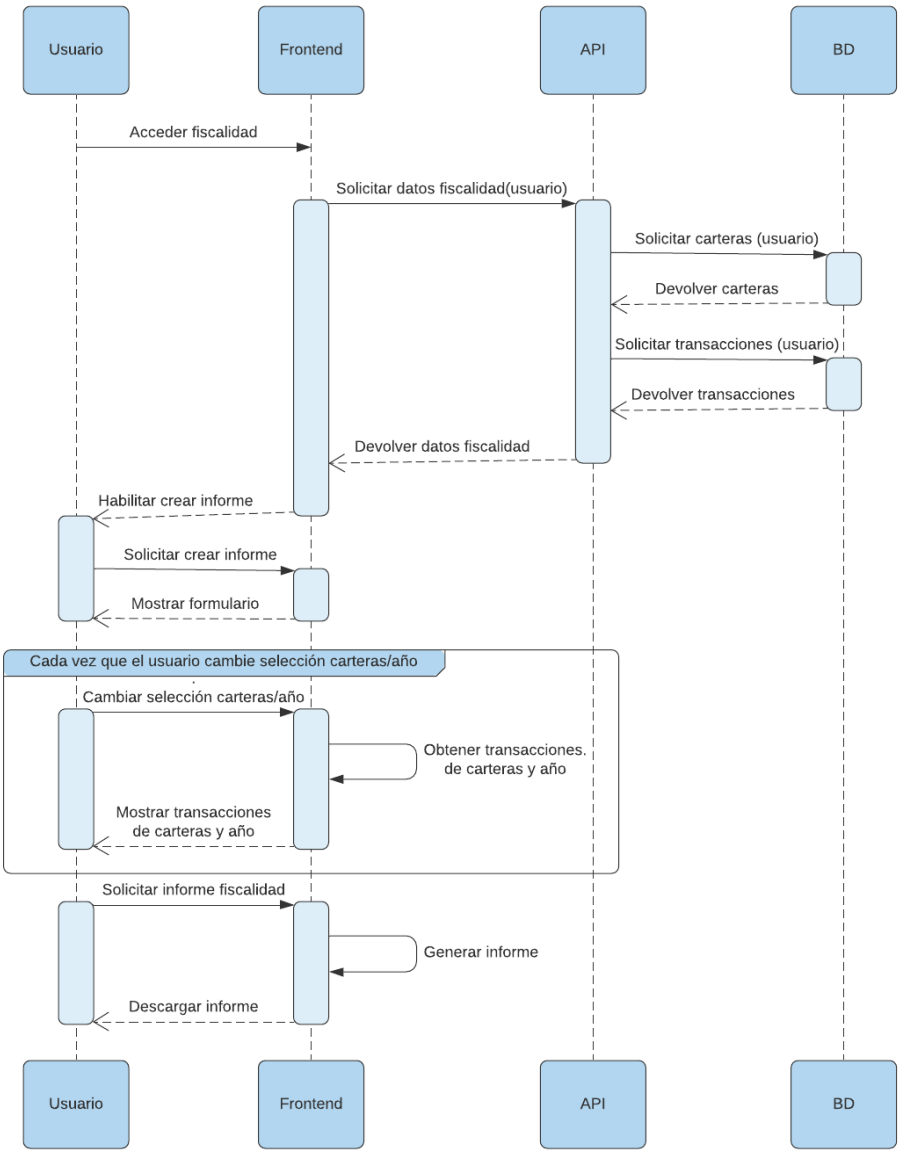


Figura 48. Diagrama de secuencia del caso de uso crear informe fiscalidad

Crear empresa:

Para que el usuario pueda crear una empresa, primero tiene que solicitar la creación en el Frontend, este le devolverá al usuario un formulario para que pueda añadir los datos de la nueva empresa. Para que el usuario pueda validar el formulario, primero tiene que rellenarlo y luego tiene que comprobar el ticker (si el ticker no está incluido en los datos de Yahoo Finance entonces no se podrá usar). Cada vez que el usuario modifique el ticker en el formulario, tendrá que comprobarlo. Para comprobarlo hará click en el botón de comprobar y el Frontend recibirá la señal. Luego, el Frontend le solicitará a la API comprobar el ticker, para que la API mande el ticker a Yahoo Finance para comprobar si existe o no. La API recibirá la respuesta y se la mandará al Frontend. Si el ticker no es válido, se le pedirá al usuario que use otro, y si es válido, se continuará con la operación.

Una vez el usuario ha rellenado correctamente el formulario con un ticker comprobado ya puede mandárselo al Frontend. Después, el Frontend envía los datos para crear la empresa a la API. La API crea en la BD el objeto de la empresa y le devuelve la respuesta al Frontend. Para terminar, el Frontend actualiza la lista de empresas en el sistema y le muestra al usuario esta lista actualizada.

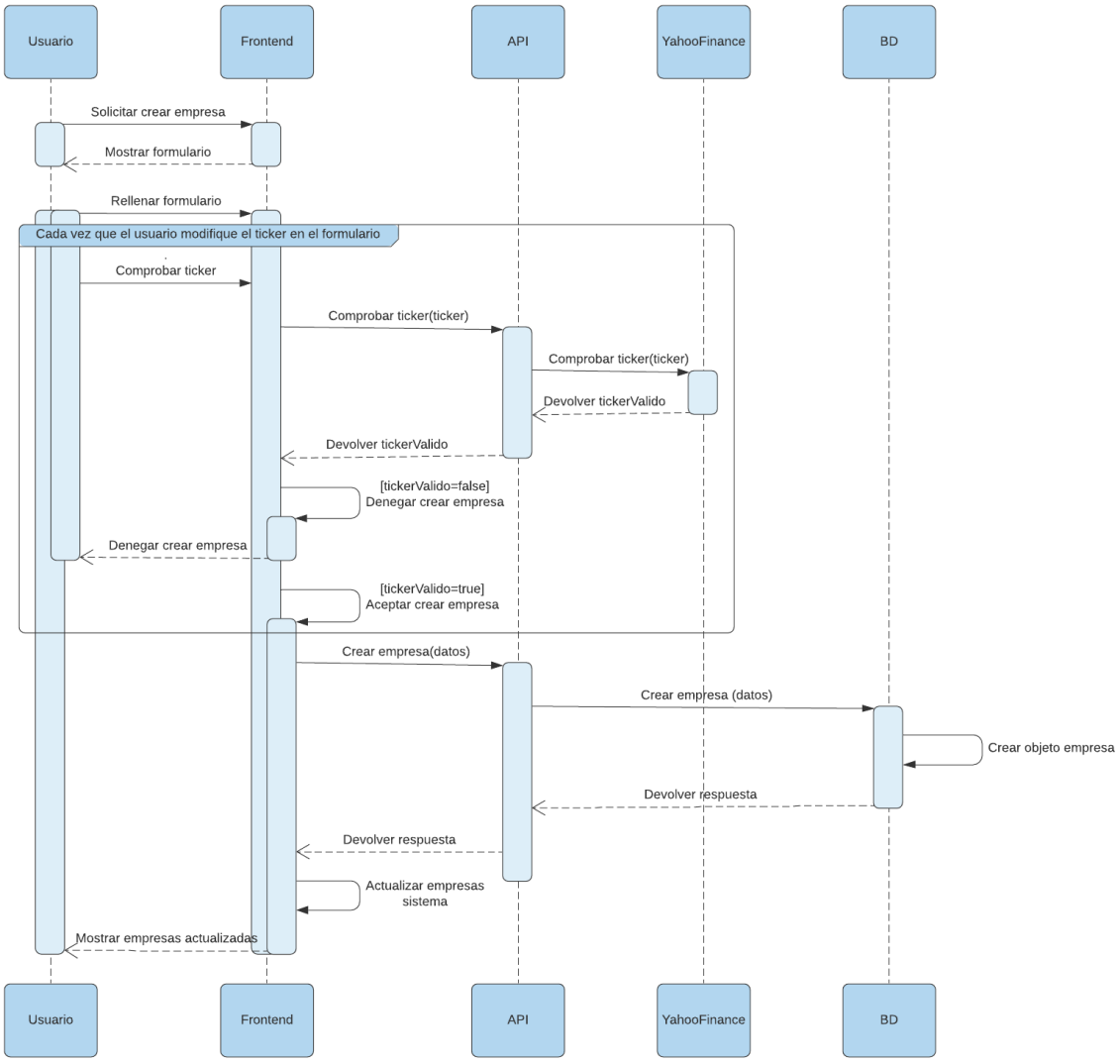


Figura 49. Diagrama de secuencia del caso de uso crear empresa

Editar empresa:

Para que el usuario pueda editar una empresa, primero tiene que solicitar la edición en el Frontend, este le devolverá al usuario un formulario con los datos de la empresa a editar para que pueda cambiarlos. Para que el usuario pueda validar el formulario, primero tendrá que comprobar el nuevo ticker (si el ticker no está incluido en los datos de Yahoo Finance entonces no se podrá usar). Cada vez que el usuario modifique el ticker en el formulario, tendrá que comprobarlo. Para comprobarlo hará click en el botón de comprobar y el Frontend recibirá la señal. Luego, el Frontend le solicitará a la API comprobar el ticker, para que la API mande el ticker a Yahoo Finance para comprobar si existe o no. La API recibirá la respuesta y se la mandará al Frontend. Si el ticker no es válido, se le pedirá al usuario que use otro, y si es válido, se continuará con la operación.

Una vez el usuario ha cambiado correctamente el formulario con un ticker comprobado ya puede mandárselo al Frontend. Después, el Frontend envía a la API los datos y el id para editar la empresa. La API edita en la BD el objeto de la empresa y le devuelve la respuesta al Frontend. Para terminar, el Frontend actualiza la lista de empresas en el sistema y le muestra al usuario esta lista actualizada.

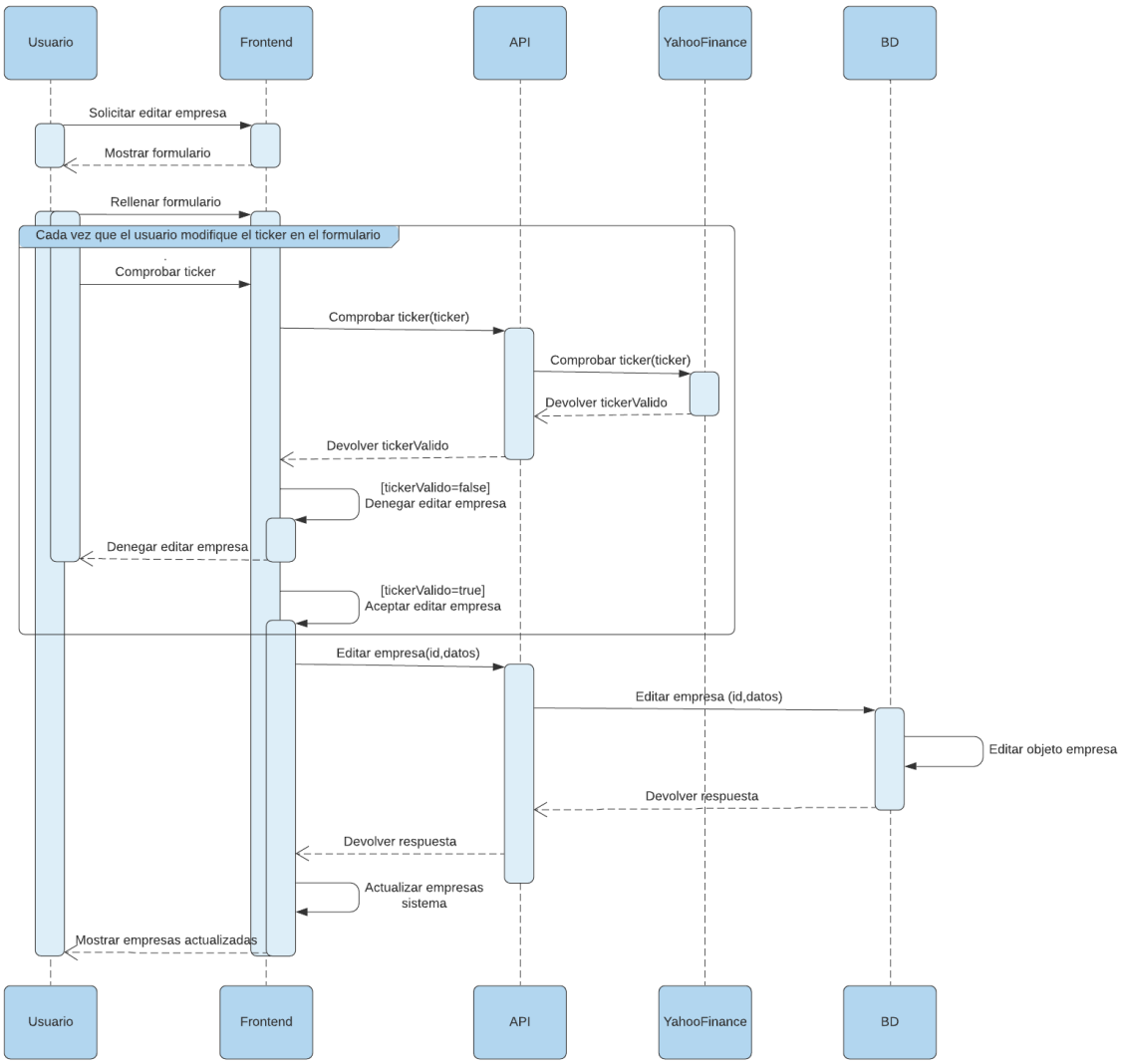


Figura 50. Diagrama de secuencia del caso de uso editar empresa

Eliminar empresa:

Para que el usuario pueda eliminar una empresa, primero solicita en el Frontend la empresa que quiere eliminar. El Frontend envía el id de la empresa que hay que eliminar a la API. La API elimina en la BD el objeto de la empresa y también todas las transacciones asociadas a esa empresa y los favoritos asociados a esa empresa. Luego, la API le devuelve la respuesta al Frontend. Para finalizar, el Frontend actualiza la lista de empresas en el sistema y le muestra al usuario esta lista actualizada.

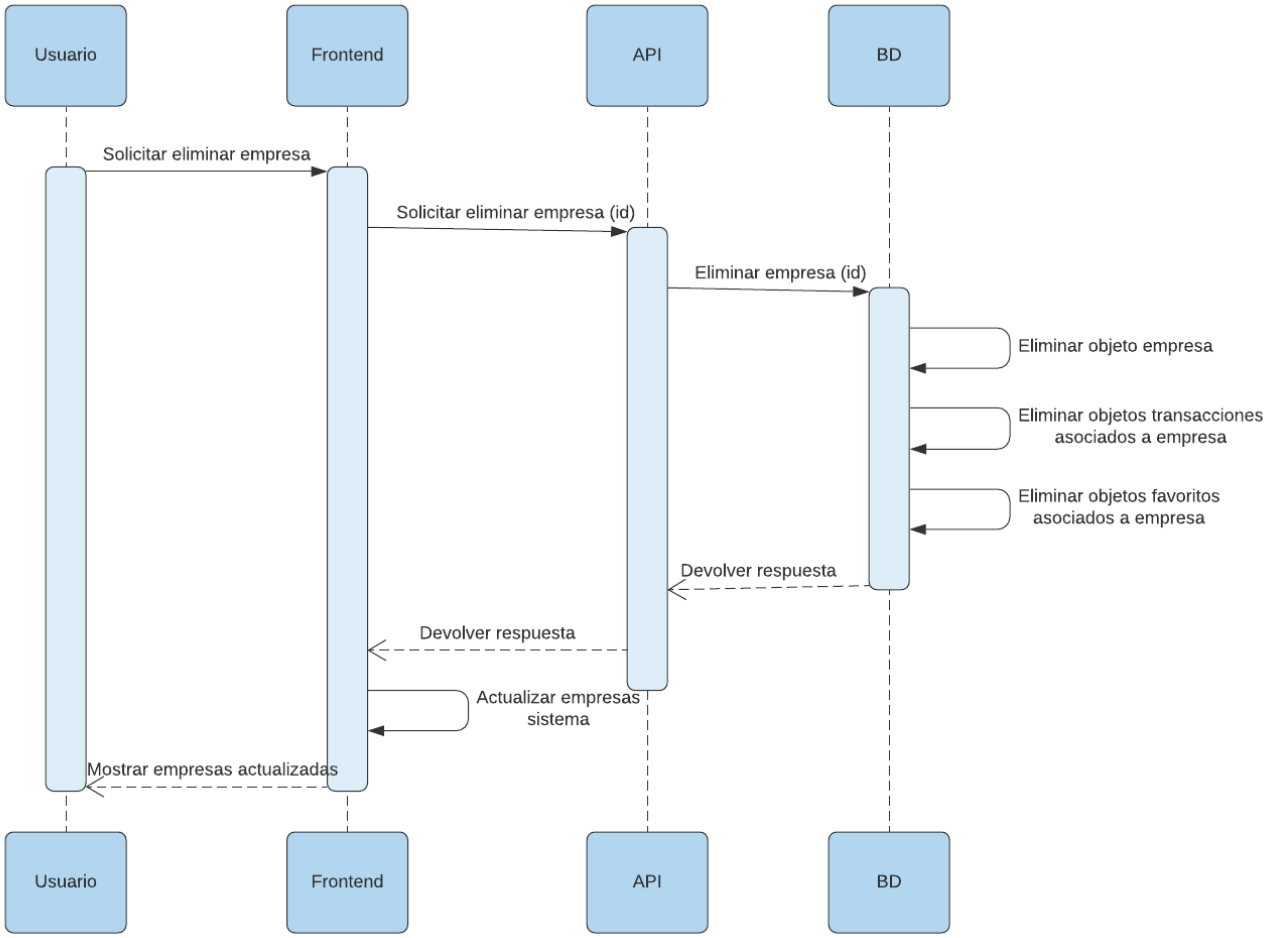


Figura 51. Diagrama de secuencia del caso de uso eliminar empresa

Cambiar idioma:

Para que el usuario pueda cambiar el idioma, primero elige en el Frontend el idioma que quiere seleccionar. El Frontend envía el id del usuario y el idioma seleccionado a la API. La API edita el objeto usuario en la BD editándole el idioma. Luego, la API le devuelve la respuesta al Frontend. Para finalizar, el Frontend actualiza el idioma de todo el sistema y le muestra el sistema actualizado al usuario.

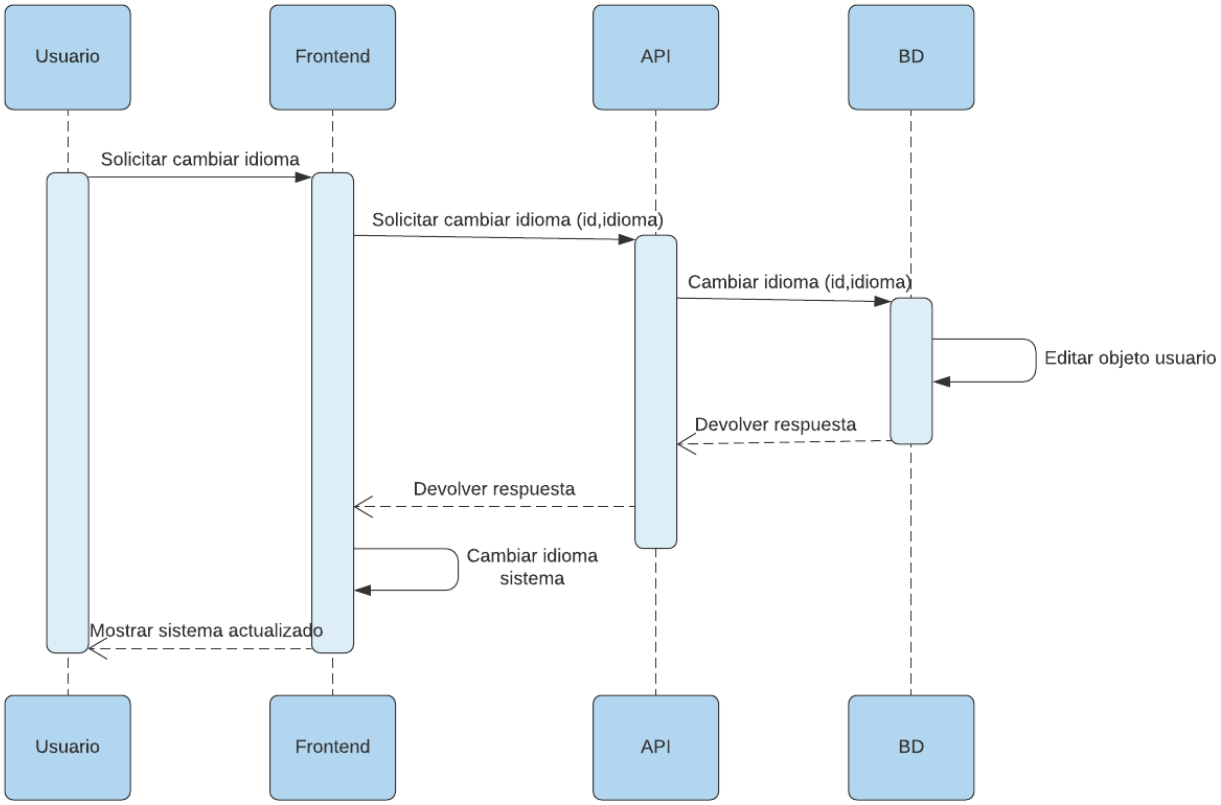


Figura 52. Diagrama de secuencia del caso de uso cambiar idioma

Cambiar divisa:

Para que el usuario pueda cambiar la divisa, primero elige en el Frontend la divisa que quiere seleccionar. El Frontend envía el id del usuario y la divisa seleccionada a la API. La API edita el objeto usuario en la BD editándole la divisa. Luego, la API le devuelve la respuesta al Frontend. Para finalizar, el Frontend actualiza la divisa de todo el sistema y le muestra el sistema actualizado al usuario.

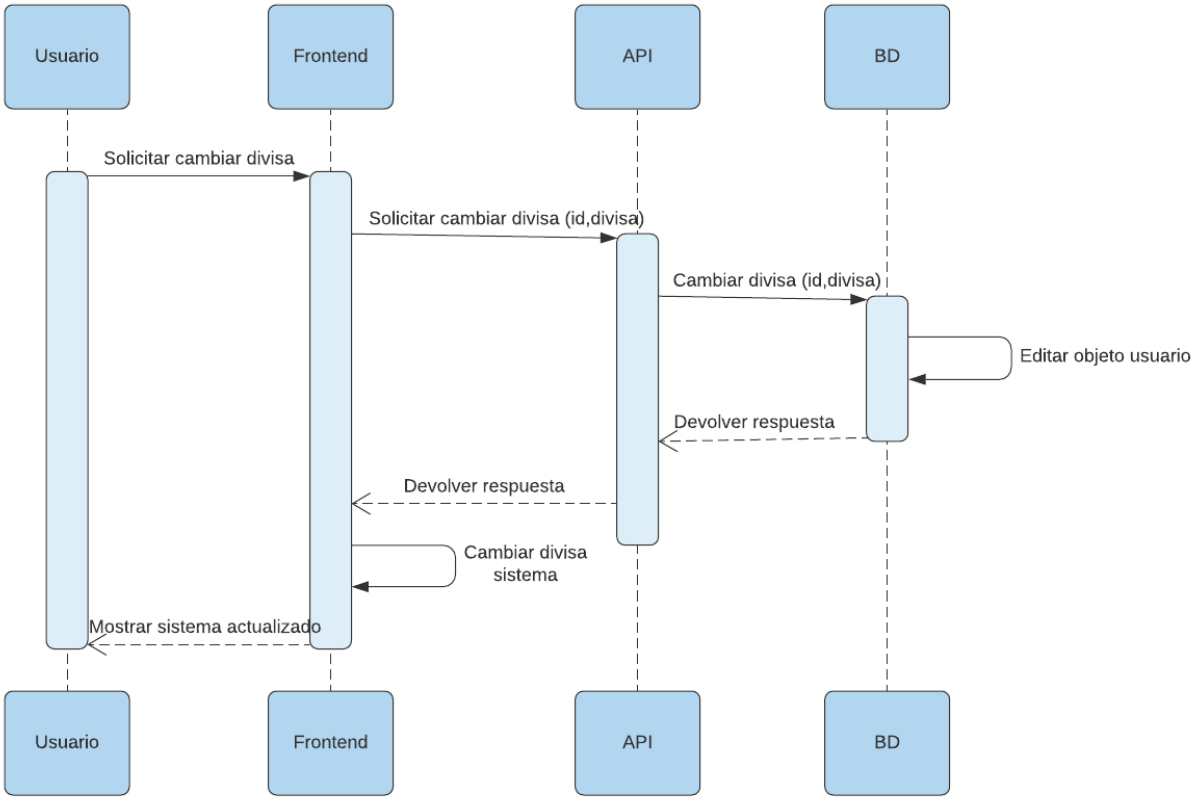


Figura 53. Diagrama de secuencia del caso de uso cambiar divisa

6. Implementación

Este capítulo servirá para comprender y detallar como se ha llevado a cabo la implementación del proyecto. Se especificará la arquitectura utilizada y la aplicación de la misma y también se hará hincapié en las tecnologías usadas y la manera de aplicar las mismas

6.1. Arquitectura de la aplicación

6.1.1. Especificaciones de arquitectura

En este apartado se definirá la arquitectura necesaria para poner en funcionamiento la aplicación web, la gestión del código y cómo se conectará con la máquina que albergue la aplicación de modo que pueda ser accesible a través de un navegador web. Así mismo, se explicará el funcionamiento de toda la arquitectura, todos los apartados de la misma y la razón de haber elegido esta arquitectura.

La arquitectura usada se basa en la arquitectura de tres niveles, añadiéndole un nivel más. En el caso de la arquitectura de tres niveles, los apartados que lo forman son los siguientes: el cliente, el servidor de negociación y la base de datos. En la arquitectura usada en este proyecto se añade un nivel más, dividiendo el servidor de negociación en dos (servidor web y API). De esta manera, el cliente se conectará al servidor WEB para poder ejecutar la aplicación en su navegador. El servidor WEB se conectará con la API, que es la que hará las llamadas a la base de datos y nada más.

Separando en dos apartados el servidor de negociación conseguimos darle un alcance mayor al proyecto y facilitar la gestión del mismo, tanto a la hora de desarrollar como a la hora de mantener actualizada la aplicación. Esto se debe a que, si hacemos un cambio mínimo en alguna de las dos partes, solo habría que subir esa parte y dejar la otra. También hay que tener en cuenta que, al estar en dos servidores totalmente diferentes, si uno de los dos servidores cae, el otro estaría totalmente funcional y sería más fácil localizar el error.

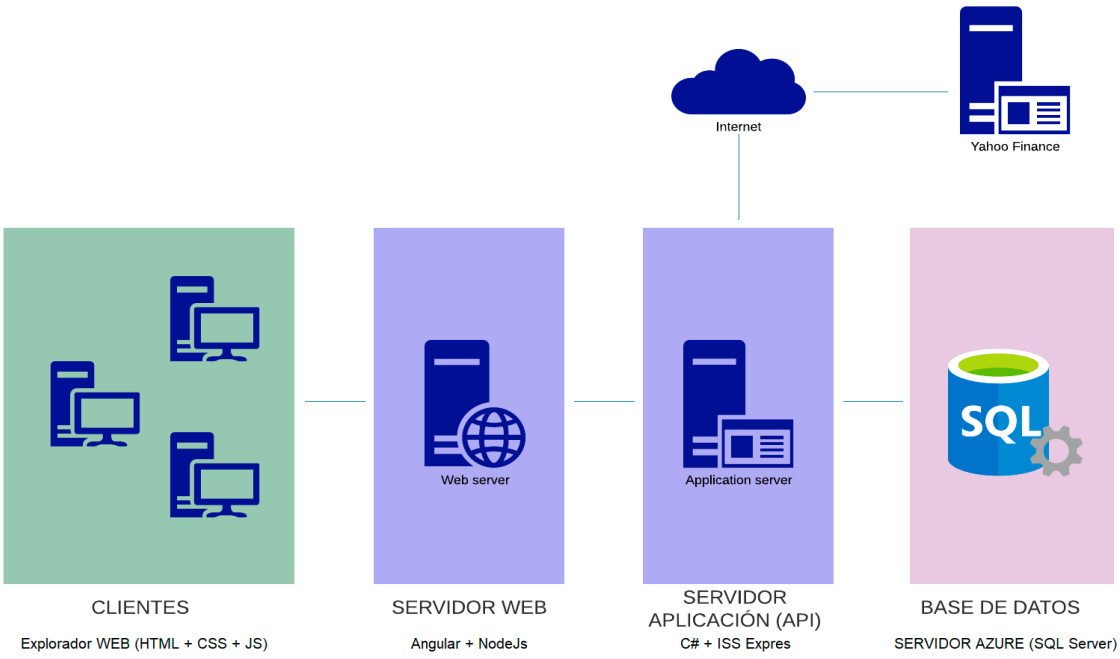


Figura 54. Esquema de la arquitectura de la aplicación

6.1.2. Definición de arquitectura

En este apartado se hará una breve descripción de cada uno de los puntos de la arquitectura. Más adelante se ahondará en sus especificaciones, en las tecnologías utilizadas y en la manera de aplicar las mismas. Las partes de la arquitectura son las siguientes:

- Clientes:

  Es la parte donde el cliente ejecuta la aplicación y la hace funcionar. El cliente ejecuta la aplicación en el dispositivo que quiera (ordenadores, tablets, dispositivos móviles…) desde cualquier lugar accediendo a la url de la web. Cada vez que se accede a la url, se "descarga" una copia de la aplicación en el dispositivo y se ejecuta esta copia en el navegador, como si de una página web normal se tratase.

- Servidor WEB:

  Como hemos dicho, cada vez que el usuario accede a una url de la aplicación, se descarga una copia de esta aplicación completa. Es decir, después de haber accedido por primera vez, si se navega por la aplicación, aunque cambiemos de páginas, no se accederá al servidor WEB otra vez, porque ya tenemos descargada toda la WEB en el navegador. Esto es gracias a la tecnología usada para desarrollar la WEB: Angular, de la que más adelante hablaremos.

- Servidor Aplicación (API):

  Este apartado es el que se ocupa de conectar la WEB con las fuentes de datos de la aplicación, que en este caso son dos (base de datos y Yahoo Finance). La API se limita a recibir llamadas de la WEB, gestionar esas llamadas y obtener los datos que solicitan, para finalmente devolver una respuesta.

- Base de datos (BD):

  La base de datos es la que se ocupa de guardar todos los datos que nutren la aplicación (usuarios, empresas, carteras…). La base de datos es una colección organizada de datos almacenados electrónicamente en un sistema de computadora. En este caso, la base de datos está almacenada en un servidor de Azure, que permite tener una gran velocidad y una amplia cobertura ante posibles pérdidas de datos.

- Yahoo Finance:

  Finalmente, en la aplicación también se usa una API externa, esta API la proporciona gratuitamente la empresa Yahoo y se llama Yahoo Finance. La plataforma de finanzas de Yahoo es conocida en el mundo de las finanzas por ser una de las más completas y de las que más datos tiene. A parte de esto, Yahoo habilita una API a la que se le pueden hacer consultas para poder obtener los mismos datos que muestran en su plataforma. La API de Yahoo Finance ayuda a consultar toda la información sobre resúmenes financieros, acciones, cotizaciones… la cual necesitamos en nuestra plataforma. Más adelante se tratará en más profundidad el funcionamiento.

6.2. Tecnologías utilizadas

Durante el siguiente capítulo describiremos lo que han sido las tecnologías aplicadas en el proyecto. Se citarán aquellas que han sido elegidas y cuyo uso ha sido general en el desarrollo de las funcionalidades de la aplicación. También se hará hincapié en aquellas más específicas que han sido elegidas para soluciones más particulares o excepcionales. A continuación, se expondrá la razón de haber elegido cada una de ellas y las ventajas que nos presentan.

6.2.2. Angular

Angular es un framework para aplicaciones web desarrollado en TypeScript[5] que se utiliza para crear y mantener aplicaciones web de una sola página. Esto quiere decir que al usar Angular solo se accede al servidor WEB la primera vez que se ejecuta la aplicación. Es decir, se descarga la aplicación en su totalidad la primera vez y ya se podrá navegar sin necesidad de volver a acudir a ese servidor. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (MVC), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles.

La biblioteca lee el HTML que contiene atributos de las etiquetas personalizadas adicionales, entonces obedece a las directivas de los atributos personalizados, y une las piezas de entrada o salida de la página a un modelo representado por las variables estándar de JavaScript.

Angular se basa en clases tipo "Componentes", cuyas propiedades son las usadas para hacer el binding de los datos con el HTML. En dichas clases tenemos variables y métodos.

Como hemos dicho, el lenguaje de programación que usa Angular es TypeScript. Este lenguaje es un superconjunto de JavaScript, que esencialmente añade tipos estáticos y objetos basados en clases. TypeScript es usado para desarrollar aplicaciones JavaScript que se ejecutarán en el lado del cliente o del servidor. TypeScript extiende la sintaxis de JavaScript, por tanto, cualquier código JavaScript existente funciona sin problemas. Está pensado para grandes proyectos, los cuales a través de un compilador de TypeScript se traducen a código JavaScript original.

Figura 55. Imagen corporativa de las ventajas de

[5] TypeScript: Es un lenguaje de programación libre y de código abierto. Es un superconjunto de JavaScript, que esencialmente añade tipos estáticos y objetos basados en clases.

6.2.2.1. Organización de ficheros

I. Carpeta "src". En la carpeta src tenemos las siguientes carpetas reseñables:

I.I. Carpeta "app": Esta carpeta contendrá toda la lógica de nuestro proyecto. Incluye los siguientes ficheros:

I.I.I. Componentes: Un componente en Angular es un bloque de código reutilizable, que consta básicamente de tres archivos: un CSS (opcional), un HTML (también conocido como plantilla o en inglés, template) y un TypeScript. El CSS servirá para darle estilo al componente, el HTML define la estructura básica del componente y el TypeScript formará toda la lógica del componente. Estos tres archivos están introducidos en una carpeta que tendrá el nombre que nosotros queramos darle (el nombre del componente).

En resumidas cuentas, estos componentes son todas las partes o ventanas que tendrá nuestra aplicación. Es decir, cuando queramos hacer una ventana nueva en la aplicación, crearemos el componente correspondiente, por lo tanto, nuestra aplicación tendrá tantas ventanas como componentes. Como normal general (que tiene sus excepciones) todos los componentes de un proyecto de Angular deben ser subcarpetas de la carpeta principal "app".

I.I.II. Componente "app": Cabe mencionar que la carpeta app en sí, también es un componente, aunque es un componente un tanto especial, ya que es el componente principal de la aplicación y la base de la WEB. Los tres archivos anteriormente mencionados que forman un componente, los podemos ver en la raíz de la carpeta app («app.component.html», «app.component.ts» y «app.component.css»). Desde este componente se llamarán a todos los demás componentes y se acabará formando la web en su totalidad. Pero como ya hemos dicho, es un componente un tanto especial, y es por eso que cuenta con otros dos archivos muy importantes en el proyecto:

I.I.II.I. app.module.ts: Este archivo es un módulo de angular. Angular nos ofrece múltiples funcionalidades y las organiza en módulos, algo así como paquetes de funcionalidades relacionadas entre sí. Por ejemplo, el módulo «FormsModule», es un conjunto de funcionalidades relacionadas con los formularios.

«AppModule» es un módulo un tanto especial, porque es donde recopilamos (entre otras cosas) la lista de componentes que van a construir nuestra app. En cuanto a las propiedades de las que consta, baste decir que las «declarations» es el array donde irán nuestros componentes y los «imports» el array donde importaremos otros módulos si fuera necesario.

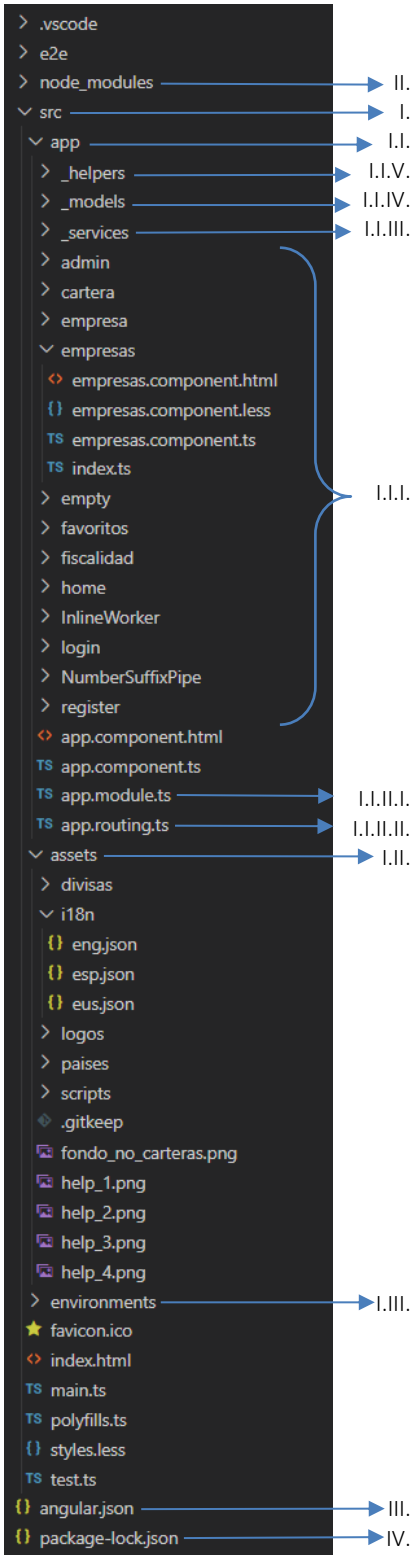


Figura 56. Organización de ficheros de Angular

I.I.II.II. app.routing.ts: En cualquier sitio web generalmente tienes varias direcciones que son entregadas por un servidor, para mostrar diferentes contenidos del sitio. Cada una de esas páginas se presenta en una ruta diferente del sitio web. Cada una de esas rutas podría tener un archivo HTML, que se sirve con el contenido de esa sección.

Sin embargo, en las aplicaciones Angular sólo tenemos una página, el index.html y toda la acción se desarrolla dentro de esa página. En Angular lo común es que el «index» sólo tenga un componente en su «BODY» y realmente toda la acción se desarrollará en ese componente. Todas las "páginas" (pantallas o componentes) del sitio web se mostrarán sobre ese índex, intercambiando el componente que se esté visualizando en cada momento.

Para facilitar la navegación por un sitio donde realmente sólo hay un «index», existe lo que llamamos el sistema de routing, que tiene el objetivo de permitir que en el sitio web haya rutas internas, respondiendo a rutas "virtuales" como las que existen en los sitios tradicionales. El sistema de routing es el encargado de reconocer cuál es la ruta que el usuario quiere mostrar, presentando la pantalla correcta en cada momento. Estos son los elementos básicos que forman parte del routing:

- El módulo del sistema de rutas: llamado «RouterModule».
- Rutas de la aplicación: es un array con un listado de rutas que nuestra aplicación soportará.
- Enlaces de navegación: son enlaces HTML en los que incluiremos una directiva para indicar que deben funcionar usando el sistema de routing.
- Contenedor: donde colocar las pantallas de cada ruta. Cada pantalla será representada por un componente.

Cabe mencionar que en este apartado también podremos decidir quien tiene acceso a cada página, esto es gracias al servicio de autenticación. Usando este servicio podemos poner la condición de que solo se pueda acceder a ciertas ventanas si previamente has hecho el login con un usuario válido, poniendo en el routing la etiqueta «[AuthGuard]». En este caso sin hacer login solo se puede acceder a las ventanas login y registro.

```
import { Routes, RouterModule } from '@angular/router';

import { HomeComponent } from './home';
import { LoginComponent } from './login';
import { RegisterComponent } from './register';
import { CarteraComponent } from './cartera';
import { EmpresasComponent } from './empresas';
import { AdminComponent } from './admin';
import { FiscalidadComponent } from './fiscalidad';
import { FavoritosComponent } from './favoritos';
import { EmpresaComponent } from './empresa';
import { EmptyComponent } from './empty';
import { AuthGuard } from './_helpers';

const routes: Routes = [
    { path: 'login', component: LoginComponent },
    { path: 'register', component: RegisterComponent },
    { path: '', component: HomeComponent, canActivate: [AuthGuard] },
    { path: 'cartera/:idCartera', component: CarteraComponent, canActivate: [AuthGuard] },
    { path: 'empresas', component: EmpresasComponent, canActivate: [AuthGuard] },
    { path: 'admin', component: AdminComponent, canActivate: [AuthGuard] },
    { path: 'fiscalidad', component: FiscalidadComponent, canActivate: [AuthGuard] },
    { path: 'favoritos', component: FavoritosComponent, canActivate: [AuthGuard] },
    { path: 'empresa/:idEmpresa', component: EmpresaComponent, canActivate: [AuthGuard] },
    { path: 'empty', component: EmptyComponent, canActivate: [AuthGuard] },

    // otherwise redirect to home
    { path: '**', redirectTo: '' }
];

export const appRoutingModule = RouterModule.forRoot(routes, { relativeLinkResolution: 'legacy' });
```

Figura 57. Archivo app.routing.ts

I.I.III. Carpeta "_services": Esta carpeta contiene los servicios usados en la aplicación. El uso de servicios es importante para nutrir de datos a los componentes. Como hemos dicho anteriormente, el protagonista en las aplicaciones de Angular es el componente, y toda la aplicación se desarrollan en base a un árbol de componentes. Sin embargo, a medida que nuestro proyecto se vuelve más complejo y de más tamaño, lo normal es que el código de los componentes también vaya aumentando, implementando mucha lógica del modelo de negocio.

Por lo tanto, es importante mantener la organización del código, manteniendo piezas pequeñas de responsabilidad reducida, para que posteriormente sea más sencillo editarlo. Además, siempre llegará el momento en el que dos o más componentes tengan que acceder a los mismos datos y hacer operaciones similares con ellos, que podrían obligarnos a repetir código. Para solucionar estas situaciones tenemos a los servicios.

Un servicio es un proveedor de datos, que mantiene lógica de acceso a ellos y operativa relacionada con el negocio y las cosas que se hacen con los datos dentro de una aplicación. Los servicios serán consumidos por los componentes, que delegarán en ellos la responsabilidad de acceder a la información y la realización de operaciones con los datos.

Se podría decir que, dentro de la aplicación de angular, los servicios son lo que conectan los componentes con la API.

A continuación, se muestra un ejemplo de uno de los servicios usados en la aplicación. Como podemos ver, se limita a hacer llamadas a la API y devolver la respuesta en formato JSON.

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { environment } from '@environments/environment';

const baseUrl = `${environment.apiUrl}`;

@Injectable()
export class FinanceService {

  constructor(private http: HttpClient) { }

  GetUsuario(usuario: string, pass: string) {
    return this.http.post<JSON>(`${baseUrl}/GetUsuario`, { usuario: usuario, pass: pass }, { withCredentials: true });
  }

  GetDatos() {
    return this.http.post<JSON>(`${baseUrl}/GetDatos`, {}, { withCredentials: true });
  }

  GetEmpresasConDatos(idUsuario: number) {
    return this.http.post<JSON>(`${baseUrl}/GetEmpresasConDatos`, { idUsuario: idUsuario}, { withCredentials: true });
  }

  GetEmpresa(idEmpresa: number, idUsuario: number) {
    return this.http.post<JSON>(`${baseUrl}/GetEmpresa`, { id: idEmpresa, idUsuario: idUsuario }, { withCredentials: true });
  }

  CambiarIdioma(idUsuario: number, idIdioma: number) {
    return this.http.post<JSON>(`${baseUrl}/CambiarIdioma`, { id: idUsuario, idioma: idIdioma }, { withCredentials: true });
  }
```

Figura 58. Archivo finance.service.ts

I.I.IV. Carpeta "_models": Esta carpeta guardará los modelos de los objetos usados en el proyecto. Estos modelos no son estrictamente necesarios, ya que el lenguaje TypeScript es muy permisivo y se admite el tipo de dato "any" el cual engloba todos los posibles tipos de variable, tanto tipos simples (string, int, boolean), tanto tipos más complejos (arrays, arrays de objetos...). Usando estos modelos para los objetos tendremos una estructura más adecuada y nos será más fácil crear objetos, ya que tendremos que crear los objetos con las variables definidas. Con esto nos ahorramos posibles errores.

A continuación, podemos ver un ejemplo del funcionamiento de los modelos. En este ejemplo tenemos un objeto «Datos», el cual contiene una variable llamada «empresas». Esta variable es un array de objetos de tipo «Empresa», definido debajo de «Datos». A la derecha de los modelos podemos ver una manera correcta de instanciar un objeto de tipo «Datos» y una manera errónea.

```
export class Datos {
    empresas: Empresa[];
}

export class Empresa {
    id: number;
    nombre: string;
    ticker: string;
    fechaInicioDatos: Date;
    fechaInicioDatosTmsp: number;
    idMercado: number;
    nombreMercado: string;
    idPais: number;
    nombrePais: string;
    idDivisa: number;
    nombreDivisa: string;
    idSector: number;
    nombreSector: string;
    idIndustria: number;
    nombreIndustria: string;
}
```

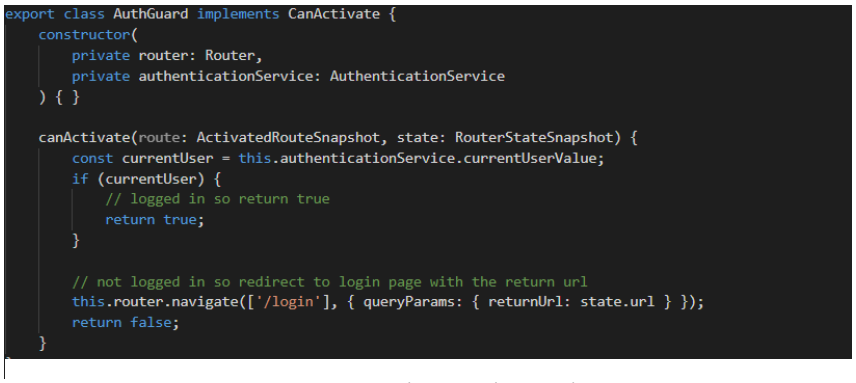
```
var pruebaDatos: Datos = {
    empresas: [
        {
            id: 1,
            nombre: "AMAZON",
            ticker: "AMZN",
            fechaInicioDatos: new Date(2000,1,1),
            fechaInicioDatosTmsp: 1000000,
            idMercado: 1,
            nombreMercado: "NASDAQ",
            idPais: 1,
            nombrePais: "EEUU",
            idDivisa: 1,
            nombreDivisa: "USD",
            idSector: 1,
            nombreSector: "Consumer Cyclical",
            idIndustria: 1,
            nombreIndustria: "Internet Retail",
        },
        //...
    ]
}
```

```
var pruebaDatos: Datos = {
    empresas: [
        {
            id: 1,
            nombre: "AMAZON",
            ticker: "AMZN",
            fechaInicioDatos: new Date(2000,1,1),
            fechaInicioDatosTmsp: 1000000,
            idMercado: 1,
            nombreMercado: "NASDAQ",
            idPais: 1,
            nombrePais: "EEUU",
            idDivisa: 1,
            nombreDivisa: "USD"
        },
        //...
    ]
}
```

Figura 59. Ejemplo del funcionamiento de los modelos

I.I.V. Carpeta "_helpers": En esta carpeta se guardarán elementos que se usen recurrentemente en el proyecto. En nuestro caso, cabe mencionar dos archivos:

- El archivo «funcionesGlobales.ts», que sirve para guardar funciones a las que se llama desde más de un fichero. De esta manera se incluyen en este archivo y se evita la repetición de funciones en el proyecto.
- El archivo «auth.guard.ts», que ya antes hemos mencionado. Hay veces que queremos que determinadas áreas de nuestra aplicación web estén protegidas y solo puedan ser accedidas si el usuario ésta logueado o incluso que solo puedan ser accedidas por determinados tipos de usuarios. Para esto se usan los guards. Los guards pueden ser extensibles para que permitan acceder bajo las condiciones que nosotros queramos.



```
export class AuthGuard implements CanActivate {
    constructor(
        private router: Router,
        private authenticationService: AuthenticationService
    ) { }

    canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
        const currentUser = this.authenticationService.currentUserValue;
        if (currentUser) {
            // logged in so return true
            return true;
        }

        // not logged in so redirect to login page with the return url
        this.router.navigate(['/login'], { queryParams: { returnUrl: state.url } });
        return false;
    }
```

Figura 60. Archivo auth.guard.ts

I.II. Carpeta “assets”: Esta carpeta contiene los recursos usados en la página WEB (imágenes, scripts, estilos específicos...). También incluye la carpeta i18n, la cual los archivos de los idiomas usados en la WEB (más adelante se explicará el funcionamiento de los idiomas).

I.II. Carpeta “environments”: En esta carpeta se definen los entornos usados en la aplicación. En nuestro proyecto nos encontramos con dos archivos, uno para el entorno de desarrollo, «environment.ts», y otro para producción, «environment.prod.ts». Cada uno de estos archivos tendrá las variables para conectarnos y consumir los datos de la API que hemos desarrollando en paralelo. En el entorno de desarrollo se usará un servidor local y en el entorno de producción se usará la API que tenemos subida en un servidor de internet. A continuación, se muestra el archivo de desarrollo «environment.ts»:

```
// This file can be replaced during build by using the `fileReplacements` array.
// `ng build --prod` replaces `environment.ts` with `environment.prod.ts`.
// The list of file replacements can be found in `angular.json`.

export const environment = {
    production: false,
    apiUrl: 'http://localhost:7205/depcrt'
};
```

Figura 61. Archivo environment.ts

II. Carpeta “node_modules”: Esta carpeta es un directorio que se crea en la carpeta raíz de nuestro proyecto cuando instalamos paquetes o dependencias. De esta forma, desde nuestro código Javascript podemos importar paquetes externos instalados, teniéndolos en nuestro proyecto local (no dependen de una ruta externa al proyecto) y sin necesidad de manipularlos manualmente.

Básicamente, cualquier paquete instalado, se almacena dentro de la carpeta «node_modules» del proyecto, en una carpeta con el nombre del paquete, junto a todos sus ficheros necesarios y dependencias respectivas en su propio «node_modules», y así sucesivamente.

Para terminar con la organización de ficheros de Angular, mencionaremos dos archivos importantes que se crean en la raíz del proyecto:

III. angular.json: Es un espacio de trabajo de Angular que proporciona valores predeterminados de configuración específicos del proyecto y de todo el espacio de trabajo para las herramientas de desarrollo y compilación proporcionadas por Angular. Los valores de ruta dados en la configuración son relativos a la carpeta del espacio de trabajo raíz.

IV. package.json: Es un archivo de definición o manifiesto para nuestro proyecto, en el cual especificamos referencias al proyecto como: autor, repositorio, versión y sobre todo las dependencias, entre otros. Dichas dependencias son las necesarias para que el proyecto trabaje y a la vez permite la descarga de estas a través de un simple comando. Ese comando se puede ejecutar en la consola del proyecto y es «npm install».

6.2.2.2. Implementación de un componente

A continuación, se mostrará el funcionamiento de los componentes en Angular y como se han usado, mostrando un ejemplo de uno de los componentes del proyecto. En este caso se usará como ejemplo el componente que crea la ventana de logueo. Esta ventana está compuesta del formulario de logueo y de varios de decoradores.

Primero, mostraremos el HTML creado, este HTML será la vista dedicada a renderizar aquellos controles con los que el usuario interaccionará.

```
<div class="container c-container">

    <div class="bg_login"></div>
    <div class="bg_login_overlay"></div>

    <div class="col-md-6 offset-md-3 mt-5">
      <!-- LOGO Y MENSAJES DE ENCIMA DEL FORMULARIO -->
      <div class="logo_container">
        <img class="logo_imagen" src="../../assets/logos/logo-nombre-negro.png">
      </div>
      <h1 class="login_headline">{{ 'nosAlegraQueVuelvas' | translate }}
       <br>{{ 'loginAquiPorfavor' | translate }}</h1>
      <p>
        {{ 'primeraVezAqui' | translate }}
        <a class="a_href" (click)="redirectRegister()">{{'registrateAqui'| translate}}</a>
      </p>

      <!-- FORMULARIO DE LOGUEO -->
      <div class="card">
        <h4 class="card-header">{{ 'login' | translate }}</h4>
        <div class="card-body">
          <form [formGroup]="loginForm" (ngSubmit)="onSubmit()">
            <div class="form-group">
              <label for="username">{{ 'usuario' | translate }}</label>
              <input type="text" formControlName="username" class="form-control"
              maxlength="50" [ngClass]="{'isinvalid': submitted && f.username.errors }"/>
              <div *ngIf="submitted && f.username.errors" class="invalid-feedback">
                <div *ngIf="f.username.errors.required">
                  {{ 'usuarioRequerido' | translate }}
                </div>
              </div>
            </div>
            <div class="form-group">
              <label for="password">{{ 'contrasena' | translate }}</label>
              <input type="password" formControlName="password" class="form-control"
                     [ngClass]="{ 'is-invalid': submitted && f.password.errors }" />
              <div *ngIf="submitted && f.password.errors" class="invalid-feedback">
                <div *ngIf="f.password.errors.required">
                  {{ 'contrasenaRequerida' | translate }}
                </div>
              </div>
            </div>
            <p-button type="submit" [disabled]="loading">
              <span *ngIf="loading" class="spinner-border spinner-border-sm mr-1"></span>
              {{ 'login' | translate }}
            </p-button>
            <div *ngIf="error" class="alert alert-danger mt-3 mb-0">{{error}}</div>
          </form>
        </div>
      </div>
    </div>

  </div>
```

En esta siguiente parte, se muestra el fichero TypeScript del componente. Este fichero será el que gestione toda la lógica del componente y lo haga funcionar. En este fichero se crea el formulario que luego utiliza la vista y también se gestiona el logueo del usuario una vez ha introducido las credenciales y ha validado el formulario.

```typescript
//DECLARAMOS LOS PAQUETES QUE SE USAN EN EL COMPONENTE
import { Component, OnInit } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { TranslateService } from '@ngx-translate/core';

import { AuthenticationService, FinanceService } from '@app/_services';
import { MessageService } from 'primeng/api';
import { AppComponent } from '@app/app.component';

//DEFINIMOS A QUE VISTA APUNTA EL COMPONENTE
@Component({ templateUrl: 'login.component.html' })

export class LoginComponent implements OnInit {

  //DECLARAMOS LAS VARIABLES GLOBALES
  loginForm: FormGroup;
  loading = false;
  submitted = false;
  returnUrl: string;
  error = '';

  constructor(
    private formBuilder: FormBuilder,
    private route: ActivatedRoute,
    private router: Router,
    private translate: TranslateService,
    private authenticationService: AuthenticationService,
    private financeService: FinanceService,
    private messageService: MessageService,
    private appComponent: AppComponent
  ) {
    //CAMBIAR EL TITULO DE LA PESTAÑA EN EL NAVEGADOR
    appComponent.cambiarTituloTab();

    //REGIRIGIR AL HOME SI YA SE HA LOGUEADO PREVIAMENTE
    if (this.authenticationService.currentUserValue) {
      this.router.navigate(['/']);
    }
  }

  ngOnInit() {
    //CREAR EL FORMULARIO
    this.loginForm = this.formBuilder.group({
      username: ['', Validators.required],
      password: ['', Validators.required]
    });

    // OBRENER LA URL DE RETORNO DE LOS PARAMETROS DE RUTA
    this.returnUrl = this.route.snapshot.queryParams['returnUrl'] || '/';
  }

  // GET PARA OBTENER LA INFORMACION DEL FORMULARIO DESDE LA VISTA
  get f() { return this.loginForm.controls; }

  // FUNCION PARA VALIDAR EL FORMULARIO
  async onSubmit() {
    this.submitted = true;
```

```
    //SI LOS DATOS DEL FORMULARIO SON INVALIDOS NO SEGUIR
    if (this.loginForm.invalid) {
      return;
    }

    //PONEMOS LA VISTA EN ESTADO DE CARGA
    this.loading = true;

    //OBTENEMOS EL USUARIO DE LA BASE DE DATOS (LLAMAMOS A LA FUNCION GetUsuario
    //DEL SERVICIO financeService, QUE DESPUES OBTENDRA EL USUARIO DE LA BD)
    this.financeService.GetUsuario(this.f.username.value,this.f.password.value).subscribe(
      (user: any) => {
        if(user.length > 0){
          //SI SE RECIBE UN USUARIO (user.length > 0) CARGAR EL SISTEMA
          var user = user[0];

          //AÑADIMOS AL USUARIO EN UN ARRAY LOS IDS DE SUS CARTERAS
          if (user.idCarteras.length === 0) user.idCarteras = [];
          else user.idCarteras = user.idCarteras.replace(/,+/g,",").split(",").map(Number)

          //CARGAMOS EN EL SISTEMA EL IDIOMA QUE EL USUARIO TIENE SELECCIONADO
          if(user.idioma == 1) user.idioma = "esp";
          if(user.idioma == 2) user.idioma = "eng";
          if(user.idioma == 3) user.idioma = "eus";

          //GUARDAMOS EL USUARIO EN EL SISTEMA DE AUTENTICACION Y REDIRIGIMOS AL LOGIN
          user.authdata = window.btoa(this.f.username.value + ':' + this.f.password.value)
          this.authenticationService.login(user);
          this.router.navigate([this.returnUrl]);
        } else {
          //SI NO SE RECIBE UN USUARIO (user.length == 0) MOSTRAMOS MENSAJE DE ERROR
          //DE CREDENCIALES INCORRECTAS
          this.error = this.translate.instant("usuarioOPassIncorrectas");
          this.loading = false;
        }
      },
      err => {
        //TRATAMOS SI LA API DEVUELVE UN ERROR
        this.messageService.add({severity:'error', summary:
          this.translate.instant('error'), detail: this.translate.instant('errorLogin')});
        this.loading = false;
      });
  }

  //FUNCION PARA REDIRIGIR A LA PAGINA DE REGISTRO
  redirectRegister(){
    this.router.navigate(['/register']);
  }

}
```

Para terminar con toda la lógica del componente, esta es la función del servicio a la que se llama, para que esta obtenga los datos de la API.

```
GetUsuario(usuario: string, pass: string) {
    return this.http.post<JSON>(`${baseUrl}/GetUsuario`, {
      usuario: usuario, pass: pass
    }, { withCredentials: true });
  }
```

6.2.2.3. Uso de lenguajes (ngx-translate)

Cuando se empezó a desarrollar la aplicación y a decidir las funciones que tendría, se decidió crear una aplicación multilenguaje (añadirle Internacionalización[6]) con tres idiomas diferentes: castellano, inglés y euskera.

Para poder llevar a cabo la internacionalización de la aplicación, se hará uso del paquete ngx-translate. Este paquete o librería nos permite la carga dinámica de archivos de traducciones en nuestras aplicaciones Angular, cuya capacidad de carga dinámica de archivos nos permite cambiar el idioma de la aplicación sin necesidad de recargar la pantalla. Trabajaremos con ficheros de extensión JSON.

Para descargar el módulo de traducciones y el cargador de traducciones de ngx-translate, desde nuestro terminal ejecutamos el siguiente comando añadiendo las dependencias de producción en nuestro package.json.

```
npm install @ngx-translate/core @ngx-translate/http-loader
```

Como nuestra aplicación va a tener tres idiomas, vamos a crear un fichero por cada idioma para añadir las traducciones en castellano, inglés y euskera. Para ello, nos dirigimos a la carpeta "assets" de nuestro proyecto Angular en "src/assets", creamos un directorio llamado "i18n" y dentro de este creamos tres ficheros JSON con los nombres esp.json, eng.json y eus.json. Estos ficheros contendrán las traducciones de cada uno de los idiomas a los que referencian. A continuación, podemos ver la estructura de uno de los archivos:

```
{
    "materialesBasicos": "Materiales Básicos",
    "productosQuimicos": "Productos Químicos",
    "serviciosDeComunicacion": "Servicios de Comunicación",
    "consumoCiclico": "Consumo Cíclico",
    "consumoDefensivo": "Consumo Defensivo",
    "energia": "Energía",
    "serviciosFinancieros": "Servicios Financieros",
    "cuidadoDeSalud": "Cuidado de la Salud",
    "industriales": "Industriales",
    "bienesRaices": "Bienes Raíces",
    "tecnologia": "Tecnología",
    ...
}
```

Como se puede observar, estos ficheros están formados por una lista de pares, las cuales la componen la palabra clave a la cual se le llamará desde cualquier componente para obtener la traducción, y la traducción en cuestión. Cada fichero contiene todas las traducciones necesarias para que la aplicación este completamente traducida. La aplicación se ha compuesto con unas 400 traducciones por fichero.

[6] Internacionalización: Corresponde al diseño y desarrollo de un producto, una aplicación o el contenido de un documento de modo tal que permita una fácil localización con destino a audiencias de diferentes culturas, regiones o idiomas.

Una vez que ya disponemos de los ficheros con los textos en los dos idiomas, lo que tenemos que hacer es configurar el servicio de traducciones para lo cual importaremos los módulos de traducción dentro del app.module.ts, necesitando importar también el módulo y el cliente http, para el correcto funcionamiento del servicio de traducciones que vamos a usar. Crearemos también dentro del archivo app.module.ts una factoría para cargar los archivos de traducciones que lo vamos a llamar "httpTranslateLoader":

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { HttpClient, HttpClientModule,HTTP_INTERCEPTORS} from '@angular/common/http';
import { TranslateLoader, TranslateModule } from '@ngx-translate/core';
import { TranslateHttpLoader } from '@ngx-translate/http-loader';

...

export function httpTranslateLoader(http: HttpClient) {
    return new TranslateHttpLoader(http);
}
```

Esta factoría se encargará de cargar en la aplicación los distintos ficheros de idiomas de la carpeta i18n dependiendo del idioma seleccionado en cada momento, por ejemplo, si el idioma seleccionado es inglés (eng), la factoría cargara el fichero assets/i18n/eng.json.

Para acabar la configuración de la aplicación, añadimos al import del app.module.ts la configuración de carga del servicio de traducciones.

```
@NgModule({
    imports: [
        BrowserModule,
        HttpClientModule,
        TranslateModule.forRoot({
          loader: {
            provide: TranslateLoader,
            useFactory: httpTranslateLoader,
            deps: [HttpClient]
          }
        }),
        ...
    ],
})
```

Ahora que ya tenemos configurados los idiomas en nuestra aplicación, necesitaremos cambiar en nuestra lógica los idiomas cuando sea necesario y también necesitaremos poder llamar a las etiquetas de idiomas. Estas serán las funciones del paquete que más se usarán:

Para conseguir la traducción de una etiqueta en la vista (HTML):

```
{{ 'materialesBasicos' | translate }}
```

Para conseguir la traducción de una etiqueta en la lógica del componente (TypeScript):

```
this.translate.instant('materialesBasicos');
```

Para cambiar el idioma que estamos usando en la lógica del componente (TypeScript):

```
translate.setDefaultLang('esp');
```

6.2.3. ASP.NET Core

ASP.NET Core es un framework de código abierto y multiplataforma para la creación de aplicaciones modernas conectadas a Internet, como aplicaciones web y APIs Web.

Se diseñó para proporcionar un framework de desarrollo optimizado para las aplicaciones que se implementan tanto en la nube como en servidores dedicados en las instalaciones del cliente.

ASP.NET Core es un rediseño completo de ASP.NET, y su arquitectura ha sido diseñada para resultar más ligera y modular. ASP.NET Core se basa en un conjunto de paquetes NuGet[7] granulares y bien factorizados. Esto te permite optimizar tu aplicación para incluir solo los paquetes NuGet que necesitas. Estas son algunas de las ventajas de ASP.NET Core:

- Seguridad estricta: tiene una tasa baja de intercambio de información y rendimiento mejorado, ya que está formado por paquetes NuGet, lo que permite un modularidad total, de esta forma solo añadiremos los paquetes con la funcionalidad que necesitemos.
- Una plataforma unificada para la creación de interfaz web y las APIs web.
- Un sistema de configuración basado en la nube. Preparado para su integración de forma sencilla en entornos en la nube.
- Las peticiones HTTP se procesan siguiendo un flujo que puede ser modificado de forma modular para adaptarse a nuestras necesidades y que nos permite poder controlar el procesado de las peticiones HTTP en nuestra aplicación.

Teniendo en cuenta todas estas ventajas, se ha elegido esta tecnología para desarrollar la API de nuestro proyecto. La API creada para el proyecto consta básicamente de tres apartados:

- Modelos: Definen la estructura que tendrán los objetos usados en la API. Estos modelos se usarán para definir que variables se le pasarán a los diferentes métodos de la API
- Controladores: Los controladores son los responsables de responder a las solicitudes realizadas en un sitio web de ASP.NET Core. Cada solicitud del explorador se asigna a un controlador determinado. Por ejemplo, Imagine que escribe la siguiente dirección URL en la barra de direcciones del explorador: «http://localhost:7205/depcrt/GetDatos». En este caso, se invoca un controlador denominado DepcrtController, y este es responsable de generar la respuesta a la solicitud del explorador. Por ejemplo, el controlador podría devolver una tabla solicitada de la base de datos o podría llevar a cabo el insert de una línea. Para hacer estas operaciones, primero tendrá que conectar con los servicios. Todos los métodos a los que se pueden acceder de los controladores de la API hay que invocarlos con un POST, esto aumenta la seguridad de las llamadas.
- Servicios: Son el último paso de la API para llevar a cabo las operaciones. Los controladores reciben las llamadas de la WEB y los datos necesarios para llevar a cabo esa llamada. Los controladores después conectan con los servicios, y estos últimos son los que directamente se comunican con la base de datos. Cuando los servicios le devuelven la información a los controladores, estos serán los que lleven a cabo la lógica de la operación (si es necesario).

[7] NuGet: Es una extensión de Visual Studio que hace más fácil agregar, eliminar y actualizar referencias a librerías y herramientas en proyectos de Visual Studio

A continuación, se muestra un proceso de cambio de datos en la base de datos. Aquí veremos el proceso del mismo desde la WEB (angular) hasta la base de datos, pasando por la API:

Primero, se le llama a la API desde la aplicación. En este caso, el nombre del método al que llamamos se llama «GuardarDatosFiscalidad» y le pasamos los datos que se ven a continuación (idUsuario, nombre, apellido…).

```
GuardarDatosFiscalidad(
      idUsuario: number, nombre: string, apellido: string, nif: string, viaPublica:
      string, numero: string, municipio: string, provincia: string,
      codigoPostal: string
    ) {
    return this.http.post<JSON>(`${baseUrl}/GuardarDatosFiscalidad`, {
      idUsuario: idUsuario, nombre: nombre, apellido: apellido, nif: nif,
      viaPublica: viaPublica, numero: numero, municipio: municipio,
      provincia: provincia, codigoPostal: codigoPostal
    }, { withCredentials: true });
  }
```

Después, recibimos la llamada en el controlador de la API. Como vemos, el método del controlador recibe una sola variable llamada model. Este modelo lo tenemos que definir nosotros y los datos tienen que tener la misma estructura que se le pasa desde la WEB. Una vez el controlador recibe la llamada con el modelo, se le llama al servicio pasándole ese mismo modelo.

```
[HttpPost("GuardarDatosFiscalidad")]
public int GuardarDatosFiscalidad(GuardarDatosFiscalidadRequest model)
{
   int result = _depcrtService.GuardarDatosFiscalidad(model);
   return result;
}
```

```
public class GuardarDatosFiscalidadRequest
{
    public int IdUsuario { get; set; }
    public string Nombre { get; set; }
    public string Apellido { get; set; }
    public string Nif { get; set; }
    public string ViaPublica { get; set; }
    public string Numero { get; set; }
    public string Municipio { get; set; }
    public string Provincia { get; set; }
    public string CodigoPostal { get; set; }
}
```

Finalmente, el servicio ejecuta la operación que tenga que ejecutar contra la base de datos y se le devuelve la respuesta al controlador. El controlador le devolverá la respuesta a la WEB para que siga el proceso que esté ejecutando.

```
public int GuardarDatosFiscalidad(GuardarDatosFiscalidadRequest model)
{
    try
    {
        string constring = ConnectionStrings.DefaultConnection(0);
        var dbacc = new BaseDataAccess(0);

        StringBuilder s = new StringBuilder();

        int response = -1;
```

```
        s.AppendLine("UPDATE Usuarios");
        s.AppendLine("SET");
        s.AppendLine("      nombre = '" + model.Nombre + "'");
        s.AppendLine("      ,apellido = '" + model.Apellido + "'");
        s.AppendLine("      ,nif = '" + model.Nif + "'");
        s.AppendLine("      ,viaPublica = '" + model.ViaPublica + "'");
        s.AppendLine("      ,numero = '" + model.Numero + "'");
        s.AppendLine("      ,municipio = '" + model.Municipio + "'");
        s.AppendLine("      ,provincia = '" + model.Provincia + "'");
        s.AppendLine("      ,codigoPostal = '" + model.CodigoPostal + "'");
        s.AppendLine("WHERE id = " + model.IdUsuario);

        response = dbacc.Update(s.ToString());

        return response;
    }
    catch
    {
        return -1;
    }
}
```

6.2.4. SQL y SQL Server

Para crear y gestionar la base de datos que nutre de datos a nuestra aplicación se ha elegido el lenguaje SQL. Este lenguaje SQL sirve para el acceso a la información almacenada en las bases de datos. Es un lenguaje sencillo de consulta, que permite realizar operaciones de selección, inserción, actualización y borrado de datos, así como operaciones administrativas sobre las bases de datos.

Dada su fuerte conexión con la teoría del modelo relacional, SQL es un lenguaje de alto nivel orientado a conjuntos de registros. Esto implica que un solo comando SQL puede equivaler a decenas o cientos de líneas de código que se tendrían que utilizar en un lenguaje de más bajo nivel orientado a registros. Como resultado de lo anterior, SQL permite lograr con mayor rapidez y facilidad la definición y manipulación de los objetos de base de datos, permitiendo así alcanzar una mayor eficiencia y productividad en el desarrollo.

Una base de datos SQL se compone de trabas. Estas tablas son el elemento más importante dentro de una base de datos ya que se utilizan para guardar los datos persistente o permanentemente. Una tabla se asemeja a una hoja de cálculo. Al igual que éstas, las tablas se componen de filas y columnas. Cada una de las columnas se denominan campos de la tabla y a cada una de las filas se les conoce como registros. Cada campo puede contener tipos de datos como: texto, números, gráficos, etc.

Ejemplo de una consulta a la tabla Usuarios en la base de datos:

```
SELECT u.id, u.usuario, u.nombre, u.apellido, u.idioma, u.divisa, u.admin,
       ISNULL(STRING_AGG(c.id, ','),'') AS idCarteras
FROM Usuarios AS u LEFT OUTER JOIN Carteras AS c ON u.id=c.idUsuario
WHERE usuario='xabiazabal'
GROUP BY u.id, u.usuario, u.nombre, u.apellido, u.idioma, u.divisa, u.admin
```

id	usuario	nombre	apellido	idioma	divisa	admin	idCarteras
1	xabiazabal	Xabi	Azabal	1	2	1	2,5,10,14

Figura 62. Resultado de una consulta a la base de datos

Para gestionar la base de datos y hacer y testear todas las operaciones en un entorno grafico de administración, se ha usado el servidor Microsoft SQL Server. Este servidor es la alternativa de Microsoft a otros potentes sistemas gestores de bases de datos. Es un sistema de gestión de base de datos relacional desarrollado como un servidor que da servicio a otras aplicaciones de software que pueden funcionar ya sea en el mismo ordenador o en otro ordenador a través de una red (incluyendo Internet).

Los servidores SQL Server suelen presentar como principal característica una alta disponibilidad al permitir un gran tiempo de actividad y una conmutación más rápida. Todo esto sin sacrificar los recursos de memoria del sistema. Gracias a las funciones de memoria integradas directamente en los motores de base de datos SQL Server y de análisis, mejora la flexibilidad y se facilita el uso. Pero quizá su característica más destacada es que ofrece una solución robusta que se integra a la perfección con la familia de servidores Microsoft Server.

Cuenta con las siguientes características:

- Soporte de transacciones.
- Escalabilidad, estabilidad y seguridad.
- Soporte de procedimientos almacenados.
- Incluye también un potente entorno gráfico de administración, que permite el uso de comandos DDL y DML gráficamente.
- Permite trabajar en modo cliente-servidor, donde la información y datos se alojan en el servidor y las terminales o clientes de la red solo acceden a la información.
- Permite administrar información de otros servidores de datos.

6.2.5. NPM

En Angular, a medida que vamos trabajando iremos necesitando agregar más paquetes para poder completar ciertos objetivos. En este punto es donde justamente entra NPM (Node Package Manager).

De sus siglas NPM (Node Package Manager) es un gestor de paquetes desarrollado en su totalidad bajo el lenguaje JavaScript, a través del cual podemos obtener cualquier librería con tan solo una sencilla línea de código, lo cual nos permitirá agregar dependencias de forma simple, distribuir paquetes y administrar eficazmente tanto los módulos como el proyecto a desarrollar en general.

Así mismo, en las actuales versiones de Angular, NPM permite a los desarrolladores instalar aplicaciones que se encuentren en el repositorio. En este mismo orden de ideas, cabe destacar que al instalar paquetes nuevos estos serán almacenados localmente en la carpeta que lleva por nombre “node_modules” (anteriormente mencionada) dentro de nuestro proyecto. Sin embargo, el desarrollador puede indicarle a NPM que instale dicho paquete de forma global, según lo considere necesario. A continuación, se muestra varios paquetes instalados desde la consola (de los que hablaremos más adelante):

```
npm install primeng –save
npm install @amcharts/amcharts4 –save
npm install jspdf jspdf-autotable –save
```

6.2.6. Yahoo Finance

Yahoo Finance es un servicio de la empresa Yahoo que proporciona información financiera y comentarios con un enfoque en los mercados globales, haciendo más hincapié en los de los Estados Unidos. Yahoo Finance ofrece información financiera, incluyendo cotizaciones de bolsa, índices bursátiles, comunicados de prensa corporativos y financieros, y foros de discusión para discutir las perspectivas de empresas y la valoración de las mismas. También ofrece algunas herramientas para el manejo organizado de finanzas personales y gestión de carteras. A nivel de cantidad de datos, Yahoo Finance es de las plataformas de finanzas que más tienen.

Así mismo, Yahoo Finance también dispone de una API gratuita. Yahoo habilita esta API a la que se le pueden hacer consultas para poder obtener los mismos datos que muestran en su plataforma. La API de Yahoo Finance nos ayuda a consultar toda la información sobre resúmenes financieros, acciones, cotizaciones… la cual necesitamos en nuestra plataforma.

Yahoo pone a nuestra disposición esta API a través de enlaces públicos, de manera que podemos acceder a todos los datos en formato JSON desde cualquier navegador. En nuestro caso, para obtener la información de las finanzas, haremos llamadas GET desde nuestra API y la convertiremos en objetos para luego poder tratarlos. A continuación, se muestra como se obtiene la relación entre el dólar y el euro (USD/EUR) llamando a la API de Yahoo desde nuestra API.

```
public double GetCambioUsdEur() {
   try {
      string sURL;
      sURL = "https://query1.finance.yahoo.com/v7/finance/quote?symbols=USDEUR=x";
      WebRequest wrGETURL;
      wrGETURL = WebRequest.Create(sURL);

      Stream objStream;
      objStream = wrGETURL.GetResponse().GetResponseStream();

      StreamReader objReader = new StreamReader(objStream);

      string sLine = objReader.ReadLine();

      GetUsdEurResponse obj = JsonConvert.DeserializeObject<GetUsdEurResponse>(sLine);

      return obj.quoteResponse.result[0].regularMarketPrice;

   } catch {
      return -1;
   }
}
```

Los enlaces de la API de Yahoo que más usaremos son los siguientes:

1. Para obtener la relación entre el dólar y el euro: https://query1.finance.yahoo.com/v7/finance/quote?symbols=USDEUR=x
2. Para obtener los precios de cotización de una empresa en el rango que queramos: https://query1.finance.yahoo.com/v8/finance/chart/GOOG?period1=0&period2=9999999999&interval=1d
3. Para obtener información general de una lista de empresas (GOOG, AAPL, AMZN y V): https://query1.finance.yahoo.com/v7/finance/quote?symbols=GOOG,AAPL,AMZN,V

Este sería el resultado que obtendríamos de una llamada a la API de Yahoo (en este caso para obtener la información de la acción de la empresa Google):


```
{
  quoteResponse: {
    result: [
      {
        language: "en-US",
        region: "US",
        quoteType: "EQUITY",
        quoteSourceName: "Delayed Quote",
        triggerable: true,
        currency: "USD",
        regularMarketVolume: 955524,
        bid: 2893.2,
        shortName: "Alphabet Inc.",
        marketState: "CLOSED",
        exchange: "NMS",
        longName: "Alphabet Inc.",
        messageBoardId: "finmb_29096",
        exchangeTimezoneName: "America/New_York",
        exchangeTimezoneShortName: "EDT",
        esgPopulated: false,
        firstTradeDateMilliseconds: 1092922200000,
        postMarketChangePercent: -0.0017285038,
        postMarketTime: 1630713576,
        postMarketPrice: 2895.45,
        postMarketChange: -0.050048828,
        regularMarketChange: 11.120117,
        regularMarketChangePercent: 0.38552886,
        regularMarketTime: 1630699202,
        regularMarketPrice: 2895.5,
        regularMarketDayHigh: 2907.54,
        regularMarketDayRange: "2870.1 - 2907.54",
        regularMarketDayLow: 2870.1,
        regularMarketPreviousClose: 2884.38,
        ask: 2895.44,
        bidSize: 9,
        fullExchangeName: "NasdaqGS",
        financialCurrency: "USD",
        regularMarketOpen: 2882.92,
        averageDailyVolume3Month: 1061406,
        averageDailyVolume10Day: 954862,
        fiftyTwoWeekLowChange: 1488.95,
        fiftyTwoWeekLowChangePercent: 1.058583,
        fiftyTwoWeekRange: "1406.55 - 2936.41",
        fiftyTwoWeekHighChange: -40.909912,
        fiftyTwoWeekHighChangePercent: -0.013931949,
        fiftyTwoWeekLow: 1406.55,
        fiftyTwoWeekHigh: 2936.41,
        trailingPE: 31.408985,
        bookValue: 355.83,
        fiftyDayAverage: 2766.8965,
        twoHundredDayAverageChangePercent: 0.1941305,
        marketCap: 1923411083264,
        forwardPE: 27.32377,
        priceToBook: 8.137313,
        displayName: "Alphabet",
        symbol: "GOOG"
      }
    ],
    error: null
  }
}
```

6.2.7. Bootstrap

Bootstrap, es un framework que permite crear interfaces web con CSS y JavaScript, cuya particularidad es la de adaptar la interfaz del sitio web al tamaño del dispositivo en que se visualice. Es decir, el sitio web se adapta automáticamente al tamaño de una PC, una Tablet u otro dispositivo. Esta técnica de diseño y desarrollo se conoce como "responsive design" o diseño adaptativo.

Cuando surgió la idea de hacer esta aplicación, una de las cosas que se quería conseguir era hacer una plataforma accesible desde cualquier dispositivo. Con esta tecnología podemos conseguir esto porque nos ayuda a definir cada parte de nuestra aplicación cuanto espacio va a ocupar en función del tamaño de la pantalla del dispositivo (organizándolo en filas y columnas). Toda la aplicación está hecha usando esta tecnología y haciendo una interfaz adaptativa.

A continuación, se muestra el comportamiento de algunas ventanas de la aplicación en diferentes dispositivos con diferentes resoluciones:

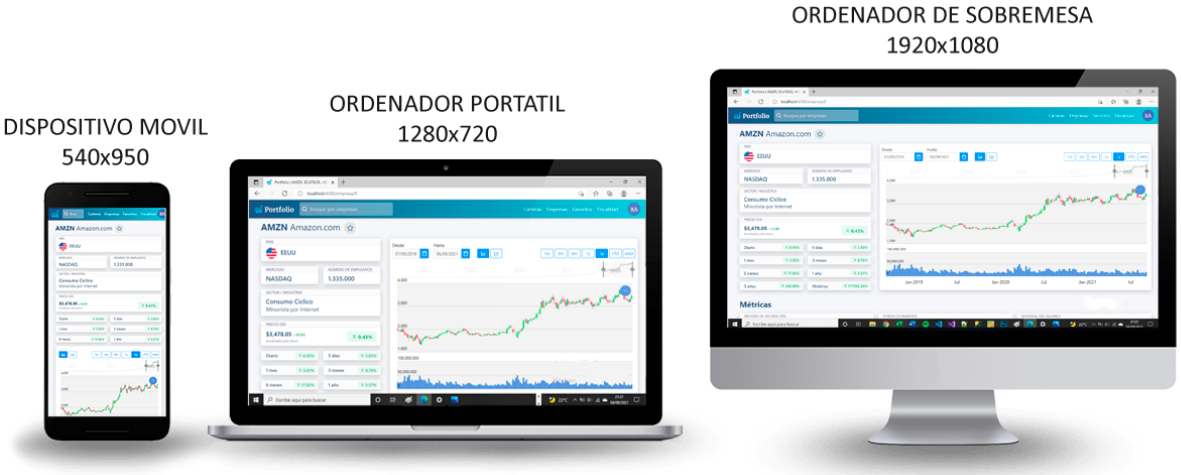


Figura 63. Comportamiento de la pantalla empresa en diferentes dispositivos

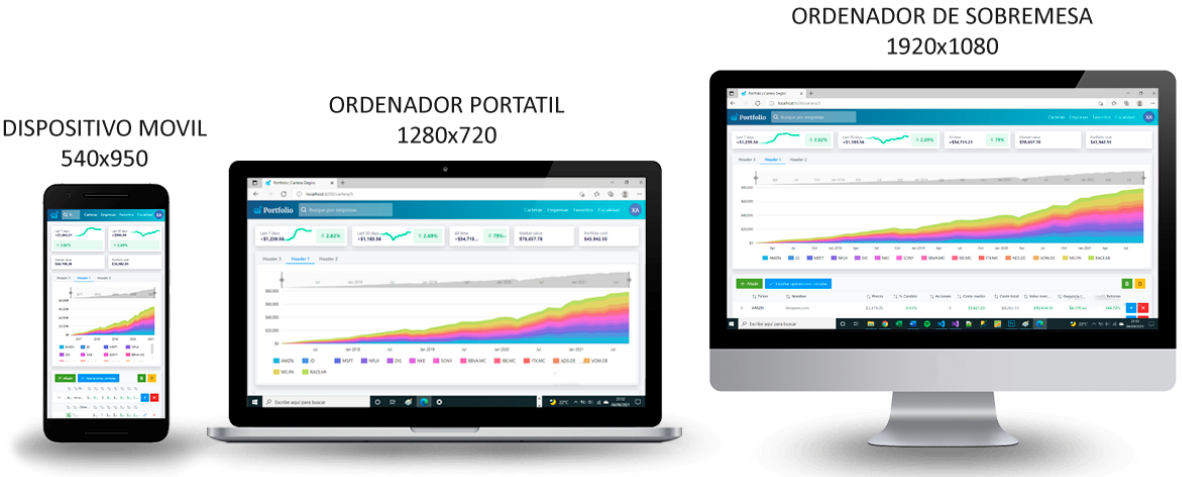


Figura 64. Comportamiento de la pantalla cartera en diferentes dispositivos

6.2.8. PrimeNG

PrimeNG es un amplio conjunto de componentes de interfaz de usuario angulares nativos de código abierto. Eso significa que para cualquier componente de interfaz de usuario que necesitemos, PrimeNG tiene una biblioteca lista para él. Esto incluye: calendarios, tablas, formularios, pop-up-s, etc.

Además, PrimeNG es un paquete muy conocido y con mucho desarrollo por detrás, esto hace que aceleremos el adelanto de nuestra aplicación y mantiene a todos los usuarios de este paquete usando los mismos componentes en todo el mundo. Esto hace que el manejo de código nuevo sea más rápido ya que todos los desarrolladores están familiarizados con PrimeNG.

6.2.9. amCharts

amCharts es una biblioteca de referencia para la visualización de datos. amCharts 4 es una biblioteca JavaScript lista para usar que le permite agregar gráficos interactivos a una amplia gama de aplicaciones. Con soporte integrado para los módulos TypeScript y ES6, es totalmente compatible con Angular.

Permite hacer gráficos de columnas, barras, líneas, áreas, paso, paso sin bandas, línea suavizada, candelabro, tarta o dónut, radar o polar, XY – dispersión o XY – burbuja, embudos, o medidores. Tiene herramientas específicas para datos financieros y visualizaciones de mapas. Y es una librería Javascript que puede descargarse gratuitamente. Nosotros haremos uso de este paquete para la creación de los gráficos que usaremos para analizar las carteras y las empresas.

6.2.10. jsPDF

jsPDF es una librería para Java Script que permite generar documentos PDF a partir de una plantilla HTML o directamente por programación. Esto nos permite poder diseñar un documento PDF de manera sencilla y adaptable. Una ventaja importante que se obtiene al utilizar esta librería es la de no utilizar recursos en el servidor ya que sólo se ejecuta del lado del cliente (navegador). Usaremos este paquete para crear los informes de fiscalidad o los informes de cartera.

7. Pruebas

Las pruebas realizadas en una aplicación es un punto de vital importancia en la fase de lanzamiento o su salida a producción. Este es un momento clave pues si no se han realizado las pruebas adecuadas se pueden encontrar vulnerabilidades de seguridad, algo que en la etapa de producción de una aplicación puede ser tortuoso de arreglar y puede ser aprovechado por agentes externos.

Este capítulo reúne la información de las pruebas que se han realizado a lo largo del desarrollo de la aplicación pasando por las pruebas de funcionalidad y finalmente las pruebas hechas con usuarios reales.

7.1. Pruebas de funcionalidad

Un proyecto de Angular tiene la posibilidad de ejecutarse en fase de desarrollo en modo debug[8], esto sirve para comprobar las trazas de errores detectados en el código desde la parte del cliente para depurarlos. Todo el desarrollo local se ha hecho en modo debug para detectar y solucionar los máximos posibles dentro de la lógica de la aplicación. También se han tratado los datos de la base de datos una gran cantidad de veces para poder detectar errores de diseño.

Tras una evaluación final en el estado actual de la aplicación, se muestran a continuación algunas pruebas con los resultados obtenidos.

Pruebas de usuario sin registrar

- Prueba 1: El usuario al registrarse utiliza un nombre de usuario que ya está en uso.
  - Entrada: Nombre de usuario ya en uso.
  - Salida esperada: El sistema avisa de que no se ha hecho el registro porque el nombre de usuario introducido ya está en uso.
  - Salida obtenida: Igual a la salida esperada.

- Prueba 2: El usuario deja la plantilla registro los campos nombre y apellido sin rellenar.
  - Entrada: Campos de nombre y apellido sin rellenar.
  - Salida esperada: Componentes con borde rojo avisando de que los campos sin escribir son obligatorios.
  - Salida obtenida: Igual a la salida esperada.

- Prueba 3: Los campos de contraseña y repetir contraseña no coinciden.
  - Entrada: Campos de contraseña y repetir contraseña no son iguales.
  - Salida esperada: Componente de repetir contraseña con borde rojo avisando de que las dos contraseñas deben ser iguales.
  - Salida obtenida: Igual a la salida esperada.

- Prueba 4: El usuario rellena el formulario de registro con todos los campos obligatorios escritos correctamente.
  - Entrada: Campos obligatorios correctamente escritos.
  - Salida esperada: Mensaje de registro exitoso.
  - Salida obtenida: Igual a la salida esperada.

[8] Debug: Se refiere a la depuración de programas es el proceso de identificar y corregir errores de programación. En inglés se conoce como debugging.

- Prueba 5: El usuario rellena el formulario de login con unos datos que no corresponden a ningún usuario existente.
  - Entrada: Credenciales de usuario incorrectas.
  - Salida esperada: Mensaje de que el usuario o la contraseña no son correctos.
  - Salida obtenida: Igual a la salida esperada.

- Prueba 6: El usuario rellena el formulario de login con unos datos correctos.
  - Entrada: Campos obligatorios correctamente escritos.
  - Salida esperada: Redirección a plantilla de inicio.
  - Salida obtenida: Igual a la salida esperada.

Pruebas de usuario registrado

- Prueba 1: El usuario rellena el formulario de crear una cartera nueva con el campo de nombre sin rellenar.
  - Entrada: Campo de nombre sin rellenar.
  - Salida esperada: Componente con borde rojo avisando de que el campo sin escribir es obligatorio.
  - Salida obtenida: Igual a la salida esperada.

- Prueba 2: El usuario rellena el formulario de crear una cartera con los campos obligatorios correctamente.
  - Entrada: Campos obligatorios correctamente escritos.
  - Salida esperada: La cartera se crea correctamente y el usuario recibe un mensaje de confirmación.
  - Salida obtenida: Igual a la salida esperada.

- Prueba 3: El usuario procede a borrar una de sus carteras.
  - Entrada: El usuario intenta borrar una de sus carteras.
  - Salida esperada: La cartera se borra correctamente y el usuario recibe un mensaje de confirmación.
  - Salida obtenida: Igual a la salida esperada.

- Prueba 4: El usuario al crear una transacción deja algún campo obligatorio sin rellenar.
  - Entrada: Campos obligatorios sin rellenar.
  - Salida esperada: Componentes con borde rojo avisando de que los campos sin escribir son obligatorios.
  - Salida obtenida: Igual a la salida esperada.

- Prueba 5: El usuario al crear una transacción todos los campos obligatorios están correctamente introducidos.
  - Entrada: Campos obligatorios correctamente introducidos.
  - Salida esperada: La transacción se crea correctamente y el usuario recibe un mensaje de confirmación.
  - Salida obtenida: Igual a la salida esperada.

- Prueba 6: El usuario al crear una transacción, intenta crear una transacción que no es viable. Por ejemplo: el usuario tiene 5 acciones registradas de una empresa e intenta introducir una transacción de venta de esa empresa de 10 acciones, por lo tanto, no es viable.
  - Entrada: Campos obligatorios correctamente introducidos con una transacción que no es viable.
  - Salida esperada: El sistema avisa al usuario de que no puede crear esa transacción porque no tiene suficientes acciones como para vender.
  - Salida obtenida: Igual a la salida esperada.

- Prueba 7: El usuario procede a borrar una de sus acciones.
  - Entrada: El usuario intenta borrar una de sus acciones.
  - Salida esperada: La acción se borra correctamente y el usuario recibe un mensaje de confirmación.
  - Salida obtenida: Igual a la salida esperada.

- Prueba 8: El usuario procede a introducir una empresa a sus favoritos.
  - Entrada: El usuario intenta introducir una empresa a sus favoritos.
  - Salida esperada: La empresa se añade correctamente a favoritos y el usuario recibe un mensaje de confirmación.
  - Salida obtenida: Igual a la salida esperada.

- Prueba 9: El usuario procede a cambiar el idioma del sistema.
  - Entrada: El usuario intenta cambiar el idioma del sistema.
  - Salida esperada: El idioma se cambia correctamente.
  - Salida obtenida: Igual a la salida esperada.

- Prueba 10: El usuario al guardar los datos de fiscalidad deja algún campo obligatorio sin rellenar.
  - Entrada: Campos obligatorios sin rellenar.
  - Salida esperada: Componentes con borde rojo avisando de que los campos sin escribir son obligatorios.
  - Salida obtenida: Igual a la salida esperada.

- Prueba 11: El usuario al guardar los datos de fiscalidad todos los campos obligatorios están correctamente introducidos.
  - Entrada: Campos obligatorios correctamente introducidos.
  - Salida esperada: Los datos de fiscalidad se guardan correctamente y el usuario recibe un mensaje de confirmación.
  - Salida obtenida: Igual a la salida esperada.

- Prueba 12: El usuario crea un informe de fiscalidad seleccionando una/s cartera/s y un año.
  - Entrada: El usuario intenta crear un informe de fiscalidad.
  - Salida esperada: El informe se crea correctamente y se descarga en el navegador del usuario.
  - Salida obtenida: Igual a la salida esperada.

- Prueba 13: El administrador al crear una empresa todos los campos obligatorios están correctamente introducidos.
  - Entrada: Campos obligatorios correctamente introducidos.
  - Salida esperada: La empresa se crea correctamente y el administrador recibe un mensaje de confirmación.
  - Salida obtenida: Igual a la salida esperada.

7.2. Pruebas con usuarios reales

Por último, hemos realizado pruebas con diferentes usuarios midiendo los tiempos de ejecución de diferentes tareas y preguntando por sus experiencias con la aplicación. Concretamente, se han medido los resultados en tres usuarios familiarizados con el mundo de las inversiones y tres usuarios que no lo están. A continuación, se muestran los diferentes tiempos:

TAREA	TIPO DE USUARIO	USUARIO	TIEMPO	MEDIA	ESTIMADO	DESVÍO
REGISTRARSE	Familiarizado	Usuario nº1	19,7	15,5	20	-22,5%
		Usuario nº2	13,1			
		Usuario nº3	13,8			
	No familiarizado	Usuario nº4	23,7	17,7	20	-11,5%
		Usuario nº5	11,5			
		Usuario nº6	17,9			
LOGIN	Familiarizado	Usuario nº1	8,2	7	5	40%
		Usuario nº2	6,2			
		Usuario nº3	6,7			
	No familiarizado	Usuario nº4	10,6	5,9	5	18%
		Usuario nº5	3,2			
		Usuario nº6	4			
CREAR CARTERA	Familiarizado	Usuario nº1	17,3	16,3	15	8,7%
		Usuario nº2	17			
		Usuario nº3	14,6			
	No familiarizado	Usuario nº4	32,7	24,8	25	-0,8%
		Usuario nº5	20,2			
		Usuario nº6	21,4			
CREAR TRANSACCIÓN	Familiarizado	Usuario nº1	22,5	28,3	30	-5,7%
		Usuario nº2	34,8			
		Usuario nº3	27,6			
	No familiarizado	Usuario nº4	67,4	57,4	50	14,8%
		Usuario nº5	57,6			
		Usuario nº6	47,2			
VER EMPRESAS	Familiarizado	Usuario nº1	38,4	43,4	50	-13,2%
		Usuario nº2	48,9			
		Usuario nº3	43			
	No familiarizado	Usuario nº4	78,3	96,9	100	-3,1%
		Usuario nº5	114,9			
		Usuario nº6	97,6			
CREAR INFORME FISCAL	Familiarizado	Usuario nº1	19,7	21,2	20	6%
		Usuario nº2	27,6			
		Usuario nº3	16,4			
	No familiarizado	Usuario nº4	28,4	30,8	35	-12%
		Usuario nº5	41,2			
		Usuario nº6	22,9			

** Todos los tiempos están indicados en segundos*

Tabla 26. Resultados de las pruebas hechas a usuarios reales

7.3. Conclusiones de las pruebas

En general, los resultados de las pruebas que hemos logrado han resultado satisfactorios. En cuanto a las pruebas de funcionalidad, todo ha ido según lo esperado y no se han detectado fallos o brechas en el sistema.

En cuanto a los resultados de las pruebas hechas a otros usuarios, las pruebas también han sido satisfactorias. En los desvíos logrados en las diferentes tareas podemos observar que los desvíos no son demasiado excesivos y que son una cosa transitoria de la ejecución de la aplicación. También podemos ver como en las tareas referentes a cosas de finanzas, los usuarios que están familiarizados con las mismas han tardado menos en ejecutarlas que los que no están familiarizados. Esto es algo que esperábamos, y es por eso se le pusieron unos estimados más bajos. Además, es algo que no nos preocupa porque esta aplicación está dirigida a gente familiarizada con las finanzas e inversiones.

En cuanto a las pruebas con usuarios reales hemos obtenido varias sugerencias para mejorar la aplicación:

- Al registrar una cuenta enviar un email al correo electrónico para confirmar el registro.
- Integrar diferentes redes sociales para poder entrar a la plataforma con las mismas (Facebook, Gmail, Twitter…).
- Añadir la posibilidad de eliminar más de una transacción al mismo tiempo.

8. Seguimiento y control

Como en todo proyecto de estas dimensiones suele suceder, ocurren ciertos desvíos de la planificación inicial, y en ocasiones desviaciones respecto a las tares definidas. Para ello, en este apartado vamos a analizar ciertos puntos clave que nos dirán el resultado de nuestro proyecto como ha sido y si hemos realizado una correcta planificación.

8.1. Control de planificación

En la siguiente tabla aparece cual es el desglose completo del tiempo invertido en horas de cada tarea planificada, pudiendo ver la diferencia entre lo estimado en un inicio del proyecto y el tiempo real. También podremos ver la desviación obtenida y el porcentaje equivalente; se han marcado como porcentaje aceptable un máximo del ±25%. Se marcan de color amarillo los valores aceptables, en rojo las desviaciones negativas (estimación demasiado positiva) y en verde las desviaciones positivas (trabajo realizado en menos horas de lo esperado).

TAREA	ESTIMACIÓN EN HORAS	TIEMPO REAL EN HORAS	DESVIACION EN HORAS	DESVÍO
P - Planificación	9	10	-1	11,1%
P.1 - Presentación proyecto	3	5	-2	66,7%
P.2 - Planificación	6	5	1	-16,7%
EH - Estudio de herramientas	35	36	-1	2,9%
EH.1 - Estudio Angular	5	8	-3	60%
EH.2 - Estudio ASP.NET Core	2	2	0	0%
EH.3 - Estudio Yahoo Finance	10	7	3	-30%
EH.4 - Estudio amCharts	10	15	-5	50%
EH.5 - Estudio general de conceptos de inversiones	8	4	4	-50%
CC - Control y continuidad	14	15	-1	7,1%
CC.1 - Revisión trabajo	4	6	-2	50%
CC.2 - Comparación trabajo	4	5	-1	25%
CC.3 - Seguimiento con el tutor	6	4	2	-33,3%
AR - Análisis de requisitos	10	15	-5	50%
DW - Desarrollo aplicación WEB	127	143	-16	12,6%
DW.1 - Crear base del proyecto	12	6	6	-50%
DW.2 - Crear interfaz de usuario (componentes)	40	46	-6	15%
DW.3 - Crear lógica de la aplicación (componentes)	60	84	-24	40%
DW.4 - Crear servicios	15	7	8	-53,3%
DA - Desarrollo API	36	30	6	-16,7%
DA.1 - Crear base de la API	2	1	1	-50%
DA.2 - Crear base de datos	4	5	-1	25%
DA.3 - Alimentar base de datos	5	7	-2	40%
DA.4 - Crear controladores	10	6	4	-40%
DA.5 - Crear servicios	10	6	4	-40%
DA.6 - Crear modelos	5	5	0	0%
I - Integración	12	12	0	0%
I.1 - Integración de la aplicación WEB	6	7	-1	16,7%
I.2 - Integración de la API	6	5	1	-16,7%
PR - Pruebas	9	10	-1	11,1%
PR.1 - Pruebas de funcionalidad	4	5	-1	25%
PR.2 - Pruebas con usuarios reales	5	5	0	0%
M - Memoria del proyecto	50	45	5	-10%
Total	**302**	**316**	**-14**	**4,6%**

Tabla 27. Comparativa de tiempo estimado y real de la planificación

8.2. Estimación y desviación de las tareas

P - Planificación: En este apartado ha ido todo según lo esperado en cuanto al tiempo estimado. Hubo un pequeño retraso en el apartado presentación del proyecto por algunos contratiempos en GAUR.

EH - Estudio de herramientas: En este apartado ha ido todo según lo esperado en cuanto al tiempo estimado, ya que ha salido prácticamente igual que el estimado. Hubo un retraso considerable a la hora del estudio de Angular y del paquete amCharts de Angular, por la complejidad de crear gráficos.

CC - Control y continuidad: En este apartado ha ido todo según lo esperado y no ha habido ningún retraso considerable.

AR - Análisis de requisitos: En este apartado se sufrió un retraso considerable porque al final el alcance del proyecto fue un poco más grande de lo esperado. Esto repercutió en que el análisis de requisitos se realizase en más tiempo del estimado.

DW - Desarrollo aplicación WEB: En este apartado se sufrió un pequeño retraso. En cuanto a estimar las horas, este era el paquete más difícil para hacerlo por ser el paquete más amplio del proyecto. El retraso más considerable ha sido en la lógica del proyecto.

DA - Desarrollo API: En este apartado se han recortado reseñablemente los tiempos estimados. Esto ha sido debido a que la lógica de la API se ha hecho en menos de lo que se esperaba (controladores y servicios).

I - Integración: En este apartado prácticamente se han clavado los tiempos estimados. Ha ido todo como se esperaba.

PR - Pruebas: En este apartado prácticamente se han clavado los tiempos estimados. Ha ido todo como se esperaba.

M - Memoria del proyecto: Este apartado se ha completado en menos tiempo del esperado.

En conclusión, para completar el proyecto de las 302 horas que se planificaron, se han empleado 316 horas. Esto supone un desvío de 14 horas que en porcentaje son un 4,6%. Este desvío es bastante aceptable en un proyecto de estas dimensiones.

8.3. Diagrama de gantt

El diagrama de Gantt es una herramienta gráfica cuyo objetivo es exponer el tiempo de dedicación para diferentes tareas o actividades a lo largo de un tiempo total determinado.

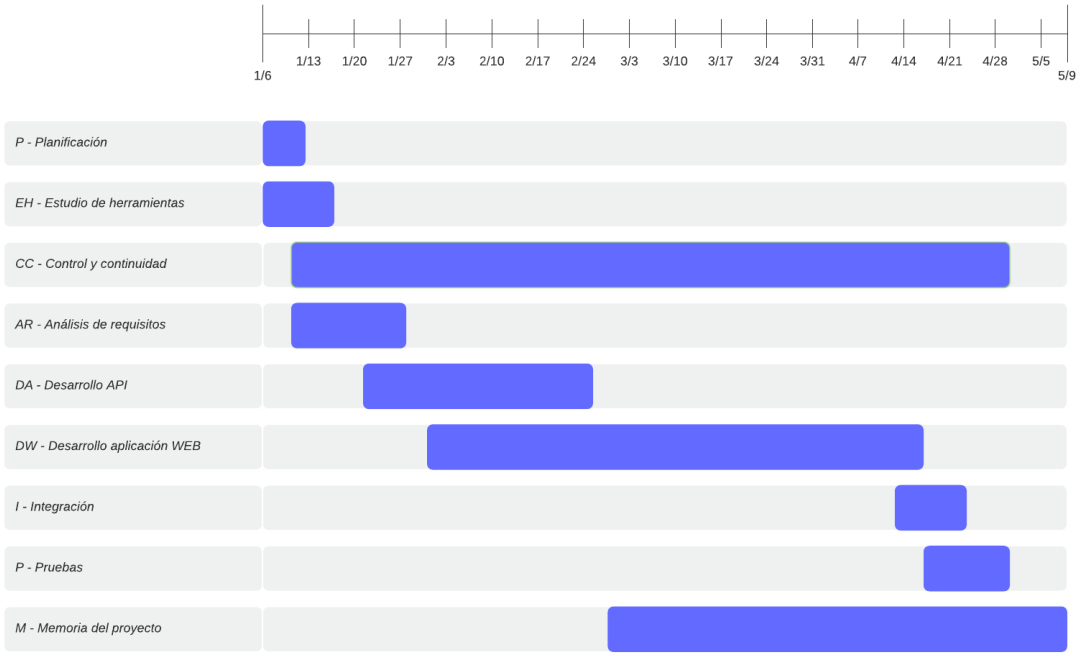


Figura 65. Diagrama de gantt del proyecto

9. Conclusiones

En este capítulo se exponen las conclusiones finales del proyecto repasando la metodología de trabajo, las tecnologías utilizadas, los conocimientos adquiridos y la solución abordada para la aplicación. Por último, se comentará en extenso acerca de las mejoras y línea futura de la aplicación y lo que ha supuesto el grado en la realización de este proyecto.

9.1. Conclusiones

En este proyecto se ha conseguido alcanzar el objetivo básico que se perseguía: Desarrollar una herramienta online capaz de gestionar las finanzas personales de las personas que lo requiriesen de una manera accesible desde cualquier lugar, tanto en un entorno local como en el entorno de desarrollo.

Se ha invertido una gran parte del tiempo en investigar las tecnologías utilizadas en entornos reales. La decisión de utilizar las tecnologías seleccionadas ha exigido una cierta formación del autor en dichas tecnologías ya que, aunque se tenían ciertos conocimientos previos sobre algunas de ellas, siempre surgen inconvenientes y situaciones nunca antes vistas durante la realización del proyecto. Además, se han usado tecnologías nunca antes aplicadas que se han tenido que aprender.

En general, ha sido una experiencia bastante enriquecedora. Al ser un proyecto bastante más prolongado en el tiempo que por ejemplo una práctica de alguna asignatura, con él se han aprendido conceptos de planificación o análisis que son pilares fundamentales de todo el desarrollo software.

Por todo lo anterior, a pesar de que siempre hay cosas que mejorar, en líneas generales el resultado de este desarrollo ha sido muy satisfactorio tanto a nivel personal como profesional.

9.2. Posibles mejoras

Como hemos dicho, en este proyecto se ha conseguido alcanzar el objetivo básico. Sin embargo, este objetivo significaba cubrir ciertos aspectos específicos del mundo de las finanzas, las cuales se consideraban que faltaban en otras aplicaciones conocidas. A parte de esas especificaciones, se podrían incluir muchas mejoras en la aplicación.

Una de las principales mejoras sería la profundidad de los datos de la plataforma. La aplicación está pensada para poder añadir infinidad de empresas (tantas como las que tiene Yahoo Finance), pero esto significaría tener que añadirlas. Es por eso que se podría tener un alcance de aplicación mayor añadiendo mayor cantidad de empresas (actualmente hay registradas 300).

De la misma manera, pasaría lo mismo con los idiomas, a parte del castellano, el inglés y el euskera, se podrían agregar más idiomas para llegar a más gente.

En cuanto a la usabilidad de la aplicación y la capacidad de analizar carteras, siempre se podrían añadir más cantidad de gráficos y métricas con la intención de tener más cantidad de datos.

Bibliografía

[1] Angular

https://angular.io/

[2] Angular (2)

https://www.tutorialesprogramacionya.com/angularya/

[3] ASP.NET Core

https://docs.microsoft.com/es-es/aspnet/core/?view=aspnetcore-5.0

[4] API ASP.NET Core

https://docs.microsoft.com/es-es/aspnet/core/tutorials/first-web-api?view=aspnetcore-5.0&tabs=visual-studio

[5] ngx-translate

https://github.com/ngx-translate/core

[6] ngx-translate (2)

https://medium.com/angular-chile/aplicaciones-multilenguaje-en-angular-7-con-ngx-translate-db8d1e7b380c

[7] SQL

https://support.microsoft.com/es-es/office/access-sql-conceptos-b%C3%A1sicos-vocabulario-y-sintaxis-444d0303-cde1-424e-9a74-e8dc3e460671

[8] amCharts

https://www.amcharts.com/

[9] jsPDF

https://github.com/MrRio/jsPDF

[10] Yahoo Finance

https://es.finance.yahoo.com/

[11] Investing

https://es.investing.com/

[12] Lucidchart

https://www.lucidchart.com/pages/es

[13] TickerTech

http://tickertech.com/