Universidad Rey Juan Carlos

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y ADMINISTRACIÓN Y DIRECCIÓN DE EMPRESAS

CURSO ACADÉMICO 2022-2023

TRABAJO FIN DE GRADO

CRYPTOMYNCE: UN SIMULADOR DE TRADING CON CRIPTOMONEDAS EDUCATIVO

Autor: Sergio Carrascosa Sánchez

Director: Francisco de Asis Gortázar Bellas

Resumen

CryptoMynce es una aplicación de trading de criptomonedas enfocada en el ámbito educativo. En ella, los estudiantes podrán realizar transacciones con una gran variedad de activos y los profesores podrán evaluarlas de una manera sencilla.

Se ha utilizado un gran abanico de tecnologías y herramientas modernas para crear este proyecto, con un enfoque especial en que la forma de trabajar sea lo más profesional posible, utilizando prácticas de desarrollo como la integración y despliegue continuos. El Frontend se ha construido utilizando la librería React, mientras que el Backend se ha desarrollado con el marco de trabajo Spring Boot. Además de estas tecnologías principales, se han empleado otras como Docker, Vite o Github Actions.

Este documento consta de tres capítulos principales. En el primero de ellos se desarrollará la motivación detrás de la creación de este proyecto. El segundo se dividirá en dos epígrafes. El primero tratará en profundidad cuáles son las características imprescindibles, la metodología de desarrollo empleada, la gestión de ramas utilizada y, finalmente, se diseñará una arquitectura del sistema a modo de solución. En el segundo epígrafe del capítulo se comentarán todos los detalles concretos de la implementación de la arquitectura a la que se llegó finalmente en el apartado anterior. Esto abarca también los entornos de ejecución necesarios, los sistemas de seguridad del sistema, las pruebas que se han realizado al software junto con su tipología y herramientas empleadas para ello, y cómo se ha desplegado el sistema en un ambiente de producción. Finalmente, en el tercer y último capítulo se comprobará que los objetivos marcados en los primeros epígrafes han sido cumplidos y se comentarán posibles mejoras para el futuro.

Índice

Capítulo I: Introducción

En determinadas asignaturas del grado de Administración y Dirección de Empresas, y otras titulaciones similares, es habitual encontrar contenidos relacionados con la gestión de activos financieros. En ellos, se enseña a los alumnos a crear carteras de inversión y a medir determinadas métricas relacionadas con la gestión de estas. En una de estas asignaturas se utiliza un simulador web para realizar transacciones basándose en el análisis técnico de estos activos. Sin embargo, este simulador es realmente simple. Tanto que no incluye gráficos ni actualizaciones de precios en tiempo real. Esto provoca que el aprendizaje no sea óptimo.

Actualmente no existe en el mercado ninguna aplicación que mezcle el mundo del trading simulado con funcionalidades para el sector educativo. Debido a ello, las entidades educativas que necesitan utilizar productos de este tipo recurren a diferentes alternativas de simulación que se pueden dividir en dos grupos fundamentales:

El primero son los simuladores sencillos como el que se describió en el primer párrafo. Estos proporcionan funcionalidades suficientes para los usuarios menos exigentes, pero no incluyen una gran selección de opciones que pueden resultar importantes. El ejemplo más famoso que corresponde a esta categoría es La Bolsa Virtual.

El segundo tipo son las pruebas gratuitas y cuentas de demostración de plataformas de compraventa real. Estos simuladores cuentan con todas las funcionalidades de un software de trading y permiten operar con dinero ficticio a sus usuarios. Sin embargo, no son la mejor opción para principiantes debido a su alta complejidad. Este tipo de simuladores son utilizados en cursos de trading por entidades de renombre como la BME o las universidades CEF y IEB.

Sin embargo, como se ha comentado anteriormente, ninguna de estas dos opciones está especializada en la educación de estudiantes, por lo que no incluyen funcionalidades al respecto.

El objetivo fundamental de este proyecto será cubrir ese hueco detectado en el mercado. Para ello, se creará una aplicación que cubra esta necesidad y sea realmente útil para este propósito, ayudando a los alumnos a aprender de una forma óptima, con un software que resulte sencillo y que facilite las labores docentes de evaluación y retroalimentación.

La elección de las criptomonedas en lugar de otros activos más convencionales, como las acciones o los bonos, es debida principalmente a la disponibilidad de la información. Mientras que de las criptomonedas es relativamente sencillo obtener un volumen de datos aceptable, los proveedores de información del resto activos son mucho más restrictivos en sus versiones gratuitas o requieren realizar pagos mensuales.

Capítulo II: Descripción informática

En este capítulo se propondrá un diseño y una implementación de un sistema informático que sirva de solución para el problema detectado en el capítulo anterior.

2.1: Diseño

A lo largo de este epígrafe se abordará la especificación de requisitos, los diferentes diagramas de casos de uso y de flujo identificados, la metodología y gestión de ramas que se seguirán, y la arquitectura del sistema. El objetivo será identificar las partes y funcionalidades a implementar.

2.1.1: Especificación de requisitos

Para comenzar a plantear una solución al problema, primero se deberá plasmar en el documento cuáles son las funcionalidades imprescindibles para que el desarrollo resulte exitoso. Para ello, se ha creado una Especificación de Requisitos de Software que distingue entre características funcionales y no funcionales.

2.1.1.1: Requisitos funcionales

Son aquellos que describen las funcionalidades tangibles del propio sistema. Se centran en qué debe hacer el software (Universidad de Granada, 2010).

- RF1: Se deben poder comprar y vender criptomonedas.
- RF2: Se deben poder evaluar las transacciones realizadas. Para ello, cada una deberá mostrar un gráfico del momento en el que se realizó, el activo en cuestión, la cantidad comprada o vendida, el precio, una justificación del motivo por el cual se ha realizado y la hora de ejecución.
- RF3: El sistema debe permitir la creación de dos tipos de usuarios nominales, profesores y estudiantes, que podrán realizar acciones diferentes según su rol.
    - RF3.1: Solo los estudiantes podrán comprar y vender los diferentes activos. Además, un estudiante podrá ver su portafolio de inversión actual, así como un histórico de todas sus transacciones y la retroalimentación escrita por el profesor.
    - RF3.2: Los profesores podrán revisar las transacciones realizadas por sus alumnos y escribir un comentario en cada una de ellas a modo de retroalimentación y evaluación.
- RF4: Deberá existir un tercer rol, el administrador, que se encargará de dar de alta profesores y estudiantes, así como asignar los alumnos que le corresponden a cada docente.
- RF5: Se deben mostrar gráficos de las diversas criptomonedas con actualización en tiempo real.

- RF6: Se debe contar con una gran variedad de criptomonedas disponibles para comprar y vender por los estudiantes.
- RF7: La aplicación deberá ser desplegada y ser accesible vía web.

2.1.1.2: Requisitos no funcionales

Determinan las normas bajo las cuales deben implementarse las funcionalidades recogidas por los requisitos funcionales (Universidad de Granada, 2010).

- RNF1: La parte visual de la aplicación estará desarrollada con la librería de JavaScript React acompañada de Joy UI, que aportará algunos componentes básicos prediseñados que agilizarán el desarrollo y facilitará la adaptabilidad de la interfaz a las diversas resoluciones.
- RNF2: El Backend de la aplicación estará programado en Java bajo el marco de trabajo Spring Boot.
- RNF3: Las bases de datos serán PostgreSQL.
- RNF4: Deberán existir mecanismos de seguridad que impidan el acceso a información de otros usuarios y la ejecución de acciones sin autorización.
- RF5: La aplicación debe adaptarse principalmente a versión de escritorio y móvil, aunque debe funcionar de manera óptima en todos los tamaños.
    - RF8.1: La versión de escritorio se optimizará para la resolución 1920x1080.
    - RF8.2: La versión móvil tomará como referencia el iPhone 13 Pro Max, que cuenta con una resolución de 428x926.

2.1.2: Diagrama de casos de uso

Una vez se han especificado los requisitos de la aplicación, se deberán identificar los diferentes tipos de usuarios y las acciones que realizará cada uno de ellos.

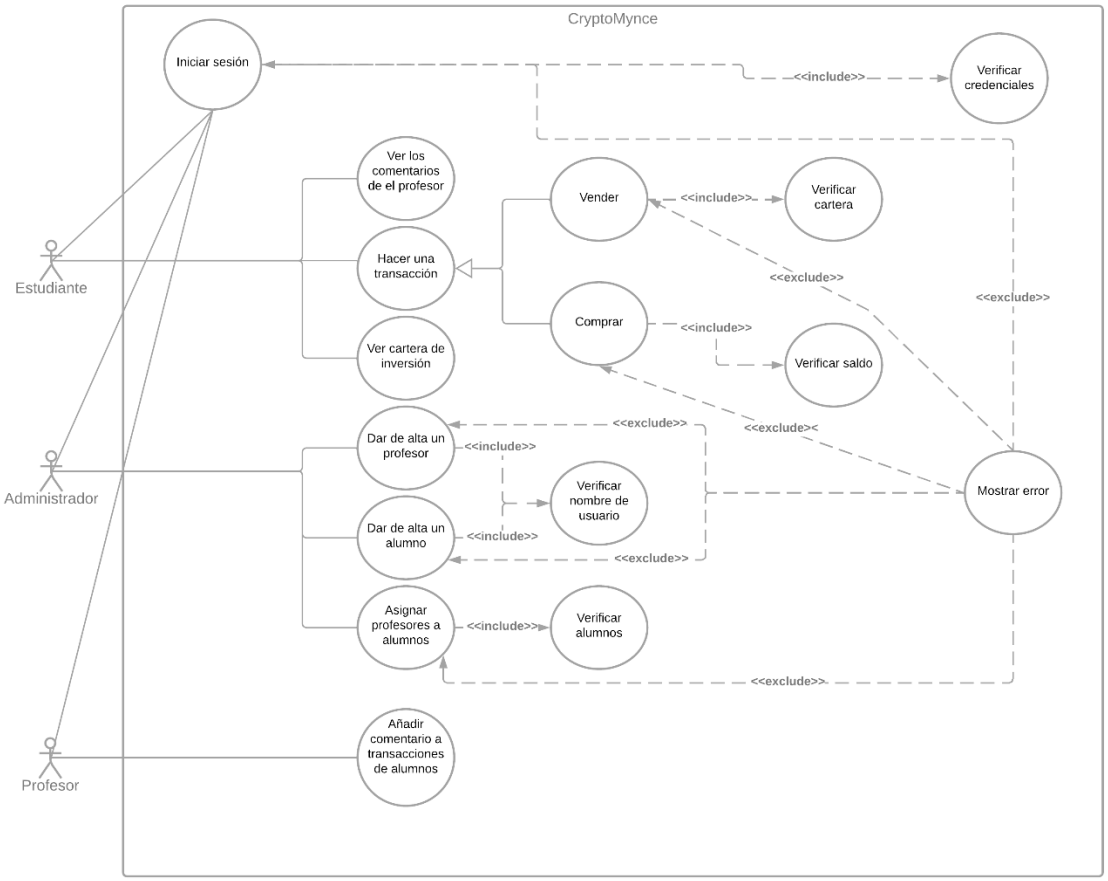


Imagen 1: Diagrama de casos de uso. Fuente: Elaboración propia.

Se han identificado tres actores que interactuarán con el sistema: estudiantes, profesores y administradores. Cada uno podrá realizar una serie de acciones de forma exclusiva, teniendo solo en común el inicio de sesión. Este inicio de sesión supondrá una verificación de las credenciales introducidas y podrá desembocar en mostrar un error si son incorrectas.

El primer actor que encontramos es el Estudiante y tiene tres casos de uso asociados. El primero será revisar la retroalimentación enviada por su profesor en las transacciones que ha realizado en el pasado. El segundo será la realización de una de esas transacciones, que podrá ser de compra o venta. Si se trata de una compra, la aplicación deberá revisar que tiene saldo suficiente en su cuenta antes de completarla y, de no ser así, deberá mostrar un error. La venta es similar, pero se debe comprobar que el estudiante cuenta en su cartera con criptomonedas suficientes para completarla. Por último, el estudiante podrá revisar su cartera de inversión y ver las criptomonedas que la componen.

El administrador tendrá tres tareas. La primera y segunda serán crear las cuentas de estudiantes y profesores, respectivamente. En ambos casos se verificará que los usuarios que se intenta dar de alta no existían previamente en el sistema y, en caso contrario, se mostrará un mensaje de error. La tercera y última tarea será asignar a cada profesor los alumnos correspondientes. Al contrario que en los dos casos de uso anteriores, en este se comprobará que las cuentas existan antes de ser asignadas.

El último actor que se encuentra en el diagrama es el profesor. Este tipo de usuario solo podrá realizar una acción, que consiste en revisar las transacciones realizadas por cada alumno que tenga asociado y enviar una retroalimentación por cada una de ellas.

2.1.3: Diagramas de flujo

Tras identificar los diferentes casos de uso procedentes de los requisitos, se realizarán diagramas de flujo para las acciones que pueden tener una complejidad mayor a la hora de ser implementadas. En este caso, solo se encuentran dificultades en el flujo de la compra y venta de criptomonedas por parte del usuario.

Este proceso comienza iniciando sesión (caso de uso que será omitido en este diagrama) y mostrando el perfil del usuario correspondiente. Una vez en el perfil, el usuario deberá poder acceder a la pestaña de “Mercados”, donde tendrá la posibilidad elegir una de las criptomonedas que se muestran en esta pantalla o buscar una criptomoneda que no aparezca en la lista. En el primer caso, al hacer clic sobre el nombre será redirigido a la pantalla en detalle de la divisa en cuestión. Si por el contrario decide hacer una búsqueda pueden ocurrir dos escenarios posibles. En caso de que el estudiante intentara buscar una criptomoneda que no esté registrada, se mostrará el mensaje de error pertinente. Sin embargo, si el activo es reconocido por el sistema, se mostrará la pantalla de la divisa en detalle. Una vez en la página de la criptomoneda, se deberán introducir los datos requeridos, como cantidad y justificación, y el estudiante deberá seleccionar si desea ejecutar una orden de compra o venta. En ambos casos, lo primero será comprobar que no hay campos vacíos y, tras esto, se procederán con el resto de las comprobaciones. Para el caso de la compra, la aplicación se asegurará de que el usuario cuenta con saldo suficiente para completar la operación, en cuyo caso mostrará un mensaje de éxito, y si no es así, mostrará un mensaje de error. Para la venta es similar, pero se comprobará que el usuario disponga en su cartera de la cantidad adecuada de la criptomoneda que quiera vender.

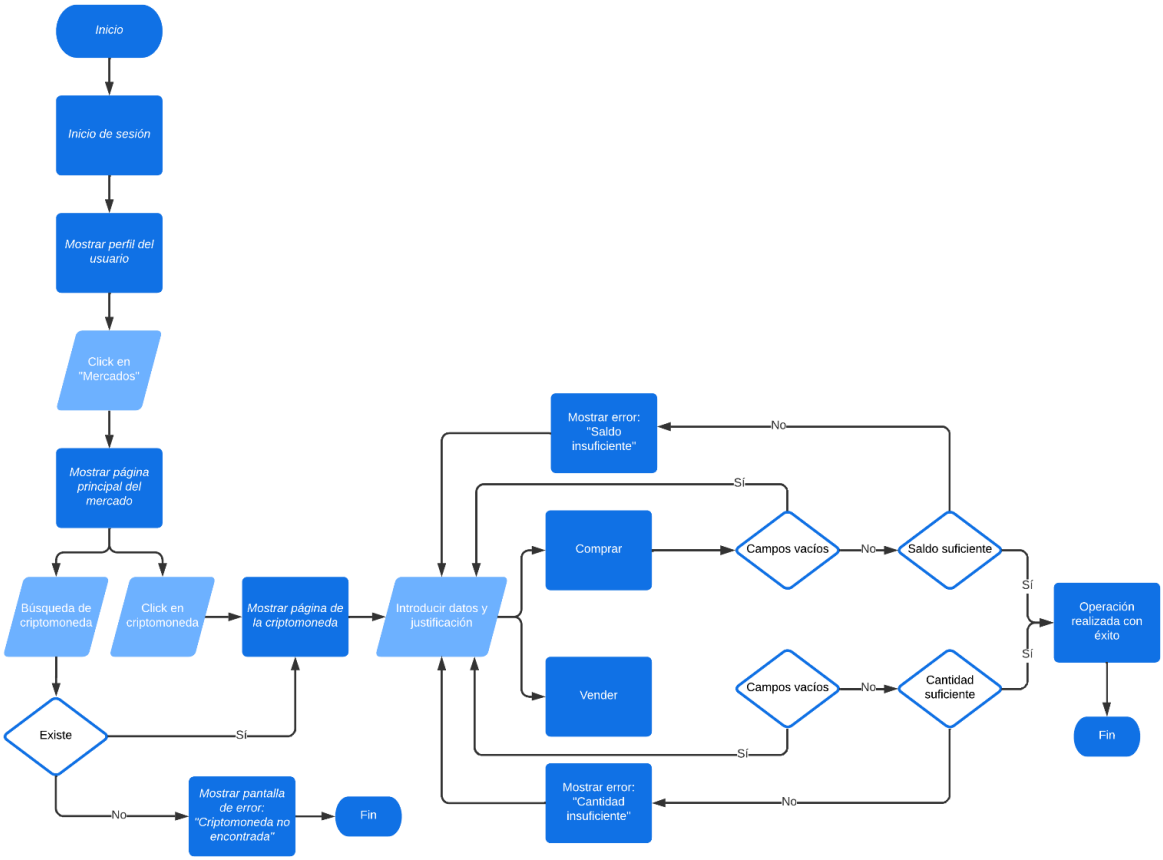


Imagen 2: Diagrama de flujo de una transacción. Fuente: Elaboración propia.

El resto de los casos de uso son muy sencillos, por lo que se considera que realizar un esquema completo no es necesario. Por ejemplo, iniciar sesión, crear estudiantes y profesores, y asignar alumnos a los docentes se puede resumir en un simple formulario que muestra un error si hay campos vacíos y realiza la verificación correspondiente a los datos. Por otra parte, ver la cartera, los comentarios del profesor o escribir un comentario son representados por camino lineal que no requiere de una toma de decisiones ni escenarios excesivamente complejos.

2.1.4: Metodología y gestión de ramas

Como metodología de trabajo se ha utilizado Kanban. La principal característica de esta forma de trabajar es la división de las tareas en tres estados: sin iniciar, en curso y finalizadas. Esto permite tener una visión global de los avances del proyecto, así como permitir una organización sencilla y visual (Santander, 2020).

En este caso, se ha recurrido a la herramienta Trello para la creación del tablero y gestión de las diversas tareas.

A la hora de desarrollar, el uso de esta metodología se ha reflejado en la gestión de ramas. Cada tarea ha sido desarrollada en una rama diferente, lo que permite avanzar cada una sin afectar a las demás. Para el identificador de estas se ha escogido el nombre de la tarea acompañado de un prefijo que indica su naturaleza. Los dos prefijos utilizados han sido:

- feature/*: Indica que la rama corresponde a una nueva funcionalidad o elemento a introducir en la aplicación. Este prefijo ha sido utilizado, por ejemplo, cuando se ha implementado la posibilidad de crear comentarios en las transacciones de los estudiantes (feature/TeacherFeedback) o cuando se ha añadido seguridad a la aplicación mediante JSON Web Tokens (feature/JWT).
- fix/*: Muestra que el objetivo de la tarea es arreglar algún problema detectado en la aplicación o llevar a cabo algún cambio que no está relacionado con nuevas funcionalidades. Este tipo de nomenclatura se ha empleado en la corrección de errores, como en el cambio de los puertos de las bases de datos debido a un error (fix/DB_Ports), o en el arreglo de los logos de cada criptomoneda cuando no se estaban mostrando de la manera correcta (fix/CoinLogos).

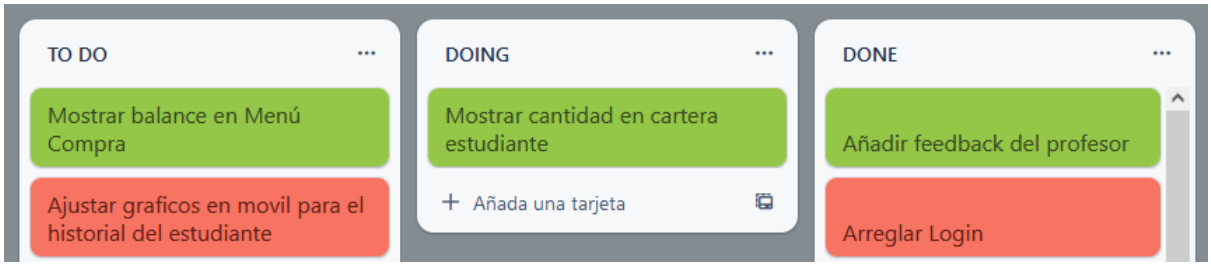


Imagen 3: Tablero Kanban utilizado en el proyecto. Fuente: Elaboración propia.

Las ramas feature y fix corresponden a ramificaciones temporales que son eliminadas cuando se ha completado la implementación de los cambios por los que surgieron. Sin embargo, hay otras dos ramas que permanecen a lo largo de toda la vida del proyecto. Estas son:

- Develop: Recoge los cambios del entorno de preproducción. Cuando se completa el desarrollo de una rama fix o feature éstas son mezcladas mediante una Pull Request con la rama develop. Cuando los cambios son aprobados finalmente, se ejecuta un workflow que despliega esta rama en el entorno de preproducción.
- Main: Es la rama principal del proyecto y corresponde con el ambiente final de producción. Una vez han sido comprobados los cambios de la rama develop ésta es mezclada finalmente con main mediante una Pull Request para realizar el pase a producción. También existe un workflow que se encarga de realizar esta acción de forma automática.

2.1.5: Arquitectura

Tras detallar los requisitos y diagramas que muestran las funcionalidades que tendrá la aplicación, es el momento de presentar los componentes software que formarán el sistema.

En primer lugar, encontramos el Frontend. Este será el componente de la aplicación con el que interactuará el usuario. Como se especificó en los requisitos, estará construido principalmente con React.

El Backend de la aplicación tiene una arquitectura un poco más compleja. Se compone principalmente de tres APIs:

La primera de ellas, UsersAPI, se encargará de manejar toda la información relacionada con los usuarios, es decir, debe poder crear y facilitar información sobre profesores, alumnos y administradores, y permitir realizar todas las acciones asociadas con ellos. Para persistir esta información hace uso de una base de datos PostgreSQL.

Por otra parte, se encuentra la API de precios, PricesAPI. Esta API permite almacenar y consultar la información de precios de las diferentes criptomonedas para crear los distintos gráficos en el Frontend y mostrar los precios de las divisas. Utiliza como fuente de datos principal la API pública de Binance, que es el mayor mercado de criptomonedas del mundo.

Todo lo anterior está orquestado bajo Docker Compose, lo que facilitará el despliegue y puesta en marcha de los componentes.

Por último, se encuentra una tercera API, desarrollada en Express, que se sitúa fuera de todo el sistema de Docker Compose. Se trata de un componente que no fue concebido como parte de esta aplicación en un primer momento, sino que su creación responde a un problema que surgió durante el desarrollo. Al comienzo del proyecto, PricesAPI era la pieza encargada de obtener los precios directamente de la API de Binance. Sin embargo, debido a problemas legales y regulatorios Binance ha tenido que limitar el uso de su API y crear otra alternativa con funcionalidades recortadas para los usuarios que consuman el servicio desde determinados países, como Estados Unidos (Binance, 2023). Esto repercutió en la aplicación debido a que el sistema estaba desplegado en un servidor estadounidense. Además, estas restricciones conllevaron limitaciones en la cantidad de datos recopilados y variedad de criptomonedas disponibles. Como esta situación tenía un impacto muy negativo en el funcionamiento del sistema, se decidió crear la API de Express. Esta API está desplegada en un servidor europeo y se encarga de realizar las peticiones que PricesAPI necesite, entregándole la información desde Europa y evitando las restricciones impuestas en Estados Unidos.

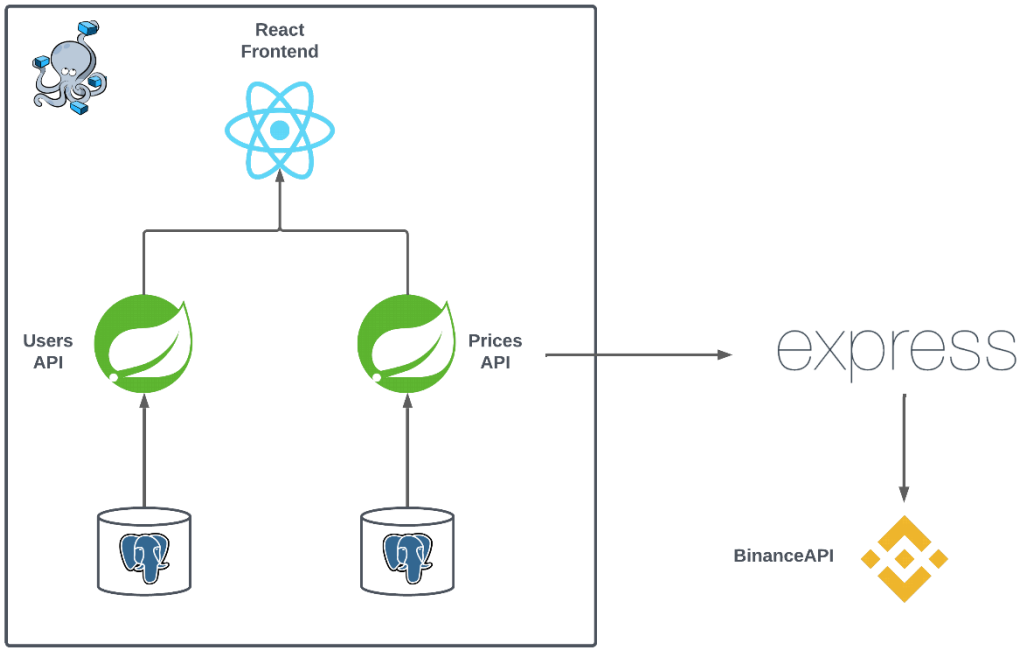


Imagen 4: Arquitectura de la aplicación. Fuente: Elaboración propia.

2.2: Implementación

A lo largo de esta sección se detallará la implementación concreta de cada sistema que se ha mencionado en el epígrafe de arquitectura. Además, se comentarán los entornos utilizados, la seguridad del sistema, las pruebas que se han implementado y la forma de realizar los despliegues a producción.

2.2.1: Interfaz de usuario

Para la implementación de la interfaz de usuario se ha utilizado como base la librería React. Al no tratarse de un marco de trabajo completo, todo la organización del proyecto se realizará a gusto del desarrollador, incluyendo funcionalidades como el enrutado, la creación de builds o la propia organización de carpetas.

2.2.1.1: Estructura de carpetas

La estructura de carpetas del proyecto permite conocer la implementación de las diferentes partes del sistema a grandes rasgos. En este caso, la carpeta de desarrollo de la interfaz de usuario del proyecto se ha organizado de la siguiente forma:

- Assets: Contiene las diferentes imágenes y recursos utilizados en la aplicación.
- Config: En su interior se puede encontrar la configuración de un tema de estilos personalizado creado a partir de JoyUI.
- Pages: En esta carpeta se almacenan las páginas principales que se renderizarán dependiendo de la URL. Se han desarrollado las siguientes:
    - MainPage: Es la página principal de la aplicación. En ella se muestra una breve descripción del proyecto. También contiene el botón que permite acceder al formulario de inicio de sesión.

Imagen 5: Página principal. Fuente: Elaboración propia.

- UserPage: Renderiza la información del usuario. Es capaz de adaptarse a cada tipo de usuario, es decir, muestra distintos componentes en función de si el usuario es un administrador, un estudiante o un profesor. Esto se consigue mediante el renderizado condicional tras recibir la información de la UsersAPI.

Imagen 6: Página de estudiante. Fuente: Elaboración propia.

- MarketPage: Muestra las nueve criptomonedas más importantes del momento y una barra de búsqueda para acceder a todas las que no se encuentren en la lista. Sirve exclusivamente para navegar a la página que detalla cada criptomoneda de forma individual, que será descrita a continuación.

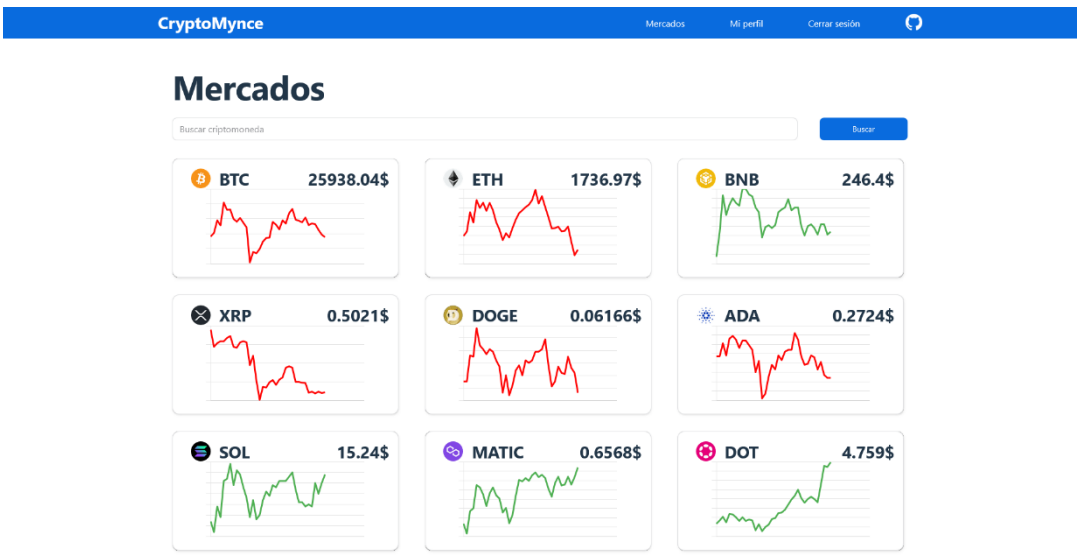


Imagen 7: Página de mercado. Fuente: Elaboración propia.

- CoinPage: En esta página se pueden encontrar los datos completos de una criptomoneda en específico. La información está representada principalmente con un gráfico de gran tamaño que actualiza el precio cada diez segundos. Además, muestra un menú donde los estudiantes deberán introducir la cantidad que quieren comprar o vender e incluir una justificación de la transacción.

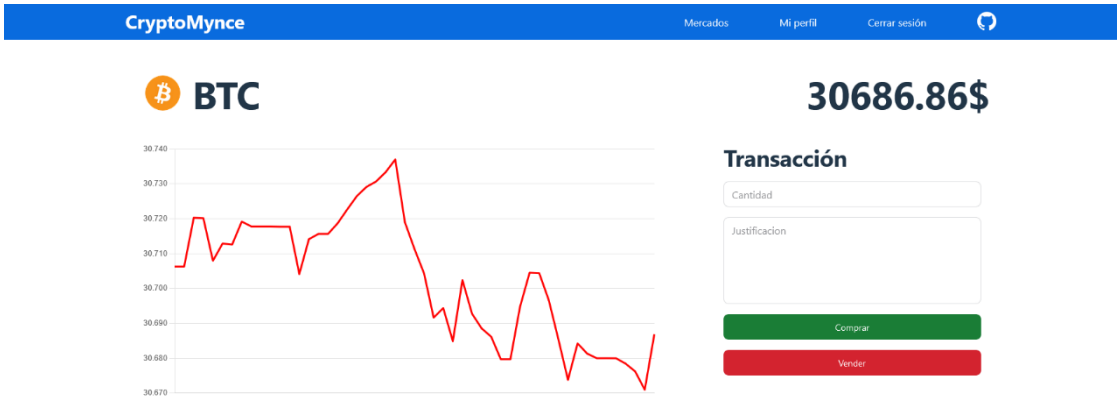


Imagen 8: Página de la criptomoneda Bitcoin. Fuente: Elaboración propia.

- NotFoundPage: Está página se renderiza cuando el usuario ha intentado acceder a una ruta que no existe. Simplemente muestra un mensaje que indica que la ruta no es válida.

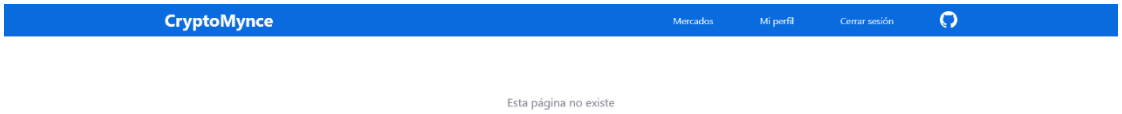


Imagen 9: Página de ruta inexistente. Fuente: Elaboración propia.

- Components: Los elementos que conforman cada página se conocen como componentes (React, 2023). Por ejemplo, en una página de usuario para un estudiante se podría visualizar uno llamado "StudentDashboard", que a su vez está integrado por diversos subcomponentes como textos, tarjetas que detallan cada transacción realizada, tarjetas que representan las criptomonedas que componen su cartera y un gráfico que resume esta información. De hecho, las propias tarjetas también están creadas a partir de otros subcomponentes. Este proceso sigue hasta llegar a componentes más simples como textos o botones. El fin detrás de esta atomización de elementos es la reutilización de estos en una gran variedad de lugares dentro de la aplicación, lo que resultará útil para no repetir líneas código y mantener lenguaje visual común (React, 2023).

- Hooks: En esta carpeta se guardan los “hooks” personalizados. En React, un “hook” es una función que permite utilizar algunas características de React, como almacenar en memoria determinada información de un componente, ejecutar código en un momento determinado, o encapsular cierta lógica para reutilizarla en diferentes lugares (React, 2023). Por defecto React incluye algunas de estas funciones especiales y son realmente útiles. Sin embargo, puede que el desarrollador encuentre otras situaciones en las que sea conveniente crear un “hook” personalizado. Por ejemplo, en esta aplicación se han utilizado “hooks” personalizados principalmente para realizar las diferentes llamadas a UsersAPI y PricesAPI. De esta forma, se consigue que los componentes sean agnósticos a la implementación de estas llamadas puesto que solo conocen el estado de su petición y el resultado de estas. Otro “hook” que se ha implementado es useAuth, que se encarga de comprobar si el usuario ha iniciado sesión o no.
- Context: Almacena los contextos utilizados en la aplicación. Un contexto es útil cuando dos elementos deben compartir información, pero se encuentran lejos en el árbol de componentes, lo cual obligaría al desarrollador a pasar los datos entre todos los nodos intermedios hasta llegar al que consumirá la información. En React, esto se conoce como “prop drilling”. Gracias a los contextos se puede evitar esta situación, compartiendo datos o funciones que podrán ser accedidas desde cualquier nodo que tenga acceso sin necesidad de ser enviados explícitamente de padres a hijos (React, 2023). En esta aplicación solo ha sido necesario crear un contexto. En él se almacena una función para obtener la información del usuario mediante una llamada a UsersAPI y una variable que indica si se ha finalizado esa petición o aún sigue en curso. Estos elementos son utilizados por componentes muy alejados, por lo que era necesario crear un contexto.

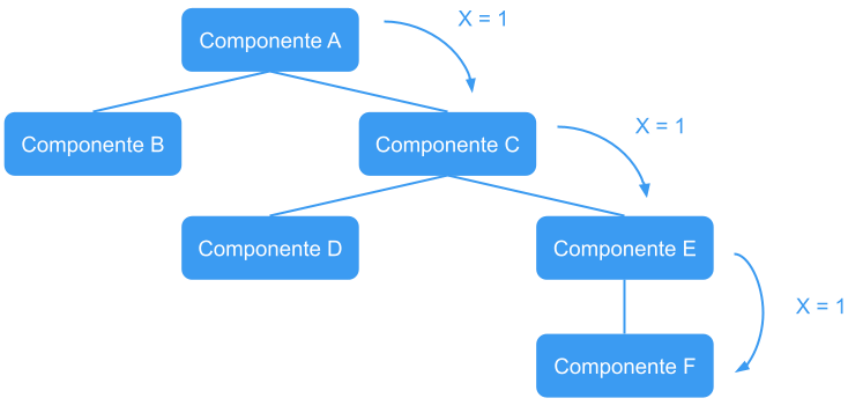


Imagen 10: Ejemplo de “prop drilling”. Fuente: Elaboración propia.

En la imagen anterior se puede observar un ejemplo de “prop drilling”. El componente A necesita compartir el valor de la variable X con el componente F. Para ello sería necesario que los componentes C y E tuvieran acceso a X exclusivamente para que pueda ser utilizada por F. Si se implementara un contexto en esta situación, el componente A podría almacenar allí la información de X y ser directamente consultada por F, sin necesidad de que C y E conozcan explícitamente su valor.

- Utils: En esta carpeta se almacenan las funciones auxiliares que pueden ser utilizadas por otros componentes y hooks del sistema. Para este proyecto sólo se ha incluido una, stringBUSD, que se ocupa de detectar las terminaciones de los nombres de las

criptmonedas acaban en el sufijo “BUSD” y eliminarlo. Es utilizada principalmente para modificar las respuestas recibidas por la API de precios.

- Ficheros Main.jsx, App.jsx y GlobalStyles.css: Main.jsx es un fichero cuya función será simplemente renderizar el componente raíz del proyecto, que se encuentra en App.jsx, a partir del cual se crean el resto. Por otro lado, GlobalStyless.css contiene una configuración simple de estilos que se aplica en toda la aplicación.

2.2.1.2: Vite

Vite es una herramienta para desarrollar y compilar proyectos con diferentes marcos de trabajo y librerías de JavaScript entre las que se encuentra React. Una de sus bondades principales, y la que le da nombre, es su extrema rapidez (Vite, 2023). En este caso se ha utilizado para crear la estructura del proyecto, crear las builds de los distintos entornos, ejecutar pruebas y elaborar la interfaz gracias al servidor de desarrollo que tiene integrado.

2.2.1.3: JoyUI

Tal y como se especificó en los requisitos, Joy UI será la biblioteca empleada como apoyo en la parte visual. Se trata de una biblioteca desarrollada por los creadores de Material UI. Aún está en fase Alpha, pero cuenta con suficientes componentes como para ser utilizada en un proyecto de este estilo.

```
<Textarea
  minRows={3}
  sx={{
    my: 1,
  }}
  onChange={(e) => setNewComment(e.target.value)}
/>
<Box
  size={"xs"}
  sx={{
    display: "grid",
    gridTemplateColumns: {
      xs: "0% 100%",
      md: "75% 25%",
      lg: "75% 25%",
      xl: "77% 23%",
    },
    placeContent: "right",
    my: 1,
  }}
>
  <Button
    loading={loading || contextLoading}
    onClick={handleSubmit}
    sx={{ gridColumn: 2 }}
  >
    {TeacherFeedbackElements.SubmitButton}
  </Button>
</Box>
```

Imagen 11: Ejemplo de uso de JoyUI en el componente TeacherFeedback. Fuente: Elaboración propia.

Estos componentes servirán como elementos básicos a partir de los cuales crear componentes propios de una mayor complejidad. Entre la gran variedad de elementos que se pueden encontrar en la librería, se han utilizado principalmente: Button, Typography, Box, CircularProgress, Card, TextArea, TextField, Avatar y diferentes Modales. Además de los propios componentes, Joy UI también pone a disposición del desarrollador una API sencilla que permite personalizar una gran cantidad de aspectos de cada uno de ellos. Incluye funcionalidades como modificar fácilmente el aspecto de los componentes a través de propiedades o añadir los estilos CSS.

Otra funcionalidad clave es la facilidad para hacer adaptables los componentes a las diferentes resoluciones de pantalla. Gracias a la propiedad sx, es posible modificar los estilos de cada componente de JoyUI para adaptarse al ancho de la ventana del navegador. En concreto, tiene predefinidos cuatro puntos según el ancho actual de la pantalla. Además, esta funcionalidad también permite utilizar variables además de las dimensiones de la pantalla, lo que facilita la reutilización de estos componentes.

```
sx={{
  display: role === "STUDENT"
  ? "block"
  : role === "TEACHER" &&{
    xs: "block",
    md: "grid",
    lg: "grid",
    xl: "grid",
  },
  gridTemplateColumns: "50% 50%",
  gap: {
    xs: 0,
    md: 7,
    lg: 7,
    xl: 9,
  },
}}
```

Imagen 12: Ejemplo de adaptabilidad al ancho de la pantalla y modificación del valor de la propiedad CSS "display" en función de la variable "role". Fuente: Elaboración propia.

Por último, cabe destacar que JoyUI permite personalizar sus componentes de una forma más profunda aún. Esto se realizará a través de un tema personalizado del sistema y permitirá modificar los valores que utiliza la librería por defecto o añadir nuevas propiedades. En concreto, esta funcionalidad se ha empleado para definir nuevos tamaños de letra, modificar algunas de las proporcionadas por defecto, y cambiar el color predeterminado de los componentes de la aplicación.

2.2.1.4: Routing

En React, al tratarse de únicamente una librería para crear interfaces de usuario y no un marco de trabajo completo como Angular o Vue, no se encuentran incluidos por defecto todos los elementos básicos de una Single Page Application. Uno de ellos es el sistema de rutas. Para solventar esto, se debe instalar otra librería externa que proporcione estas funcionalidades. En

este caso se ha elegido React Router, que es una de las alternativas más populares del mercado y resulta realmente sencilla de usar.

```
<Routes>
  <Route path="/" element={<MainPage />}></Route>
  <Route element={<PrivateRoutes />}>
    <Route path="/market" element={<MarketPage />}></Route>
    <Route path="/coins/:coin" element={<CoinPage />}></Route>
    <Route
      path="/users/:user"
      element={
        <UserContext.Provider
          value={{ loading: true, GetUserData: () => {} }}
        >
          <UserPage />
        </UserContext.Provider>
      }
    ></Route>
    <Route path="*" element={<NotFoundPage />} />
  </Route>
</Routes>
```

Imagen 13: Rutas de la aplicación React. Fuente: Elaboración propia.

Con esta librería se define cada ruta utilizando el componente Route, y se agrupan dentro del componente padre Routes. En cada una, se definirá la dirección asociada con la propiedad "path" y se escogerá el componente a renderizar con "element".

Para el caso de las páginas de criptomonedas y usuarios, se necesita utilizar un misma ruta, pero personalizarla para cada elemento concreto. Por ejemplo: "coins/BTC" y "coins/ETH" o "users/Sergio" y "users/Paula". Para tener una parte variable dentro de una ruta, React Router ha creado los segmentos dinámicos. Se escriben con la nomenclatura "/parteEstática/:parteDinámica", y permiten recuperar la parte dinámica como parámetro para utilizarla dentro del componente.

También se ha contemplado el caso de que el usuario intente acceder a una ruta inexistente. Para que la aplicación no muestre una página totalmente en blanco, se ha descrito la ruta por defecto, "*", que renderiza un mensaje de error.

Por último, se puede apreciar que hay un conjunto de rutas que tienen una ruta padre. El grupo está formado por todas las rutas menos la que renderiza la página de inicio. Esto se debe a que esas direcciones solo pueden ser accedidas si el usuario ha iniciado sesión, es decir, son rutas privadas. En React Router no hay una forma oficial de crear este tipo protección, sino que el desarrollador debe crear su propia solución al problema. En este caso, se ha creado una ruta de orden superior que renderiza un componente llamado PrivateRoutes. Este elemento se encarga de comprobar que el usuario está autenticado, haciendo uso del "hook" useAuth. En caso de que lo esté, le permite acceder a las rutas protegidas gracias al componente Outlet que proporciona React Router. Por el contrario, si no se ha iniciado sesión, el usuario será redirigido a la página principal.

```
export const PrivateRoutes = () => {
  const isAuthenticated = useAuth();
  return isAuthenticated ? <Outlet /> : <Navigate to="/" />;
};
```

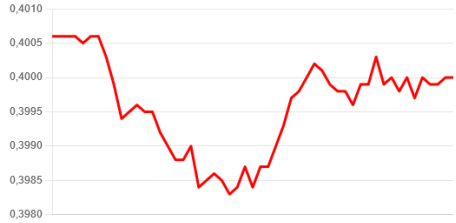
Imagen 14: Componente PrivateRoutes. Fuente: Elaboración propia.

Realmente el uso de rutas privadas no es una forma de seguridad en sí misma, puesto que se ejecuta en el navegador y un usuario avanzado podría manipularlo para saltarse estas restricciones. En esta aplicación se han implementado medidas adicionales, que serán desarrolladas en un epígrafe exclusivo más adelante.

2.2.1.5: Gráficos

Como se puede observar en los requisitos, los gráficos son un elemento indispensable en esta aplicación. Para su creación se ha utilizado la librería react-chartjs-2, que pone a disposición del desarrollador componentes de React que contienen los gráficos de la famosa librería Chart.JS, y propone una forma sencilla de modificar sus parámetros para personalizarlos. En este caso, se ha hecho uso de dos formatos:

El primero de ellos es el gráfico de línea, que ha sido utilizado para la representación de los precios de las diversas criptomonedas en la aplicación. Gracias a la posibilidad de configurar el gráfico, se les ha dotado de algunas funciones adicionales. Entre ellas destaca el cambio del color dependiendo de los valores recibidos en forma de lista. En caso de que el primer elemento de la lista sea menor que el último, se mostrará de color verde indicando que el precio ha aumentado. En el caso contrario, se reflejará que el precio es inferior coloreando el gráfico de rojo.



Imágenes 15 y 16: Gráficos de línea creados con react-chartjs-2. Fuente: Elaboración propia.

El segundo es el gráfico circular, que se ha empleado como representación gráfica del peso de cada elemento en la cartera del estudiante. Esto permite mostrar de una forma visual y rápida cómo el usuario ha dividido el capital entre los diferentes activos que posee.

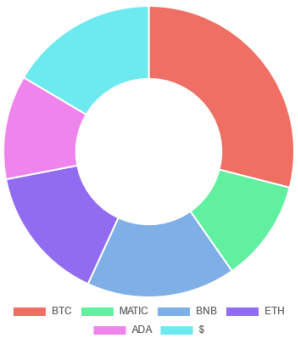


Imagen 17: Gráfico circular creado con react-chartjs-2. Fuente: Elaboración propia.

2.2.2: UsersAPI

Esta API REST es la más compleja de las tres. Mientras que las otras dos APIs, que se explicarán en los epígrafes siguientes, son únicamente para consultar información, esta es la que permite realmente llevar a cabo todas las acciones que solicitan los usuarios.

Como se ha especificado en los requisitos, está implementada utilizando Spring Boot, concretamente en su versión 2.7.4.

A continuación, se procederá a explicar las diferentes entidades y endpoints que se han desarrollado para dotar a la aplicación de la funcionalidad deseada.

2.2.2.1: UserProfile

UserProfile es la clase que contiene la información del usuario básico. Como atributos cuenta con un Id, nombre de usuario, correo electrónico, contraseña y listado de roles. En cuanto a los métodos, sólo implementa getters y setters.

Esta clase es utilizada para crear los administradores, puesto que no requieren de información adicional. Sin embargo, estudiantes y profesores sí necesitan atributos añadidos como el saldo en la cuenta o el listado de alumnos asociados. Es por ello que estos tipos de usuarios utilizan como base UserProfile, pero extienden su funcionalidad mediante herencia. Más adelante se explicarán en mayor profundidad.

La clase UserProfile está anotada como entidad para poder almacenar su información en una base de datos (Spring Framework, 2023). Por hacer uso de herencia, además se debe especificar una estrategia para que Spring organice la información de las clases hijas (Baeldung, 2023). En este caso se ha elegido crear una tabla por cada clase. Finalmente, los accesos a la información persistida se consiguen gracias a la interfaz UserProfileRepository. Ésta hereda de JpaRepository y permite realizar las operaciones de consulta, guardado y modificación de la información de una manera muy sencilla (Simplilearn, 2023). Aunque incluye por defecto una gran cantidad de métodos realmente útiles, se ha implementado además otro llamado findByName, que permite al desarrollador obtener un usuario utilizando únicamente el nombre.

2.2.2.2: Student

Como se ha mencionado anteriormente, la clase Student hereda de UserProfile. Además de todos los métodos y atributos de su clase padre, también añade algunos adicionales para implementar las acciones necesarias de un estudiante.

Básicamente agrega tres atributos: balance, portfolio y tradeHistory. El primero de ellos simplemente almacena el saldo disponible que tiene el estudiante para realizar las compras de criptomonedas. El segundo es un mapa que tiene como claves las criptomonedas que componen la cartera del usuario y como valores la cantidad que posee de cada una de ellas. Por último, también se puede encontrar un historial de transacciones implementado mediante una lista.

Como métodos tiene getters, setters y alguna función auxiliar. Entre ellas, cabe destacar dos funciones, addToPortfolio y sellFromPortfolio. El objetivo de ambas es completar una transacción que se ha solicitado, modificando los tres atributos que se han descrito anteriormente para poder añadir o eliminar un activo de la cartera. En ambos se hace una

comprobación inicial. Para el caso de una compra se utilizará el primer método y la comprobación será referente al saldo disponible. En el segundo caso, la venta, se comprobará que la cantidad que se quiere vender es inferior o igual a lo que posee el usuario en su cartera para esa criptomoneda concreta. Si cualquiera de estas comprobaciones fallara, se lanzaría una excepción. Finalmente, se procede a modificar el saldo y añadir o reducir la cantidad del activo en cartera.

Para el almacenamiento en base de datos no será necesario utilizar otro JpaRepository gracias a la anotación de la estrategia de herencia en la clase UserProfile. Esto se ha implementado de esta manera para simplificar la recuperación de la información y la inyección de las dependencias en las clases que requieran de ella.

2.2.2.3: Teacher

Al igual que la clase para el estudiante, los profesores también heredan de UserProfile. En este caso los añadidos son mínimos, incluyendo únicamente un atributo adicional que se encarga de recoger los estudiantes asociados al profesor. Esto se implementa a través de un conjunto, dado que el orden no es relevante.

Como métodos adicionales se incluyen el getter y setter del conjunto acompañados de algunos otros que sirven para simplificar las acciones de añadir y borrar elementos.

Tampoco es necesario crear un JpaRepository específico para los profesores por la misma razón que en el caso del estudiante.

2.2.2.4: Trade

Esta clase se encarga de representar las transacciones que realizan los estudiantes. Tiene un gran número de atributos, pero como métodos solo implementa getters y setters. Dentro de los atributos se encuentra la asignación de un número identificativo único para cada transacción, lo que permite su fácil recuperación en el futuro. Además, se especifica el tipo de transacción realizada (compra o venta), así como la cantidad de criptomonedas que el alumno desea adquirir o vender, y su precio en el momento de la transacción. Es importante destacar que los alumnos deben proporcionar una justificación clara y detallada para cada transacción que realicen. Además, se ha decidido almacenar una lista con los precios del gráfico en el momento de la transacción, lo que permitirá representar el gráfico en la interfaz de usuario. Por último, se registra la fecha de creación de cada transacción y se recopila la retroalimentación enviada por los profesores para evaluar el desempeño de los alumnos en la realización de transacciones con criptomonedas.

En este caso, para simplificar el acceso a la información de esta entidad en base de datos si será necesario hacer uso de un JpaRepository específico.

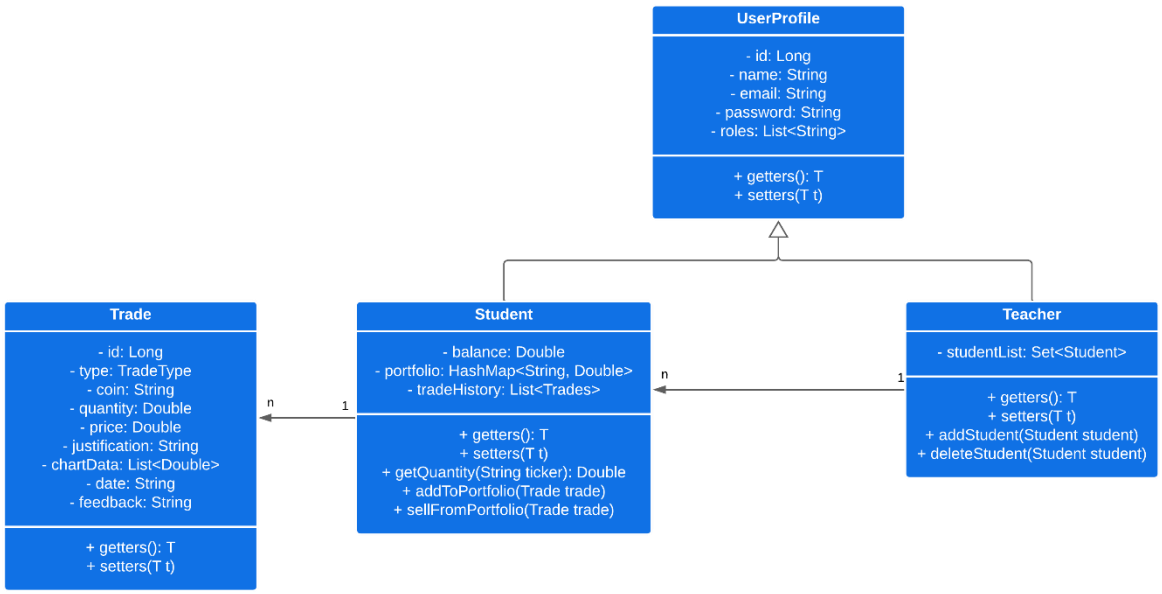


Imagen 18: Diagrama de clases. Fuente: Elaboración propia.

2.2.2.5: Endpoints

Los endpoints de esta API están agrupados en dos controladores. El primero tiene como objetivo gestionar las operaciones relacionadas con la autenticación de usuarios. El segundo realiza el resto de las operaciones.

2.2.2.5.1: AuthRestController

Los endpoints son los siguientes:

- /signup: Se encarga de crear los usuarios. Recibe un Data Transfer Object llamado SignUpDTO, que contiene los datos necesarios para dar de alta un usuario entre los que se incluye una lista de roles. Lo primero que realizará el endpoint será comprobar que el usuario no existe en el sistema. A continuación, se encargará de crear el estudiante, profesor o administrador utilizando los datos proporcionados. En caso de que se solicite crear un usuario con un rol diferente a esos tres, responderá con un código 400 al solicitante.
- /login: Este endpoint comprobará si las credenciales del usuario que está tratando de iniciar sesión son correctas. En caso de que lo sean responderá con un nuevo token. En el epígrafe de seguridad se profundizará en ellos.

2.2.2.5.2: UserRestController

Los endpoints que este controlador pone a disposición de los usuarios son los siguientes:

- /users/{username}: El objetivo es devolver la información del usuario solicitado. El primer paso será comprobar que efectivamente existe en el sistema. Una vez asegurada su existencia, se obtendrá la información utilizando el repositorio de usuarios UserProfileRepository. A continuación, se verificará si es un estudiante, profesor o administrador a través del tipo del objeto. Por último, se crea el DTO correspondiente, StudentDTO, TeacherDTO o AdminDTO, y se envía como respuesta.

- /students/{username}/trade: Este endpoint se encarga realizar las transacciones pedidas por los estudiantes. Como el proceso de compra y venta es muy similar, se ha optado por utilizar un solo endpoint en lugar de dos. Se recibe un TradeDTO en el cuerpo de la petición con toda la información necesaria para poder crear un objeto de la clase Trade. En funcionamiento es el siguiente: en la ruta del endpoint el nombre de usuario corresponde al estudiante que realiza la transacción, por lo que se deberá comprobar si existe en el sistema. Tras esto, se obtendrá su información y se comprobará que se trata de un estudiante. A continuación, se bifurcan los caminos de la compra y la venta. Para el primer caso, se creará un nuevo Trade y se añadirá a la cartera del estudiante. En caso de que el método llamado lance una excepción porque no sea posible completar la compra, se responderá al usuario con un código HTTP 402. En la realización de una venta se sigue un procedimiento similar, pero se emplea otro método de la clase Student destinado a este cometido.
- /teacher/{username}: Se ha utilizado para gestionar la asignación de estudiantes a un profesor. Este endpoint recibe en el cuerpo de la petición una lista con todos los nombres que se desean asignar mediante un objeto StudentListDTO. En primer lugar, se comprueba que el nombre de usuario del profesor realmente exista. A continuación, se recorre la lista recibida comprobando que los nombres son correctos y añadiéndolos. En caso de encontrar un estudiante cuyo nombre de usuario no esté previamente registrado en el sistema o ya estuviera asignado al profesor, se responderá con un error y se cancelará la asignación de todos los estudiantes de la lista.
- teachers/{teacherUsername}/students/{studentUsername}/trades/{id}/feedback: Por último, se encuentra el endpoint que permite a los profesores añadir comentarios a una transacción de un alumno. En este caso, los tres elementos están identificados como variables dentro de la propia ruta. El primer paso será comprobar que todos ellos existen en el sistema y, en caso de que no sea así, responder con un código 404. También se responderá con este código en caso de que alguno de ellos falte. Por último, tras rescatar los tres elementos de su repositorio correspondiente, se agrega el comentario a la transacción indicada. Esta acción solo se completará tras asegurar que el estudiante indicado está asignado en la lista del profesor y que la transacción efectivamente fue realizada por el alumno especificado.

2.2.3: PricesAPI

Como se ha explicado en los epígrafes anteriores, este componente de la aplicación se encarga de entregar a la interfaz los datos actualizados de todas las criptomonedas disponibles en Binance. Para ello, debe ser capaz de persistir la información obtenida y proveer endpoints que permitan al resto de componentes acceder a esos datos.

2.2.3.1: Obtener información de precios

Para obtener la información se ha implementado un componente Spring llamado ScheduledTasks. Este componente tiene diferentes funciones:

- fetchApiCall: Es la función principal. Se encarga de pedir la información de las criptomonedas a la API de Express o directamente a la API de Binance.US. Se utiliza la primera API cuando se necesitan datos para producción y la segunda durante la realización de pruebas y el tiempo de desarrollo. Esta decisión ha sido tomada para no

sobrecargar el servicio de Express con un exceso de llamadas innecesarias que se producen mientras se ejecutan las pruebas o se desarrolla el sistema, escenarios para los cuales no es realmente necesaria una API tan completa. En ambos casos las peticiones han sido implementadas utilizando la librería RestTemplate. Una vez se obtiene la información, se deberá filtrar antes de entregarla como resultado de la función. Es importante realizar este filtrado para utilizar únicamente los datos que son de interés y descartar el resto. En este caso es necesario, puesto que la API de Binance proporciona información de las criptomonedas con varios tipos de cambios diferentes. Por ejemplo, el precio de Ethereum es recibido en dólares, euros y Bitcoins, entre otros. Para este proyecto se ha decidido que se utilizará únicamente el precio de cada criptomoneda en dólares.

- fetchLastValues: Esta función está programada para realizarse en cada diez segundos con el fin de tener actualizados todos los precios. En un primer lugar llama a la función fetchApiCall para obtener las datos actualizados y filtrados. Tras esto, su objetivo será persistir los datos de cada criptomoneda. En caso de encontrar en la respuesta una divisa no registrada previamente, también se encargará de crear una entidad que se almacenará en la base de datos.
- fetch30mValues: El comportamiento es igual que el anterior, pero se ejecuta cada treinta minutos. En este caso, su objetivo será persistir información en un horizonte temporal más amplio para poder formar gráficos que muestren el comportamiento de la criptomoneda a lo largo del día.
- fetch1dValues. Funciona de la misma manera que los dos anteriores, pero solicita los datos de manera diaria.

2.2.3.2: Persistir información de precios

Toda la información obtenida por las acciones programadas se almacena en base de datos. Al utilizar Spring Boot, es necesario crear una entidad. En esta API se ha creado una llamada Coin, que tiene un Id, una etiqueta de cotización que guarda el nombre de cada criptomoneda, el precio actual y tres listas. Estas listas guardan la información de los precios obtenidos cada diez segundos, treinta minutos y un día. Además, para acceder a la información de estas entidades de tipo Coin de una forma sencilla, se ha creado CoinRepository.

Cuando se ejecutan las acciones programadas, que han sido descritas en el epígrafe anterior, se rescatará la criptomoneda del repositorio a través de su etiqueta y se hará uso de los métodos addLastPrice, add10sPrice y add30mPrice y add1dPrice para actualizar las listas de precios. La colección de precios que se guarda cada diez segundos y la lista que almacena los valores cada treinta minutos tienen ciertas características especiales. La primera puede almacenar un máximo de cincuenta registros. Esto se debe a que esta información será utilizada para mostrar los gráficos a muy corto plazo y con una tasa de refresco alta, por lo que no es necesario mantener un gran volumen de información. Por otra parte, la segunda lista sólo mantendrá los precios del día actual. Esto se consigue borrando sus valores cuando se ejecuta la tarea programada fetch1dValues.

2.2.3.3: Proveer información de precios

Para responder a las solicitudes de datos procedentes del resto de componentes de la aplicación, han sido creados cuatro endpoints. El primero de ellos, "/last/{ticker}", envía el último precio registrado de la criptomoneda identificada por la etiqueta de cotización (ticker). Los otros tres endpoints son: "/10s/{ticker}", "/30m/{ticker}" y "/1d/{ticker}". Todos devuelven al solicitante la información de las listas de precios que se han guardado para la etiqueta pedida. En los cuatro endpoints se accede al CoinRepository para comprobar si la criptomoneda existe en el sistema. En caso negativo, se envía un código 404 indicando que no está presente. Por otro lado, si la información está presente, se envía al usuario junto con un código HTTP 200 para indicar que la solicitud ha resultado exitosa.

2.2.4: ExpressAPI

Esta API tiene una implementación muy sencilla, puesto que su único objetivo es enviar la información que la API de precios necesite en cada momento.

Dispone de un único endpoint accesible bajo la ruta "/prices". Cuando se reciba una petición, el sistema realizará una solicitud a la API de oficial de Binance con la que se obtendrá información de los precios de más de trescientas criptomonedas al mismo tiempo.

2.2.5: Docker

En el apartado de arquitectura se comentó que todo el sistema principal de la aplicación estaba orquestado por un fichero Docker Compose. Esta tecnología permite crear y ejecutar varios contenedores Docker al mismo tiempo y conectarlos entre sí.

2.2.5.1: Contenedores

Es por ello por lo que cada parte principal de la aplicación (excepto la API Auxiliar de Express) tiene definido un fichero Dockerfile. Esto le permitirá al Compose crear un contenedor específico para cada una. Además, las bases de datos PostgreSQL son ejecutadas y generadas gracias a Docker, pero haciendo uso de una imagen obtenida de DockerHub.

En el fichero Docker Compose se pueden definir además diferentes variables para los entornos en los que deba ejecutarse el sistema. Se han utilizado estas variables para definir cierta información de tal forma que sea sencillo modificarla y, a su vez, permanezca de manera oculta utilizando un sistema de secretos proporcionado por la plataforma de despliegue. Además, para cada entorno se ha creado un fichero Docker Compose que modifica ciertos aspectos. Por ejemplo, para el entorno de desarrollo se han definido archivos Dockerfile diferentes a los utilizados en el resto de los ambientes. Esto se puede ver reflejado en el Docker Compose por la modificación del campo "dockerfile" de las tres piezas fundamentales, la interfaz, UsersAPI y PricesAPI. En el caso de la interfaz, también ha sido necesario generar una variable ENV que se encuentra dentro del apartado de argumentos del Docker Compose. Esta variable permite definir el ambiente en que se encuentra el sistema con el fin de crear las builds

adecuadas, puesto que es necesario especificarlo en la ejecución de un comando dentro de su Dockerfile. Más adelante en el documento se detallarán los ambientes de la aplicación.

```
services:
  prices-api:
    build:
      context: ./prices-api
      dockerfile: ./Dockerfile
    environment:
      - POSTGRES_HOST=${DB_HOST}
      - POSTGRES_DB=${DB_DB}
      - POSTGRES_USER=${DB_USER}
      - POSTGRES_PASSWORD=${DB_PASS}
      - POSTGRES_PORT=5432
      - PRICES_URL=${PRICES_URL}
      - PRICES_PASS=${PRICES_PASS}
    ports:
      - "8080:8080"
    depends_on:
      - pg
  users-api:
    build:
      context: ./users-api
      dockerfile: ./Dockerfile
    environment:
      - POSTGRES_HOST=${DB_HOST_2}
      - POSTGRES_DB=${DB_DB_2}
      - POSTGRES_USER=${DB_USER_2}
      - POSTGRES_PASSWORD=${DB_PASS_2}
      - POSTGRES_PORT=5433
      - ADMIN_PASS=${ADMIN_PASS}
    ports:
      - "8081:8081"
    depends_on:
      - pg2
  ui:
    build:
      context: ./ui
      args:
          ENV: prod
      dockerfile: ./Dockerfile
    ports:
      - "80:80"
    links:
      - prices-api
      - users-api
    depends_on:
      - prices-api
      - users-api
  pg:
    image: "postgres:9.6-alpine"
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_HOST=${DB_HOST}
      - POSTGRES_DB=${DB_DB}
      - POSTGRES_USER=${DB_USER}
      - POSTGRES_PASSWORD=${DB_PASS}
    volumes:
       - postgres-data:/var/lib/postgresql/data
  pg2:
    image: "postgres:9.6-alpine"
    ports:
      - "5433:5432"
    environment:
      - POSTGRES_HOST=${DB_HOST_2}
      - POSTGRES_DB=${DB_DB_2}
      - POSTGRES_USER=${DB_USER_2}
      - POSTGRES_PASSWORD=${DB_PASS_2}
    volumes:
       - postgres-data2:/var/lib/postgresql/data
volumes:
  postgres-data:
  postgres-data2:
```

Imágenes 19 y 20: Archivo docker-compose.yaml. Fuente: Elaboración propia.

2.2.5.2: Persistencia

Para asegurar la persistencia del contenido de las bases de datos, es fundamental contar con un espacio donde almacenar los datos cuando se detiene la ejecución de los contenedores de PostgreSQL. Para ello, se hará uso de los volúmenes proporcionados por Docker. Estos funcionan como un sistema de almacenamiento independiente al ciclo de vida del sistema y que puede ser accedido desde cualquier contenedor. Cada vez que se crean los contenedores, se comprobará la existencia de estos volúmenes que se han definido, y en caso de existir, serán utilizados por la bases de datos para recuperar y almacenar la información necesaria durante la ejecución (Docker, 2023).

Este sistema de persistencia resulta de vital importancia cuando se despliega el sistema en producción, debido a que los datos de los usuarios no deben ser borrados en caso de que el sistema deba reiniciarse.

2.2.6: Entornos

A pesar de que se han ido mencionando a lo largo del documento, aún no se han detallado más profundamente los entornos concretos que se han utilizado en este proyecto.

2.2.6.1: Desarrollo

Como su propio nombre indica, este será el entorno empleado durante el desarrollo de la aplicación. Tiene asociado un fichero Docker Compose llamado docker-compose.dev.yaml. En este modo se utilizan los Dockerfile de desarrollo que han sido definidos para cada uno de los sistemas principales, interfaz, UsersAPI y PricesAPI.

Tanto en UsersAPI como en PricesAPI lo único que varía es que al no recibir la información de las variables de entorno de Docker Compose los sistemas utilizan la configuración por defecto recogida en el fichero su fichero application.properties.

Por otro lado, la interfaz genera una build de desarrollo utilizando el fichero con extensión ".env" que le indica el Dockerfile correspondiente. En este fichero están definidas las URLs de UsersAPI y PricesAPI que durante el desarrollo serán en la propia máquina local.

Por último, cabe mencionar que en este entorno el sistema no cuenta con las bases de datos PostgreSQL, sino que hace uso de bases de datos en memoria H2. Esta decisión se debe a que realmente no es necesario persistir la información para evitar perderla cuando se detiene la ejecución.

2.2.6.2: Pruebas

Para el entorno de pruebas la configuración es realmente parecida a la de desarrollo. De hecho, para las APIs se utilizan los mismos ficheros Dockerfile puesto que la configuración necesaria es exactamente la misma que para ese otro ambiente. Tampoco se utilizarán las bases de datos PostgreSQL por el mismo motivo que en el caso anterior.

Sin embargo, a pesar de ser una configuración similar, requiere de la creación de un fichero Docker Compose propio para poder incluir un contenedor adicional que permitirá realizar pruebas end-to-end con Cypress.

2.2.6.3: Producción

Para el entorno de producción también se utilizará un archivo Docker Compose propio. En este caso sí contiene todo el sistema tal y como se explicó en todos los apartados anteriores, incluyendo las bases de datos de producción para persistir la información.

En este caso se utilizan los ficheros Dockerfile para generar cada contenedor específico con la configuración adecuada.

2.2.6.4: Preproducción

Hasta este momento todos los entornos que se han descrito eran los más habituales en todo tipo de proyectos. Sin embargo, el entorno de preproducción no es tan habitual, sobre todo

en aplicaciones de tamaño reducido como esta. Este entorno se ha considerado de vital importancia para poder realizar los despliegues en producción con la certeza de que todo el sistema funciona según lo esperado. Permitirá probar el sistema en un ambiente totalmente real, permitiendo también asegurar que no hay problemas en el pase a producción causados por la plataforma de despliegue.

Por ejemplo, gracias a esta configuración de producción ha sido posible detectar problemas que solo ocurrían al tener desplegada la aplicación. Un ejemplo es la visualización de los iconos de las criptomonedas, que no eran mostrados en la plataforma por un problema en las rutas. Otro ejemplo es que al desplegar la aplicación ocurría un error al enviar la retroalimentación a los usuarios.

Esto se ha llevado a cabo mediante otro fichero Docker Compose muy similar al de producción. Realmente el único cambio que se puede encontrar es el argumento que recibe el Dockerfile de la interfaz de usuario. Esto permite producir la build adecuada y que la interfaz reciba las URLs correctas de las APIS correspondientes, que son diferentes a las de producción.

2.2.7: Seguridad

Uno de los requisitos fundamentales de este proyecto era la seguridad de la información para evitar posibles riesgos. Esto cobra más sentido aun teniendo en cuenta que el sistema estará desplegado en la nube y podrá ser accesible por cualquier persona.

En los epígrafes siguientes se explicarán las medidas de seguridad que se han implementado en cada una de las APIs que componen la aplicación.

2.2.7.1: UsersAPI

2.2.7.1.1: Endpoints

En relación con la seguridad de los endpoints en la API de usuarios se han establecido una serie de objetivos:

- Solo los usuarios con rol de administrador podrán crear cuentas de usuario.
- Un estudiante o profesor solo podrá acceder a su propia información.
- Un estudiante solo podrá comprar o vender criptomonedas a su propio nombre.
- Un profesor solo podrá evaluar a estudiantes que estén en su lista.

Es importante definirlos porque entonces se comprenderá que no es suficiente con una simple autorización basada en roles, excepto para el caso de los administradores. Esto es debido a que, si solo se comprueba el rol, usuarios con un mismo rol podrían acceder a la información de otros e incluso realizar acciones a su nombre. Por ejemplo, un estudiante podría hacer transacciones a nombre de otro, porque la autorización basada en roles lo permitiría.

Para solucionar esto se ha implementado un componente de Spring que aplica una restricción a la petición indicada. Este componente se llama UserSecurity y tiene un único método que recibe un objeto de tipo Authentication y un String con el nombre de usuario. Se encarga de comprobar que el usuario que está autenticado actualmente en el sistema es el mismo que el sujeto de la acción y, por lo tanto, está autorizado para completarla.

```
@Component("userSecurity")
public class UserSecurity {
    public boolean isUserAuthorized(Authentication authentication, String username) {
        return authentication.getName().equals(username);
    }
}
```

Imagen 21: Componente UserSecurity. Fuente: Elaboración propia.

Esto se puede agregar a las restricciones de autorización de roles mediante el uso de control de acceso basado en expresiones que proporciona Spring. Para ello, se utiliza el método "access" que permite asegurar un endpoint con una expresión. La expresión que recibe este método estará compuesta por una sentencia hasRole() y la utilización del componente UserSecurity.

```
http.authorizeRequests().antMatchers("/login")
    .permitAll();
http.authorizeRequests().antMatchers("/signup")
    .hasRole("ADMIN");
http.authorizeRequests().antMatchers("/teacher/{username}")
    .hasRole("ADMIN");
http.authorizeRequests().antMatchers("teachers/{teacherUsername}/students/{studentUsername}/trades/{id}/feedback")
    .access("hasRole('TEACHER') and @userSecurity.isUserAuthorized(authentication,#teacherUsername)");
http.authorizeRequests().antMatchers("/users/{username}")
    .access("hasAnyRole('STUDENT', 'TEACHER', 'ADMIN') and @userSecurity.isUserAuthorized(authentication,#username)");
http.authorizeRequests().antMatchers("/students/{username}/trade")
    .access("hasRole('STUDENT') and @userSecurity.isUserAuthorized(authentication,#username)");
```

Imagen 22: Autorización de endpoints en UsersAPI. Fuente: Elaboración propia.

Un ejemplo podría ser el siguiente: Un estudiante cuyo nombre de usuario es Sergio utiliza el endpoint "/students/Sergio/trade" para realizar la compra de una nueva criptomoneda. Spring comprobará que el usuario que ha realizado la petición tiene el rol de estudiante y además su nombre de usuario es Sergio. En caso de que otra estudiante llamada Paula utilizara el mismo endpoint, no coincidiría el usuario de la llamada con el de la dirección del endpoint, por lo que el sistema no autorizaría la transacción. Si solo se hubiera utilizado la autorización por roles esta segunda acción si estaría permitida, porque el usuario tenía el rol requerido.

Sin embargo, todavía no se ha explicado cómo se determina quién es el usuario que ha realizado la petición. Para identificarlos, se han utilizado JSON Web Tokens. Estos tokens son cadenas de texto que permiten el traspaso de información en formato JSON de manera segura (Peyrott, 2018). Se han implementado utilizando la librería Java JWT desarrollada por Auth0. Gracias a ella, es posible acceder a determinados métodos que facilitan la implementación de esta medida de seguridad.

El funcionamiento habitual de los JSON Web Tokens suele ser el siguiente: cuando un usuario ha iniciado sesión de forma satisfactoria, la API genera un token que contiene cierta información y está firmado con una clave privada. Responde al usuario con el token, y el resto

de las llamadas que deberán contenerlo en la cabecera de autenticación. Este token será utilizado por la API para saber quién hace cada petición (Morgan, 2017).

Para realizar las acciones relacionadas con los tokens se ha creado una componente de Spring llamado JwtUtil. Este componente contiene las tres operaciones: crear token, verificar token y obtener el usuario de un token.

Los tokens serán creados por el endpoint "/login". En una primera instancia, se comprobará que las credenciales son correctas y, posteriormente, se creará un token utilizando el nombre del usuario. Finalmente se responderá enviando ese token.

Para verificar los tokens en cada llamada a la API, se ha implementado un filtro personalizado llamado JwtFilter. Este filtro se ejecutará una vez por llamada al heredar de la clase OncePerRequestFilter e implementará el método doFilterInternal. En él se comprobará que la cabecera contenga el filtro y siga el esquema Bearer. Posteriormente se verificará el token y, en caso de ser válido se obtendrá la información decodificada. Utilizando esta información es posible extraer el nombre del usuario que se había almacenado en el token original. Este nombre de usuario se utilizará para obtener la información del usuario mediante el método loadByUsername del servicio RepositoryUserDetailsService que implementa la interfaz UserDetailsService. Gracias a este método es posible acceder a la base de datos de UserProfile para obtener el nombre de usuario, la contraseña y los roles, todo ello en un objeto UserDetails. Este mecanismo es realmente común en las implementaciones de seguridad que utilizan Spring Security. En caso de no encontrar el usuario en esa base de datos se lanzará una excepción. Si se encuentra un objeto UserDetails válido, se crea un objeto de autenticación con los detalles y se añade al contexto de seguridad.

La única llamada que no será filtrada por JwtFilter será, precisamente, la que crea el propio token, "/login".

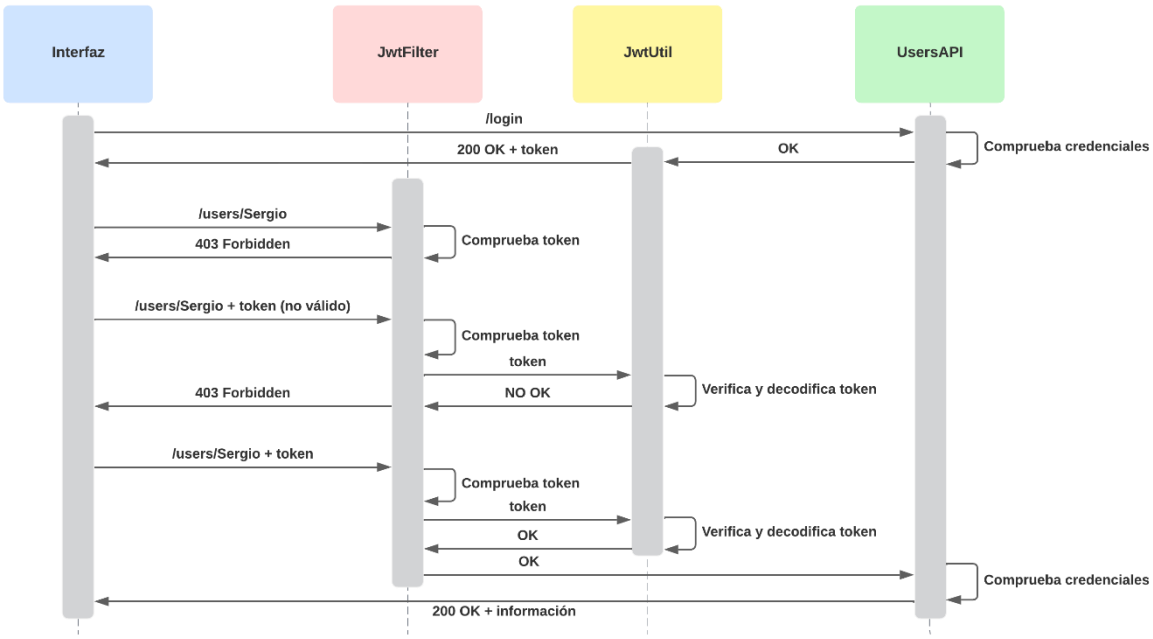


Imagen 23: Diagrama de secuencia de JSON Web Tokens. Fuente: Elaboración propia.

2.2.7.1.2: Base de datos

La seguridad de la información de los usuarios también es crucial. Por ello se han encriptado las contraseñas en base de datos. Gracias a esto, en caso de que un usuario malicioso consiguiera obtener los datos, no podría utilizar las contraseñas para suplantar la identidad de ningún otro usuario. Esta medida se ha implementado utilizando BCryptPasswordEncoder.

2.2.7.2: PricesAPI y ExpressAPI

Ninguna de estas APIs está protegida. Esto se debe principalmente a que los usuarios solo pueden consultar la información y no se dispone de ningún endpoint que modifique realmente la base de datos.

2.2.8: Pruebas

Para comprobar que el software realmente funciona como se requiere, han sido implementadas una serie de pruebas para cada parte principal de la aplicación y también para todo el sistema en conjunto.

2.2.8.1: Pruebas en la interfaz de usuario

Para la interfaz de usuario se ha elegido Vitest como marco de trabajo para la realización de las pruebas. Esta decisión ha supuesto un reto, debido a que esta tecnología se encuentra actualmente en sus primeros años de vida y la cantidad de información disponible al respecto es escasa. En conjunto con Vitest también se ha utilizado React Testing Library para acceder e interactuar con los diferentes componentes del DOM de una forma sencilla y poder así comprobar su funcionamiento.

En este caso, se han desarrollado pruebas unitarias para cada componente personalizado y para todas las funciones auxiliares asociadas a estos.

La cobertura de código en la interfaz de usuario ha sido del 98,28%. Este dato ha sido obtenido gracias a Vitest.

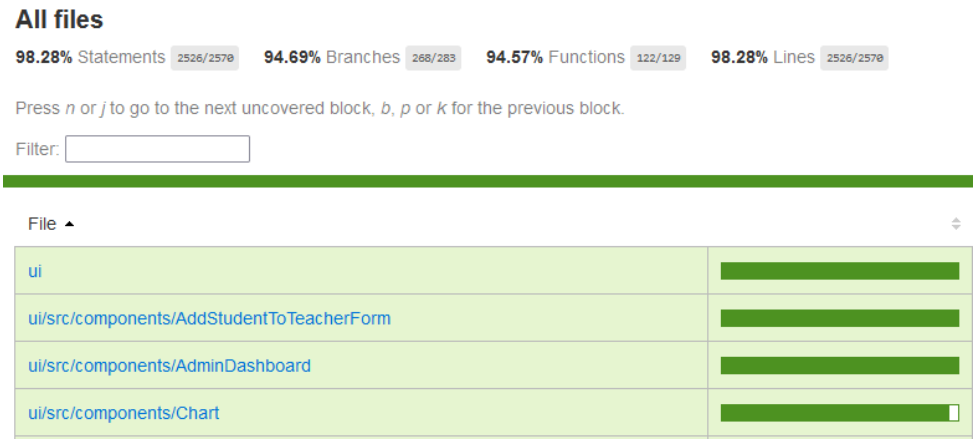


Imagen 24: Cobertura código de la interfaz de usuario. Fuente: Elaboración propia.

Estas pruebas se ejecutan automáticamente, gracias a un workflow creado con Github Actions, cuando se realiza un cambio en cualquier rama o se abre una Pull Request de solicitud de cambios en preproducción o producción.

2.2.8.2: Pruebas en UsersAPI y PricesAPI

Las pruebas para ambas APIs se han dividido en dos tipos, unitarias y de integración.

Las primeras han sido implementadas haciendo uso de JUnit 4 como marco de trabajo principal y de Mockito para simular las dependencias cuando ha sido necesario.

Por otra parte, las de integración han consistido en la realización de pruebas a los endpoints de cada API utilizando la librería Rest Assured.

Para realizar el análisis de cobertura se ha utilizado Java Code Coverge, que genera un fichero HTML con toda la información al respecto cuando se ejecuta el comando "mvn test". En este caso, la cobertura de PricesAPI es del 97,6% mientras que la de UsersAPI asciende al 92,5%.

Imagen 25: Cobertura código de PricesAPI y UsersAPI. Fuente: Elaboración propia.

Estas pruebas unitarias y de integración para ambas APIs se ejecutan bajo las mismas circunstancias que las pruebas de la interfaz de usuario.

2.2.8.3: Pruebas de extremo a extremo (End-to-End)

Las pruebas extremo a extremo son esenciales para comprobar que el sistema funciona correctamente en su conjunto (Schmitt, 2023).

Han sido implementadas utilizando Cypress. La razón principal detrás de la elección de esta librería es su popularidad y la existencia de una imagen Docker que permite utilizar este software en un contenedor independiente. Esta imagen se ha añadido a un fichero de Docker Compose específico para la ejecución de pruebas extremo a extremo. Gracias a este archivo se han podido ejecutar todas las partes del sistema y posteriormente utilizar Cypress para realizar las pruebas. Esta solución ha sido la más sencilla y acertada que se ha encontrado.

Este tipo de pruebas, al requerir gran tiempo de ejecución, solo se llevan a cabo cuando se crea una Pull Request con el objetivo de añadir cambios a los entornos de preproducción y producción.

2.2.9: Despliegue

El despliegue de la aplicación es el proceso a través del cual una software pasa a estar disponible para que sea utilizado por los usuarios finales (VMware, 2023).

Como se detalló en el epígrafe de arquitectura, esta aplicación se divide en dos piezas. El sistema principal, que contiene la funcionalidad troncal, y una API auxiliar que permite obtener los precios sin restricciones. A raíz del problema que se ha explicado anteriormente, resulta indispensable que ambas partes estén desplegadas en plataformas separadas puesto que la API auxiliar debe encontrarse alojada en un servidor europeo y el sistema principal está en un servidor de Estados Unidos.

2.2.9.1: Sistema principal

En este caso se ha elegido Okteto para llevar a cabo los despliegues del sistema Docker Compose, debido a que permite realizarlos de forma gratuita y sin necesidad de agregar una tarjeta de crédito. Esta plataforma realmente no está enfocada a realizar despliegues de aplicaciones como tal, sino que propone un sistema de desarrollo alternativo, creando entornos de desarrollo en la nube de una forma sencilla (Okteto, 2023). Sin embargo, permite generar espacios idénticos a uno de producción, por lo que es una buena alternativa para un proyecto como este.

Se han desplegado en esta plataforma dos entornos de los que se comentaron en un epígrafe anterior, producción y preproducción. Gracias a que Okteto pone a disposición de los desarrolladores una interfaz de línea de comandos, se podrán realizar los despliegues a producción de manera automática mediante Github Actions. En concreto, se han creado dos workflows, uno para cada entorno, aunque ambos funcionan de la misma manera. Se pueden ejecutar de forma manual o, lo más habitual, cuando se realiza un cambio en su rama correspondiente. El primer paso de los workflows es la descarga la interfaz de Okteto para poder ejecutar determinados comandos. Después se inicia sesión utilizando un token de la plataforma. En este caso, al tratarse de un repositorio público, se ha almacenado esa información en un secreto de Github para evitar posibles problemas. A continuación, se elige el espacio donde se desea desplegar la aplicación y finalmente se ejecuta un último comando que completa el despliegue.

```yaml
name: deploy-develop
on:
  push:
    branches: [develop]
  workflow_dispatch:

jobs:
  deploy_preproduction:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Get Okteto CLI
        run: curl https://get.okteto.com -sSfL | sh

      - name: Okteto login
        run: okteto context --token ${{ secrets.OKTETO_TOKEN }} use

      - name: Go to namespace
        run: okteto namespace --personal tfg-preprod-sergiocarrascosasanchez

      - name: Deploy
        run: okteto deploy --file docker-compose.preprod.yaml --wait --build
```

Imagen 26: Workflow de despliegue a preproducción. Fuente: Elaboración propia.

Por último, cabe mencionar que Okteto permite insertar variables de entorno a modo de secretos en el fichero Docker Compose. Esto se ha utilizado para ocultar determinada información sensible como son todo tipo de contraseñas, usuarios y URLs, puesto que el repositorio que contiene el proyecto es público y accesible por cualquier usuario de Github.

2.2.9.2: ExpressAPI

Se eligió Render como plataforma de despliegue. En este caso, es posible desplegar aplicaciones NodeJS sin requerir una tarjeta de crédito y permite al desarrollador elegir el lugar geográfico en el que se encuentra el servidor.

No se han creado workflows ni acciones programadas para generar los despliegues. En su lugar se han utilizado las ventajas de la propia plataforma, que detecta automáticamente cada cambio producido en el repositorio de Github y lo despliega.

Capítulo III: Conclusiones

3.1: Principales logros alcanzados y conclusiones

Si se comparan los requisitos definidos al principio del documento con el resultado final obtenido tras la implementación se puede concluir que todos han sido satisfactoriamente cumplidos.

Este proyecto fue comenzado en septiembre de 2022 y terminado en mayo de 2023. A lo largo de estos ocho meses he conseguido aumentar mi nivel de programación en todas las áreas del desarrollo web de manera exponencial. Esto se debe a la cantidad de retos a los que me he enfrentado para conseguir cumplir todas las metas que me propuse en un primer momento.

En primer lugar, React. Desarrollar el Frontend de la aplicación utilizando esta librería fue uno de los principales retos de este proyecto. A lo largo del doble grado que estoy cursando no se ha estudiado desarrollo Frontend más allá del uso básico de HTML y CSS explicado de manera superficial. Personalmente, el diseño siempre ha sido algo que ha llamado mi atención, por lo que quería que fuera una pieza importante en este proyecto. Para conseguirlo, decidí utilizar una tecnología moderna y actual que fuera demandada en el mercado. Al no tener ningún conocimiento de marcos de trabajo ni librerías de JavaScript, decidí decantarme por React. Todo lo aprendido ha sido de forma totalmente autodidacta, lo cual ha supuesto un reto mayúsculo en multitud de aspectos. Además, ha provocado que en ocasiones haya tenido que refactorizar código que escribí al comienzo. Por ejemplo, el inicio de sesión. Al no tener grandes conocimientos, no utilicé renderizado condicional empleando ternarias, si no que empleé sentencias IF-ELSE. Además, escribí las llamadas a la API en el mismo archivo que el renderizado del componente. El resultado funcionaba, pero era completamente ilegible y poco mantenible. Por ello, cuando avancé con el proyecto y aprendí esos conceptos decidí refactorizar el código y mejorarlo para que estuviese en sintonía con el resto de la aplicación. Otro aspecto de React que tuve que aprender y que ha supuesto un gran reto es la realización de pruebas. De nuevo por no tener experiencia previa en este sentido. Además, decidí utilizar Vitest, que es un marco de trabajo de muy reciente creación y del que hay poca información más allá de la documentación oficial. Además, esta documentación en ocasiones me ha resultado complicada de entender debido a la brevedad de las explicaciones y poca variedad de ejemplos. A pesar de todo, React es una tecnología que me ha gustado especialmente utilizar. Considero que la libertad que ofrece al programador a la hora de desarrollar mejora la experiencia de una manera increíble. De hecho, fue mi parte favorita.

Docker también era una tecnología que, aunque la había utilizado en algún momento puntual, nunca había profundizado en ella. Realmente creía que tenía una complejidad muy elevada y, además, yo no encontraba realmente su utilidad. Sin embargo, me ha resultado mucho más sencilla de lo esperado y realmente útil. De hecho, al tener una aplicación dividida en pequeñas partes, resultaba conveniente poder arrancar todo el sistema con un único comando en apenas unos segundos. También ha sido imprescindible para la ejecución de pruebas y despliegues con Github Actions.

El desarrollo de las API en Spring fue relativamente diferente al conocer previamente la tecnología que he utilizado. Esto se debe a que en la universidad sí se ha profundizado en

ella. Incluso en el apartado de pruebas. Aun así, siempre surgen obstáculos que se deben superar a lo largo del camino y temas que se decidieron desarrollar en una profundidad mayor. Un apartado que he tenido que aprender para la realización de este proyecto ha sido la seguridad de las APIs en Spring, la cual desconocía por completo.

3.2: Trabajos futuros

A pesar de haber cumplido todos los objetivos propuestos siempre hay espacio para una mejora.

Uno de los principales puntos que considero que se puede mejorar es en la representación de los gráficos. No porque sean de mala calidad o insuficientes para el propósito de un proyecto de este tamaño. Realmente la razón subyace en el tipo de gráfico. Para la inversión se pueden utilizar diferentes representaciones de los precios que tienen los activos. Uno de los más habituales es el gráfico de línea que se ha utilizado en este proyecto. Otro de los más comunes es el de velas japonesas. Este segundo gráfico aporta una cantidad de información muy superior al primero y permite tomar mejores decisiones de inversión. En concreto, muestra una vela por cada unidad de tiempo. Las velas señalan el precio del activo en el momento inicial y en el final, además de indicar cómo ha fluctuado en ese periodo.

Imagen 27: Gráfico de velas japonesas de BTCUSD. Fuente: Trading View.

De hecho, el motivo fundamental por el que finalmente se utilizó el gráfico de línea es la obtención de la información. Al tener que utilizar servicios de terceros, la mayoría tenían un coste muy elevado o resultaban insuficientes para los requisitos del proyecto, incluso para la frecuencia de refresco de los gráficos de línea. La única API que ofrecía unas características aceptables era la de Binance. Además, era un proveedor fiable y reconocido. Este servicio sí que ofrecía la posibilidad de obtener la información para poder representar gráficos de velas japonesas, pero las limitaciones de la API hacían que resultara imposible cumplir el resto de los requisitos asociados a los gráficos. Por ello, una posible mejora en este sentido sería realizar una inversión monetaria para obtener acceso a un servicio que proporcione una información más completa y permita cumplir los requisitos al mismo tiempo.

Otra mejora relativa a los gráficos está relacionada con la posibilidad de dibujar y trazar líneas en ellos. Esto ayuda a analizar los patrones identificados por los alumnos de una forma

más sencilla. Sin embargo, esta funcionalidad no es trivial de implementar y se decidió no incluirla porque excedía con creces los objetivos de un proyecto de esta dimensión.

Otro punto que se debe mejorar es la posibilidad de editar algunos elementos, como las justificaciones creadas por los estudiantes o los comentarios de retroalimentación. Este apartado se ha decidido no implementar por falta de tiempo. Además, se ha considerado que era prioritario crear un Producto Mínimo Viable completo que recogiera toda la funcionalidad principal de la aplicación en lugar de dedicar parte del tiempo a una tarea secundaria como es su modificación. Esto es debido a que estas acciones suponen la creación de numerosos endpoints, formularios y hooks, además de todos los tests asociados a ellos. A cambio, se ganaría una funcionalidad accesoria que no resulta clave para resolver el problema a tratar.

Bibliografía

Baeldung. (26 de Mayo de 2023). *Hibernate Inheritance Mapping*. Obtenido de https://www.baeldung.com/hibernate-inheritance

Binance. (2023). Obtenido de https://www.binance.com/en/terms

Docker. (2023). *Docker Compose overview*. Obtenido de https://docs.docker.com/compose/

Morgan, Á. J. (25 de Septiembre de 2017). *Securizar un API REST utilizando JSON Web Tokens*. Obtenido de https://www.adictosaltrabajo.com/2017/09/25/securizar-un-api-rest-utilizando-json-web-tokens/

Okteto. (2023). *Welcome to Okteto!* Obtenido de https://www.okteto.com/docs/

Peyrott, S. (2018). *JWT Handbook.*

React. (2023). *Built-in React Hooks*. Obtenido de https://react.dev/reference/react

React. (2023). *Passing Data Deeply with Context*. Obtenido de https://react.dev/learn/passing-data-deeply-with-context

React. (2023). *Your First Component*. Obtenido de https://react.dev/learn/your-first-component

Santander. (2020). *Metodologías de desarrollo de software: ¿qué son?* . Obtenido de https://www.becas-santander.com/es/blog/metodologias-desarrollo-software.html

Schmitt, J. (6 de Abril de 2023). *What is end-to-end testing?* Obtenido de https://circleci.com/blog/what-is-end-to-end-testing/

Simplilearn. (13 de Febrero de 2023). Obtenido de https://www.simplilearn.com/tutorials/jpa-tutorial/spring-boot-jpa

Spring Framework. (2023). *Accessing Data with JPA*. Recuperado el 2023, de https://spring.io/guides/gs/accessing-data-jpa/

Universidad de Granada. (2010). Obtenido de https://elvex.ugr.es/idbis/db/docs/design/2-requirements.pdf

Vite. (2023). *Getting Started*. Obtenido de https://vitejs.dev/guide/

VMware. (2023). *What is Application Deployment?* Obtenido de https://www.vmware.com/topics/glossary/content/application-deployment.html

Anexo I: Pantallas de la interfaz gráfica

En esta sección se recogen todas las páginas de la interfaz gráfica de usuario que ha sido desarrollada utilizando React.

Imagen 28: Pantalla principal. Fuente: Elaboración propia.

Imagen 29: Formulario de inicio. Fuente: Elaboración propia.

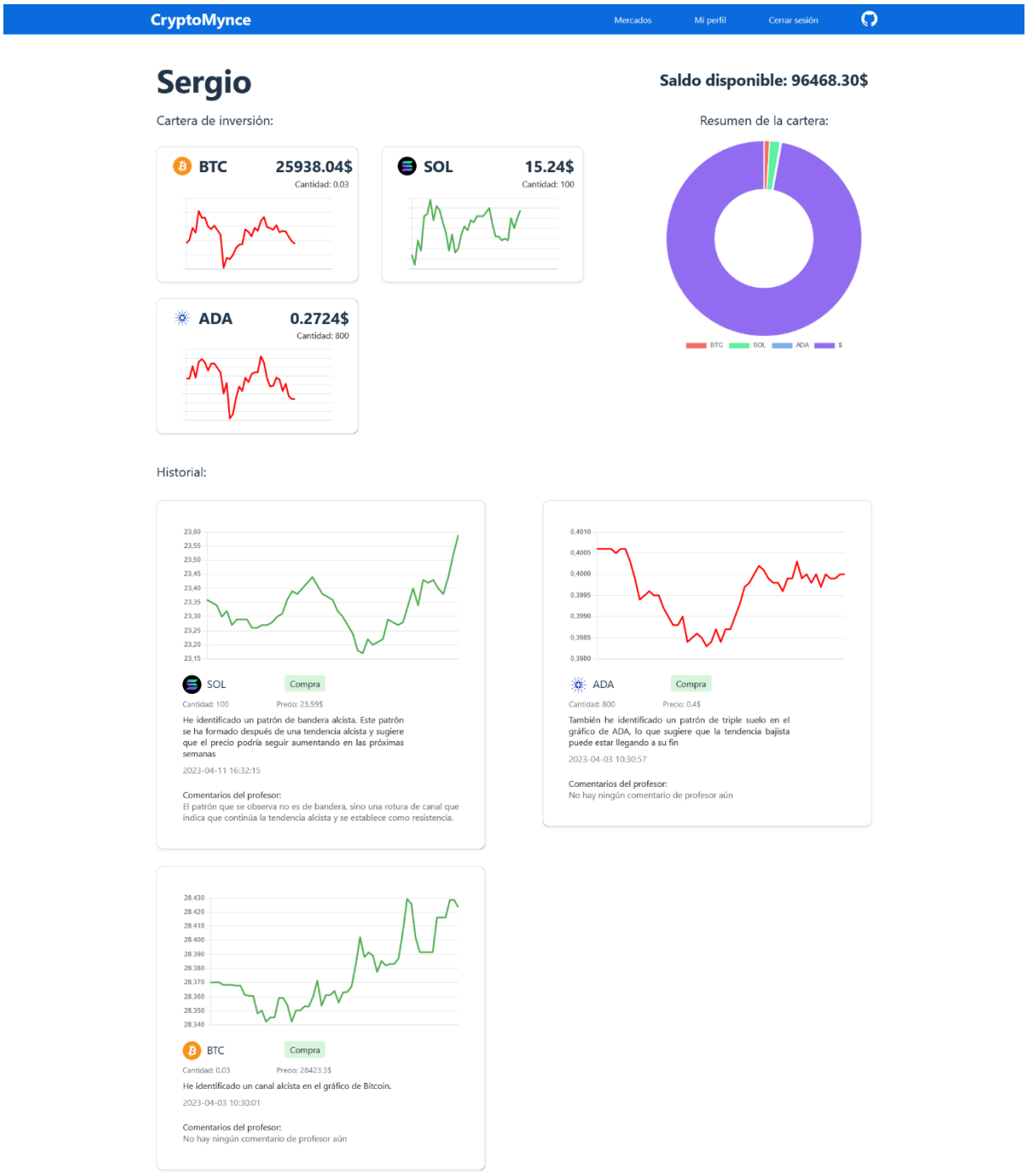


Imagen 30: Página de un estudiante. Fuente: Elaboración propia.

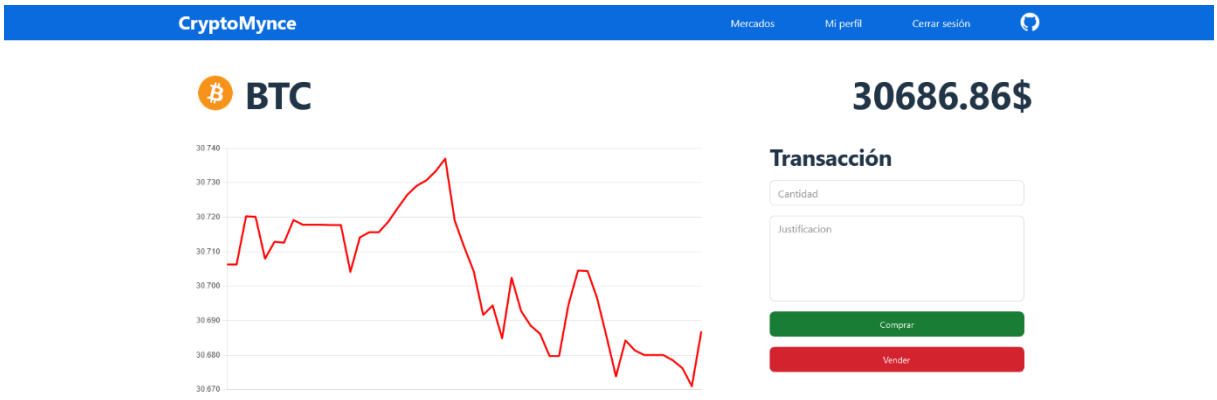


Imagen 31: Página de una criptomoneda en detalle. Fuente: Elaboración propia.

Imagen 32: Página de mercado. Fuente: Elaboración propia.

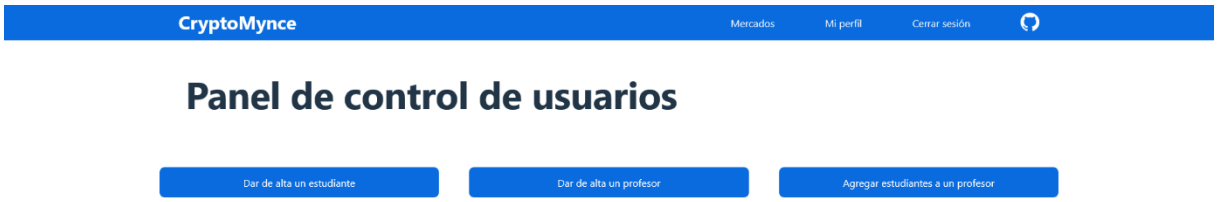


Imagen 33: Panel de control para el administrador. Fuente: Elaboración propia.

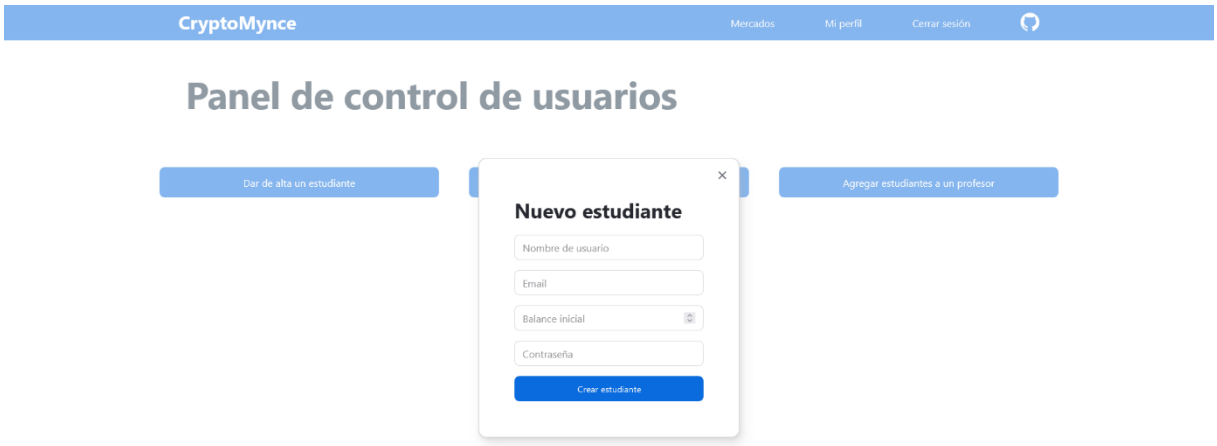


Imagen 34: Formulario para crear un nuevo estudiante. Fuente: Elaboración propia.

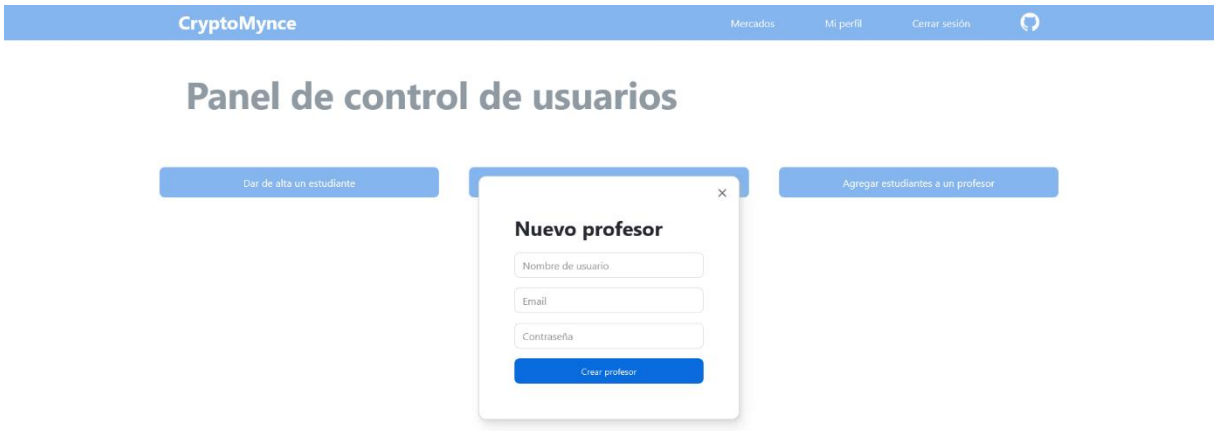


Imagen 35: Formulario para crear un nuevo profesor. Fuente: Elaboración propia.

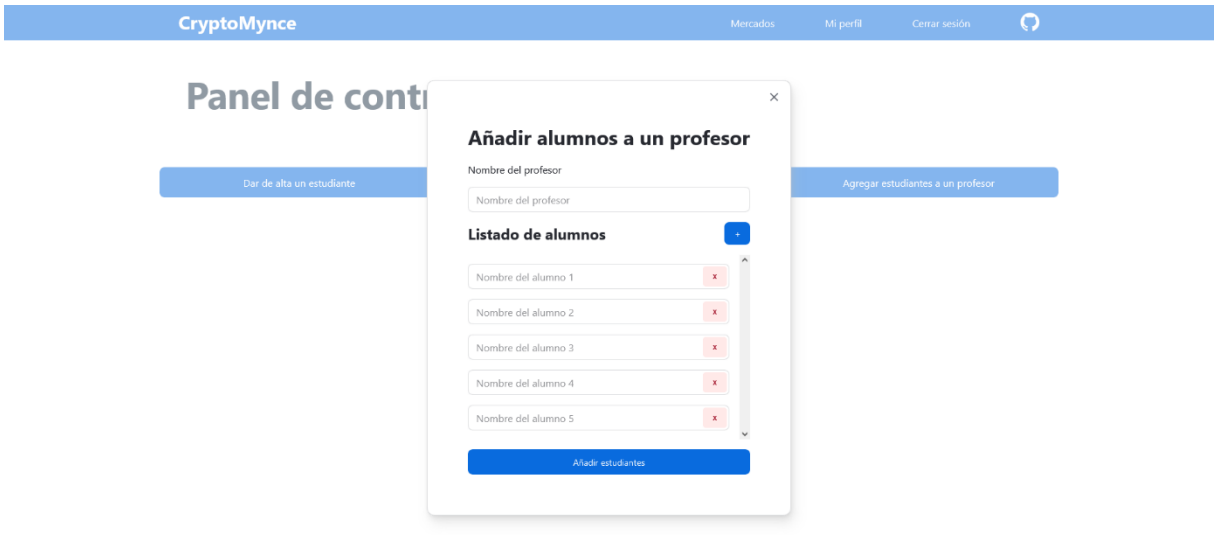


Imagen 36: Formulario para asignar estudiantes a un profesor.Fuente: Elaboración propia.

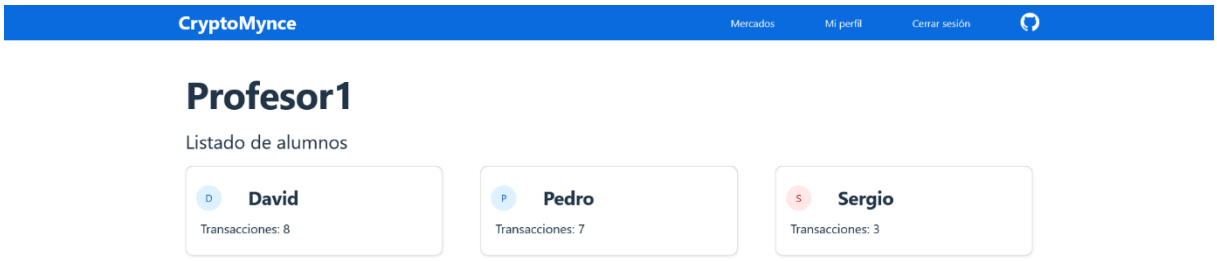


Imagen 37: Página del profesor. Fuente: Elaboración propia.

Imagen 38: Transacciones realizadas por un estudiante desde el panel del profesor. Fuente: Elaboración propia.