UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

SISTEMA DE VALIDACIÓN DE ACCESO A EVENTOS MEDIANTE NFTS

DAVID MOLINA MESA

Director/a
CARME QUER BOSOR (Departamento de Ingenieria de Servicios y Sistemas de Información)

Titulación
Grado en Ingeniería Informática (Ingeniería del Software)

Memoria del trabajo de fin de grado

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

28/06/2024

Agradecimientos

En primer lugar, agradecer a la profesora Carme Quer por haber aceptado la propuesta de este proyecto y haber llevado un seguimiento del mismo durante todas las etapas de este.

En segundo lugar, a mis compañeros del equipo de Mobile de Basetis por los conocimientos aprendidos en materia de desarrollo Android que, aunque se trate de un proyecto de React Native, he podido aplicar en este proyecto.

Por último, quisiera agradecer a familiares y amigos por el apoyo brindado durante este último cuatrimestre. Especialmente a mi pareja Lola por ser mi sujeto de pruebas de las aplicaciones, a Marina por ofrecerse a dejarme un teléfono Android con NFC y a Pablo y Dani por ayudarme a fondear billeteras de criptomonedas con *tokens* de *testnets*.

Resumen

La industria de los eventos sufre de actividades fraudulentas como el fraude de entradas y la reventa especulativa. Estas prácticas aparecen especialmente en eventos muy demandados y se saldan con cientos e incluso miles de afectados. No solo afecta a los asistentes, sino también a los propios organizadores y actuantes de los eventos.

La motivación de este proyecto es crear un sistema de validación de acceso basado en *blockchain* utilizando NFTs. El resultado es el desarrollo de dos aplicaciones móviles: una para los asistentes y otra para los validadores, que juntas simulan un caso de uso real y proporcionan una solución innovadora para abordar los problemas mencionados.

En este proyecto, de carácter académico, se desarrollan las aplicaciones móviles, un *smartcontract* para la *blockchain* de Ethereum y una API alojada en un servidor *backend.* A lo largo de este documento, se detallan las fases como si se tratase de un desarrollo real.

Resum

La indústria dels esdeveniments pateix d'activitats fraudulentes com el frau d'entrades i la revenda especulativa. Aquestes pràctiques apareixen especialment en esdeveniments molt demandats i es salden amb centenars i fins i tot milers d'afectats. No només afecta als assistents, sinó també als propis organitzadors i actuants dels esdeveniments.

La motivació d'aquest projecte és crear un sistema de validació d'accés basat en *blockchain* utilitzant NFTs. El resultat és el desenvolupament de dues aplicacions mòbils: una per als assistents i una altra per als validadors, que juntes simulen un cas d'ús real i proporcionen una solució innovadora per a abordar els problemes esmentats.

En aquest projecte, de caràcter acadèmic, es desenvolupen les aplicacions mòbils, un *smartcontract* per a la *blockchain* de Ethereum i una API allotjada en un servidor *backend*. Al llarg d'aquest document, es detallen les fases com si es tractés d'un desenvolupament real.

Abstract

The event's industry suffers from fraudulent activities such as ticket fraud and speculative reselling. These practices appear especially in high-demand events and result in hundreds and even thousands of people affected. It not only impacts the attendees, but also the organizers and performers of the events themselves.

The motivation of this project is to create a blockchain-based access validation system using NFTs. The result is the development of two mobile applications: one for attendees and one for validators, which together simulate a real use case and provide an innovative solution to address the mentioned problems.

In this academic project, mobile applications, a smartcontract for the Ethereum blockchain and an API hosted on a backend server are developed. Throughout this document, the phases are detailed as if it were a real development.

Tabla de contenidos

1. Introducción

Este proyecto es el resultado de un Trabajo de Final de Grado (TFG) en Ingeniería Informática, orientado a la especialización en Ingeniería del Software, que se ha llevado a cabo en la Facultat d'Informàtica de Barcelona, integrante de la Universitat Politècnica de Catalunya. Ha sido dirigido por la profesora Carme Quer y propuesto por el estudiante con el objetivo de profundizar y enriquecer su conocimiento en el desarrollo *blockchain*.

1.1 Contexto del proyecto

En la última década, la industria de eventos ha experimentado un crecimiento exponencial al punto que ha ido diversificándose en varios formatos como pueden ser los conciertos y festivales de música, eventos deportivos, conferencias, exposiciones culturales y parques temáticos, entre otros. Sin embargo, el auge de esta industria ha traído consigo una serie de desafíos que actualmente están por resolver. De entre estos desafíos destacan dos prácticas muy comunes: la primera, el fraude de entradas y la segunda, la reventa especulativa.

El fraude de entradas es una de las prácticas más extendidas. Este sucede cuando un estafador vende entradas falsas o inexistentes a un comprador, el cual más tarde descubre que ha sido estafado al canjear su entrada intentando acceder al evento. Este fraude se presenta en distintas modalidades como pueden ser la venta de entradas falsificadas, la venta de entradas duplicadas o la venta de entradas no entregadas (al comprador).

Por otro lado, la reventa especulativa, se trata de un fenómeno donde un comprador adquiere inicialmente las entradas para después venderlas a un precio superior al original. Esto sucede especialmente en eventos muy demandados como pueden ser conciertos de musicales o eventos deportivos. Un ejemplo de este fenómeno ocurrió en mayo del 2023, cuando el grupo *Coldplay* actuaba en el Palau Sant Jordi de Barcelona. El rápido *sold-out* del grupo musical provocó que muchas personas se quedaran sin entrada y tuvieran que recurrir a la reventa.

Cabe destacar, que también ha llegado a ocurrir reventa especulativa de entradas que son falsas, de modo que, en ciertas ocasiones, ambas prácticas pueden ir de la mano. Es por ello que el caso expuesto anteriormente resultó en que, más adelante, los *Mossos d'Esquadra* recibieron más de 50 denuncias y detectaron a más de 300 estafados [1].

Estas prácticas tienen un impacto muy negativo, puesto que no solo afectan a los propios consumidores, sino también a los organizadores de los eventos, a los actuantes, a las plataformas de venta de entradas e incluso al mercado secundario de revendedores.

Durante los últimos años, el avance de la tecnología *blockchain* y la llegada de los NFT ha traído consigo nuevas soluciones para abordar parte de los desafíos de la industria de los eventos. Es por ello, que, en este TFG, se propone y desarrolla una solución basada en tecnología *blockchain*. Esta

solución consiste en un sistema de validación de las entradas a eventos a través de NFT gobernados por un *smartcontract* con el fin de regular la emisión, transferencia y validación de las entradas. El sistema desarrollado consiste tanto en la aplicación móvil de los asistentes como en la de los validadores para así simular un caso de uso real.

1.2 Términos y conceptos

En esta sección se define conceptos que se utilizan a partir de ahora en esta memoria. Pese a que la tecnología *blockchain* empezó a ser factible hace más de una década, el uso de esta tecnología no ha sido tan popular hasta hace unos años y por ende no es muy conocida ni entendida. Además de ello, se explican otros conceptos claves para la realización del proyecto.

Blockchain

Cuando se habla de tecnología *blockchain* se habla de un tipo de bases de datos, también se conoce como *ledger* digital descentralizado, que sirve específicamente para registrar de forma segura transacciones. Los datos de la *blockchain* se organizan en bloques ordenados cronológicamente y protegidos por *hashes* criptográficos [2].

Smartcontract

Un *smart contract* es un programa almacenado en la *blockchain* con el que se puede interactuar. Este programa una vez desplegado en la *blockchain* no se puede modificar, se ejecuta automáticamente y se comporta tal y como el código lo describe. Se le llama contrato por el símil que presenta respecto a un contrato ordinario [3]. En este proyecto se desarrolla uno para la gestión de los eventos y sus entradas.

Decentralized Application (DApp)

Una DApp es una aplicación que opera en una red descentralizada *blockchain*. A diferencia de las aplicaciones tradicionales que operan en un servidor o red de servidores controlados por una entidad, estas lo hacen en una red distribuida de computadores. Hacen uso de los *smartcontracts* [4].

Non-Fungible Token (NFT)

Un NFT es un identificador único digital que se almacena en una *blockchain*. Este mismo puede guardar información como pueden ser documentos digitales (imágenes, video, audio, etc.). Se dice que es no-fungible, ya que no puede ser copiado, substituido o subdivido por otro (esto último es posible en el caso de las criptomonedas). Un NFT tiene un propietario, el cual puede transferirlo [5].

Ethereum

Ethereum es una *blockchain* descentralizada que nació en 2015 y permite desplegar y ejecutar Dapps y *smartcontracts*. Estos se ejecutan en la *Ethereum Virtual Machine* (EVM). La criptomoneda

nativa de esta red es el Ether (ETH) la cual sirve como “gasolina” para poder realizar transacciones y ejecutar Dapps en la red [6]. Se necesita utilizar su *blockchain* en el proyecto para desplegar la Dapp que se ha desarrollado y programar un *smartcontract* en Solidity (su lenguaje de programación).

Billetera de criptomonedas

Una billetera de criptomonedas es un software o hardware diseñado para almacenar claves privadas y públicas de los usuarios. Estas permiten enviar, recibir y monitorear criptomonedas o NFTs. Es necesario una billetera para que los usuarios puedan firmar las transacciones que realicen en la aplicación desarrollada [7].

Web3

Web3, abreviatura de Web 3.0, representa la próxima evolución de Internet, centrada en la descentralización, la privacidad del usuario y la interoperabilidad. A diferencia de las fases anteriores de la web (Web 1.0, que era principalmente de solo lectura, y Web 2.0, caracterizada por la interactividad y las redes sociales), Web3 utiliza tecnologías *blockchain*, *smartcontracts* y criptomonedas para crear una Internet más abierta [8]. Atendiendo a las características de este proyecto, este se considera un proyecto Web3.

Código *Quick Response* (QR)

Un código QR es un tipo de código de barras bidimensional que se puede escanear usando la cámara de un teléfono inteligente u otro dispositivo especializado. Se utilizan comúnmente para almacenar URLs o enlaces, información de contacto, o textos, y son útiles para acceder rápidamente a información en línea o transferir datos de manera sencilla entre dispositivos. En el contexto de este proyecto es útil para controlar el acceso de los asistentes [9].

Near-field communication (NFC)

Esta tecnología permite establecer una comunicación inalámbrica de corto alcance entre dispositivos. Los estándares NFC están basados en RFID y FeliCa. Hoy en día esta tecnología se utiliza sobre todo en dispositivos móviles y *wearables* para realizar pagos, controles de acceso y transferencias de datos, entre otros [10]. De cara a este proyecto, concretamente, se utiliza NFC Tipo 4, ya que es el que tiene mejores especificaciones en cuanto a velocidad y memoria. En el contexto del proyecto, NFC sirve para gestionar el control de acceso de los asistentes en la aplicación de los validadores.

Host-based Card Emulation (HCE)

Es una tecnología que permite que los dispositivos móviles actúen como tarjetas inteligentes sin necesidad de un hardware específico, emulando su comportamiento (como por ejemplo el de una

tarjeta de crédito). En lugar de almacenar datos en un componente físico, la información de la tarjeta se almacena y se gestiona a través del *software* del dispositivo móvil [11]. Para el contexto de este proyecto, se utiliza para emular una tarjeta capaz de ser escaneada mediante NFC. Se utilizará en la aplicación de los asistentes para que la entrada pueda ser leída a través de NFC en la aplicación de los validadores.

1.3 Objetivo y alcance del proyecto

En esta sección se describe primeramente el objetivo y subobjetivos que se pretenden alcanzar en el proyecto. Después, en la figura 1.1, se detallan los componentes que formaran parte de la arquitectura del sistema objetivo de este proyecto. Finalmente, se detalla el alcance del proyecto.

Desarrollar los componentes software necesarios para disponer de un sistema de validación de acceso de entradas a eventos mediante NFTs que evite malas prácticas como el fraude de entrada y la reventa especulativa de entradas.

Este objetivo se descompone en los subobjetivos siguientes:

1. **Desarrollar una aplicación móvil para los asistentes a eventos:** Permitir a los usuarios asistentes adquirir, almacenar, gestionar y validar sus entradas para eventos de manera segura y eficiente.
    a. Proporcionar una interfaz de usuario amigable e intuitiva para así facilitar que el usuario pueda interactuar cómodamente con sus NFTs siguiendo recomendaciones de diseño UX/UI.
    b. Ofrecer diferentes métodos de validación de las entradas. En este caso mediante un código QR o bien usando la tecnología NFC del dispositivo (es decir, acercándose a un receptor para validar la entrada inalámbricamente).

2. **Desarrollar una aplicación móvil para los validadores de eventos:** Permitir a los usuarios validadores de entradas realizar su trabajo de manera eficiente.
    a. Ofrecer capacidades de escaneo de códigos QR o tecnología NFC para validar las entradas.

3. **Desarrollar un *smartcontract* para la emisión y transferencia de entradas:** Asegurar la autenticidad, unicidad, y transparencia de las entradas mediante el uso de *smartcontracts* en la *blockchain* de Ethereum.
    a. Diseñar *smartcontracts* que permitan la transferencia segura entre usuarios y la validación en el acceso.

4. **Crear una API REST y desplegarla en un servidor *backend*:** La idea de esta API REST es ser utilizada en la aplicación de los validadores para evitar tener que firmar las transacciones a través de una aplicación de billetera cada vez que se realice una validación.

En la figura 1.1, se puede observar los diferentes componentes a desarrollar para el sistema, además de los actores principales y aplicaciones auxiliares. En la parte izquierda de la figura, se puede observar que los asistentes hacen uso de la aplicación de los asistentes, además de las billeteras de criptomonedas. Cabe destacar que estas últimas no forman parte del desarrollo de este proyecto, pero son necesarias en ciertos momentos para poder realizar algunas operaciones necesarias para la funcionalidad de la aplicación de los asistentes.

Después, como eje central del sistema se encuentra el *smartcontract* el cual se encarga de la gestión de los eventos y la gestión de las entradas.

Por otro lado, en la parte derecha se muestra la aplicación de los validadores. Esta es usada por los validadores en los eventos. A diferencia de la aplicación de los asistentes, esta se conecta a una API REST que se desarrolla con el fin de realizar transacciones con el *smartcontract* sin necesidad de utilizar aplicaciones de billeteras.

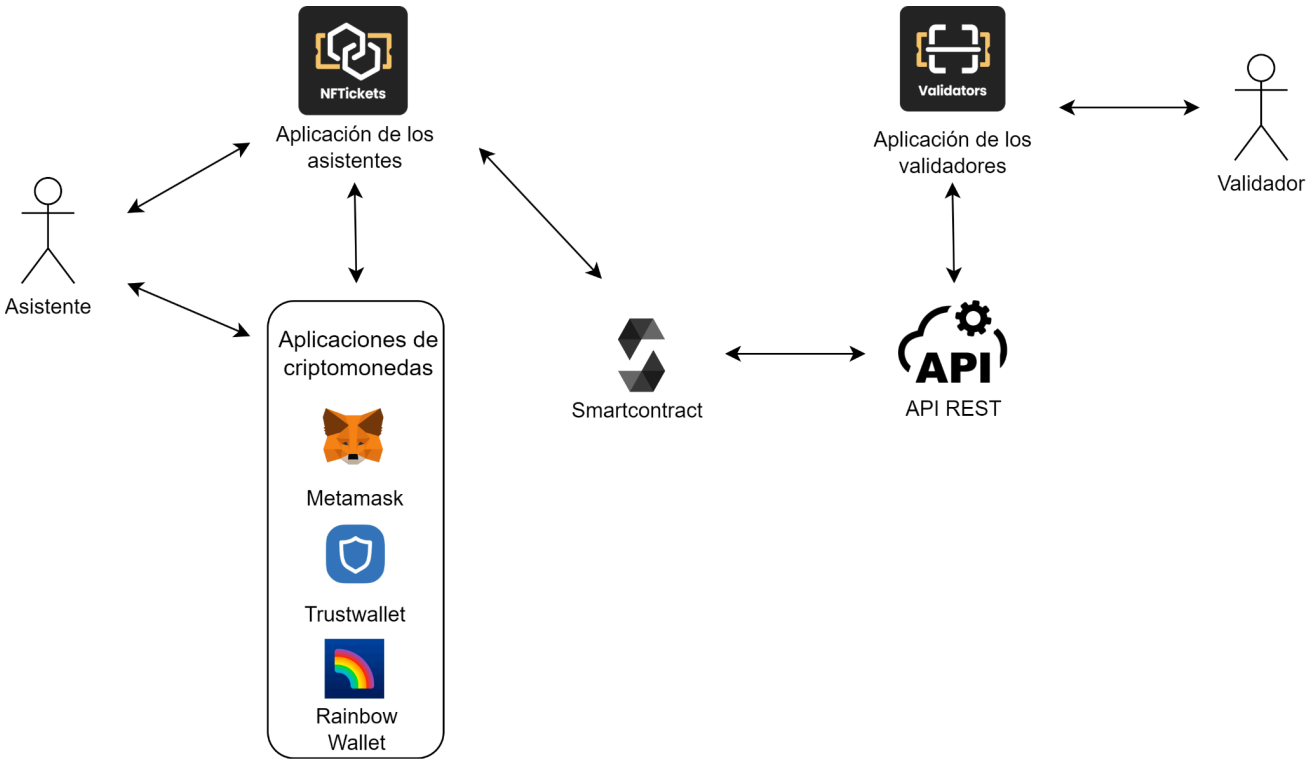


Figura 1.1: Diagrama del los elementos del proyecto. Fuente: Elaboración propia

1.3.1 *Out-lists*

A continuación, para terminar de aclarar qué es lo que se va a desarrollar en este sistema, se muestran las siguientes tablas. Estas consisten en unas *out-lists*, donde se definen los objetivos que se pretenden alcanzar (*In*), aquellos que quedan fuera del alcance (*Out*) y los que quizás podrían incluirse (*May be*). En la tabla 1.1 se define la *out-list* de desarrollos, en la tabla 1.2 la *out-list* de la aplicación de validadores y finalmente en la tabla 1.3 la *out-list* de la aplicación de los asistentes.

Desarrollos

In	***Out***	***May be***
Desarrollar una aplicación para los asistentes	-	-
Desarrollar una aplicación para los validadores	-	-
-	-	Desarrollar una aplicación para los organizadores
-	Desarrollar una aplicación de billetera de criptomonedas	-

Tabla 1.1: *Out-list* de desarrollos. Fuente: Elaboración propia

Aplicación de los validadores

In	***Out***	***May be***
-	Permitir la autenticación	-
Permitir seleccionar un evento entre los disponibles para validar sus entradas	-	-
Permitir validar las entradas mediante escaneo de código QR	-	-
Permitir validar las entradas mediante escaneo NFC	-	-
Permitir cambiar el método de validación en cualquier momento	-	-
-	Permitir visualizar un registro de las entradas validadas	-
-	Permitir gestionar su jornada laboral	-

Tabla 1.2: *Out-list* de la aplicación de asistentes. Fuente: Elaboración propia

Aplicación de los asistentes

In	*Out*	*May be*
-	Permitir la creación de perfiles y cuentas de usuario	-
-	Permitir autenticarse mediante usuario-contraseña o servicios de autenticación como los de Google o Microsoft	-
Permitir conectarse mediante diferentes aplicaciones de billeteras de criptomonedas (Metamask, Truswallet, Rainbow, etc)	-	-
Permitir visualizar los eventos disponibles	-	-
Permitir buscar por nombre entre los eventos disponibles	-	-
-	-	Permitir realizar búsqueda con filtros avanzados sobre los eventos
Permitir adquirir entradas de los eventos	-	-
-	Permitir seleccionar sectores donde ubicarse del evento (sala, butaca, sector del escenario, etc)	-
Permitir visualizar sus entradas adquiridas	-	-
Permitir buscar por nombre entre las entradas adquiridas	-	-
-	-	Permitir realizar búsqueda con filtros avanzados
-	-	Permitir vender sus entradas
Permitir transferir sus entradas a otras billeteras	-	-
Permitir validar sus entradas mediante código QR	-	-
Permitir validar sus entradas mediante NFC	-	-
Permitir visualizar datos sobre la red a la que están conectados, su saldo y la billetera conectada.	-	-

Tabla 1.3: *Out-list* de la aplicación de validadores. Fuente: Elaboración propia

1.4 Partes interesadas

A continuación, en este apartado, se enumeran las principales partes interesadas en este proyecto.

Desarrollador del proyecto

El desarrollador será el encargado de diseñar y desarrollar el sistema que se ha planteado previamente. Se encargará de la evolución de este durante todo el proceso de desarrollo. Puesto que el proyecto se trata de un TFG, a este rol le repercute en su evaluación académica.

Directora del proyecto

La directora de este proyecto es Carme Quer. Su rol consiste en dar soporte al desarrollador del proyecto durante su diseño y desarrollo con el fin de conseguir éxito en este trabajo.

Organizadores de eventos

Entre ellos se incluyen promotores, productores y las propias organizaciones de eventos (entre los que se incluyen el equipo que válida las entradas en las inmediaciones del evento). Se pueden beneficiar de tener un sistema seguro y transparente para la emisión y gestión de las entradas. Además, les puede permitir combatir el fraude, mejorando la experiencia del comprador y muy probablemente su rentabilidad.

Asistentes a los eventos

Este grupo es clave entre las diferentes partes interesadas, ya que la solución está diseñada principalmente para la mejora de su experiencia. La facilidad de uso, la seguridad y la confiabilidad del sistema son fundamentales para satisfacer sus necesidades.

Plataformas de venta de entradas de terceros

Estas se verán especialmente afectadas porque ahora los usuarios con intención de venderlas simplemente no podrán hacerlo si el *smartcontract* que lo regula no permite su transferencia.

Actuantes de los eventos

Los artistas, músicos, deportistas y demás protagonistas de los eventos están interesados en que no haya conflictos en la venta de entradas y acceso a sus espectáculos. Un sistema eficiente para la gestión de entradas puede tener un impacto positivo en su relación con los fans e imagen de marca. Además de que se puede evitar que otros sujetos se lucren a costa de su actividad profesional.

Autoridades gubernamentales

Las autoridades tienen interés en asegurar prácticas comerciales justas, transparentes y seguras para los consumidores. Podrían influir en la adopción de tal sistema a través de regulaciones.

1.5 Estado del arte

En esta sección se presentan las alternativas que existen en el mercado a la propuesta, el análisis de sistemas similares y finalmente, la justificación de la necesidad del desarrollo de este proyecto.

1.5.1 Análisis de alternativas existentes

Las prácticas que se intenta evitar mediante el producto desarrollado en este TFG ya han sido atacadas con otras herramientas existentes en el mercado anteriormente. Estas soluciones alternativas a la propuesta en este proyecto aborda el problema sin hacer uso de la tecnología *blockchain*, es decir, son soluciones con herramientas tradicionales. En este apartado se destaca cuáles son y se explica sus debilidades y fortalezas.

- **Sistemas de entradas nominativas:** Una entrada nominativa es aquella que está asociada a la identidad del comprador. De esta manera se impide la reventa no autorizada. En este caso, él comprador debe acudir al evento con su identificación oficial. Para este caso, a diferencia de la propuesta de este proyecto, la posibilidad de transferir la entrada queda totalmente anulada.
- **Sistemas limitadores de compra de entradas:** Algunos organizadores de eventos optan por esta medida que previene de la acumulación de entradas. Sin embargo, estos sistemas son fáciles de sortear utilizando múltiples cuentas.
- **Plataformas de reventa autorizadas:** Esta alternativa se sustenta gracias a que algunos organizadores colaboran con plataformas de reventa. En ellas se permite la reventa de entradas a precios controlados.
- **Programas de fidelidad y membresías:** Esta opción consiste en ofrecer acceso prioritario a la compra de entradas a sus miembros. De esta manera se logra reducir la cantidad de entradas disponibles para la reventa, pero no se aborda el problema del fraude.
- **Sistemas con códigos QR dinámicos:** Tal vez esta alternativa es la más eficaz entre las que se nombran en este apartado. Los códigos QR dinámicos (que cambian periódicamente) evitan la duplicidad de las entradas y por ende el fraude. Este sistema únicamente exige al usuario disponer de un dispositivo móvil con acceso a Internet.

1.5.2 Análisis de sistemas similares

En este apartado se describirán algunos sistemas similares existentes en el mercado. A diferencia de los descritos en el apartado anterior, estos son soluciones basadas en las mismas tecnologías que se utilizan en este proyecto. Es por ello que todas las que se describen a continuación implementan soluciones *blockchain*.

- **Oveit:** Esta empresa norteamericana ofrece soluciones tecnológicas para la gestión de eventos y espacios de entretenimiento. Entre uno de sus proyectos se encuentra un sistema para validar entradas a eventos con NFTs. Disponen de la aplicación del asistente y el validador [12].
- **EntryNFT:** Esta empresa española presta servicios para *tokenizar* [13] las entradas de tus eventos. En este caso permiten una reventa controlada de estos y ofrecen una aplicación para poder transferirlos [14].
- **RacksLabs:** Empresa española de consultoría de software web3. Poseen una colección de NFTs llamada *MrCrypto* [15] que otorgan ventajas exclusivas a sus propietarios. Una de ellas es que les permite identificarse en los eventos que realiza la empresa [16].

1.5.3 Justificación

Este proyecto se distingue en algunos conceptos con los sistemas similares vistos en la subsección anterior por los motivos que se menciona a continuación.

En primer lugar, la solución propuesta en este proyecto no se trata de un Saas (*Software as a Service*) como los proyectos de Oveit y EntryNFT, sino que se trata de una aplicación gratuita donde los usuarios solo tendrán que hacerse cargo del coste de las propias transacciones de la red sobre la que se opera.

En segundo lugar, la solución propuesta permite la compra de las entradas desde la propia aplicación, además de poder almacenarlas y transferirlas.

Por otro lado, desde la perspectiva del autor del proyecto, la ejecución de este proyecto constituye una oportunidad de aprendizaje y desarrollo profesional en la tecnología *blockchain* que en un futuro le podría servir de utilidad realizando alguna actividad laboral en proyectos con esta tecnología.

1.6 Riesgos

A continuación, se lista los riesgos principales a los que el proyecto está expuesto debido a la naturaleza del mismo y su alcance.

1.6.1 Inexperiencia con tecnología *blockchain*

El autor del proyecto no tiene mucha experiencia desarrollando Dapps, lo cual requerirá un esfuerzo para adaptarse a estas tecnologías y llevar un buen ritmo de trabajo.

1.6.2 Falta de tiempo

Puesto que se trata de un TFG, el tiempo disponible para desarrollar el proyecto es bastante limitado.

1.6.3 Compatibilidad de las aplicaciones a nivel de plataforma

Puede darse el caso, que, durante el desarrollo de las aplicaciones, algunos recursos de las librerías que se proporcionan no sean compatibles con Android o iOS.

1.6.4 Rendimiento de las aplicaciones

La velocidad de las transacciones *blockchain* depende de lo saturada que esté la red sobre la que se opera. Un mal rendimiento de esta afecta directamente a la experiencia del usuario.

1.7 Metodología

En esta sección se presenta la metodología de trabajo y las herramientas de seguimientos usadas durante el proyecto.

1.7.1 Metodología de trabajo

De cara a la metodología de trabajo que se utiliza, es *Scrum* de *Agile* una tecnología en la que el autor tiene alguna experiencia fruto de proyectos reales de *software*.

El trabajo planificado se fragmenta en historias de usuario (tareas) y estas mismas se reparten entre los *sprints* que se llevan a cabo durante todo el proyecto. Estas tareas se puntúan en función de las horas de desarrollo que se estima que se puede tardar en realizarlas. Estos *sprints* tienen una duración de 2 semanas.

De cara a las ceremonias típicas de *Scrum*, se ha tenido que hacer algunas pequeñas modificaciones, ya que en este caso el equipo del proyecto es un único desarrollador. Las modificaciones para cada ceremonia son las siguientes:

- ***Sprint planning*:** En esta ceremonia se define el objetivo del *sprint* basándose en la prioridad que se hayan dado a las historias de usuario en el *backlog*. Como tal, esta ceremonia no tiene ningún cambio más allá de que se realiza individualmente.

- ***Dailies*:** Normalmente, esta reunión sirve para dar contexto al resto del equipo de qué es lo que se trabajó el día anterior, qué se trabajará durante el día y si hay algún bloqueo o problema presente para así mirar de solventarlo cuanto antes. Para este proyecto simplemente se revisa en pocos minutos lo que se ha trabajado el día anterior y se planifica lo que se trabajará durante el día. Simplemente, sirve para hacer una organización diaria.

- ***Sprint Review*:** En esta sesión se revisan los resultados finales del *sprint* para así evaluar el trabajo completado durante el *sprint*. La idea es poder hacer una pequeña demostración del

proyecto con lo que se ha conseguido agregar. Se aprovecha esta ceremonia para validar los avances. Acto seguido se realiza el *Sprint Retrospective*.

- ***Sprint Retrospective*:** En esta reunión se hará una valoración de cómo ha ido el *sprint* para así ver qué hay que seguir haciendo, cambiar o dejar de hacer.

Cabe mencionar que, al ser *sprints* de 2 semanas, todas las ceremonias, a excepción de las *Dailies*, son bisemanales.

1.7.2 Herramientas de seguimiento

En el siguiente apartado se especifican las herramientas utilizadas durante el desarrollo del proyecto para tener un seguimiento de este:

- **Taiga:** Esta aplicación web es una herramienta de gestión de proyectos *Agile* que permite organizar las historias de usuario, puntuarlas, definirlas y registrar incidencias, entre otros. Será útil para la gestión del proyecto, ya que permite mostrar una planificación del sprint y gestionar el estado de las tareas.

- **GitHub:** De cara a la gestión del código, se utiliza un repositorio en GitHub para el control de versiones. Se sigue el flujo de GitFlow para así tener una buena gestión de las versiones del código. Este flujo se caracteriza por tener dos ramas, *master* y *develop*. La rama *master* contiene el código en su estado más estable, destinado a producción, donde cada *feature* implementada y sometida a tests con resultados positivos es fusionada. Por otro lado, *develop* sirve como rama de integración para las nuevas *features* (que tendrán una rama para cada una de ellas), donde se acumulan los cambios y mejoras en espera de ser promovidos a *master* tras un ciclo de la fase de testeo.

- **Aplicaciones de Google:** Por otro lado, se usan las aplicaciones de Google Drive para guardar y compartir ficheros y Gmail y Google Meet para la comunicación con la directora del proyecto.

2. Planificación inicial

En este capítulo se presenta la planificación de tareas inicial realizada al principio del proyecto. En el capítulo que tiene por título Ejecución del Proyecto se presentan las diferencias acontecidas durante el proyecto respecto a la planificación inicial. Se divide el capítulo en calendario donde se puede encontrar la distribución de horas realizadas durante los días que dura el proyecto, la descripción de las tareas a realizar durante el proyecto, los recursos necesarios, el resumen de recursos, horas por tareas, el diagrama de Gantt y el estudio de riesgos realizado a inicios del proyecto.

2.1 Calendario

El proyecto tiene una duración de 4 meses, con fecha de inicio el 19 de febrero y fecha límite el 25 de junio, donde se realizará la presentación del proyecto. Teniendo en cuenta que la dedicación de un Trabajo Final de Grado son 540 horas aproximadamente y que se dispone de unos 127 días hasta la fecha límite. Lo cual supone una carga diaria de 4 horas de dedicación al proyecto. Para poder hacer un uso eficiente del tiempo, las 28 horas semanales de dedicación se dividirán entre 5 días de trabajo con jornadas de 6 horas.

2.2 Tareas

A continuación, se definen las tareas a realizar durante el proyecto. Estás se dividen entre tareas de gestión y documentación, que como su nombre indica, son aquellas relacionas con la elaboración de los documentos del proyecto y las tareas de desarrollo, las cuales se refieren al trabajo técnico a realizar.

2.2.1 Tareas de gestión y documentación

- **GD1 - Contextualización y alcance (25h)**. Lectura de los documentos relativos al primer entregable y redacción de este. En este documento se define el contexto, justificación, alcance y metodología. **Dependencias:** Ninguna.
- **GD2 - Planificación temporal (8h)**. Lectura de los documentos relativos al segundo entregable y redacción de este. En este documento se define la planificación del trabajo. **Dependencias: GD1**.
- **GD3 – Gestión económica y sostenibilidad (10h)**. Lectura de los documentos relativos al tercer entregable y redacción de este. En este se define el presupuesto del proyecto y se elabora un informe de sostenibilidad. **Dependencias: GD2**.
- **GD4 - Integración documento final (20h).** Integrar todas las entregas anteriores en una y realizar los cambios pertinentes para librar un mejor documento final de la gestión del proyecto. **Dependencias: GD3**.

- **GD5 - Documentación del proyecto (80h).** Documentar las fases del desarrollo del proyecto. **Dependencias: GD4**.
- **GD6 - Elaboración presentación final (20h).** Preparar las diapositivas a mostrar en la presentación final, preparar un guión para este y una demostración en vivo del proyecto. **Dependencias: GD5**.
- **GD7 - Reuniones de seguimiento con la directora (10h)**. Se realizarán reuniones bisemanales para mostrar el estado del proyecto y comentar avances o bloqueos. **Dependencias:** Ninguna.

2.2.2 Tareas de desarrollo

Fase Inicial

- **FI1- Formación (100h)**. Formación en *smartcontracts*, estándares de Ethereum, NFTs, integración de Metamask en proyectos *mobile*, etc. **Dependencias:** Ninguna.
- **FI2 - Especificación de requisitos (15h)**. Identificar los diferentes requisitos y funcionalidades del sistema. **Dependencias:** Ninguna.
- **FI3 - Diseño de las aplicaciones (10h)**. Realizar mockups para representar la UI de nuestras aplicaciones. **Dependencias: FI2**.
- **FI4 - Preparación del entorno (25h)**. Crear repositorios en GitHub, crear proyecto en Taiga y añadir las tareas, preparar servicios API necesarios, crear billeteras y fondearlas con *tokens* para probar en *testnets* e instalación de software necesario para el desarrollo. **Dependencias: FI2**.

Smartcontracts

- **SC – Crear el *smartcontract* (20h)**: Crear y desplegar el *smartcontract* que gestiona los eventos, sus entradas, su transferencia, visualización y canje. **Dependencias: FI1** y **FI4**.

Backend

- **BC – Crear API *backend* (20h)**: Crear y desplegar un *backend* para así evitar conectar el frontend de las aplicaciones con el smartcontract directamente. **Dependencias: SC1**.

Aplicación para los asistentes

- **AA1 - Autenticar usuarios (5h):** Integrar Metamask con la aplicación de React Native y gestionar el inicio de sesión del usuario. **Dependencias: FI4**.
- **AA2 - Consultar entradas en propiedad (15h):** Implementar una pantalla donde el usuario podrá visualizar sus entradas adquiridas. Se necesitará interactuar con el *smartcontract* para

recuperar los NFTs asociados a la dirección de la cartera del usuario y listarlos. **Dependencias: BC**.

- **AA3 - Consultar detalle entrada en propiedad (15h):** Implementar una pantalla donde el usuario pueda visualizar toda la información relacionada con la entrada. **Dependencias: AA2**.
- **AA4 - Transferir una entrada en propiedad (20h)**: Implementar un formulario donde el usuario pueda especificar a qué billetera desea transferir la entrada seleccionada. Se debe gestionar la transferencia con el *smartcontract*. **Dependencias: AA3**.
- **AA5 - Consultar eventos (20h)**: Implementar una pantalla donde se muestren los eventos disponibles para que el usuario puede adquirir entradas. Para listarlos se necesita interactuar con el *smartcontract*. Se debe crear una pantalla para poder visualizar la entrada seleccionada. **Dependencias: BC**.
- **AA6 - Adquirir una entrada (10h):** Implementar una pantalla donde se muestre toda la información relacionada con la entrada para poder adquirirla. Se necesitará interactuar con el *smartcontract*. **Dependencias: AA5**.
- **AA7 - Canjear entrada con código QR (20h)**: Implementar un *pop-up* que muestre el QR de la entrada. Se necesitará interactuar con el *smartcontract*. **Dependencias: AA2**.
- **AA8 - Canjear entrada con NFC (20h)**: Implementar un *pop-up* que indique al usuario que esta emitiendo una señal NFC para validar la entrada. Se necesitará interactuar con el *smartcontract*. **Dependencias: AA2**.

Aplicación para los validadores (35h)

- **AV1 - Validar entrada con código QR (20h)**: Implementar una pantalla para escanear entradas mediante código QR. Se necesitará interactuar con el *smartcontract.* **Dependencias: AA7**.
- **AV2 - Validar entrada con NFC (20h)**: Implementar una pantalla para escanear entradas mediante NFC. Se necesitará interactuar con el *smartcontract.* **Dependencias: AA8**.

2.3 Recursos

Recursos humanos: Estos son el desarrollador del proyecto (**D**) y la directora de este (**C**).

Recursos materiales: Escritorio, ordenador y dos dispositivos móviles para probar los casos de uso.

Recursos software:

- **AF – Alchemy Faucet:** Herramienta que proporciona *tokens* gratuitos para *testnets*.
- **AS – Android Studio:** Necesario para configurar un emulador Android.
- **B – Brave**: Navegador para buscar información y otros servicios en línea.
- **CG – ChatGPT**: IA generativa para tareas genéricas.
- **F – Figma:** Plataforma de diseño para crear UIs.

- **GA – Galileo AI:** Aplicación web para crear diseños de UI mediante IA.
- **GH - GitHub:** Sistema para el control de versiones con Git.
- **GP – Gant Project:** Software para crear el diagrama de Gantt.
- **GW – Google Workspace:** Conjunto de herramientas de productividad y comunicación.
- **RI – Remix IDE:** Editor de código online para desarrollar, compilar y desplegar *smartcontracts*.
- **T – Taiga:** Herramienta de gestión de proyectos ágiles.
- **TA – Thirdweb API:** Plataforma que ofrece herramientas para facilitar el desarrollo de *Dapps*.
- **VSC – Visual Studio Code:** Editor de código para desarrollar el proyecto.
- **Z – Zotero:** Herramienta para la gestión de referencias y citas bibliográficas.

2.4 Estimaciones y diagrama de Gantt

2.4.1 Estimaciones

Código	Tarea	Horas	Dependencias	Recursos
GD1	Contextualización y alcance	25h	-	D, C, B, CG, GW, Z
GD2	Planificación temporal	8h	GD1	D, C, B, CG, GP, GW, Z
GD3	Gestión económica y sostenibilidad	10h	GD2	D, C, B, CG, GW, Z
GD4	Integración del documento final	20h	GD3	D, C, B, CG, GW, Z
GD5	Documentación del trabajo	80h	GD4	D, C, B, CG, GW, Z
GD6	Elaboración presentación final	20h	GD5	D, C, B, CG, GW,
GD7	Reuniones con la directora	10h	-	D, C, B, T, GW
FI1	Formación	100h	-	D, AS, B, CG, RI, TA, VSC
FI2	Especificación de requisitos	15h	FI1	D, B, CG, GW, Z
FI3	Diseño de las aplicaciones	10h	FI2	D, B, CG, F, GA
FI4	Preparación del entorno	25h	FI2	D, AF, AS, B, CG, GH, T, TA, VSC
SC	Crear el *smartcontract*	20h	FI4	D, B, CG, GH, RI, TA
BC	Crear API *backend*	20h	SC1	D, B, CG, GH, RI, TA, VSC
AA1	Autenticar usuarios	5h	FI4	D, B, CG, GH, TA, VSC
AA2	Consultar entradas en propiedad	15h	AA1, BC	D, AS, B, CG, GH, TA, VSC
AA3	Consultar detalle entrada en propiedad	15h	AA2	D, AS, B, CG, GH, TA, VSC
AA4	Transferir una entrada en propiedad	20h	AA3	D, AS, B, CG, GH, TA, VSC
AA5	Consultar eventos disponibles	20h	BC	D, AS, B, CG, GH, TA, VSC
AA6	Adquirir una entrada	10h	AA5	D, AS, B, CG, GH, TA, VSC
AA7	Canjear entrada con código QR	20h	AA2	D, AS, B, CG, GH, TA, VSC
AA8	Canjear entrada con NFC	20h	AA2	D, AS, B, CG, GH, TA, VSC
AV1	Validar entrada con código QR	20h	AA7	D, AS, B, CG, GH, TA, VSC
AV2	Validar entrada con NFC	20h	AA8	D, AS, B, CG, GH, TA, VSC
TOTAL		528h		

Tabla 2.1: Estimaciones, dependencias y recursos de tareas. Fuente: Elaboración propia

2.4.2 Diagrama de Gantt

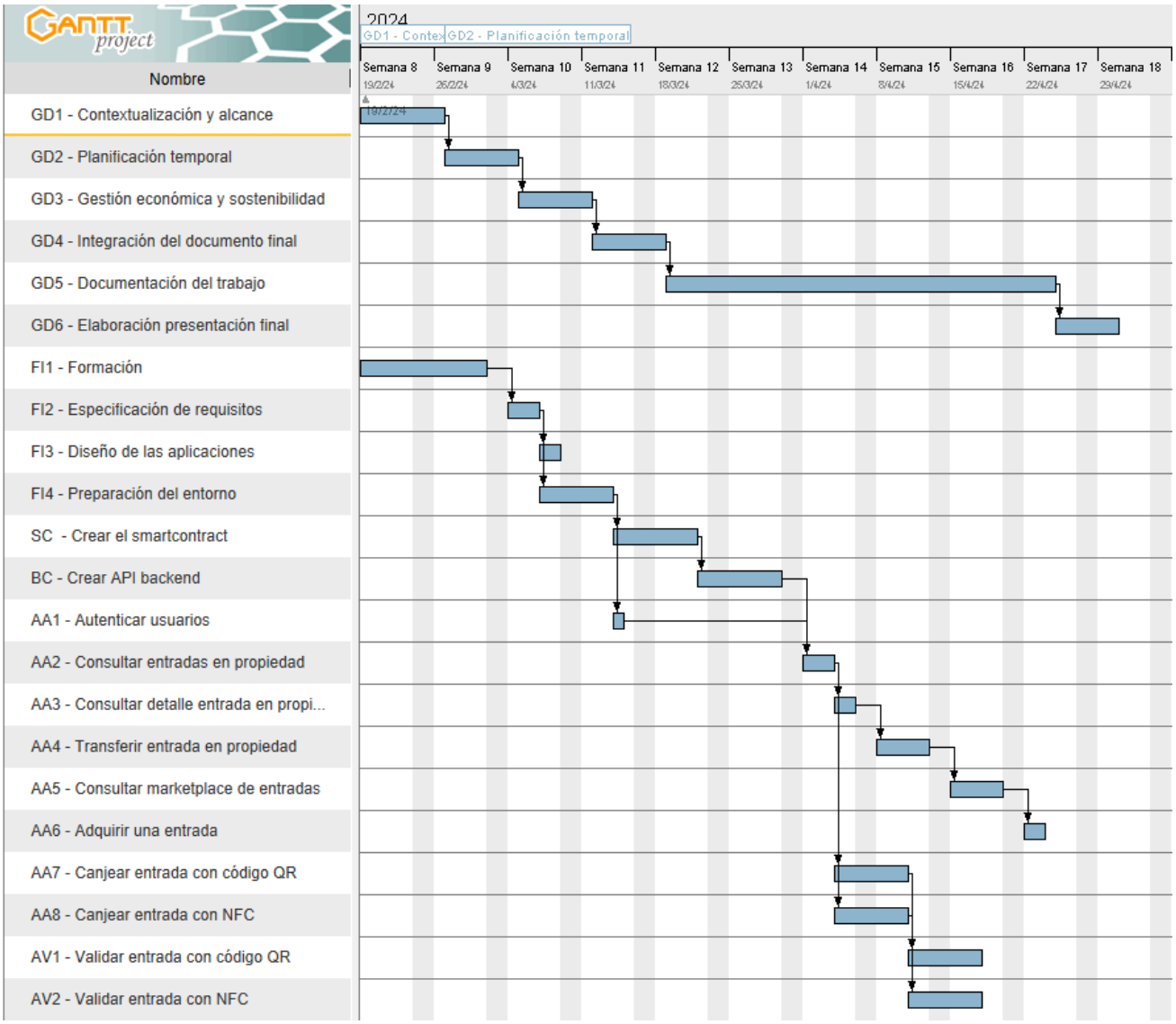


Figura 2.1: Diagrama de Gantt. Fuente: Elaboración propia

2.5 Gestión del riesgo

En la sección 2.4 se definieron algunos riesgos que podían afectar al desarrollo del proyecto. A continuación, se identifica posibles estrategias para evitar o mitigarlos.

2.5.1 R1 – Inexperiencia con tecnología *blockchain*

Descripción: Al ser el primer contacto con esta tecnología, se corre el riesgo de quedarse estancado o progresar muy lentamente por falta de conocimiento para resolver según que problemáticas.

Probabilidad: Baja - Media.

Severidad: Media. En el peor de los casos aumentaría el número de horas por tarea.

Estrategia de mitigación: Dedicar más horas a formarse y si es preciso ajustar el *backlog*.

2.5.2 R2 – Falta de tiempo

Descripción: Debido a la naturaleza del proyecto, al ser un TFG el tiempo disponible está muy acotado.

Probabilidad: Baja.

Severidad: Alta. Si faltase tiempo para desarrollar el proyecto, se tendrían que recortar funcionalidades influyendo negativamente en el resultado final.

Estrategia de prevención: Establecer prioridades en las tareas y desarrollar un timeline para ejecutar cada funcionalidad en un periodo de tiempo. La organización es clave.

Estrategia de mitigación: Ajustar el *backlog* (en última instancia).

2.5.3 R3 – Compatibilidad de las aplicaciones a nivel de plataforma

Descripción: Algunas funcionalidades pueden no tener soporte en alguna de las plataformas a desarrollar.

Probabilidad: Media – Alta.

Severidad: Alta. Se dejaría de continuar el desarrollo en la plataforma afectada.

Estrategia de prevención: En este caso se utilizaría React Native con Expo [17], ya que permite el desarrollo multiplataforma. Hoy en día es un *framework* muy robusto, pero siempre cabe la posibilidad de lidiar con alguna incompatibilidad. Para evitar encontrarse con esto a mitad del desarrollo, sería interesante probar previamente las funcionalidades que pueden ser más problemáticas en pequeños proyectos de prueba (como podría ser el acceso a la cámara para el QR, el lector NFC o el HCE).

Estrategia de mitigación: En el mejor de los casos se podría migrar el proyecto a React Native CLI [18], que permite más flexibilidad y control sobre la configuración del proyecto y las dependencias nativas. En el peor de los casos se debería renunciar a desplegar el proyecto en la plataforma afectada.

2.5.4 R4 – Rendimiento de las aplicaciones

Descripción: Puesto que se hace uso de APIs de terceros, librerías externas y la fluidez de las transacciones dependen del estado de la red, podría desencadenar en un rendimiento bajo del sistema afectado a la experiencia del usuario.

Probabilidad: Media. Normalmente, funciona bien, pero puede tener periodos de mayor demanda.

Severidad: Baja – Media. El usuario percibirá la aplicación más lenta, lo cual entorpecería su uso en general.

Estrategia de prevención: Para evitar esta problemática, se puede utilizar una *sidechain* [19] de Ethereum como puede ser Polygon [20] u Optimistic Ethereum [21]. Estas son soluciones de *Layer 2* [22].

Estrategia de mitigación: Plantear trabajar en una red distinta compatible con la EVM de modo que se pueda aprovechar el *smartcontract* desarrollado con Solidity. Una muy buena alternativa podría ser la *BNB Chain de* Binance [23].

3. Estudio económico inicial

En este capítulo se presenta la estimación inicial del coste del proyecto considerando recursos humanos, materiales, de software y otros recursos necesarios. También se expone el mecanismo que se usa al final del proyecto para ver la desviación total en horas y coste estimados. En el capítulo que tiene por título Ejecución del Proyecto se presentan las diferencias del coste final del proyecto con el estimado inicialmente.

3.1 Identificación y estimación del coste

En los siguientes apartados se estima los costes del proyecto. Estos están directamente relacionados con sus recursos. Se estimarán los costes como si de un proyecto real se tratase.

3.1.1 Recursos humanos

Pese a que el proyecto será realizado por una sola persona, se calcula su coste dependiendo de los roles que desarrollan en cada etapa. En la siguiente tabla se puede ver los precios por hora de cada rol. Estos precios han sido obtenidos basándose en el portal Glassdoor referente a los salarios por hora en la consultora Accenture en España [24].

Rol	Precio por hora	Precio por hora con coste SS
Jefe de proyecto [JP]	31,00 €	40,30 €
Analista programador [AP]	14,00 €	18,20 €
Diseñador UX/UI [D]	12,00 €	15,60 €
Programador *Mobile* [PM]	13,00 €	16,90 €
Programador *blockchain* [PB]	15,00 €	19,50 €

Tabla 3.1: Recursos humanos del proyecto y sus costes. Fuente: Elaboración propia

Ahora se realiza el cómputo total:

Código	Tarea	Horas por tarea	Horas según rol					Coste
			JP	AP	D	PM	PB	
GD1	Contextualización y alcance	25h	15	10				786,50 €
GD2	Planificación temporal	8h	4	4				234,00 €
GD3	Gestión económica y sostenibilidad	10h	10					403,00 €
GD4	Integración del documento final	20h	10	10				585,00 €
GD5	Documentación del trabajo	80h	20	60				2.080,00 €
GD6	Elaboración presentación final	20h	20					806,00 €
GD7	Reuniones con la directora	10h	10					403,00 €
FI1	Formación	100h				50	50	1.820,00 €
FI2	Especificación de requisitos	15h	5	10				383,50 €
FI3	Diseño de las aplicaciones	10h			10			156,00 €
FI4	Preparación del entorno	25h	5			10	10	565,50 €
SC	Crear el *smartcontract*	20h					20	390,00 €
BC	Crear API *backend*	20h					20	390,00 €
AA1	Autenticar usuarios	5h				5		84,50 €
AA2	Consultar entradas en propiedad	15h				10	5	266,50 €
AA3	Consultar detalle entrada en propiedad	15h				10	5	266,50 €
AA4	Transferir una entrada en propiedad	20h				10	10	364,00 €
AA5	Consultar eventos disponibles	20h				15	5	351,00 €
AA6	Adquirir una entrada	10h				5	5	182,00 €
AA7	Canjear entrada con código QR	20h				15	5	351,00 €
AA8	Canjear entrada con NFC	20h				15	5	351,00 €
AV1	Validar entrada con código QR	20h				15	5	351,00 €
AV2	Validar entrada con NFC	20h				15	5	351,00 €
TOTAL		528h						11.921,00 €

Tabla 3.2: Costes de las tareas del proyecto según roles. Fuente: Elaboración propia

3.1.2 Recursos materiales

En la siguiente tabla X se observa los costes y amortizaciones de los recursos materiales que se usa para el proyecto. El coste de amortización se calcula de la siguiente manera:

$$Coste\ amortización = \frac{Coste\ (Euro)}{Vida\ útil\ (Años) \times Horas\ de\ uso/Año} \times Duración\ del\ proyecto\ (Horas)$$

Recurso	Coste	Vida útil (Años)	Amortización
PC i513600KF RTX 4070	1523,00 €	6	73,43 €
Monitor Benq Mobiuz EX2710Q	260,00€	10	7,52 €
Móvil Xiaomi Redmi Note 12	230,00 €	3	13,86 €
Móvil Xiaomi Mi 8 Lite	215,00 €	3	20,73 €
TOTAL			115,54 €

Tabla 3.3: Costes y amortización de recursos materiales. Fuente: Elaboración propia

3.1.3 Recursos *software*

En el proyecto, todas las herramientas de *software* que se usan son gratuitas, por ende, no hace falta contabilizarlo.

3.1.4 Otros recursos

También se deberían contabilizar otros gastos no menos importantes como podrían ser el mobiliario, la energía y los servicios de Internet. Estos dos últimos, su coste será la suma de los cuatro meses que se requiera el servicio (213,80 € y 207,8 € respectivamente).

Recurso	Coste	Vida útil (Años)	Amortización
Escritorio	200,00 €	10	3,61 €
Silla	160,00 €	5	5,78 €
Electricidad	53,45 €	-	-
Internet	51,95 €	-	-
TOTAL			9,39 €

Tabla 3.4: Costes y amortización de otros recursos. Fuente: Elaboración propia

Con lo cual la suma total será de 430,99 €.

Se debe incluir también los gastos que pueden derivar de los riesgos e imprevistos que se vieron en secciones anteriores. Los riesgos están asociados a las tareas de los roles de programador

Mobile y programador *Blockchain*, dependiendo la tarea, se multiplican las horas de desviación por el precio por hora dependiendo el rol encargado.

Riesgo	Probabilidad	Tiempo	Rol afectado	Coste
Inexperiencia con tecnología *blockchain*	25%	20h	PB	390,00 €
Falta de tiempo	15%	-	PM, PB	-
Compatibilidad de las aplicaciones a nivel de plataforma	50%	25h	PM	422,50 €
Rendimiento de las aplicaciones	30%	5h	PB	97,50 €
TOTAL				910,00 €

Tabla 3.5: Análisis de riesgos y costes asociados. Fuente: Elaboración propia

3.1.4 Estimación total del coste

Para finalizar, en la siguiente tabla se representa la suma presupuestaria total del proyecto. Cabe decir que, se ha establecido unos costes de contingencia del 15%.

	Coste
Recursos humanos	11.921,00 €
Recursos materiales	115,54 €
Otros recursos	430,99 €
Imprevistos	910,00 €
Contingencia	2.006,63 €
TOTAL	15.384,15 €

Tabla 3.6: Resumen de costes totales del proyecto. Fuente: Elaboración propia

3.2 Control de gestión

Todo lo que se ha ido desarrollando hasta ahora son estimaciones, es por ello que es necesario saber cuál es el tiempo real dedicado a cada tarea y así poder comparar con estas estimaciones iniciales.

Para lograr tener un buen control de los recursos humanos del proyecto, se necesita registrar las horas de trabajo reales dedicadas a cada tarea para así poder vislumbrar las posibles desviaciones y tomar las riendas lo antes posible ajustando la planificación temporal. Las horas trabajadas se registran en las tareas de Taiga, de este modo se puede obtener indicadores y gráficos útiles para ver con qué fluidez se están desarrollando las tareas del proyecto.

4. Especificación de requisitos

En este capítulo, se detalla la especificación de requisitos del proyecto, una tarea fundamental en el desarrollo de cualquier sistema informático. El objetivo de la especificación de requisitos de un sistema software es definir de manera clara y detallada las necesidades y expectativas de los usuarios y las partes interesadas.

4.1 Requisitos funcionales

Los requisitos funcionales describen qué debe permitir hacer un sistema software. A continuación se listan los requisitos funcionales para cada aplicación del sistema. Estos definen qué debe permitir hacer cada una de ellas.

Requisitos funcionales de la aplicación para los asistentes:

- **RFA1**: La aplicación debe permitir vincular una billetera de criptomonedas e iniciar sesión con ella.
- **RFA2**: La aplicación debe permitir visualizar los eventos disponibles de los organizadores.
- **RFA3**: La aplicación debe permitir buscar un evento mediante su nombre como palabra clave.
- **RFA4**: La aplicación debe permitir visualizar la información de un evento.
- **RFA5**: La aplicación debe permitir comprar una entrada a un evento.
- **RFA6**: La aplicación debe permitir visualizar las entradas compradas por el usuario.
- **RFA7**: La aplicación debe permitir visualizar la información de cada entrada adquirida por el usuario.
- **RFA8**: La aplicación debe permitir buscar una entrada entre las adquiridas mediante el nombre del evento como palabra clave.
- **RFA9**: La aplicación debe permitir transferir una entrada a otra billetera.
- **RFA10**: La aplicación debe permitir validar una entrada mediante escaneo de código QR.
- **RFA11**: La aplicación debe permitir validar una entrada mediante escaneo NFC.
- **RFA12**: La aplicación debe permitir cerrar sesión con la billetera vinculada para poder conectar otra.

Requisitos funcionales de la aplicación para los validadores:

- **RFV1**: La aplicación debe permitir seleccionar el evento para el cual se van a validar entradas.
- **RFV2**: La aplicación debe permitir validar una entrada mediante escaneo de código QR.
- **RFV3**: La aplicación debe permitir validar una entrada mediante escaneo NFC.
- **RFV4**: La aplicación debe permitir cambiar entre método de escaneo fácilmente.

- **RFV5**: La aplicación debe informar del estado de la validación de la entrada.

4.2 Requisitos no funcionales

A diferencia de los requisitos funcionales, los requisitos no funcionales describen cómo debe comportarse el sistema en lugar de qué debe permitir hacer. Estos se enfocan en la calidad y los criterios de operación del sistema. A continuación se definen los siguientes requisitos no funcionales de las aplicaciones a desarrollar en función de su prioridad (de mayor a menor):

Número	1
Tipo de requisito	Usabilidad
Descripción	La interfaz de usuario debe ser intuitiva y fácil de usar para personas con diferentes niveles de habilidad tecnológica.
Justificación	La usabilidad es un factor crítico que impacta directamente la adopción y satisfacción del usuario. Una interfaz complicada o no intuitiva puede llevar a frustraciones, errores en el uso y, eventualmente, a una tasa de abandono más alta de la aplicación. Mejorar la usabilidad asegura que los usuarios puedan aprovechar todas las funcionalidades ofrecidas sin dificultades, lo que se traduce en una experiencia de usuario positiva
Condición de satisfacción	**Tiempo de aprendizaje**: El sistema deberá ser lo suficientemente intuitivo, de modo que los usuarios nuevos deban poder realizar funciones básicas como buscar eventos, visualizar detalles de eventos, y comprar una entrada y validarla en menos de cinco minutos después de su primera interacción con la aplicación.

Tabla 4.1: Requisito no funcional número 1. Fuente: Elaboración propia

Número	2
Tipo de requisito	Rendimiento
Descripción	La aplicación debe ser capaz de procesar rápidamente las solicitudes, especialmente las relacionadas con la compra y transferencia de entradas, para garantizar una experiencia fluida y sin demoras para los usuarios.
Justificación	Un rendimiento eficiente es crucial para mantener la confianza y la satisfacción del usuario, especialmente en un entorno donde las transacciones de tiempo crítico, como la validación de entradas, requieren rapidez y fiabilidad
Condición de satisfacción	**Eficiencia de procesamiento**: Las operaciones de escritura en la *blockchain*, deben optimizarse para minimizar los tiempos de espera y maximizar la eficiencia de la transacción. No debería tardar más de 1 minuto realizarse una operación de escritura.

Tabla 4.2: Requisito no funcional número 2. Fuente: Elaboración propia

Número	3
Tipo de requisito	Disponibilidad
Descripción	La aplicación debe garantizar que los usuarios pueden acceder en cualquier momento y desde cualquier lugar.
Justificación	Una alta disponibilidad es esencial en la aplicación, puesto que una interrupción del servicio que no permitiera al usuario vender o transferir sus entradas podría resultar en la perdida de ingresos y con ello la confianza en el sistema.
Condición de satisfacción	**Tiempo de *uptime***: Tomando como referencia un mes natural, se garantizará que el sistema estará disponible al usuario durante el 99% de este (no deberá poder estar más de 43 minutos fuera de servicio).

Tabla 4.3: Requisito no funcional número 3. Fuente: Elaboración propia

4.3 Modelo conceptual de los datos

En esta sección se muestra en la figura 4.1 un diagrama siguiendo el modelo conceptual de datos UML. En ella se puede visualizar los datos necesarios para el funcionamiento del sistema a desarrollar.

En el diagrama se describe las clases de objetos de los que es necesario almacenar datos, y las asociaciones que existen en el dominio de la venta de entradas entre estos objetos. El lenguaje UML permite definir restricciones que los datos deben satisfacer para ser correctos, pero algunas restricciones no se pueden expresar en este lenguaje. Es por este motivo, que después del diagrama se han incluido otras restricciones adicionales que los datos del sistema deben satisfacer.

Para terminar esta sección también se incluye una descripción del diagrama, que incluye cada clase de objeto, así como de los atributos y asociaciones que se almacenaran para cada clase.

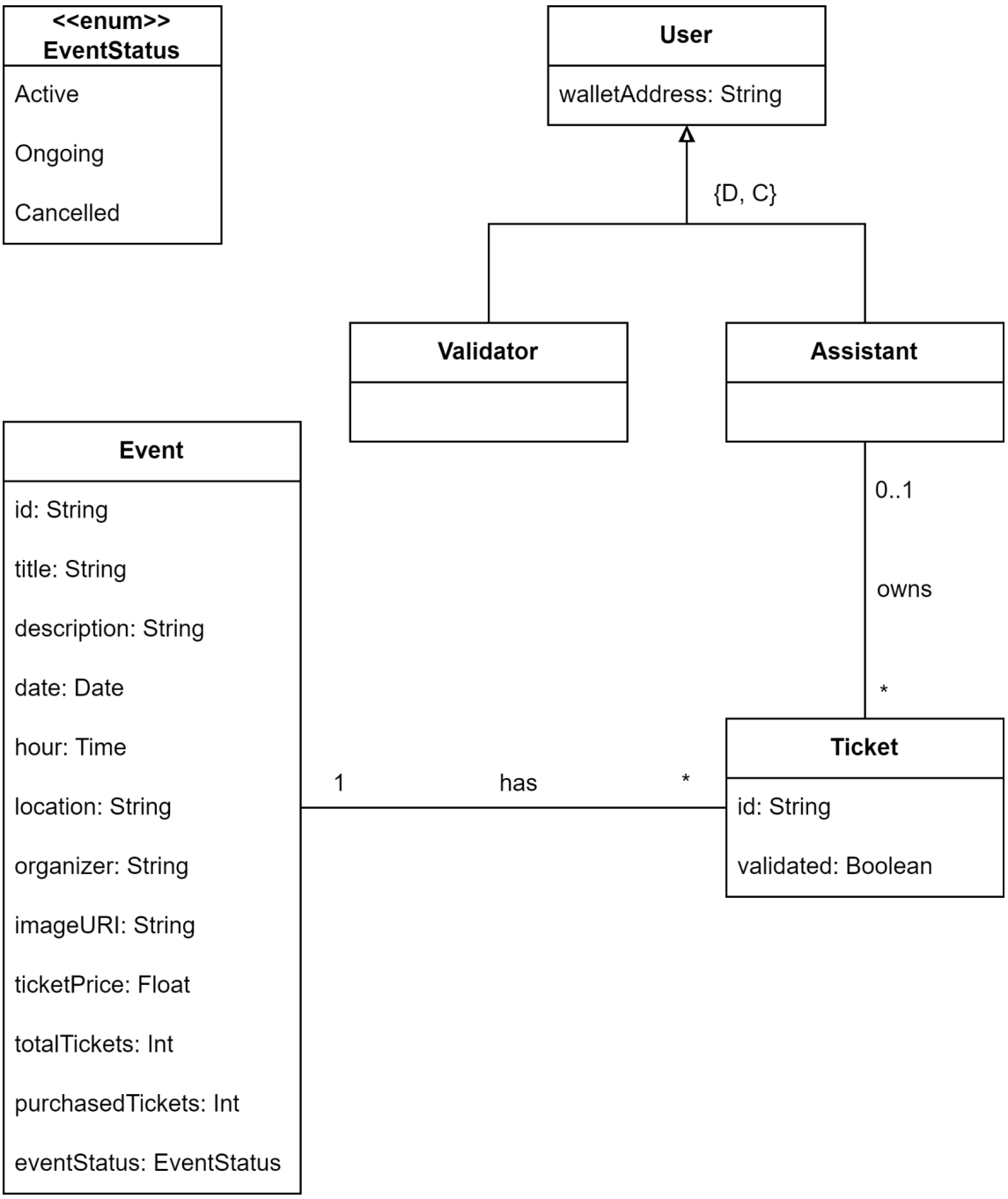


Figura 4.1: Diagrama conceptual de los datos UML. Fuente: Elaboración propia

Restricciones textuales:

RT1: Event(id), Ticket(id), User(walletAddress)

RT2: totalTickets >= 1 (las entradas totales para un evento debe ser al menos de una unidad)

RT3: purchasedTickets >= 0 (el número entradas compradas no puede ser negativo)

RT4: ticketPrice >= 0 (el precio de las entradas no puede ser negativo)

Descripción del diagrama

El diagrama UML incluye tres clases principales. A continuación se describe cada una de ellas:

Clase *Event*: Representa un evento registrado en el sistema. Sus atributos son los siguientes:

- *id: identificador del evento*
- *title*: nombre del evento
- *description*: descripción del evento
- *date*: fecha del evento
- *hour*: hora del evento
- *location*: lugar del evento
- *organizer*: organizador del evento
- *imageURI*: URI para una imagen del evento
- *ticketPrice*: precio de una entrada del evento
- *totalTickets*: número total de entradas disponibles del evento
- *purchasedTickets*: número de entradas que se han comprado del evento
- *eventStatus*: el estado del evento

Clase *Ticket*: Representa una entrada perteneciente a un evento. Sus atributos son los siguientes:

- *id*: identificador de la entrada, se corresponde con el identificador del *token* NFT
- *validated*: indica si esa entrada ya ha sido validada

Clase *User*: Representa el usuario del sistema. En este caso, esta clase tiene dos subclases diferenciadas, el usuario asistente y el usuario validador. El atributo de la clase padre es el siguiente:

- *walletAddress:* dirección de la billetera de Ethereum del usuario

Subclase *Assistant*: Representa un usuario asistente a eventos. Se entiende por usuario asistente aquel que participa en la asistencia los eventos mediante la compra de entradas. Hace uso de la aplicación de los asistentes.

Subclase *Validator*: Representa un usuario validador de entradas. Hace uso de la aplicación de los validadores. En el sistema, los usuarios validadores comparten todos la misma dirección de billetera.

Enumeración *EventStatus*: Representa los diferentes estados en los que se encuentra un evento. Estos puede ser:

- *Active*: El evento está disponible para su compra de entradas pero no validación

- *Ongoing*: El evento está disponible para validar las entradas a los usuarios asistentes que las poseen
- *Cancelled*: El evento ha sido cancelado.

4.4 Historias de usuario

En esta sección se detallan las historias de usuario del sistema acordes a los requisitos funcionales del sistema. Se describen en formato tabla, con un identificador de la historia, su nombre, descripción y criterios de aceptación. Primero se listan los de la aplicación de los asistentes:

UCA01	**Iniciar sesión**
Como usuario asistente, **quiero** poder iniciar sesión en la aplicación, **para** hacer uso de las funcionalidades de esta.	
● El usuario selecciona la billetera con la que quiere iniciar sesión ● Se abre la aplicación de billetera del usuario, este confirma la vinculación ● Se muestra la billetera con la que el usuario va a iniciar sesión en la pantalla de inicio. ● Se habilita un botón para navegar al resto de la aplicación	

UCA02	**Visualizar eventos disponibles**
Como usuario asistente, **quiero** poder visualizar todos los eventos disponibles de los organizadores, **para** poder elegir eventos que me interesen y planificar mi asistencia.	
● Se muestran en la pantalla de exploración todos los eventos disponibles en este momento. ● Se muestra un mensaje en caso de no haber eventos disponibles.	

UCA03	**Búsqueda de eventos**
Como usuario asistente, **quiero** poder buscar por nombre entre los eventos disponibles de los organizadores, **para** que pueda visualizar un evento en concreto.	
● Se despliega una barra de búsqueda. ● Se muestran los eventos que coinciden en nombre con la palabra de la búsqueda. ● Se muestra un mensaje en caso de no haber ninguna coincidencia.	

UCA04	**Visualización de un evento**
Como usuario asistente, **quiero** poder visualizar la información de un evento de los organizadores, **para** que pueda obtener información acerca de este.	
● Se muestra una pantalla con la información relativa al evento	

UCA05	**Comprar una entrada**
Como usuario asistente, **quiero** poder comprar una entrada a un evento, **para** que pueda recibir acceso a este	
● Se bloquea el botón de compra si el usuario ya tiene una entrada para ese evento. ● Se sustituye el botón de compra por un mensaje si el evento ha sido cancelado ● Si el usuario pulsa el botón, se abre la aplicación de la billetera para que el usuario firme la transacción. ● Si la compra se realiza satisfactoriamente, se navega a una pantalla donde se muestra el detalle de la compra. ● Se muestra mensaje de error en caso de fallar la transacción. ● Se reduce la disponibilidad de entradas del evento en una unidad. ● Se extrae el importe de la entrada de la billetera del usuario y se transfiere a la billetera del organizador.	

UCA06	**Búsqueda de entradas**
Como usuario asistente, **quiero** poder buscar por nombre entre mis entradas adquiridas, **para** que pueda visualizar una entrada en concreto.	
● Se despliega una barra de búsqueda. ● Se muestran las entradas que coinciden en nombre con la palabra de la búsqueda. ● Se muestra un mensaje en caso de no haber ninguna coincidencia.	

UCA07	**Visualizar entradas compradas**
Como usuario asistente, **quiero** poder visualizar las entradas que he adquirido, **para** que pueda gestionar mis entradas.	
● Se muestra una pantalla con todas las entradas adquiridas por el usuario. ● Si no tiene ninguna entrada adquirida, se muestra un mensaje conforme a ello.	

UCA08	**Visualizar detalle entrada**
Como usuario asistente, **quiero** poder visualizar la información de una entrada en concreto, **para** que pueda gestionarla o visualizar su información.	
● Se muestra un modal con la información de la entrada. ● Si el estado del evento de la entrada está activo, aparece un botón para poder transferirla ● Si el estado del evento de la entrada está en curso, aparece un botón para poder validarla ● Si el estado del evento de la entrada es cancelado, aparece un modal indicando que el evento ha sido cancelado y se han devuelto los fondos al usuario	

UCA09	**Transferir una entrada**
Como usuario asistente, **quiero** poder transferir una entrada que poseo, **para** enviarla a otra billetera.	
● Se muestra una pantalla con la información de la entrada que se va a transferir y un campo para insertar la dirección de la billetera a la que se desea enviar. ● Se muestra un modal para confirmar que el usuario desea enviar esta entrada. ● Se abre la aplicación de la billetera para que el usuario firme la transacción. ● Se muestra un mensaje de transferencia satisfactoria si se transfiere correctamente. ● Se muestra un mensaje de error si se transfiere a la misma dirección que posee la entrada. ● Se muestra un mensaje de error si no se introduce una dirección válida de Ethereum. ● Se muestra un mensaje de error si falla la transacción.	

UCA10	**Validar entrada mediante escaneo de código QR**
Como usuario asistente, **quiero** poder validar mi entrada mediante escaneo de código QR, **para** poder validarla y acceder al evento.	
● Se muestra una opción para escoger el método de validación de la entrada. ● Se muestra el código QR de la entrada. ● Se muestra un mensaje de entrada validada satisfactoriamente si se realiza correctamente.	

UCA11	**Validar entrada mediante escaneo NFC**
Como usuario asistente, **quiero** poder validar mi entrada mediante escaneo NFC, **para** poder validarla y acceder al evento.	
● Se muestra una opción para escoger el método de validación de la entrada. ● Se muestra un mensaje con la indicación de cómo posicionar el dispositivo para canjear la entrada. ● Se activa la vibración del dispositivo cuando se valida. ● Se muestra un mensaje de entrada validada satisfactoriamente si se realiza correctamente.	

UCA12	**Cerrar sesión**
Como usuario asistente, **quiero** poder cerrar la sesión con la billetera vinculada, **para** que pueda conectar otra billetera distinta.	
● Se pulsa el botón de cerrar sesión. ● Se muestra aviso conforme la acción que se va a realizar. ● Si el usuario acepta, se navega a la pantalla de inicio de sesión. ● Si el usuario lo deniega, se mantiene en la misma pantalla.	

A continuación se listan los de la aplicación de los validadores:

UCV01	**Seleccionar un evento**
Como usuario validador, **quiero** poder seleccionar un evento, **para** empezar a validar las entradas de este.	
● El usuario selecciona el evento al cual va a validar entradas. ● Se navega a una pantalla donde el usuario puede seleccionar un método de validación (NFC o QR).	

UCV02	**Validar mediante código QR**
Como usuario validador, **quiero** poder validar las entradas de un evento mediante escaneo QR, **para** poder otorgar o denegar el acceso a un asistente.	
● El usuario escanea una entrada mediante QR. ● Se muestra un mensaje acorde a si la validación ha sido correcta o incorrecta (caso de error).	

UCV03	**Validar mediante NFC**
Como usuario validador, **quiero** poder validar las entradas de un evento mediante escaneo NFC, **para** poder otorgar o denegar el acceso a un asistente.	
● El usuario escanea una entrada mediante NFC. ● Se muestra un mensaje acorde a si la validación ha sido correcta o incorrecta (caso de error).	

UCV04	**Cambiar método de validación**
Como usuario validador, **quiero** poder cambiar el método de validación fácilmente, **para** poder ser más ágil con el escaneo de las entradas.	
● Si el usuario está escaneado con QR, pulsa el botón de “Cambiar a NFC” y se navega a la pantalla de escaneo mediante NFC. ● Si el usuario está escaneado con NFC, pulsa el botón de “Cambiar a QR” y se navega a la pantalla de escaneo mediante QR.	

5. Arquitectura del sistema

En este capítulo se detalla la arquitectura del sistema. Primeramente, se analiza la arquitectura física seguida de la arquitectura lógica. Más adelante se muestran algunos diagramas de secuencia representativos del sistema. Seguidamente, se hace un análisis de los patrones de diseño utilizados en el proyecto. Por último se describe como se almacenan los datos en el sistema desarrollado.

5.1 Arquitectura física

La arquitectura física de un sistema define la interconexión y configuración de los componentes hardware del mismo. Es decir, todos los aspectos tangibles para el soporte, ejecución y gestión del sistema. Para ilustrar de manera más entendible cómo se ha planteado la arquitectura para este sistema, en la figura 5.1 se muestra la arquitectura física del sistema, además de los principales *frameworks* para entender la función de cada componente físico

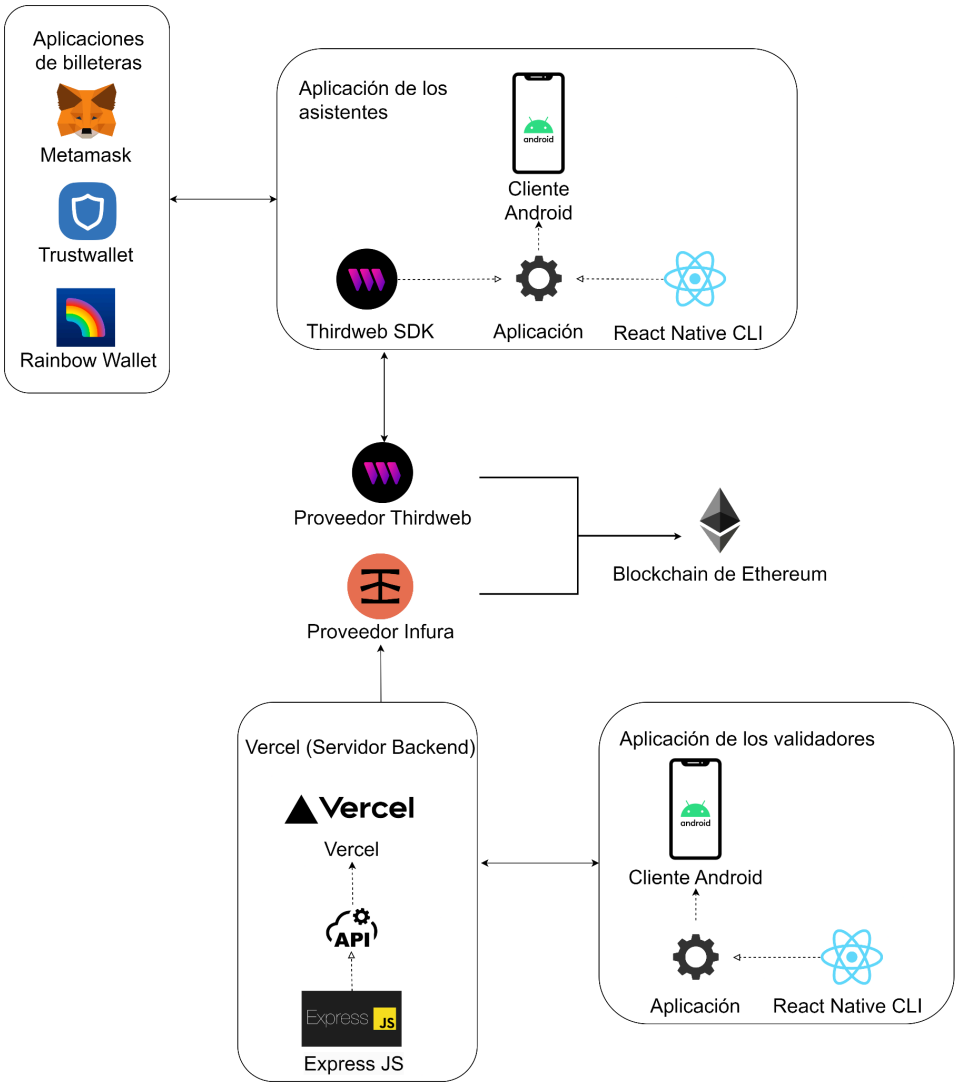


Figura 5.1: Diagrama de arquitectura física. Fuente: Elaboración propia

Como se puede observar en la figura 5.1, se ha dividido la arquitectura física en cuatro bloques principales: la aplicación de los asistentes, la aplicación de los validadores, las aplicaciones de billeteras y el servidor *backend*. Por otro lado, se dispone de dos proveedores de infraestructura *blockchain* y la red de Ethereum, estos tres últimos se detallarán al final.

Aplicación de los asistentes

Esta aplicación se ejecuta en los dispositivos móviles Android de los asistentes y ha sido desarrollada en React Native CLI. La idea inicial era hacerlo con Expo, pero no es posible integrarlo con el SDK de Thirdweb. De todos modos, esta implicación no afecta al desarrollo.

Este *framework* de Javascript permite desarrollar aplicaciones multiplataformas (Android e iOS) de manera nativa. Sin embargo, al no tener las herramientas necesarias para el desarrollo de iOS, la ejecución del proyecto solo se ha centrado en el desarrollo de Android.

Se ha optado por la elección de este *framework* y no otro, puesto que los escasos recursos que se han encontrado para hacer desarrollos web3 que requieran firmar las transacciones en plataformas móviles son para este mismo.

Por otro lado, se tiene el SDK de Thirdweb. Esta empresa ofrece una plataforma de desarrollo web3 muy completa. Entre sus herramientas se ha seleccionado el SDK para React Native, ya que permite conectar la billetera de los usuarios a la aplicación e interactuar con el *smartcontract*. Cabe destacar que este componente es clave en todo el proyecto por el hecho de que dentro de las opciones que había para desarrollar la aplicación era la única que verdaderamente se había probado que funcionaba.

Aplicaciones móviles de billeteras

Se tratan de aplicaciones de billeteras de criptomonedas que permiten firmar transacciones. Se necesitan para que cuando los usuarios de la aplicación de los asistentes realicen alguna función que requiera escribir en la *blockchain* se autorice la transacción. Como ya se ha explicado en la sección 1.3 *Out-lists* no es parte del desarrollo del proyecto. Gracias al SDK de Thirdweb se pueden conectar las billeteras de las aplicaciones de Metamask, TrustWallet y Rainbow. Aunque el SDK permite conectar muchas más para este proyecto, se han considerado únicamente estas tres por su popularidad.

Aplicación de los validadores

A diferencia de la aplicación de los asistentes, esta no hace uso del SDK de Thirdweb. Esta decisión se debe a un factor muy importante, la agilidad del validador realizando su labor. Cuando se valida una entrada se requiere que el usuario validador firme una transacción en la *blockchain*. Si se realizase este proceso mediante el SDK de Thirdweb se necesitaría iniciar alguna aplicación

de billetera (como las comentadas en el párrafo anterior) y autorizar la transacción. En un contexto de control de acceso en un evento es completamente ineficiente por el tiempo que se demora entre el escaneo de la entrada y la firma de la transacción. Es por ello que a raíz de este problema se ha considerado tener una billetera única destinada a las transacciones de los validadores alojada en un servidor *backend*. De este modo, la firma de las transacciones se realizará a través de una API REST, sin necesidad de realizar firmas en aplicaciones externas.

Servidor *Backend*

Como se ha descrito en el párrafo anterior, se requiere de una API REST para realizar la validación de las entradas en la aplicación de los validadores. Esta API, desarrollada con el *framework* de Express.js, estará disponible en una instancia de Vercel [25]. Desde la aplicación de los validadores se realizan las peticiones contra este servidor. La firma de la transacción en este caso no requiere de firmar manualmente con una billetera, puesto que las claves privadas de la billetera del validador están alojadas en el servidor *backend*. De este modo se consigue realizar escrituras en el *smartcontract* sin necesidad de usar una billetera para firmar.

Proveedor de infraestructura *blockchain* Infura

Los proveedores de infraestructura *blockchain* proporcionan acceso a nodos completos de Ethereum. Esto es útil, ya que simplifica el acceso a la red de Ethereum. Si no se usase un proveedor, sería necesario ejecutar un nodo de Ethereum propio y mantenerlo.

Para la aplicación de los validadores, al establecer conexión con un *backend* se necesita conectarlo a un proveedor. El motivo por el que no está conectado al proveedor de Thirdweb es debido a que no ha sido posible crear el proyecto *backend* con su herramienta Engine que permite tener una API REST de del *smartcontract* usando su propio proveedor [26]. Tras buscar información en la comunidad de Discord de Thirdweb acerca de los problemas que daba la configuración de Engine se ha terminado descartando su uso y optado por crear el proyecto mediante Express.js y desplegarlo en un servidor.

En este caso se ha optado por el servicio que ofrece Infura, ya que es gratuito y cumple perfectamente con los requisitos para el sistema. No obstante, cabe destacar que existen otros proveedores también gratuitos como Alchemy [27] o Moralis [28] que también servirían para este propósito.

Proveedor de infraestructura *blockchain* Thirdweb

Este proveedor se utiliza para conectarse a la *blockchain* a través de la aplicación de los asistentes únicamente. Este mismo ya viene configurado al usar el SDK de Thirdweb.

Red de Ethereum

A través de los proveedores mencionados anteriormente se accede a la red de Ethereum donde se despliega el *smartcontract* con el que se interactúa.

5.2 Arquitectura lógica

En esta sección se define la arquitectura lógica del sistema. Esta se refiere a la estructura y organización interna del *software*, centrándose en los componentes, módulos y cómo interactúan entre sí para lograr los objetivos del sistema. A diferencia de la arquitectura física, que describe la infraestructura física subyacente (como servidores, redes y dispositivos de hardware), la arquitectura lógica se centra en los aspectos conceptuales y funcionales del sistema.

Primeramente, se muestra la arquitectura lógica a nivel de componentes que forman la aplicación, indicando la máquina o servidor donde se ejecuta. Como se puede observar en la figura 5.2, el centro del sistema es el *smartcontract* alojado en la *blockchain* de Ethereum. Esto es así, puesto que es él quien expone los eventos disponibles con toda su información, se encarga de la emisión de las entradas de estos, así como la gestión de los mismos.

Para interactuar con el *smartcontract*, tanto la aplicación de los asistentes como el servidor *backend* establecen una conexión a través del protocolo de transporte HTTPS y mediante el protocolo de aplicación JSON-RPC [29].

Por último, la aplicación de los validadores establece conexión HTTPS con el servidor *backend* haciendo uso del protocolo de aplicación REST (ya que la API es de este tipo).

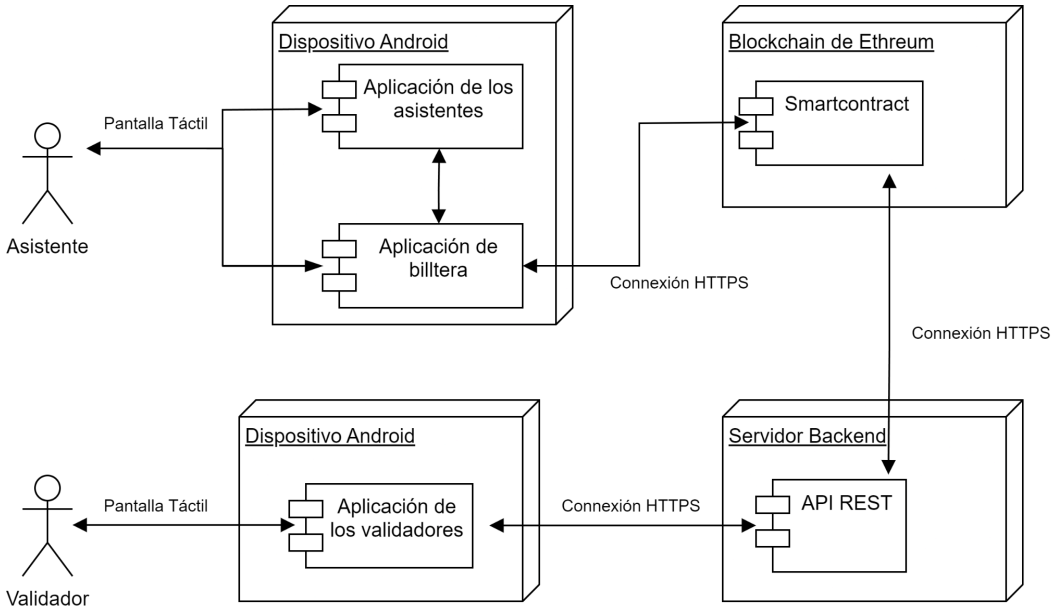


Figura 5.2: Arquitectura lógica de las aplicaciones. Fuente: Elaboración propia

Para finalizar con esta sección se explica cómo está estructurada la arquitectura de cada componente a nivel de carpetas y clases. Empezando por la de las aplicaciones móviles y después se analizará la arquitectura de la API REST.

5.2.1 Arquitectura en las aplicaciones

La arquitectura que se ha decidido seguir se conoce como *Screaming Architecture* [30]. Esta arquitectura propuesta por Bob Martin consiste en reflejar claramente qué es lo que hace nuestra aplicación. Esta es agnóstica al *framework,* de modo que puede ser implementada en cualquier proyecto.

El motivo por el que se ha escogido esta arquitectura es por el hecho de que mediante la organización de directorios que sugiere permite visualizar de manera muy clara sus elementos acorde a las funcionalidades de nuestra aplicación. Además de ello, evita tener carpetas con una lista extensa de ficheros (como por ejemplo una carpeta *components* con todos y cada uno de los componentes de nuestra aplicación). Esta arquitectura es útil sobre todo en proyectos grandes que puedan ir escalando, no obstante se ha considerado que es una buena práctica aplicarla en proyectos pequeños como este por la claridad y enfoque que aporta.

Cabe destacar, también, que se ha decidido utilizar la librería de Mobx [31], ya que proporciona una reactividad automática y facilita el manejo de estados.

A continuación, se describe la función de cada directorio dentro de la arquitectura, sus directorios internos. En la figura 5.3, se puede observar una vista general de ellos:

Core: Este directorio alberga los elementos fundamentales y esenciales de la aplicación y que son independientes de cualquier funcionalidad específica.

- **Entities**: Aquí se definen las estructuras de datos que representan conceptos claves del dominio, como pueden ser los eventos y entradas.
- **Repository**: En este directorio únicamente reside en un fichero donde se realizan las peticiones al exterior, es decir, al *smartcontract*.
- **Utils**: Este directorio contiene funciones y utilidades de uso general que se utilizan en toda la aplicación. Las utilidades pueden incluir funciones de formateo de fechas, de texto, manejo de errores, etc.

Adapters: Dentro de este se almacena código que permite formatear los datos que se recibe externamente a un formato concreto para después usarlo más cómodamente en otras partes de la aplicación.

Features: Esta es la carpeta donde se ve reflejado todo el negocio de la aplicación. Es por ello que en ella se alojan directorios con el nombre de la funcionalidad y después, dentro de él, tres subdirectorios clave:

- **Components**: Componentes de interfaz de usuario (UI) específicos de esa característica. Aquí podría haber la *card* de un evento o una entrada.
- **Screens**: Pantallas de la aplicación relacionadas con esa característica.
- **Stores**: Aquí se encuentran los almacenes de MobX relacionados con esa función. Estos almacenan y gestionan el estado de la aplicación relacionado con la funcionalidad en cuestión.

Ui: Este directorio contiene componentes de interfaz de usuario comunes reutilizables y estilos globales que se aplican en toda la aplicación. Los componentes aquí son independientes de cualquier lógica de negocio específica y pueden ser utilizados en múltiples partes de la aplicación. En este mismo directorio se encuentra un directorio con componentes como botones, campos de texto, iconos, píldoras y barra de navegación, entre otros. También se encuentra un directorio con la definición de los temas de la aplicación (colores, tamaños de texto, tamaños de *padding*, estilos de fuentes, etc).

Navigation: Contiene configuraciones y definiciones relacionadas con la navegación dentro de la aplicación. Pueden incluir archivos que describen los objetos que reciben las diferentes pantallas al navegar entre sí o las rutas entre pantallas.

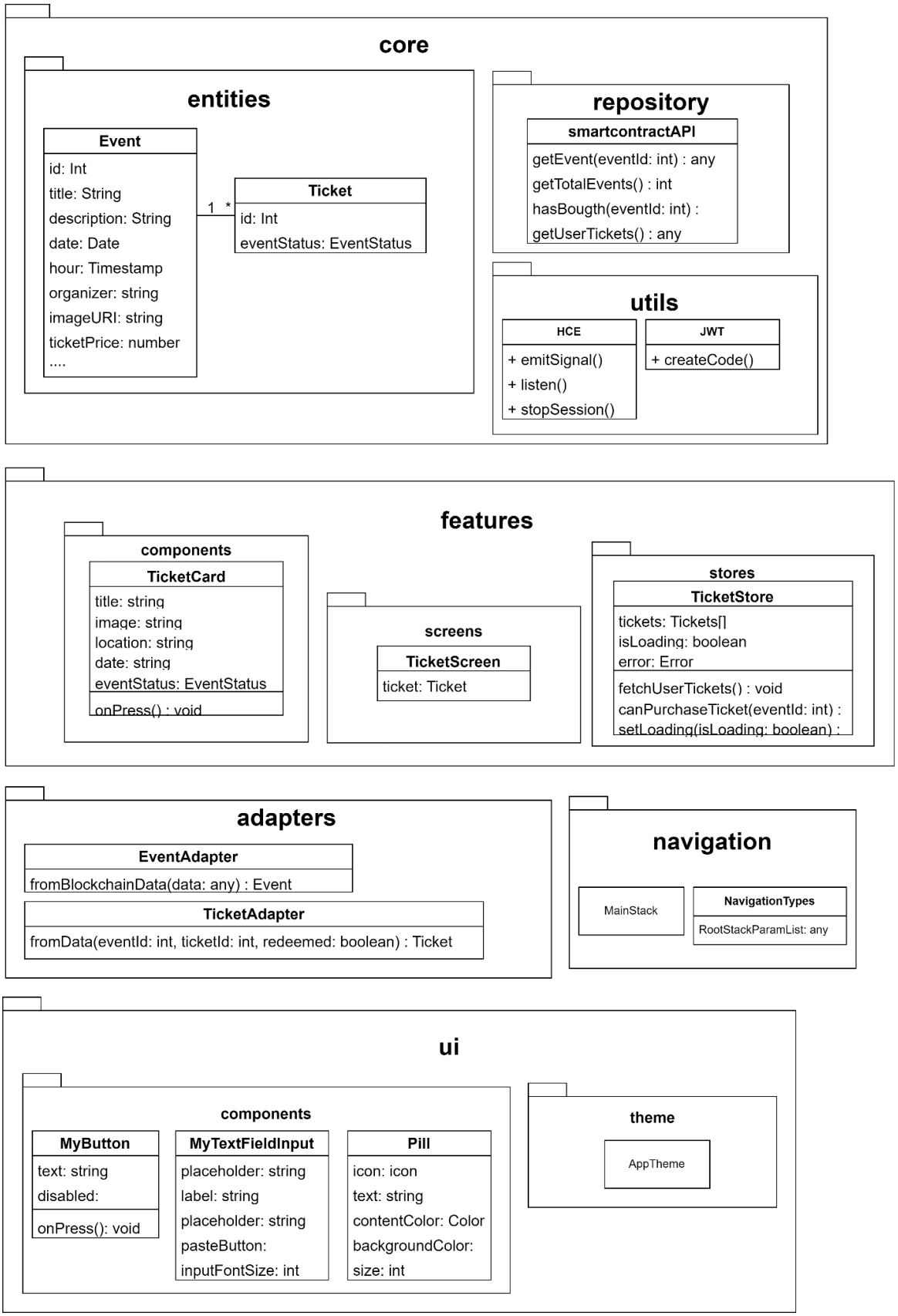


Figura 5.3: Arquitectura lógica a nivel de directorios. Fuente: Elaboración propia

5.2.2 Arquitectura de la API REST

Para el proyecto de la API REST, se ha adoptado la arquitectura *Router-Controller-Service*, la cual se integra de manera óptima con el desarrollo en Express.js [32]. Esta arquitectura se enfoca en la separación de responsabilidades y la modularización del código, facilitando la escalabilidad y el mantenimiento del sistema.

A continuación se detallan los componentes principales de esta arquitectura:

- ***Routes***: Las *routes* se definen en el directorio *routes*, donde cada ruta específica de la API se mapea a un controlador correspondiente y se les especifica los métodos HTTP que requiera (GET, POST, PUT, DELETE, PATCH…).
- ***Controllers***: En el directorio *controllers*, cada controlador maneja las solicitudes HTTP recibidas desde las rutas. Estos controladores encapsulan la lógica para procesar las solicitudes, interactuando con los servicios necesarios para realizar operaciones específicas.
- ***Services***: Estos últimos se encuentran en el directorio *services* y contienen la lógica de negocio principal de la aplicación. Estos servicios son invocados por los controladores para ejecutar operaciones complejas como acceso a datos, cálculos, integraciones externas, entre otros.

Para representar con más claridad como se estructura esta arquitectura y contextualizar con el proyecto de la API REST, en la figura 5.4 se ilustran las diferentes *routes*, *controllers* y *services* que se requieren en este sistema.

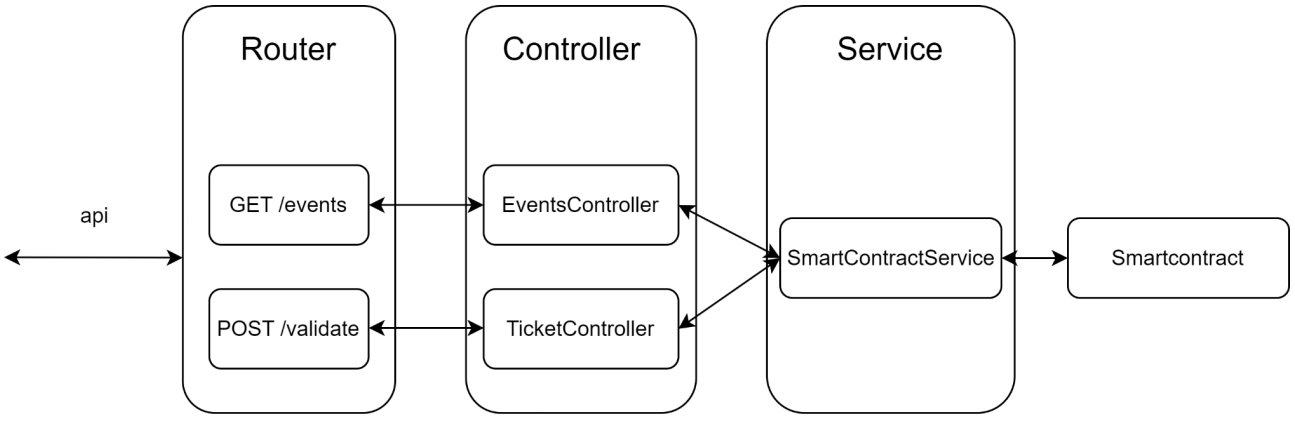


Figura 5.4: Arquitectura de la API REST. Fuente: Elaboración propia

Como se puede observar en la figura 5.4, únicamente se hace uso de dos rutas. Un método GET a */events* y un método POST */validate*. Ambas llamadas convergen en el mismo *service*, el *SmartContractService*. Este contiene todas las llamadas al *smartcontract* desplegado en la *blockchain*.

5.3 Ejemplo de diagrama de secuencia

En esta sección se presenta un diagrama de secuencia relativo a como listan los eventos en la aplicación para asistentes. Como se puede observar en la figura 5.5, se realiza una llamada *fetchEvents* contra la *EventStore*. Esta después interactúa con la clase *SmartContractApi* a través de la llamada *getAllEvents()*. Esta misma clase recoge la instancia del contrato de la *ContractStore* para realizar la llamada *call("getAllEvents")* la cual es una llamada al *smartcontract* alojado en la *blockchain*. El resultado que emite esta función no viene el formato adecuado para ser manejado por el resto las clases de la aplicación. Es por ello que se itera por cada elemento presente en el resultado emitido, llamando a la clase *EventAdapter* para ejecutar *fromBlockchainData()* que devuelve cada evento en el formato correspondiente. Después cada uno de esos elementos se inserta en un *array* que es devuelto posteriormente por la *EventStore*.

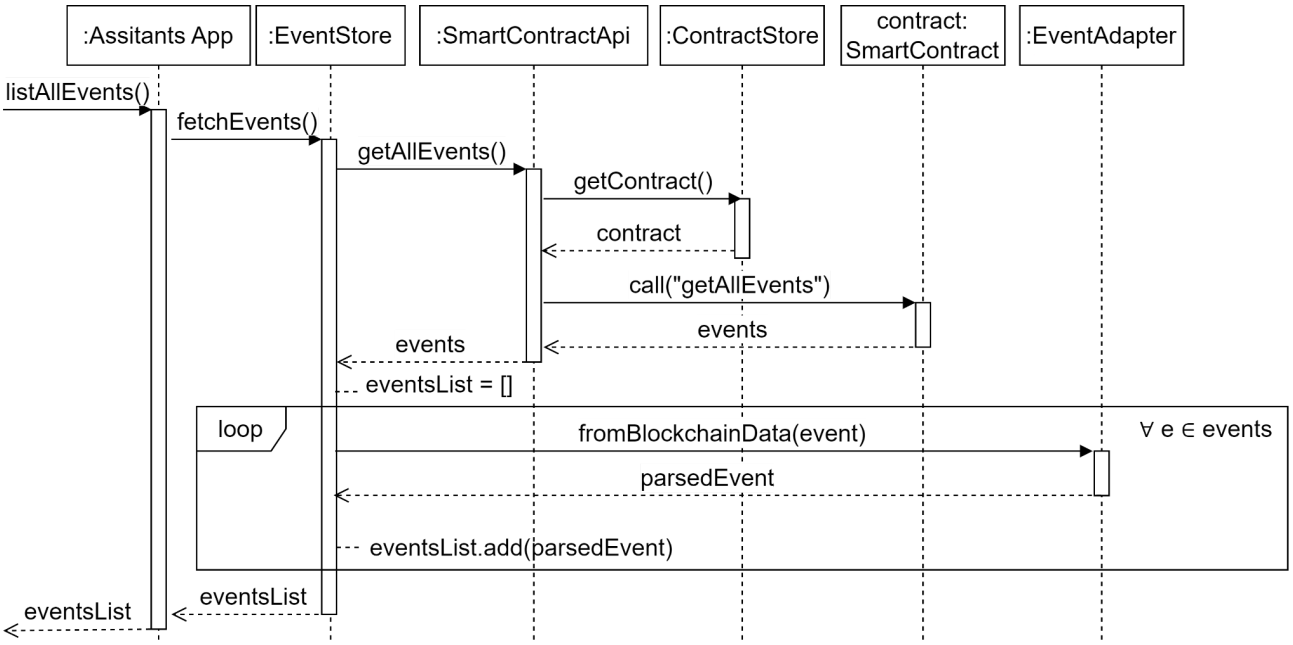


Figura 5.5: Ejemplo de diagrama de secuencia de llamada *listAllEvents()*. Fuente: Elaboración propia

5.4 Patrones de diseño utilizados

Durante esta sección se muestran los principales patrones de diseño que se han utilizado a la hora de desarrollar las aplicaciones del sistema.

Patrón *Observer*

Este patrón de comportamiento es muy común cuando se trabaja realizando aplicaciones reactivas. Consiste en, como su nombre indica, observar uno o más objetos a la espera de un cambio en su estado. Cuando el estado de estos objetos cambia, todos sus dependientes son notificados y actualizados de manera automática. En el contexto de este desarrollo se aplica al usar la librería de Mobx y también al hacer uso de *hooks* de React como *useEffect*. A continuación, en la figura 5.6 se muestra un UML de ejemplo de como se implementa en el caso de Mobx.

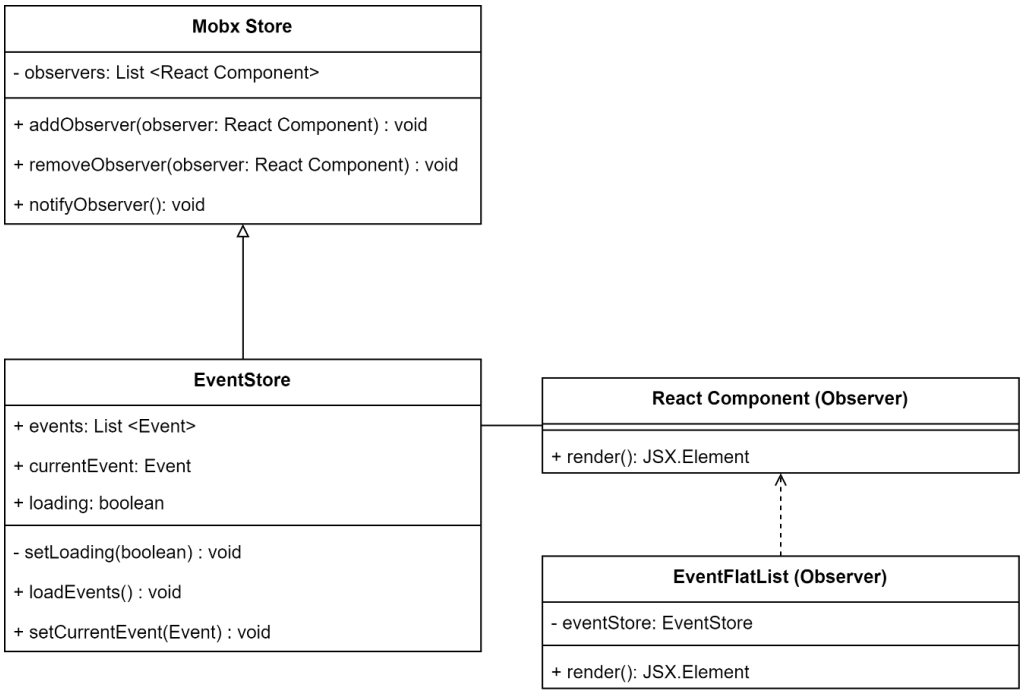


Figura 5.6: UML representativo del patrón *Observer* con Mobx. Fuente: Elaboración propia

Patrón *Singleton*

Se conoce como *Singleton* por el hecho de que permite asegurar que una clase tiene una instancia única y a su vez proporciona un acceso global a esta misma. Este patrón creacional también se encuentra al hacer uso de Mobx. Un ejemplo de esto es la instancia única de la clase ApiService del proyecto de la aplicación de los validadores, como se puede ver en la figura 5.7. Si se quisiera acceder a una de las funciones públicas de esta clase, primero se llamaría al método *getInstance(),* para acceder a la instancia de la clase y a partir de aquí bastaría con realizar las llamadas con esta misma. Es muy útil de cara a la gestión de recursos.

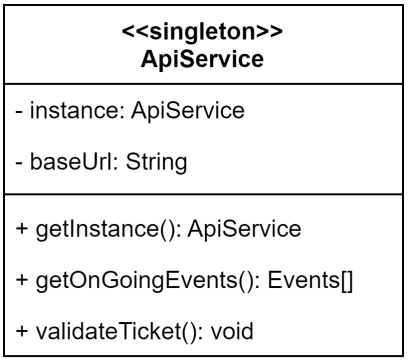


Figura 5.7: UML de la clase Singleton ApiService. Fuente: Elaboración propia

Patrón *Adapter*

Por último, se puede destacar también el uso del patrón *Adapter* en las aplicaciones del sistema. Este patrón estructural facilita la colaboración entre objetos con interfaces incompatibles. Funciona creando una clase intermedia que actúa como traductor entre dos interfaces distintas, permitiendo que trabajen juntas sin necesidad de modificar sus estructuras. En el contexto de este proyecto, el patrón *Adapter* se aplica cuando se leen datos de la *blockchain*, de la API REST o entre clases del dominio, encapsulando las diferencias y proporcionando una interfaz uniforme para su interacción. Cabe mencionar que no solo mejora la mantenibilidad y extensibilidad del código, sino que también facilita las pruebas y la integración de nuevos componentes en el futuro.

5.5 Gestión de los datos

En esta sección se explica cómo se gestionan los datos en todo el sistema. Habitualmente, los sistemas tradicionales implementan una base de datos centralizada donde almacenan toda la información necesaria para hacer funcionar el sistema. En el caso de este proyecto, esta base de datos no existe, puesto que es el *smartcontract* donde mediante unas estructuras de datos se representan y almacenan los datos del sistema para así conseguir que la aplicación de los asistentes sea una Dapp.

Es cierto que podría extraerse cierta información del *smartcontract* y gestionarla fuera de la *blockchain* como podría ser la información de los eventos (nombres, lugares, descripciones, etc). Esto no aporta ninguna mejora significativa, puesto que las lecturas de la blockchain son rápidas y en este caso solo haría más complejo el sistema.

Además de ello, el hecho de mantener estas estructuras de datos en el *smartcontract* garantiza la inmutabilidad e integridad de los datos, eliminando el riesgo de manipulación o pérdida de dichos datos gracias a las propiedades de la *blockchain*. De igual modo, se aprovecha la transparencia y automatización de los *smartcontracts* en el sentido de que las transacciones y cambios de estado son visibles y verificables por cualquier usuario. A continuación se muestran las estructuras que se han considerado para almacenar los datos y representar relaciones.

Estructura *Event*

Almacena los datos de un evento. Ver figura 5.8.

```
Unset
struct Event {
       uint256 id;
       string title;
       string organizer;
       string description;
       string location;
       string date;
       string hour;
       string imageURI;
       uint256 ticketPrice;
       uint256 totalTickets;
       uint256 purchasedTickets;
       EventStatus eventStatus;
}
```

Figura 5.8: Bloque de código estructura *Event* en el *smartcontract*. Fuente: Elaboración propia

Estructura *Ticket*

Almacena los datos de una entrada. Ver figura 5.9.

```
Unset
struct Ticket {
	uint256 id;
	uint256 occasionId;
	bool validated;
}
```

Figura 5.9: Bloque de código estructura *Ticket* en el *smartcontract*. Fuente: Elaboración propia

Diccionario *events*

La primera estructura es el diccionario *events*. A través de una clave, en concreto el identificador del evento, se devuelve un objeto *Event*. Este objeto es una *struct* que almacena los datos del evento. Esta estructura es de alcance público. Ver figura 5.10.

```
Unset
mapping(uint256 => Event) public events;
```

Figura 5.10: Bloque de código diccionario *events* en el *smartcontract*. Fuente: Elaboración propia

Diccionario *hasBought*

Después tenemos la el diccionario anidado *hasBought.* Este permite, dado un identificador de evento como clave del diccionario exterior, obtener otro diccionario donde su clave es la dirección de billtera de un usuario. Cuando se accede al valor del diccionario interior se obtiene un booleano que indica si esa dirección tiene una entrada para ese evento. Es decir, dado un identificador de evento, y una dirección de billetera, se puede leer si posee una entrada para dicho evento. Esta estructura es de alcance público. Ver figura 5.11.

```
Unset
mapping(uint256 => mapping(address => bool)) public hasBought;
```

Figura 5.11: Código diccionario *hasBought* en el *smartcontract*. Fuente: Elaboración propia

Diccionario *userTickets*

Por otro lado, para ver las entradas que posee un usuario se ha construido otro diccionario *userTickets* que dada una dirección de billetera, se retorna un arreglo de objetos Ticket. Estas son las entradas que pertenecen a esa billetera. Esta estructura es de alcance privado. Ver figura 5.12.

```
Unset
mapping(address => Ticket[]) private userTickets;
```

Figura 5.12: Código diccionario *userTickets* en el *smartcontract*. Fuente: Elaboración propia

Diccionario *eventAssitants*

Por último, se dispone del diccionario *eventAssistants* que permite devolver una lista de direcciones de billetera dada un identificador de evento. Esta estructura es de alcance privado. Ver figura 5.13.

```
Unset
mapping(uint256 => address[]) private eventAssistants;
```

Figura 5.13: Código diccionario *eventAssistants* en el *smartcontract*. Fuente: Elaboración propia

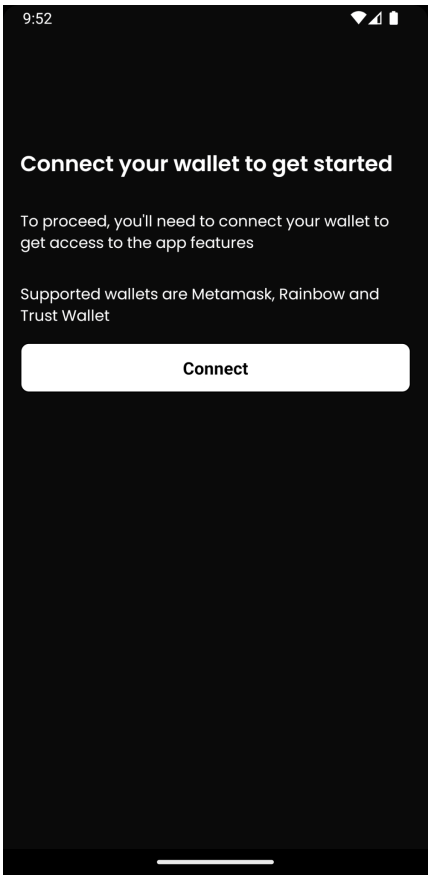
5.6 Diseños de las interfaces

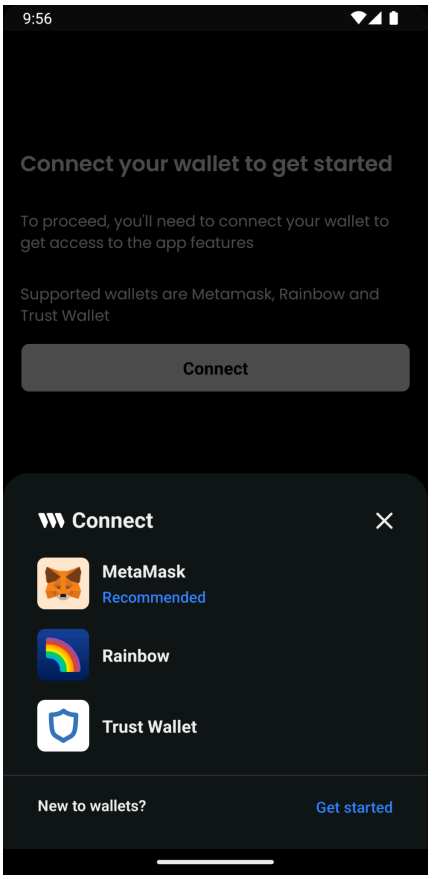
Esta sección está dedicada a explicar cómo se organizan las diferentes pantallas de las aplicaciones. Para ello se incluyen figuras de las pantallas con una descripción de su contenido y posibles acciones y navegabilidades. También se ilustra mediante diagramas UML el diseño externo de las aplicaciones.

5.6.1 Interfaz de la aplicación de los asistentes

Pantalla de inicio de sesión

Esta es la pantalla donde aterriza el usuario nada más entrar a la aplicación. En ella se muestran algunas instrucciones para poder utilizar la aplicación. Cuando se pulsa el botón de “Connect” se despliega una *BottomSheet* para seleccionar la cartera con la que el usuario se quiere conectar. Una vez realizado este paso se navega a la aplicación de billetera, y si el usuario autoriza dicha conexión, aparece su cartera vinculada y un botón de “Go to the App” para acceder a las funcionalidades de la aplicación. Ver figuras 5.14, 5.15 y 5.16.





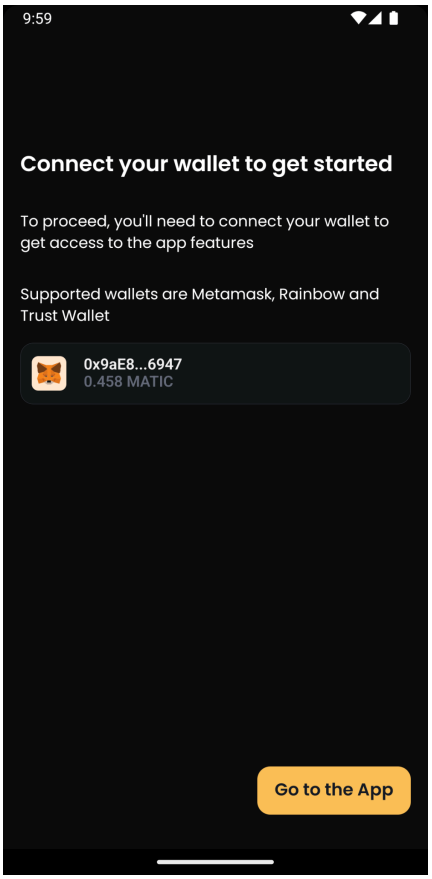


Figura 5.14 **Figura 5.15** **Figura 5.16**

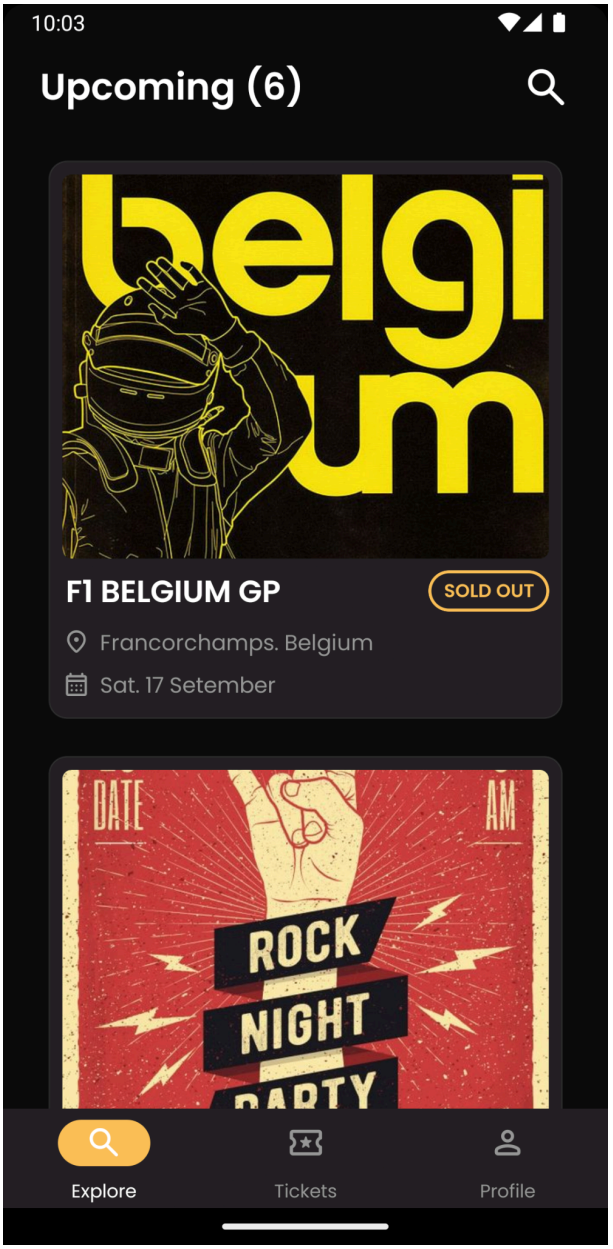
Figura 5.14: Pantalla de inicio sin conectar billetera. Fuente: Elaboración propia

Figura 5.15: Pantalla de inicio con *BottomSheet* selección de billetera. Fuente: Elaboración propia

Figura 5.16: Pantalla de inicio con billetera conectada. Fuente: Elaboración propia

Pantalla de eventos disponibles

En esta pantalla aparecen un listado de los eventos disponibles. Para cada evento se muestra una tarjeta con datos como el nombre del evento, el lugar y la fecha donde transcurre. Además, también se le añade una etiqueta a la derecha del título indicando las entradas que quedan disponibles en venta o si ha habido *sold-out*. Utilizando el buscador de la parte superior se puede realizar una búsqueda por nombre de un evento. Ver figuras 5.17 y 5.18.



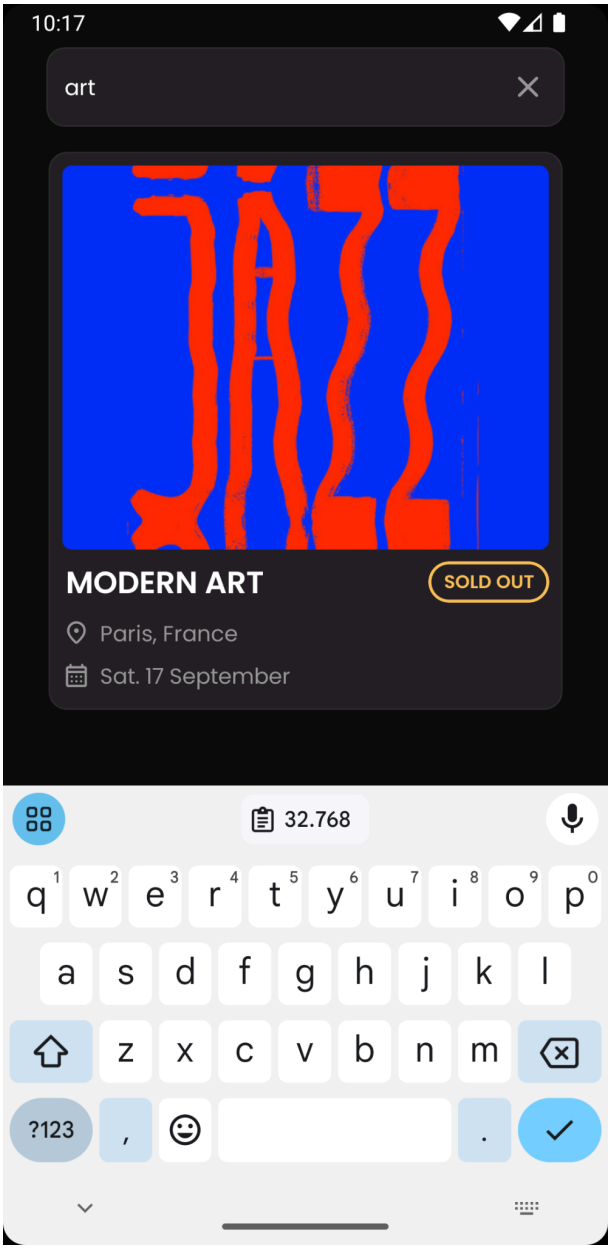


Figura 5.17

Figura 5.18

Figura 5.17: Pantalla de eventos disponibles. Fuente: Elaboración propia

Figura 5.18: Pantalla de eventos disponibles con búsqueda activada. Fuente: Elaboración propia

Pantalla del evento

En esta pantalla se muestra toda la información del evento como puede ser el organizador del evento, la fecha, la hora, el lugar y detalles del evento. También en la parte inferior aparece un botón para realizar la compra de una entrada para el evento, indicando su precio en cuestión. En caso de ya poseer una entrada para ese evento, el botón se deshabilita y muestra un texto de “Adquired”. Si el evento está cancelado, desaparece dicho botón y muestra un mensaje acorde a ello. Ver figuras 5.19 y 5.20.

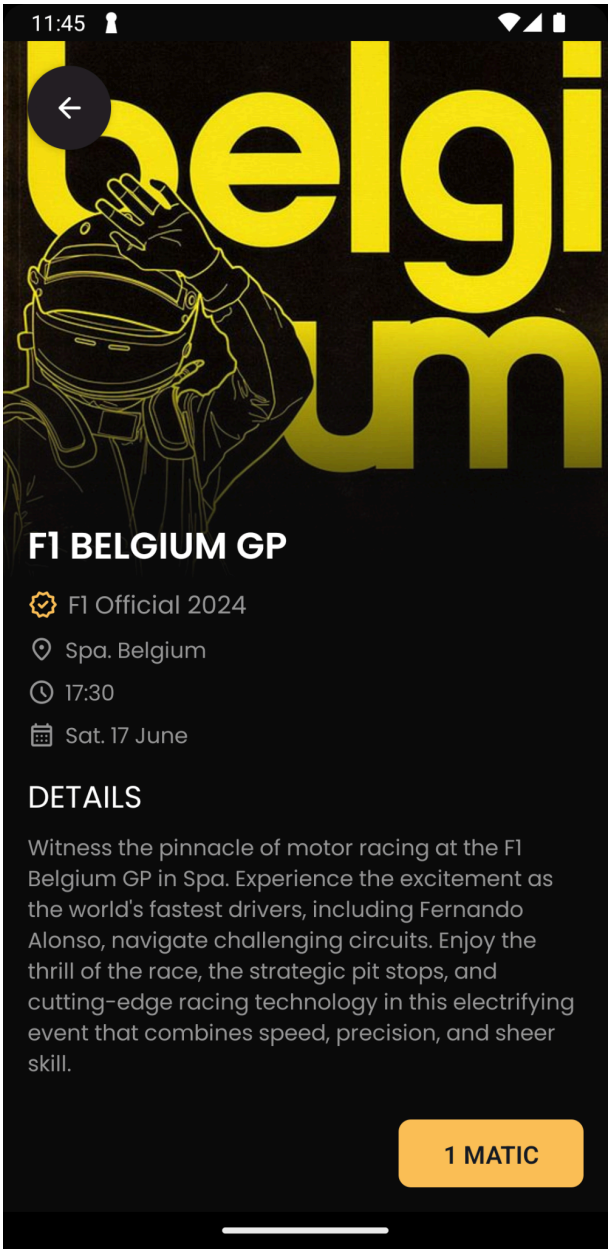


Figura 5.19

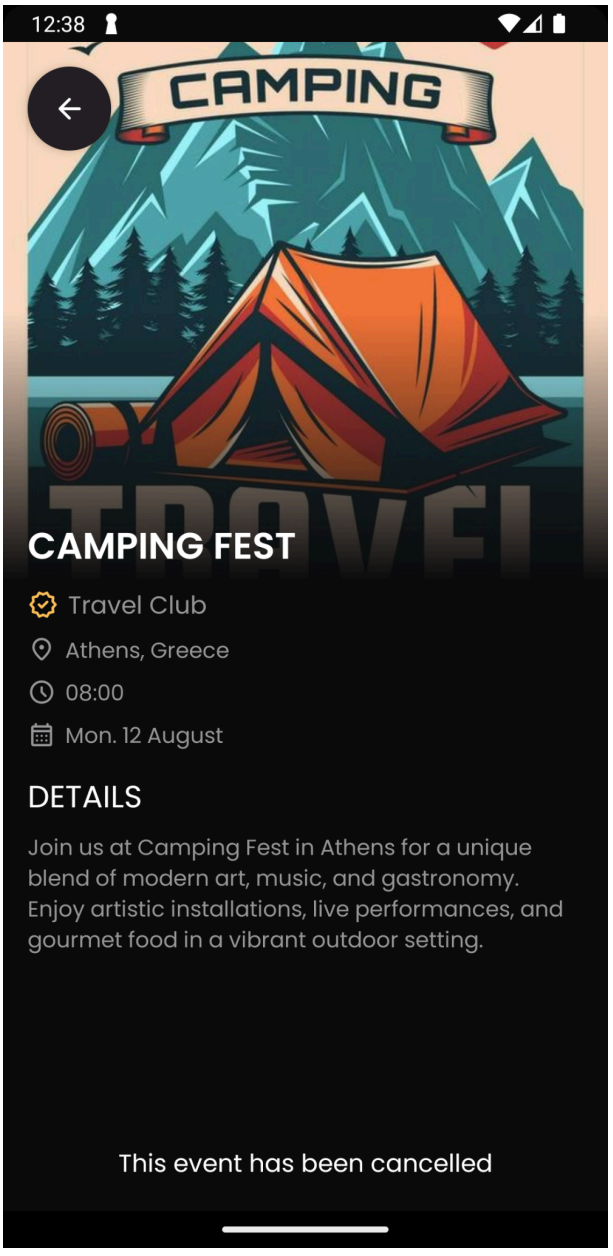


Figura 5.20

Figura 5.19: Pantalla del evento en con entrada ya adquirida. Fuente: Elaboración propia

Figura 5.20: Pantalla de evento cancelado. Fuente: Elaboración propia

Pantalla de compra realizada

Si el usuario realiza la compra de la entrada satisfactoriamente, se le redirige a una pantalla donde se muestra el detalle de esta. En esta pantalla dispone de información de la entrada adquirida y un botón de “See my tickets” para navegar a la pantalla donde aparecen sus entradas adquiridas. Ver figura 5.21.

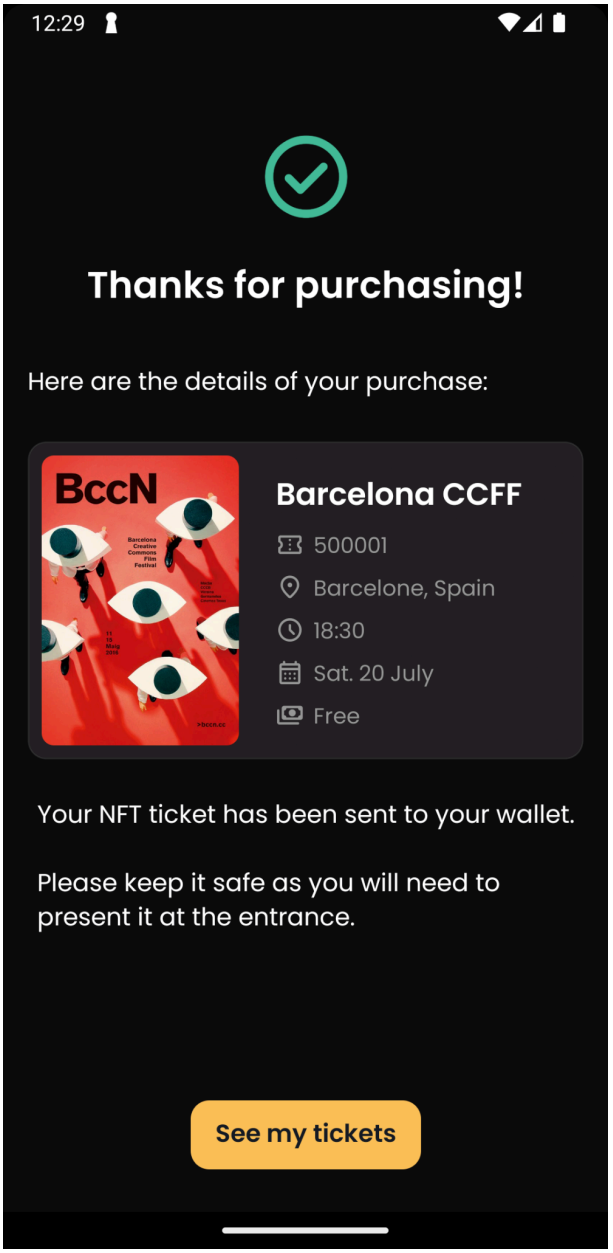


Figura 5.21: Pantalla de compra realizada. Fuente: Elaboración propia

Pantalla de las entradas adquiridas

En esta pantalla se muestran las entradas que ha adquirido el usuario. De igual modo que en la pantalla de los eventos, se pueden buscar por nombre. En esta pantalla las entradas se listan en siguiendo un orden concreto. Primero las entradas que están habilitadas para ser validadas, después las entradas que aún no están habilitadas para su validación, a continuación las que están validadas y por último aquellas entradas cuyos eventos han sido cancelados. Ver figuras 5.22 y 5.23.

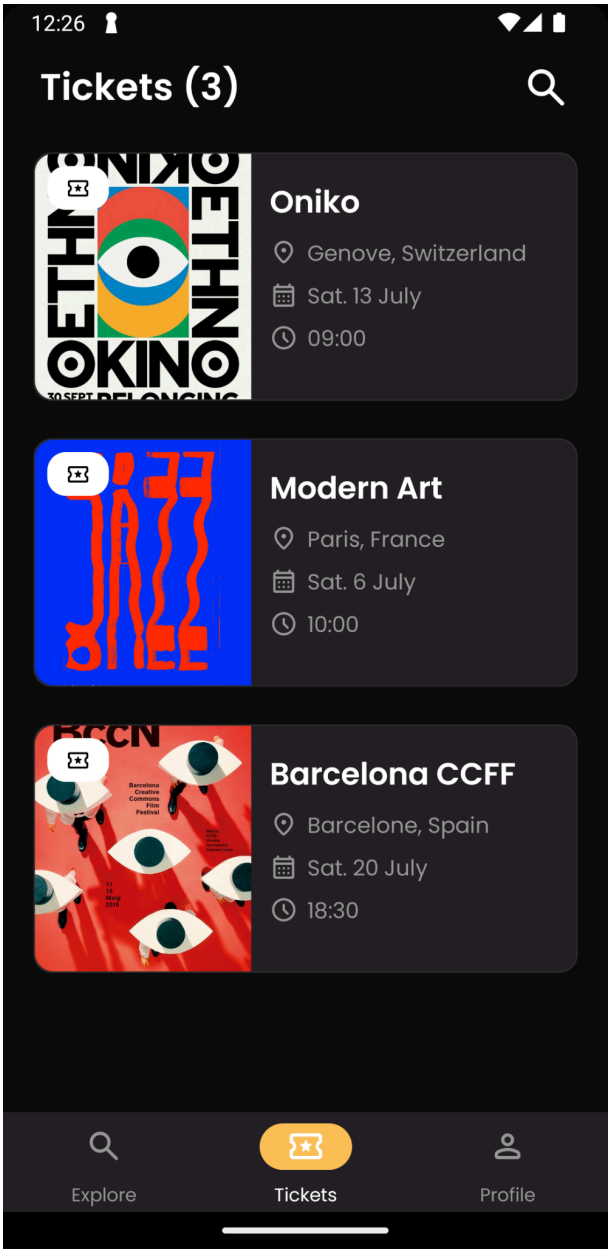


Figura 5.22

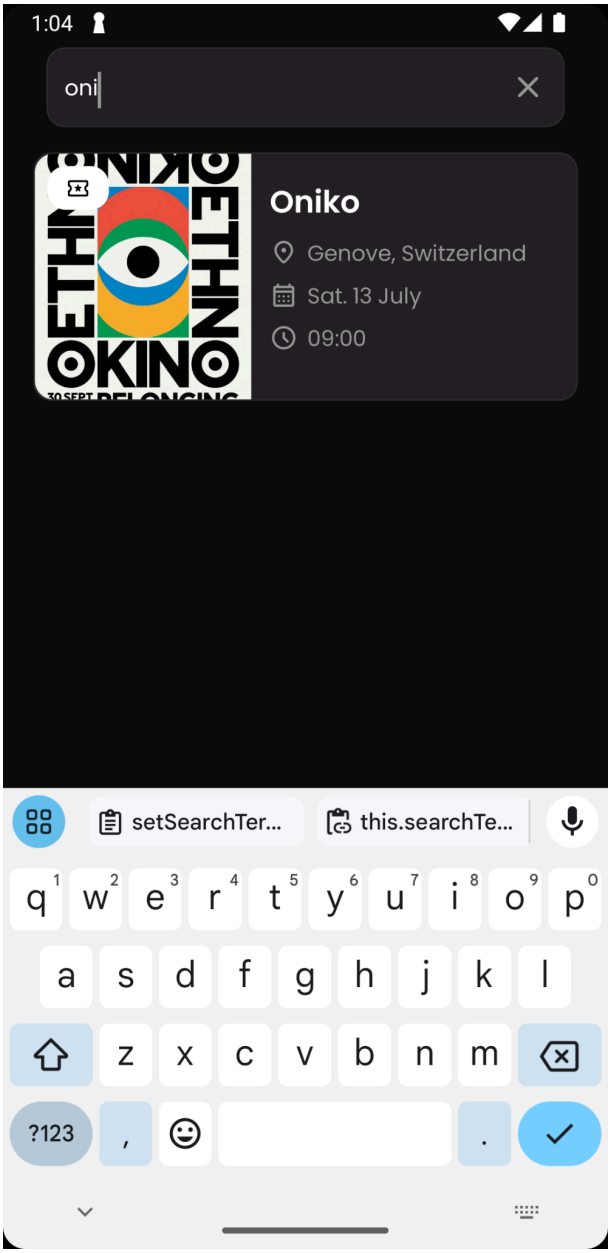


Figura 5.23

Figura 5.22: Pantalla de las entradas adquiridas. Fuente: Elaboración propia

Figura 5.23: Pantalla de las entradas adquiridas con búsqueda activada. Fuente: Elaboración propia

Pantalla-modal de información de una entrada

Cuando el usuario selecciona una de sus entradas, aparece este modal, donde se muestra la entrada seleccionada. Entre su contenido se encuentran datos del evento y el identificador de la entrada. Si la entrada está lista para ser validada, aparece un botón de “Validate” en la parte inferior para iniciar su proceso de validación.

Si, por el contrario, no está disponible para ser validada, aparece un mensaje indicando su estado y un botón de “Transfer” en la parte inferior para poder transferir la entrada si el usuario lo desea. Ver figuras 5.24 y 5.25.

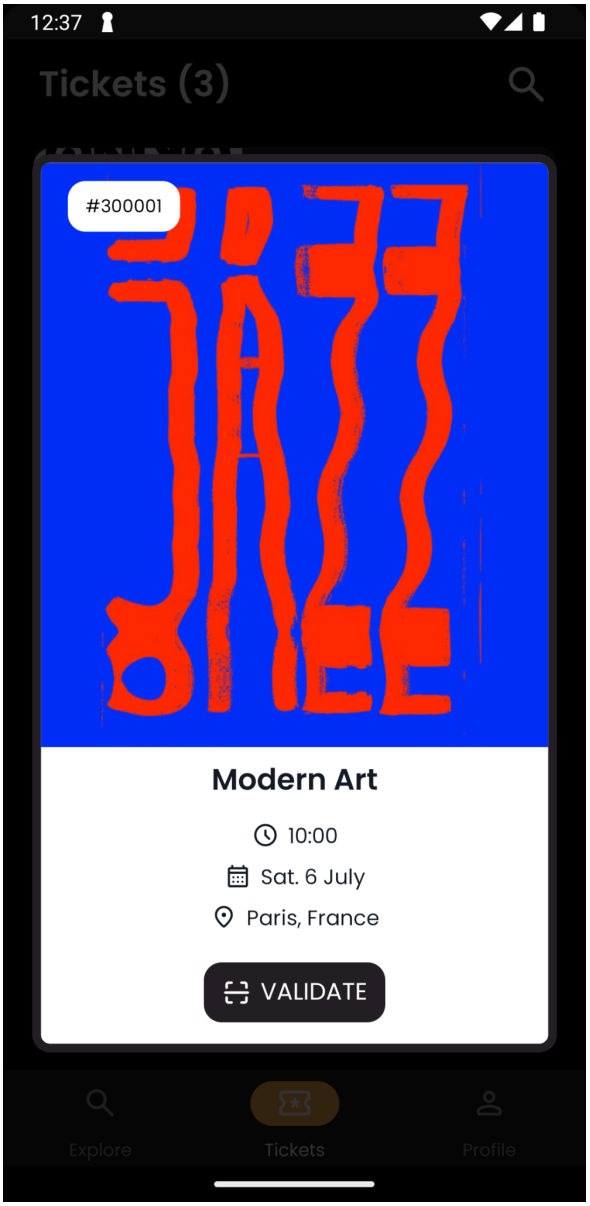


Figura 5.24

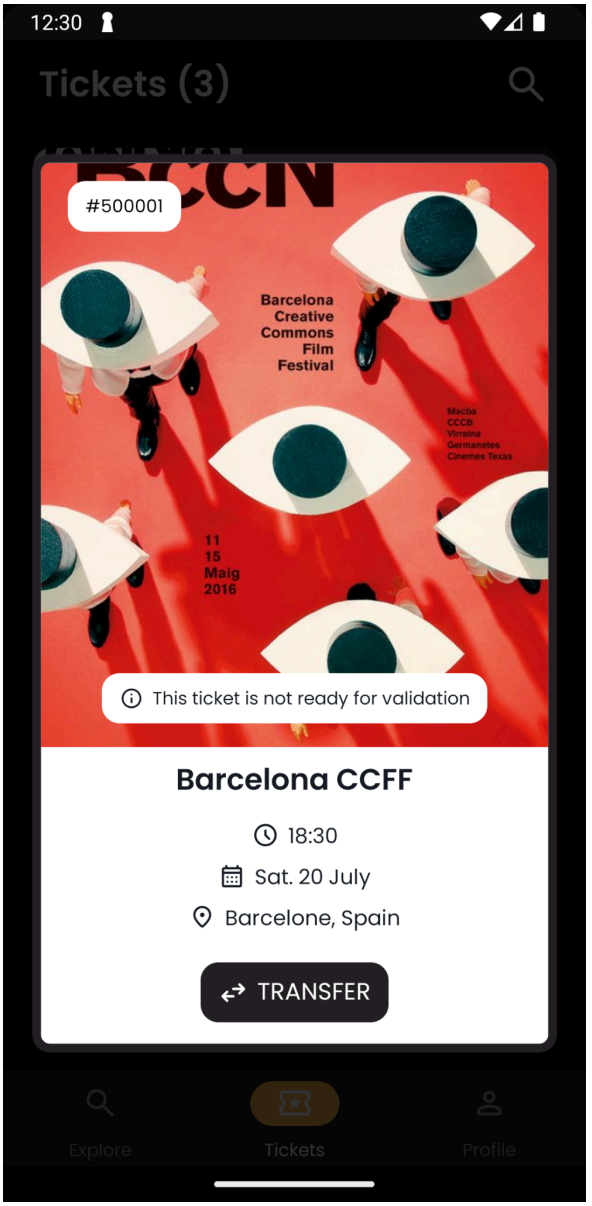


Figura 5.25

Figura 5.24: Pantalla-modal de información de una entrada lista para validar. Fuente: Elaboración propia

Figura 5.25: Pantalla-modal de información de una entrada transferible. Fuente: Elaboración propia

Pantalla-modal de selección del método de validación

Cuando el usuario procede a la validación de una entrada, primero se le permite escoger entre los dos diferentes métodos disponibles. Seleccionando la imagen que corresponde al método seleccionado, navega la pantalla de validación con este. Ver figura 5.26.

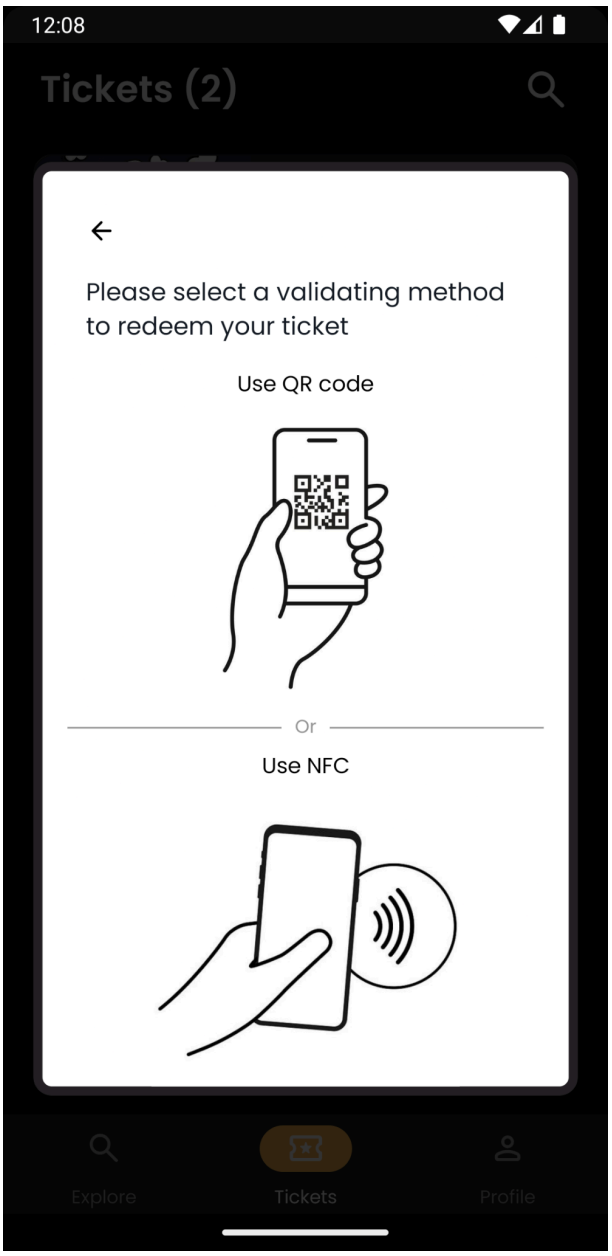


Figura 5.26: Pantalla-modal de selección del método de validación. Fuente: Elaboración propia

Pantalla-modal de validación mediante código QR

Si el usuario ha seleccionado validar mediante código QR en este paso se le muestra unas instrucciones para validar la entrada y el código de esta . Una vez es validado correctamente, se navega a la pantalla de entrada validada satisfactoriamente. Ver figura 5.27.

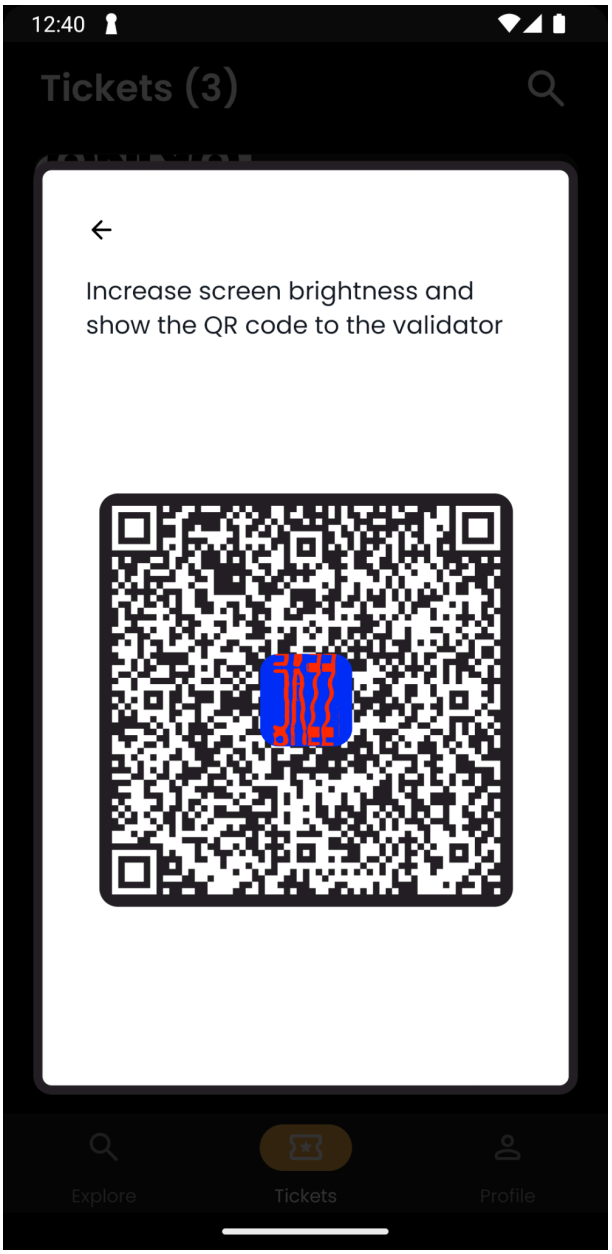


Figura 5.27: Pantalla-modal de validación mediante código QR. Fuente: Elaboración propia

Pantalla-modal de validación mediante NFC

Si el usuario ha seleccionado validar mediante NFC en este paso se le muestra una instrucción de lo que debe hacer con su dispositivo. Una vez es validado correctamente, se navega a la pantalla de entrada validada satisfactoriamente. Ver figura 5.28.

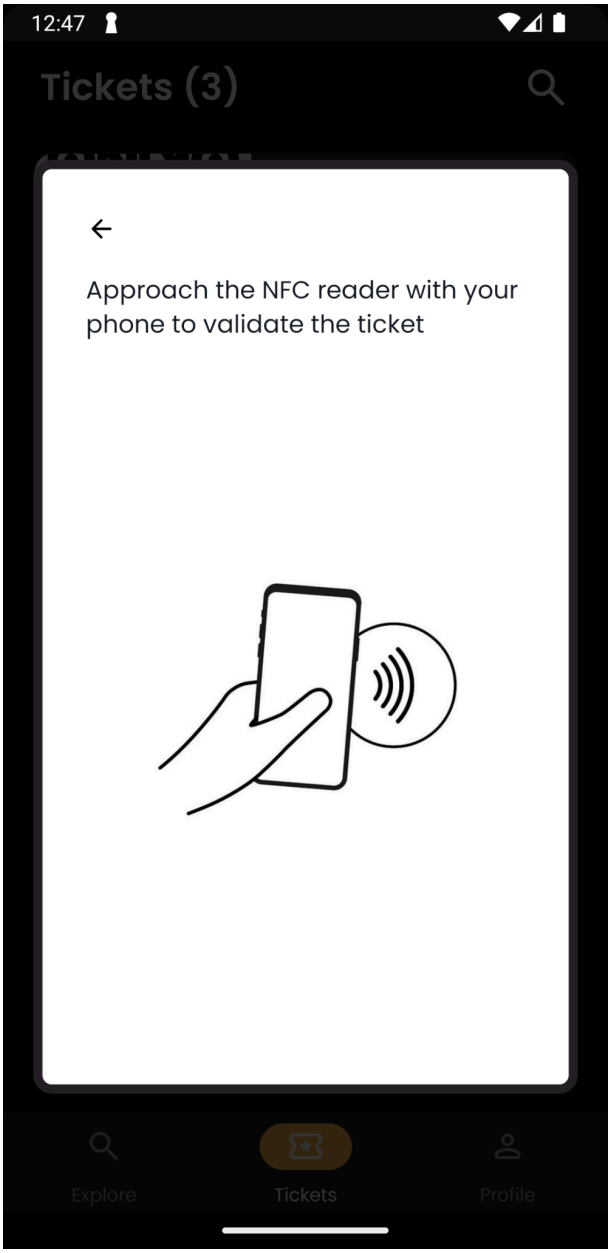


Figura 5.28: Pantalla-modal de validación mediante NFC. Fuente: Elaboración propia

Pantalla-modal de entrada validada satisfactoriamente

Si la entrada del usuario ha sido validada, se navega a la siguiente pantalla donde se le muestra un mensaje indicando que su entrada se ha validado satisfactoriamente. Ver figura 5.29.

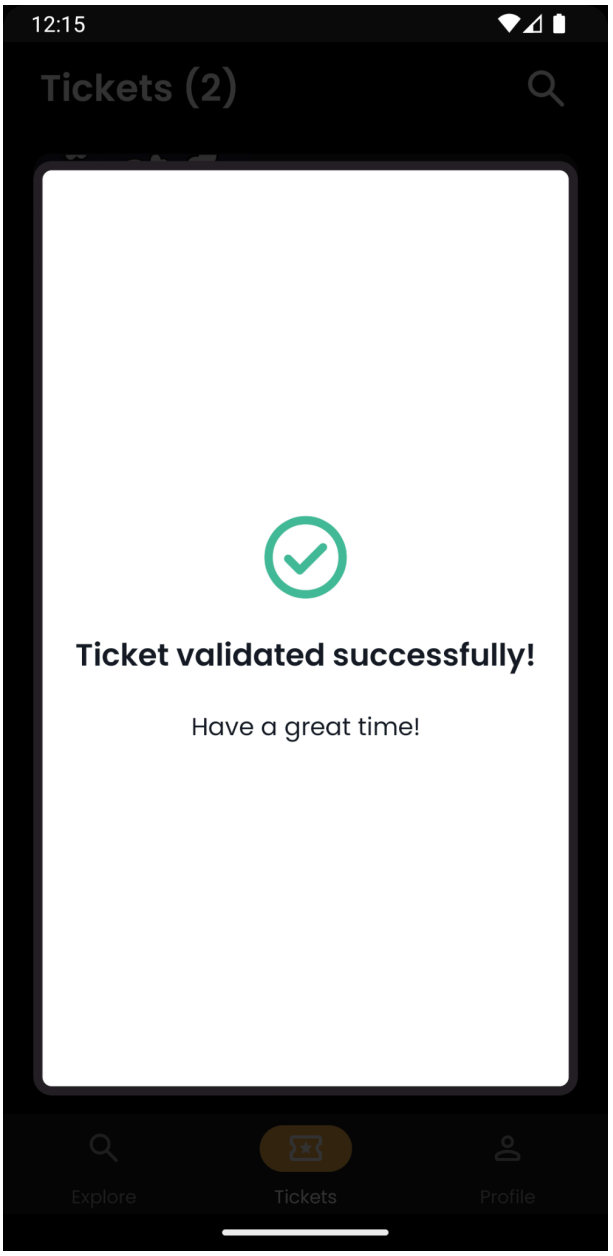


Figura 5.29: Pantalla-modal de entrada validada satisfactoriamente. Fuente: Elaboración propia

Pantalla de transferencia de una entrada

Si el usuario quiere transferir su entrada, se le muestra el siguiente formulario donde introduciendo una dirección válida de Ethereum (la de la cartera de destino) puede navegar hacia la confirmación de la transferencia.

Si el usuario pulsa el botón con el icono de portapapeles, se le pega el contenido de su portapapeles en el campo del formulario. Si la dirección es válida, se habilita un botón de “Next” para avanzar a la confirmación de la entrada. Ver figuras 5.30, 5.31 y 5.32.

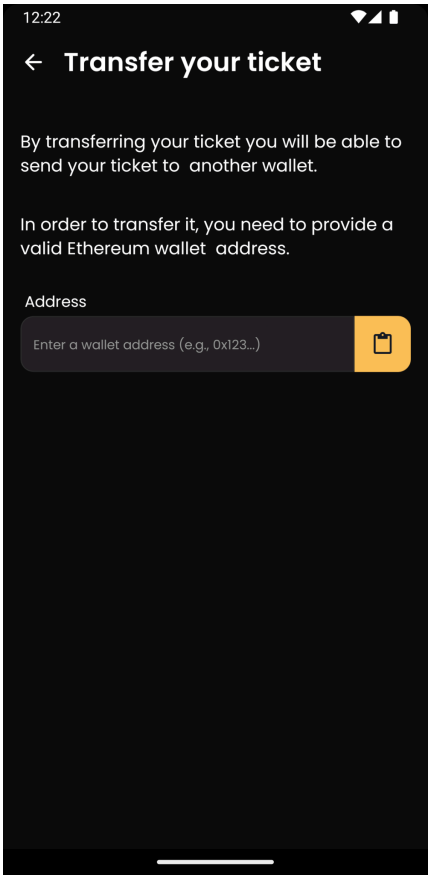


Figura 5.30

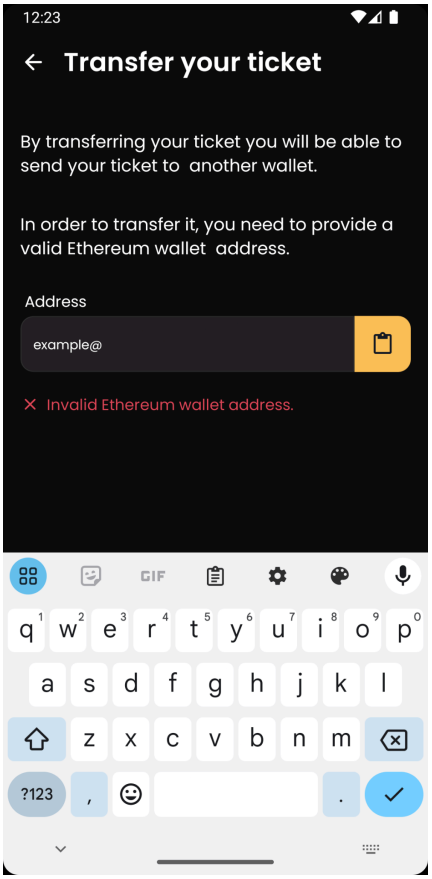


Figura 5.31

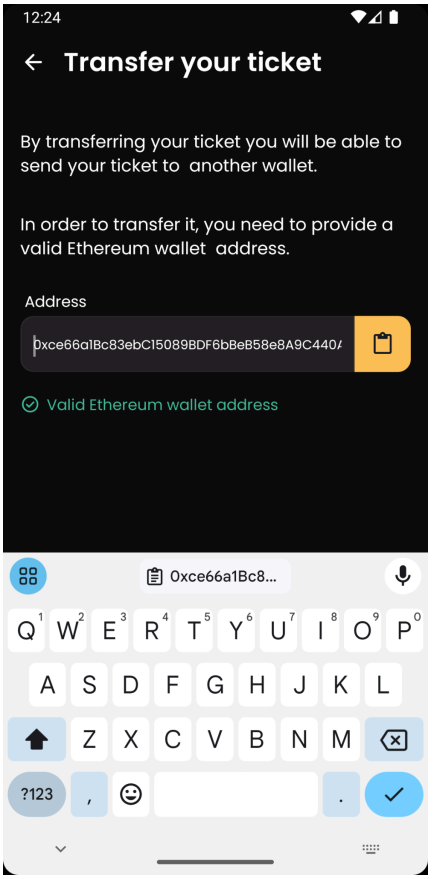


Figura 5.32

Figura 5.30: Pantalla de transferencia de una entrada. Fuente: Elaboración propia
Figura 5.31: Pantalla de transferencia de una entrada con dirección inválida. Fuente: Elaboración propia
Figura 5.32: Pantalla de transferencia de una entrada con dirección válida. Fuente: Elaboración propia

Pantalla de confirmación de transferencia

Por último se muestra una pantalla de confirmación de la transferencia a realizar. En esta pantalla se indica la entrada que se va a transferir y de qué dirección de origen hacia qué dirección de destino. Se requiere además que el usuario marque el *checkbox* de confirmación para proceder. Una vez pulsa el botón se abre la aplicación de billetera y puede firmar la transacción. Cuando esta es firmada y la transferencia realizada satisfactoriamente, se muestra un modal indicando dicho evento. Al pulsar el botón de “Close” se vuelve a la pantalla de las entradas del usuario. Ver figuras 5.33 y 5.34.

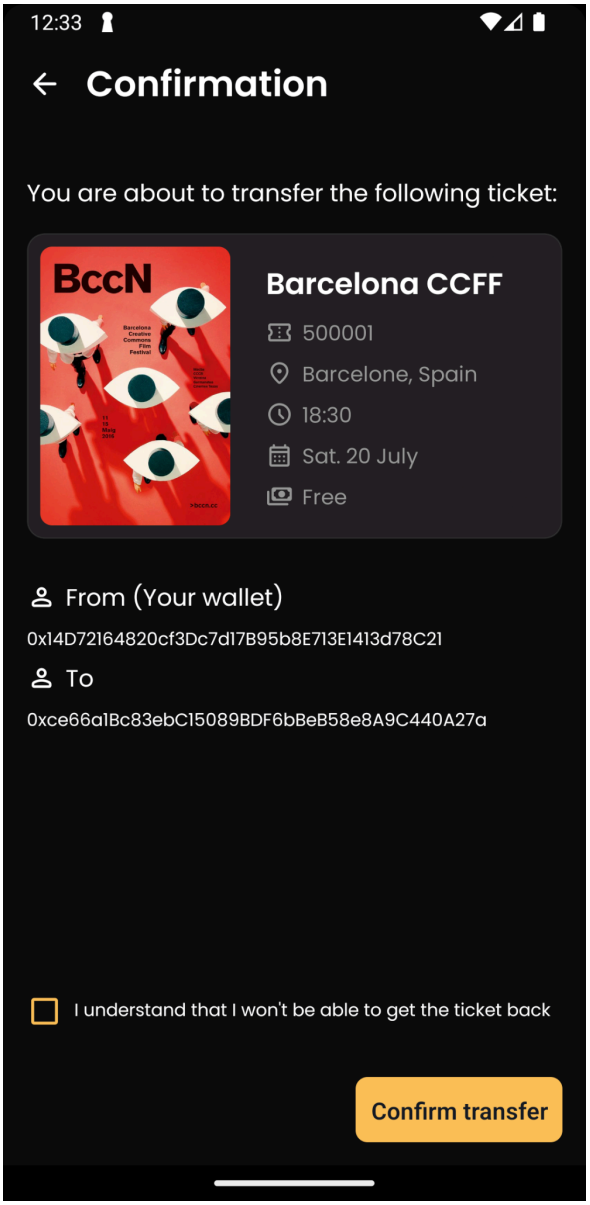


Figura 5.33

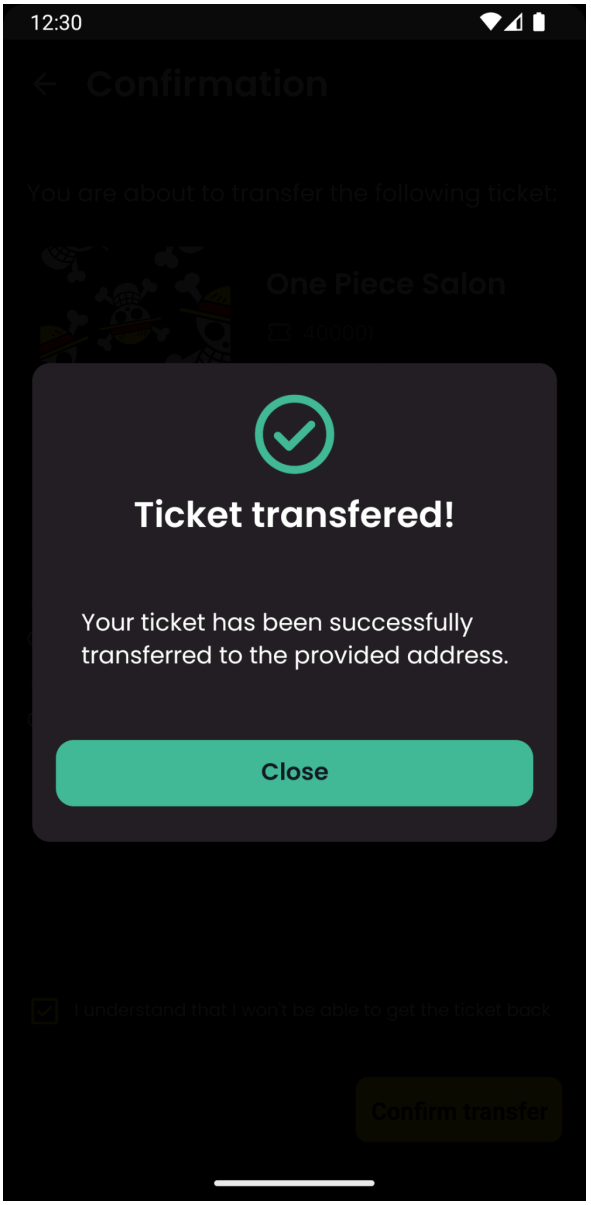


Figura 5.34

Figura 5.33: Pantalla de confirmación de transferencia. Fuente: Elaboración propia

Figura 5.34: Modal de confirmación de entrada transferida. Fuente: Elaboración propia

Pantalla del perfil del usuario

En esta pantalla se muestran datos relevantes del usuario. Aquí se pueden ver datos como la dirección de la cartera conectada, el saldo que esta contiene, la aplicación de billetera que se ha conectado y la red en la que se está operando. Nótese que aunque se le haya bautizado a esta pantalla con ese nombre, como tal la aplicación no gestiona perfiles de usuarios. Ver figura 3.35

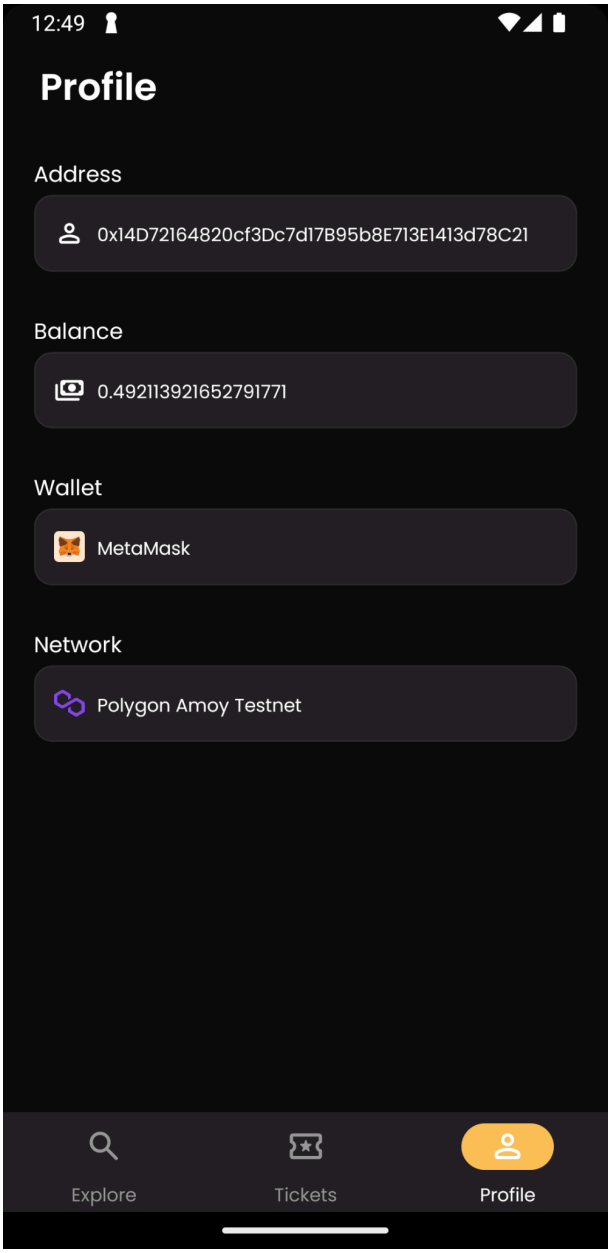


Figura 5.35: Pantalla del perfil del usuario. Fuente: Elaboración propia

Una vez mostradas las pantallas de la aplicación, en la figura 5.36 puede observarse el diseño resultante.

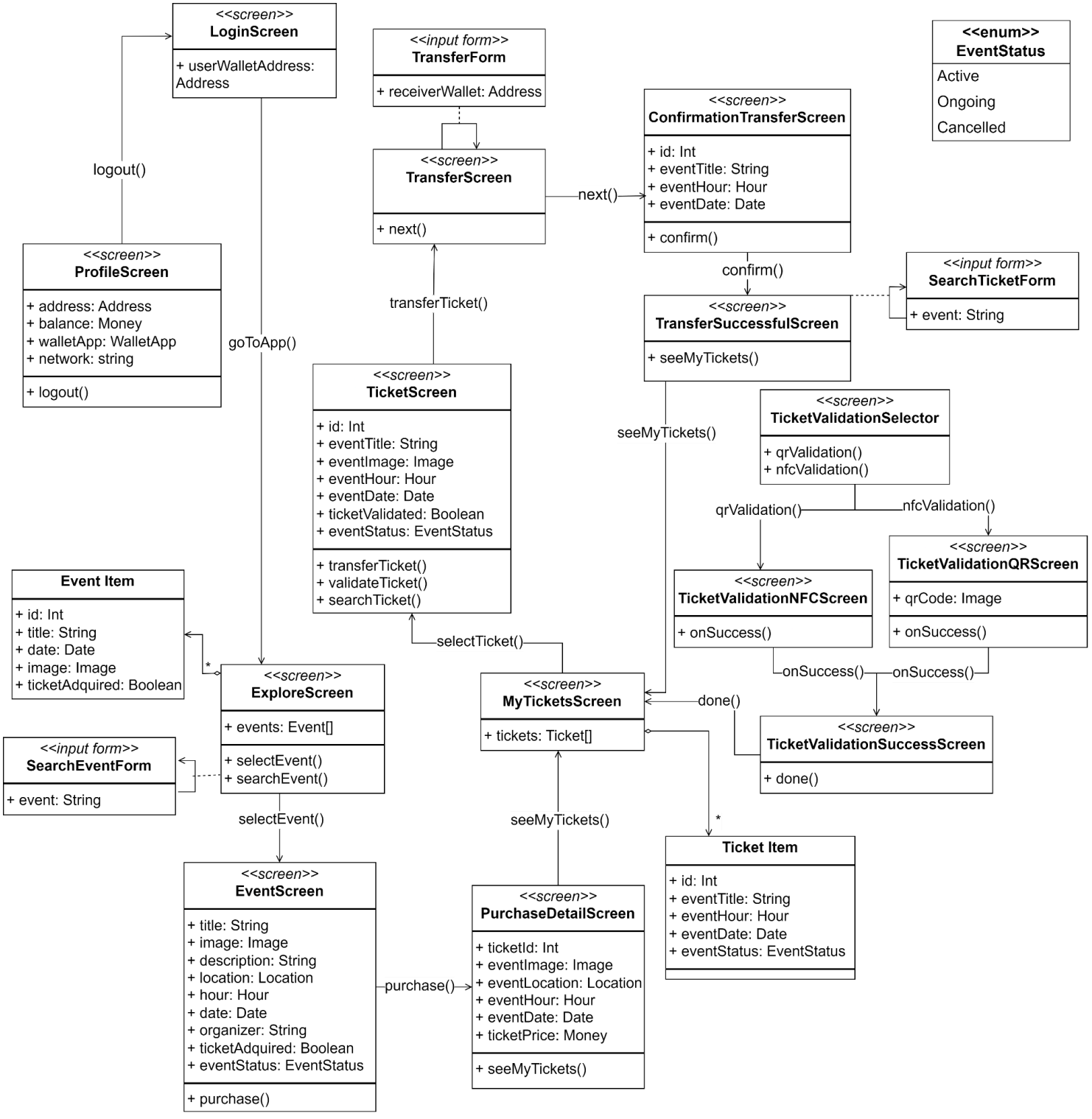


Figura 5.36: Diseño externo de la aplicación de los asistentes. Fuente: Elaboración propia

5.6.2 Interfaz de la aplicación de los validadores

Pantalla de selección de validación del evento

Esta es la pantalla inicial de la aplicación. En esta pantalla se muestran los eventos que actualmente están en curso bajo el título de “On going”. El usuario puede seleccionar el evento que al cual va a validar las entradas. Si lo hace, navega a la pantalla de selección del método de validación. Ver figura 5.37.

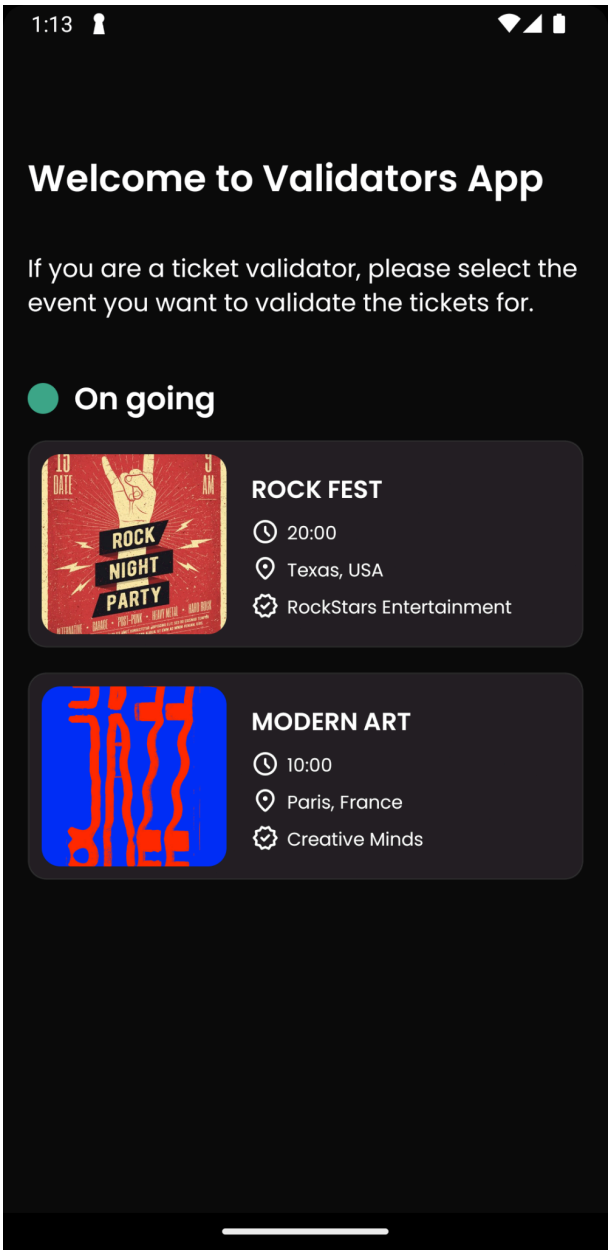


Figura 5.37: Pantalla de selección de validación del evento. Fuente: Elaboración propia

Pantalla de selección del método de validación

En esta otra pantalla el usuario selecciona con qué método va a validar las entradas. Además de ello se le dan unas indicaciones remarcando que podrá cambiar el método de validación en cualquier momento. Dispone de dos botones. El primero de “Use QR” que al pulsarlo navega a la pantalla de validación mediante código QR y el segundo de “Use NFC” que al pulsarlo navega a la pantalla de validación mediante NFC. Ver figura 5.38.

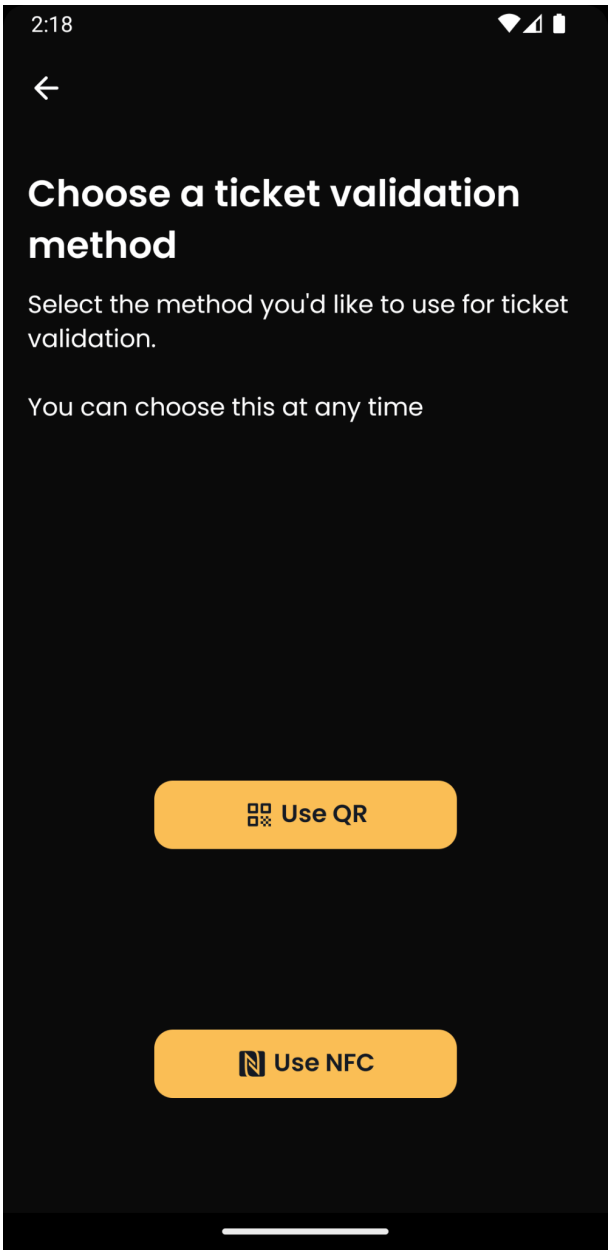
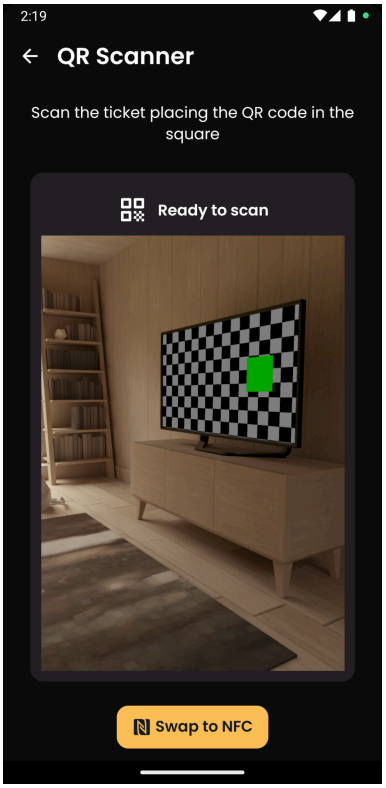


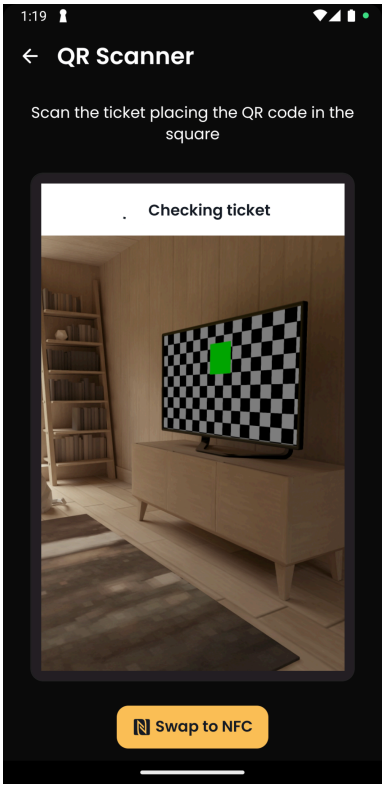
Figura 5.38: Pantalla de selección del método de validación. Fuente: Elaboración propia

Pantalla de validación mediante código QR

En esta pantalla se muestra una ventana donde el validador puede ver donde su cámara está apuntando. En esta misma ventana, para ayudar con la validación de la entrada, se le muestra en la parte superior el estado de la validación. “Ready to scan” indica que la aplicación está lista para escanear un nuevo código QR, “Checking status” indica que se está comprobando que la entrada sea válida. Si esta es válida, el estado cambia a “Ticket validated”, si esa entrada ya se ha validado previamente cambia al estado de “Ticket already validated” y si, por el contrario, esa entrada no corresponde al evento o no se ha escaneado siquiera un código QR se cambia al estado de “Error validating ticket”. Finalizado el escaneo, sea válido o erróneo, regresa automáticamente al estado inicial.

Por último, si nos fijamos en la parte inferior de la pantalla, se ha incluido un botón para cambiar a directamente la validación por NFC. El motivo de incluir este botón en esa parte es por una cuestión de usabilidad, ya que en un contexto donde acuden asistentes queriendo validar su entrada con diferentes métodos, los pasos que deben realizarse en la aplicación deberían ser los mínimos. También la posición del botón en ese lugar es por un tema de accesibilidad para el usuario. Ver figuras 5.39, 5.40, 5.41, 5.42 y 5.43.





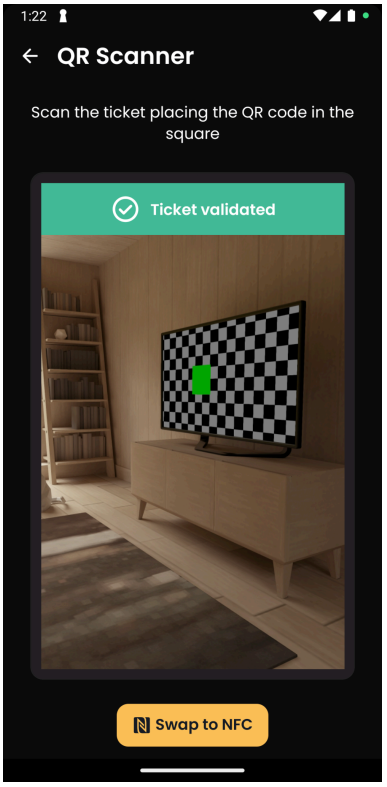
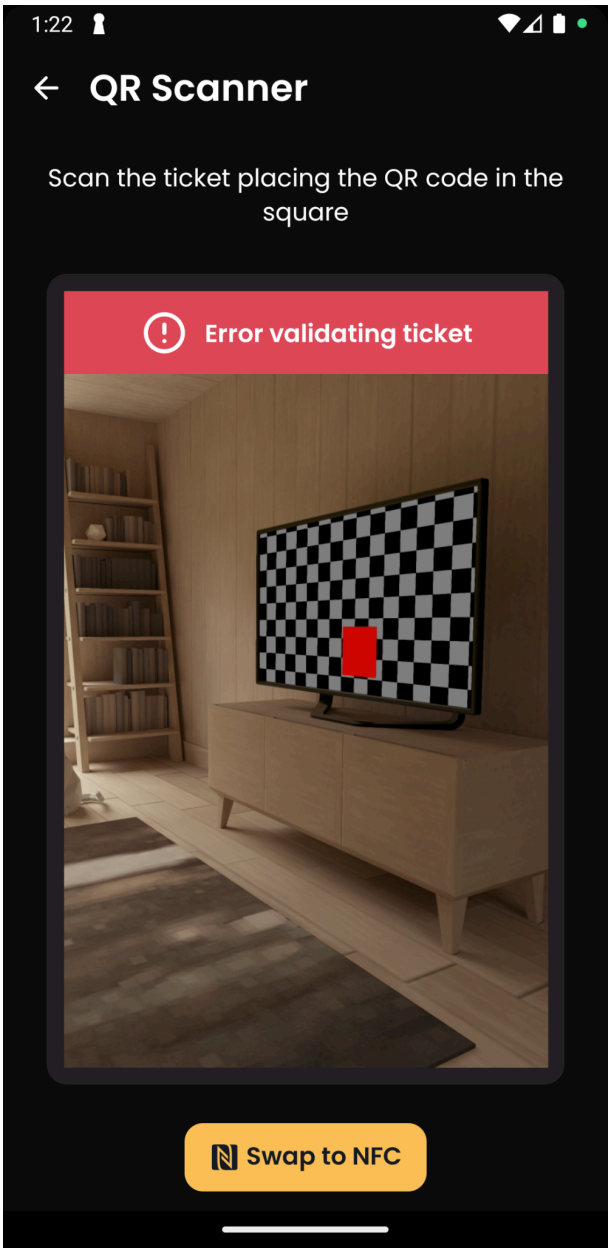


Figura 5.39 **Figura 5.40** **Figura 5.41**

Figura 5.39: Pantalla de validación mediante QR en “Ready”. Fuente: Elaboración propia

Figura 5.40: Pantalla de validación mediante QR en “Checking ticket”. Fuente: Elaboración propia

Figura 5.41: Pantalla de validación mediante QR en “Ticket Validated”. Fuente: Elaboración propia



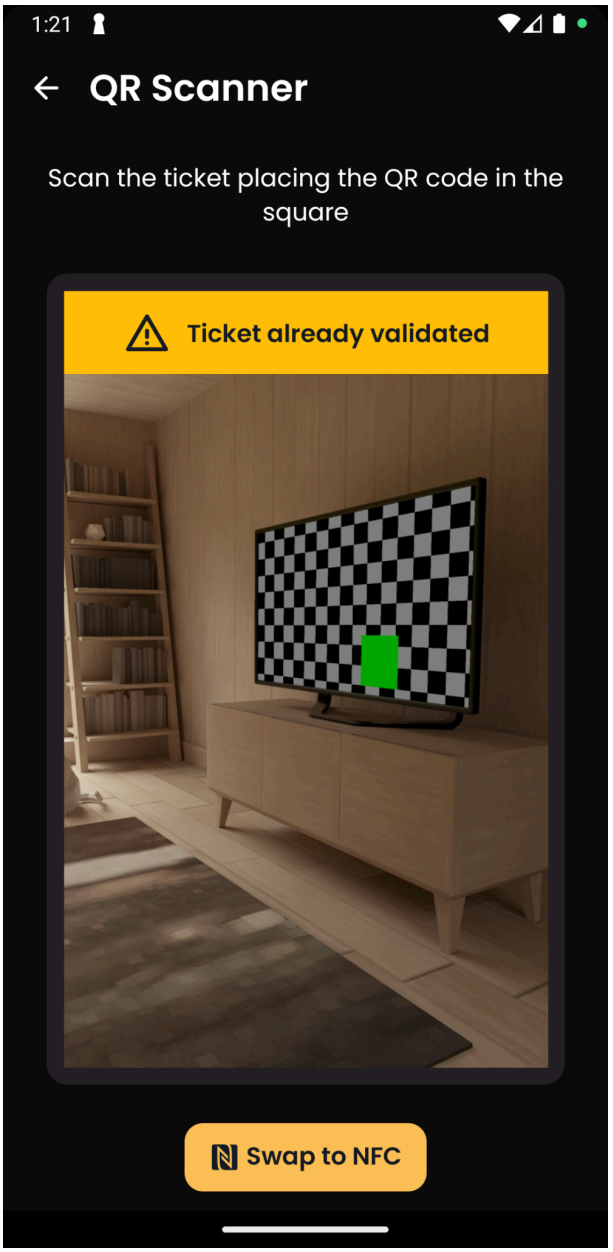


Figura 5.42

Figura 5.43

Figura 5.42: Pantalla de validación mediante QR en “Error validating ticket”. Fuente: Elaboración propia

Figura 5.43: Pantalla de validación mediante QR en “Ticket already validated”. Fuente: Elaboración propia

Pantalla de validación mediante NFC

Esta pantalla incluye unas instrucciones para indicar al validador como debe usar el escaneo NFC. Debajo de estas se encuentra un botón que al pulsarlo activa la lectura NFC. Este botón muestra los mismos estados que se muestran en el texto de la pantalla de validación mediante QR.

Igual que en la pantalla anterior, se dispone de un botón para navegar directamente a la pantalla de escaneo mediante QR. Ver figuras 5.44, 5.45, 5.46, 5.47, 5.48 y 5.49.

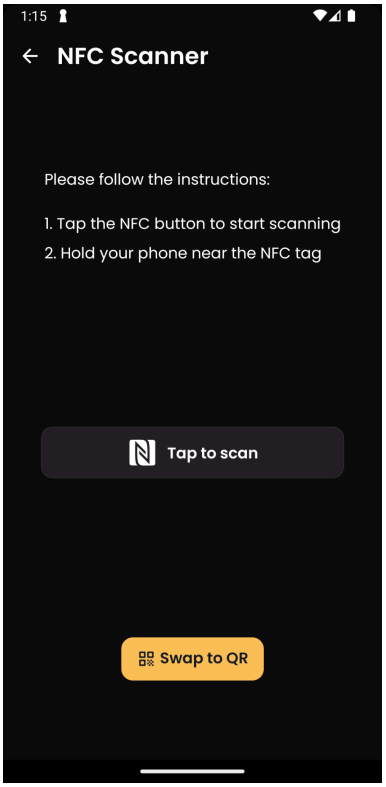


Figura 5.44

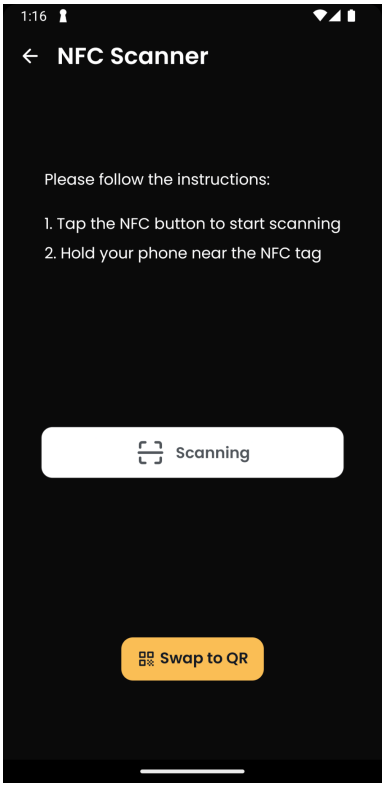


Figura 5.45

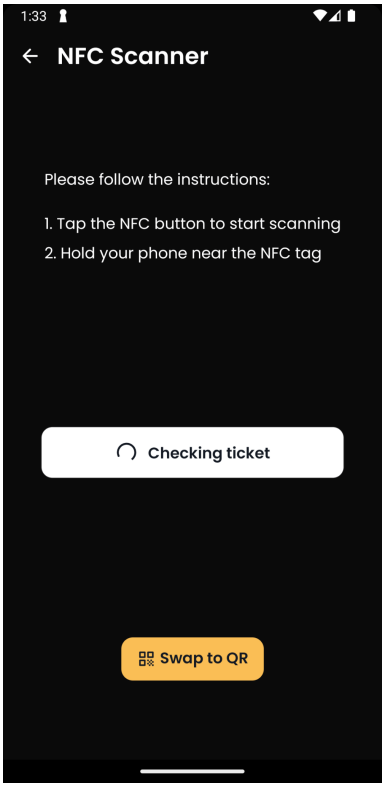
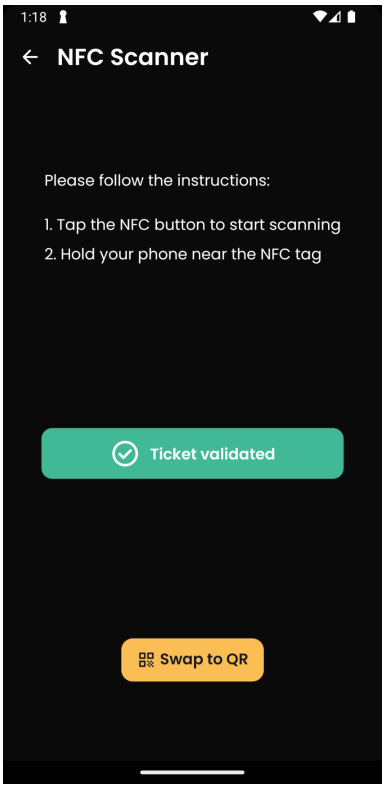
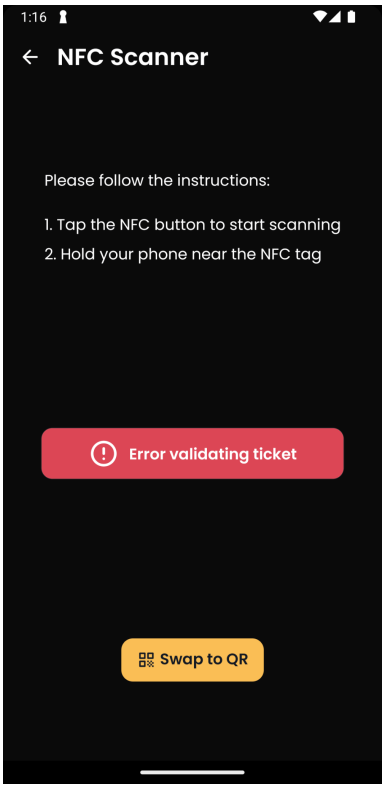


Figura 5.46

Figura 5.44: Pantalla de validación mediante NFC en “Tap to scan”. Fuente: Elaboración propia
Figura 5.45: Pantalla de validación mediante NFC en “Scanning”. Fuente: Elaboración propia
Figura 5.46: Pantalla de validación mediante NFC en “Checking ticket”. Fuente: Elaboración propia





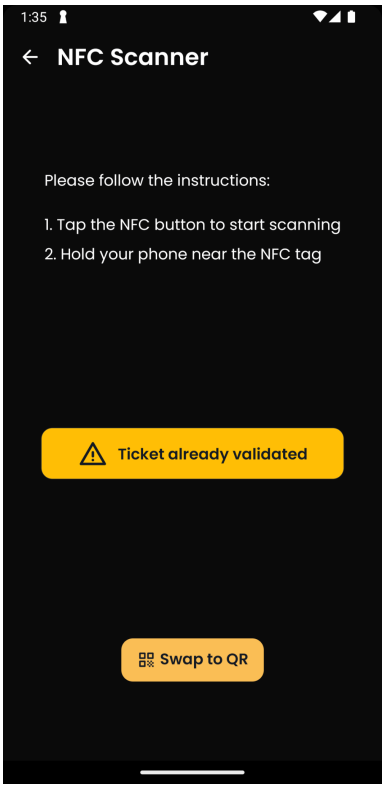


Figura 5.47 **Figura 5.48** **Figura 5.49**

Figura 5.47: Pantalla de validación mediante NFC en “Ticket validated”. Fuente: Elaboración propia

Figura 5.48: Pantalla de validación mediante NFC en “Error validating ticket”. Fuente: Elaboración propia

Figura 5.49: Pantalla de validación mediante NFC en “Ticket already validated”. Fuente: Elaboración propia

Y por último, en la figura 5.50 se muestra, un diagrama más sencillo, para la aplicación de los validadores:

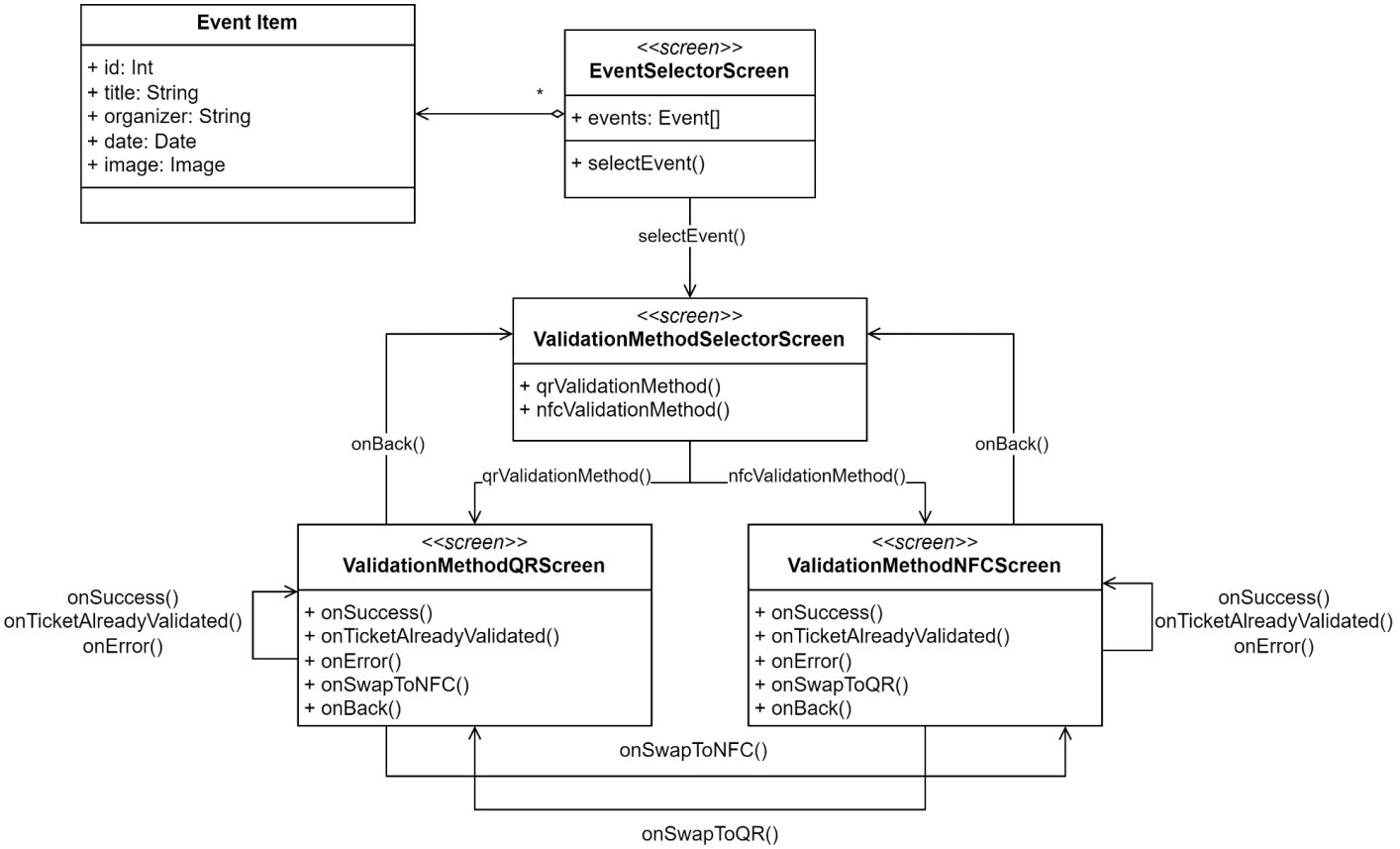


Figura 5.50: Diseño externo de la aplicación de los validadores. Fuente: Elaboración propia

6. Implementación

En este capítulo se detallan los aspectos más importantes relativos a la implementación del proyecto. Primeramente, se verán las tecnologías y lenguajes de programación utilizados, más adelante se muestran las herramientas para el desarrollo y finalmente se muestran algunas piezas de código relevantes en la implementación.

6.1 Tecnologías y lenguajes de programación utilizados

Como se ha detallado con anterioridad, el sistema está compuesto por dos aplicaciones móviles, un *smartcontract* alojado en la *blockchain* de Ethereum y un *backend* donde una API hace de intermediario entre el *smartconctract* y la aplicación de los validadores. A continuación se detallan las tecnologías que se han utilizado para implementar cada uno de estos componentes.

6.1.1 Aplicaciones móviles - React Native con TypeScript

Estas aplicaciones se han desarrollado utilizando React Native y TypeScript. La elección de React Native, detallada en el capítulo 5.1 Arquitectura física, se debe a que era el único *framework* compatible con las herramientas de desarrollo web3 que habían demostrado funcionar correctamente. Inicialmente, se consideró también Kotlin Multiplatform [33] (un *framework* de Kotlin en auge), pero se optó por React Native debido a varias ventajas específicas.

Una ventaja destacada de React Native es la capacidad de ver los cambios en la aplicación sin necesidad de recompilar todo el proyecto, gracias a su característica de hot reloading. Además, React Native cuenta con una extensa colección de librerías externas desarrolladas por la comunidad, lo que facilita la integración de funcionalidades adicionales. Por ejemplo, se han utilizado librerías para el escaneo mediante QR y NFC.

La elección de este *framework* también se ha basado en la experiencia previa con React Native en pequeños proyectos personales y en el uso de React para desarrollo web. React y React Native comparten muchas características debido a su base común en JavaScript, lo que hace la transición entre ambos más fluida.

Para mejorar la experiencia de desarrollo y la calidad del código, se ha utilizado TypeScript en lugar de JavaScript como lenguaje principal. TypeScript es un superconjunto de JavaScript que añade tipos estáticos opcionales y se transpila a JavaScript. Esto permite un tipado fuerte en el código, proporcionando mayor robustez. Una ventaja clave de TypeScript es la detección de errores en tiempo de compilación, en contraste con JavaScript, que lo hace en tiempo de ejecución. Esto ahorra numerosas horas de depuración. El tipado fuerte también contribuye a un código más escalable, mantenible y legible [34].

6.1.2 Smartcontract - Solidity

El *smartcontract* del sistema ha sido desarrollado con Solidity como lenguaje de programación y Hardhat como entorno de desarrollo. Esta combinación ofrece una experiencia robusta y eficiente para la creación, despliegue y pruebas del *smartcontract*.

La elección de Solidity viene dada porque dentro de los lenguajes de programación compatibles con la EVM es el más común, con mayor soporte y documentación. Además de ello, es un lenguaje con una sintaxis muy similar a otros lenguajes orientados a objetos como pueden ser C++ o Java lo cual para un estudiante del grado es fácil de comprender.

Por otro lado, se ha optado por Hardhat por su extensa documentación, soporte y ejemplos disponibles. Este entorno proporciona un desarrollo robusto y eficiente al permitir escribir, compilar y probar el *smartcontract* localmente sin necesidad de desplegarlo [35]. Además, Hardhat ofrece la capacidad de escribir y ejecutar pruebas automatizadas contra el *smartcontract* utilizando TypeScript, lo cual ha sido esencial para garantizar la calidad y seguridad del código. No obstante, para este proyecto, se podría haber optado por otro entorno de desarrollo como puede ser Truffle [36].

6.1.3 Servidor *backend* - Node.js con Express.js

Para la implementación del *backend*, se ha optado por Node.js debido a su capacidad para manejar eficientemente operaciones de entrada/salida y su amplio ecosistema de módulos. Además, algunas bibliotecas clave como Ethers.js [37] y web3.js [38] para conectarse al *smartcontract* estaban disponibles para este entorno de ejecución.

Para gestionar las llamadas HTTP y desarrollar la API REST del sistema, se había elegido Express.js debido a su facilidad de uso y adecuación para este propósito. Express.js proporcionaba una solución simple pero efectiva para manejar rutas, *middleware* y solicitudes HTTP en la aplicación. Además, se ha utilizado TypeScript en el *backend* debido a las ventajas que ofrece como se ha comentado anteriormente.

6.2 Herramientas de desarrollo

En esta sección se enumeran las diferentes herramientas que se han utilizado durante el desarrollo del proyecto.

6.2.1 Visual Studio Code

Este IDE de Microsoft, se ha utilizado para el desarrollo de todos los componentes del sistema. El principal motivo de su uso es la personalización que permite (colores, fuentes, temas, extensiones, formateo de código, etc.), la integración con GitHub y el reconocimiento de tipos de TypeScript. A

mediados del desarrollo se ha utilizado también la herramienta de inteligencia artificial de GitHub Copilot [39]. Si bien esta inteligencia artificial, no genera muy buen código a voluntad, sí que ayuda en el autocompletado de líneas o resolución de errores.

6.2.2 Android Studio

Este IDE únicamente se ha utilizado para instalar un emulador de Android donde poder ejecutar las aplicaciones móviles mientras se desarrollaba el proyecto.

6.2.3 *Testnets* y *faucets* de criptomonedas

Durante el desarrollo del proyecto, se ha ido cambiando la blockchain de pruebas que se ha utilizado para desplegar el *smartcontract* e interactuar con él.

Inicialmente, se consideró utilizar la *testnet* de Polygon Mumbai. Durante los primeros meses esta funcionó correctamente, pero a principios de abril entró en desuso y la alternativa que ofrecía Polygon era Amoy. El problema de Amoy era que el único verificador de *smartcontracts* de esta red en su momento, OkLink, no funcionaba correctamente. Si el *smartcontract* no se verifica, no puede ser utilizado, de modo que había que buscar otra alternativa.

Es por ello que se migró el desarrollo a la *testnet* de Binance y gran parte del desarrollo se hizo sobre esta. No obstante, el único *faucet* que proporcionaba *tokens* dejó de suministrarlos y se tuvo que migrar a otra nueva *testnet* por miedo a quedarse sin fondos durante el desarrollo.

Durante unas semanas se utilizó la red de Optimism Sepolia. Los *tokens* para esta *testnet* se obtienen minando, ya que se basa en *Proof of Work (PoW)* [40]. Esto trae consigo algunos inconvenientes y es que la velocidad de las transacciones era inferior a las realizadas en las otras testnets.

Más adelante, Polygon desplegó un verificador de *smartcontracts* para Amoy que sí funcionaba correctamente. Finalmente, el desarrollo se ha proseguido con este, puesto que hay *tokens* disponibles en los *faucets* y la *testnest* es rápida.

6.3 Aspectos relevantes del código de la implementación

6.3.1 Gestión de los eventos y entradas en el *smartcontract* de Solidity

El *smartcontract* es una pieza clave en todo el sistema desarrollado en este proyecto. Es por ello que este *smartcontract* debe disponer de las funcionalidades necesarias para gestionar correctamente los eventos y entradas del sistema.

Con el fin de conseguir este objetivo se ha utilizado el estándar de NFT ERC721. Utilizando la librería de Openzelin se permite la posibilidad de hacer que el *smartcontract* implementando herede de otro que permita la manipulación de NFTs [41]. Este es el ERC721URIStorage. Parte de las funciones de este incluyen el *minteo* (cuando un usuario compra una entrada), la transferencia de *tokens* (cuando un usuario transfiere su entrada a otro) y la consulta de la propiedad de ese NFT.

A continuación se explican las partes más relevantes para la implementación del *smartcontract*:

Modifier "*onlyOwner*"

Un *modifier* en Solidity es una pieza de código que se puede reutilizar en varias funciones para añadir lógica común de manera sencilla. En el caso del *modifier "*onlyOwner" implementado en este *smartcontract*, es utilizado para restringir el acceso a ciertos métodos, permitiendo que solo el propietario del *smartcontract* pueda ejecutarlas. Esto garantiza que ciertas funciones no puedan ser ejecutadas por aquellos que no tienen la condición de ser propietarios del *smartcontract*. Cabe decir que otra manera de gestionar la autorización de ejecución de ciertos métodos podría ser usando roles, pero en el caso de este proyecto ya es suficiente con este simple *modifier*. En el bloque de código de la figura 6.1 se muestra un ejemplo de como se utiliza:

```
Unset
function validateTicket(uint256 eventId, uint256 ticketId, address userAddress)
public onlyOwner {
        // Código omitido
}
```

Figura 6.1: Bloque de código ejemplo de *modifier onlyOnwer* en la función *validateTicket()*.

Fuente: Elaboración propia

Función de *minteo (_safeMint)*

La función *_safeMint* es una función interna heredada de la implementación del estándar ERC721 de OpenZeppelin. Se utiliza para crear un nuevo *token* NFT y asignarlo a una dirección específica de manera segura. En la implementación del *smartcontract*, esta función es llamada cuando el usuario quiere adquirir una entrada. Para *mintear* el *token*, se llama a esta función pasándole como parámetro la dirección del usuario quiere comprar la entrada (para hacerlo propietario de esta) y el identificador de la entrada. En la figura 6.2 se muestra un bloque de código de ejemplo:

```
Unset
function mint(uint256 _occasionId) public payable {
  // Código omitido
  uint256 ticketId = _occasionId * 100000 +occasions[_occasionId].purchasedTickets;
  userTickets[msg.sender].push(Ticket(ticketId, _occasionId, false));
  _safeMint(msg.sender, ticketId);
 }
```

Figura 6.2: Bloque de código ejemplo de *_safeMint* en la función *mint()*. Fuente: Elaboración propia

Función de transferencia *(transferFrom)*

La función *transferFrom* es una función pública heredada del estándar ERC721. Permite a los propietarios de los *tokens* transferirlos a otras direcciones. Esta función, dentro del contexto del *smartcontract* se utiliza para transferir las entradas entre diferentes billeteras. En la figura 6.3 se muestra un bloque de código de ejemplo:

```
Unset
function transferTicket(address to, uint256 tokenId, uint256 occasionId) public {
    // Código omitido
    Ticket storage ticket = userTickets[msg.sender][ticketIndex];
    require(!ticket.redeemed, "This user has already redeemed this ticket");
    transferFrom(msg.sender, to, tokenId);
    removeTicketFromUser(msg.sender, tokenId);
    // Código omitido
}
```

Figura 6.3: Bloque de código ejemplo de *transferFrom()* en la función *transferTicket()*.

Fuente: Elaboración propia

Función de "quema" *(_burn)*

La función _burn es una función interna heredada del estándar ERC721 de OpenZeppelin. Permite destruir un *token* específico. En la implementación del código se utiliza cuando se cancela un evento. Los *tokens minteados* como entradas son destruidos y posteriormente las billeteras poseedoras les son devueltos sus fondos. En la figura 6.4 se muestra un bloque de código de ejemplo:

```
Unset
function refundUsers(uint256 occasionId) private onlyOwner {
     // Código omitido
     Ticket[] memory tickets = userTickets[assistant];
     for (uint j = 0; j < tickets.length; j++) {
       Ticket memory ticket = tickets[j];
       if (ticket.occasionId == occasionId) {
           _burn(ticket.id);
           hasBought[occasionId][assistant] = false;
       }
     }
 }
```

Figura 6.4: Bloque de código ejemplo de ***_burn()*** en la función *refundUsers()*.

Fuente: Elaboración propia

Función de comprobación del propietario (*ownerOf*)

Esta función perteneciente al estándar ERC721 permite ver quién es el propietario de un token *minteado* en el smartcontract pasándole como parámetro el identificador del token. En este caso, esta función no se implementa en el *smartcontract, pero* se hace uso de esta desde la aplicación de los asistentes para ver si un usuario ha adquirido una entrada para cierto evento.

Funciones auxiliares

Por último, cabe destacar que la gestión de los eventos y entradas requieren de otras estructuras de datos y funciones para satisfacer algunas funciones.

Un ejemplo de esto sucede cuando se transfiere una entrada de un usuario a otro. Para este caso se implementa una estructura de datos que guarda las entradas adquiridas por una dirección de Ethereum. Cuando una entrada se transfiere, se necesita hacer uso de la función de *transferFrom*

del estándar ERC721 pero a su vez gestionar internamente la actualización de las entradas de los usuarios.

Uso de *event* de Solidity

Los *events* permiten a los desarrolladores emitir señales cuando se llama alguna función y enviar algún dato. Estos mismos pueden ser escuchados desde otras aplicaciones en tiempo real. Esto resulta especialmente útil cuando se necesita que nuestra aplicación reaccione a algo que ha ocurrido en el *smartcontract*. En el caso de este proyecto, se utilizan estos eventos en las funciones cuando se realizan funciones como las de comprar una entrada, transferirla o validar satisfactoriamente una entrada. Cuando estos eventos se emiten, contienen una información que puede ser leída desde las aplicaciones. En la figura 6.5 se muestra un bloque de código de ejemplo:

```
Unset
function validateTicket(
        uint256 eventId,
        uint256 ticketId,
        address userAddress
    ) public onlyOwner {
       // Código omitido
        Ticket storage ticket = userTickets[userAddress][ticketIndex];
        require(!ticket.redeemed, "This user has already redeemed this ticket");

        ticket.redeemed = true;
        emit TicketRedeemed(ticketId, userAddress);
    }
```

Figura 6.5: Bloque de código ejemplo de uso de *event* en la función *validateTicket()*.

Fuente: Elaboración propia

6.3.2 Uso de JWT en la validación de entradas

Para asegurar que las entradas de los asistentes sean validados en el momento adecuado (cuando estos las presentan al validador) y no sean reutilizadas, se necesita de un mecanismo que garantice que el identificador de la entrada sea dinámico en el tiempo y esté cifrado de modo que solo desde la aplicación de validadores se pueda validar dicha entrada.

Una manera de implementar este mecanismo es generando un código dinámico utilizando JSON Web Tokens (JWT). Estos permiten transmitir información de manera segura. Comúnmente son

utilizados en procesos de autenticación, pero también se pueden usar en un intercambio de información, como es el caso de esta funcionalidad del sistema [42].

Este código generado más adelante es leído en mediante escaneo de código QR o NFC. El JWT incluye el identificador de la entrada, el identificador del evento y la dirección del usuario propietario, después se firma con una clave secreta para asegurar su integridad y autenticidad. JWT además permite asignarle un tiempo de expiración al *token* que genera, que para este caso ha sido de 30 segundos, de modo que así se pueda asegurar que este caduca pasado cierto periodo de tiempo.

La clave secreta con la que se firma el JWT se ha generado previamente y está presente en la aplicación de validadores y en la aplicación de los asistentes para poder validar que el contenido no ha sido alterado y está siendo leído por un sistema autorizado. En el siguiente bloque de código de la figura 6.6 se muestra como se implementa una función que construye el *token* JWT haciendo uso de la función *sign()* de la librería react-native-pure-jwt [43]:

```javascript
export async function createToken(
  ticketId: string,
  userAddress: string,
): Promise<string> {
  try {
    const result = await sign(
      {
        iss: 'nftickets-app',
        exp: new Date().getTime() + 30 * 1000, //30s
        ticketId: ticketId,
        user: userAddress,
      },
      JWT_SECRET_KEY,
      {
        alg: 'HS256',
      },
    );
    return result;
  } catch (error) {
    throw new Error('Error creating token');
  }
}
```

Figura 6.6: Bloque de código uso de *sing()*. Fuente: Elaboración propia

Cuando la entrada es presentada, la aplicación del validador escanea el código QR o lee mediante NFC el mensaje del HCE. Después se descifra el contenido escaneado, obteniendo el identificador de la entrada, el evento y la dirección del usuario. En el siguiente bloque de código de la figura 6.7 se puede observar como se implementa una función que descifra el *token* JWT haciendo uso de la función *decode()* de la librería react-native-pure-jwt:

```javascript
export async function decodeToken(token: string): Promise<any> {
  try {
    if (!isValidJwtToken(token)) {
      throw new Error('This is not a valid code');
    }
    const decoded = await decode(token, JWT_SECRET_KEY, {
      skipValidation: false,
    });
    return decoded;
  } catch (error) {
    throw new Error('Error decoding token');
  }
}

export function extractPayload(token: any) {
  const {ticketId, user} = token.payload;
  return {ticketId, user};
}

function isValidJwtToken(token: string): boolean {
  const parts = token.split('.');
  return parts.length === 3;
}
```

Figura 6.7: Bloque de código uso de *decode()*. Fuente: Elaboración propia

6.3.3 Reactividad en las aplicaciones móviles

Como se ha mencionado en la sección 5.2 Arquitectura Lógica, para el manejo de estados dentro de las aplicaciones se ha utilizado la librería Mobx. Esta librería permite tener un contexto global para algunas variables de la aplicación, lo que facilita su uso en diferentes partes del proyecto. Además, permite que dichas variables sean observadas, de modo que cualquier cambio en ellas pueda reflejarse automáticamente en las pantallas de la aplicación.

Un punto a favor de usar Mobx es que optimiza el refresco de los elementos observados mediante etiquetas específicas que proporciona. Estas etiquetas permiten que solo los elementos afectados

por un cambio en el estado sean actualizados, mejorando así el rendimiento general de la aplicación.

A continuación, se explican las etiquetas que se han utilizado en el proyecto, poniendo como ejemplo la *store* de la validación mediante código QR en la aplicación de los validadores:

- ***observable***: La etiqueta *observable* se usa para marcar la propiedad *status* de la *store*. Esto permite que Mobx observe los cambios en esta propiedad y actualice automáticamente cualquier componente que dependa de ella.
- ***action***: La etiqueta *action* se utiliza para marcar el método *validateQRCode.* Esto indica a Mobx que cualquier cambio en los datos observables dentro de este método debe ser tratado como una única transacción atómica, optimizando así la actualización de los componentes.
- ***computed***: La etiqueta *computed* se usa para la propiedad *isReady*, que es una propiedad derivada. Esta etiqueta permite que Mobx recalculé el valor de esta propiedad solo cuando los datos observables de los que depende cambian, mejorando el rendimiento.
- ***runInAction***: Dentro del método *validateQRCode*, se usa *runInAction* para asegurarse de que los cambios en los datos observables (*status* en este caso) se realicen de manera transaccional y eficiente.

En la figura 6.8 pueden apreciarse como se utilizan estas etiquetas:

```javascript
class ValidationQRStore {
  status: ValidationStatusType = 'ready';

  constructor() {
    makeObservable(this, {
      status: observable,
      validateQRCode: action,
      isReady: computed,
    });
  }

  async validateQRCode(qrCode: string) {
    this.setStatus('loading');
    try {
      await validateData(qrCode);
      runInAction(() => {
        this.setStatus('success');
      });
    } catch (error: any) {
      if (error.message === 'Ticket alredy validated') {
        runInAction(() => {
          this.setStatus('warning');
        });
      } else {
        runInAction(() => {
          this.setStatus('error');
        });
      }
    } finally {
      await new Promise(resolve => setTimeout(resolve, 3000));
      runInAction(() => {
        this.setStatus('ready');
      });
    }
  }

  private setStatus(status: ValidationStatusType) {
    this.status = status;
  }

  get isReady() {
    return this.status === 'ready';
  }
}
export default new ValidationQRStore();
```

Figura 6.8: Bloque de código de *ValidationQRStore* usando Mobx. Fuente: Elaboración propia

Después en la pantalla de validación mediante código QR se utiliza la propiedad observada *status* colocando el elemento de la pantalla que se va a actualizar dentro el elemento *<Observable/>* para así únicamente refrescar dicha parte de la pantalla. Por otro lado, en el *handleBarCodeScanned* se hace uso de las *actions* de la *store validateQRCode()* como se puede ver en la figura 6.9.

```javascript
const QrScannerScreen: React.FC = () => {

  const handleBarCodeScanned = async (event: any) => {
    if (!ValidationQRStore.isReady) {
      return;
    }
    await ValidationQRStore.validateQRCode(event.nativeEvent.codeStringValue)
  };

  return (
    <SafeAreaView style={styles.container}>
      <MyText
        text="Scan the ticket placing the QR code in the square"
        styles={styles.advice}
      />
      <View style={styles.outerWrapper}>
        <View style={[styles.cameraWrapper]}>
          <CameraComponent onReadCode={handleBarCodeScanned} />
          <Observer>
            {() => (
              <ValidationStatusBar
                method="qr_code"
                status={ValidationQRStore.status}
                styles={styles.validationStatus}
              />
            )}
          </Observer>
        </View>
      </View>
      <Button
        title="Swap to NFC"
        leadingIcon="nfc"
        onPress={() => {
          navigation.pop();
          navigation.navigate('NFCScannerScreen', {});
        }}
      />
    </SafeAreaView>
  );
};
```

Figura 6.9: Bloque de código de *QrScannerScreen*. Fuente: Elaboración propia

6.3.4 Componentes propios y estandarización de estilos en las aplicaciones

Con el fin de conseguir un código de calidad reutilizable, modulable y mantenible a largo plazo, se han creado componentes de UI propios que encajen con los estilos de las aplicaciones. De esta manera se evita tener que reescribir estilos cada vez que se añade un elemento de UI. Si bien esta práctica no es compleja, ayuda a que el código este mejor estructura y facilita la legibilidad. En el caso de este proyecto se ha trabajado elaborando componentes propios o creando *wrappers* (envoltorios) de componentes ya existentes.

Un ejemplo sencillo de esto es el componente *MyText*. Este componente se renderiza como una línea de texto en pantalla. Para generar este componente, se utiliza el componente nativo *Text* pero aplicándole una serie de modificaciones de estilos. Así se consigue que cuando se invoque a *MyText* este ya tenga los estilos por defecto para así estar en consonancia con el resto de componentes de la aplicación.

Además de ello, a este componente se le atribuye una serie de parámetros para que en caso de necesitar modificar alguna propiedad se pueda fácilmente reutilizando el componente sin tener que crear uno nuevo. Cabe destacar también, que el trabajar con Typescript permite añadir tipados únicos a las propiedades del componente.

A continuación, en la figura 6.10, se muestra un bloque de código del componente descrito en esta subsección.

```
Unset
interface MyTextProps {
  text: string;
  color?: string;
  variant?: |'title'| 'body'| 'caption'| 'button'| 'header'| 'subtitle';
  styles?: TextStyle | TextStyle[];
}

const MyText: React.FC<MyTextProps> = ({
  text,
  color = AppTheme.colors.onSurface,
  variant = 'body',
  styles,
}) => {
  return (
    <Text style={[getStyle(variant), textStyle.default, styles, {color}]}>
      {text}
    </Text>
};
```

Figura 6.10: Bloque de código del componente *MyText*. Fuente: Elaboración propia

6.3.5 Conexión entre la aplicación de validadores, la API REST y el *smartcontract*

Como ya se ha visto en la sección 5.1 Arquitectura física, entre la aplicación de validadores y el *smarcontract* se encuentra la API REST por motivos de eficiencia y usabilidad de la aplicación como se ha ido describiendo.

En esta subsección se expone como se realizan las llamadas a esta API y como esta misma interactúa con el *smartcontract*. A continuación, se muestran bloques de código de la secuencia de llamadas entre componentes del sistema para validar una entrada mediante código QR. No obstante, cabe destacar que esta misma llamada se realiza en el caso de validación mediante NFC pero desde su store correspondiente.

Empezando desde la aplicación de validadores, la primera llamada de la secuencia se realiza cuando el usuario escanea un código QR, desde la *store ValidationQRStore* se llama a la función *validateData()* donde tras pasarle como parámetro el *token* JWT escaneado del QR se descifra y se envía su contenido a través de la función de *ApiService*. En la figura 6.11 se muestra su implementación.

```javascript
export default async function validateData(data: string) {
  try {
    const decoded = await decodeToken(data);
    const payload = extractPayload(decoded);
    await ApiService.getInstance().validateTicket(
      EventStore.currentEvent!.id.toString(),
      payload.ticketId,
      payload.user,
    );
  } catch (error) {
    throw error;
  }
}
```

Figura 6.11: Bloque de código de la función *validateData()* de *ValidationQRStore.*

Fuente: Elaboración propia

La clase *ApiService* contiene las llamadas contra la API REST del sistema. Se ha utilizado Axios para hacer llamadas HTTPS. En el siguiente bloque de código de la figura 6.12 se muestra como hace una llamada POST al *endpoint* en concreto pasándole los parámetros correspondientes:

```javascript
async validateTicket(eventId: string, ticketId: string, address: string):
Promise<void> {
    try {
      await axios.post(`${this.baseUrl}/validate`, {
        eventId,
        ticketId,
        address,
      });
    } catch (error: any) {
      if (error.response.status === 400) {
        throw new Error('Ticket alredy validated');
      }
      throw new Error('Error validating ticket');
    }
  }
```

Figura 6.12: Bloque de código de la función *validateTicket()* de *ApiService.*

Fuente: Elaboración propia

Una vez realizada esta llamada, la API REST alojada en Vercel se encarga de procesarla. Para esta llamada, se accede a la siguiente ruta expuesta en el bloque de código de la figura 6.13.

```javascript
router.post("/validate", (req, res) => TicketController.validateTicket(req, res));
```

Figura 6.13: Bloque de código de la ruta para validar una entrada en la API REST.

Fuente: Elaboración propia

Ahora *TicketController* llama a su función *validateTicket()*, en cuya implementación se llama a la función *validateTicket()* de la instancia de la clase *SmartContractService.* En la figura 6.14 se muestra el bloque de código de esta función.

```javascript
public async validateTicket(req: Request, res: Response): Promise<void> {
    try {
      const { eventId, ticketId, address } = req.body;
      await smartContractService.validateTicket(eventId, ticketId, address);
      res.send("Ticket validated");
    } catch (error: any) {
      if (error.code === "CALL_EXCEPTION") {
        res.status(400).send("Ticket already validated");
      } else {
        res.status(500).send("Internal server error");
      }
    }
  }
```

Figura 6.14: Bloque de código de la función *validateTicket()* de *TicketController*.

Fuente: Elaboración propia

Para finalizar, esta última llamada, ya sí que accede a las funciones propias del *smartcontract* desarrollado como se puede observar en la figura 6.15.

```javascript
public async validateTicket(eventId: string, ticketId: string, address: string)
:Promise<any> {
    const tx = await this.contractInstance.validateTicket(
      eventId,
      ticketId,
      address
    );
    await tx.wait();
    return tx;
  }
}
```

Figura 6.15: Bloque de código de la función *validateTicket()* de *SmartContractService*

Fuente: Elaboración propia

6.4 Problemas emergentes durante la implementación

Esta sección está dedicada a explicar los problemas que han ido surgiendo a lo largo del desarrollo que han requerido de realizar ajustes para conseguir los objetivos que se han planteado al inicio de este.

6.4.1 Problemas de instalación e inestabilidad del SDK de Thirdweb para React Native

Este es quizás uno de los problemas que más tiempo ha requerido para ser solventado. Como ya se explica en la sección 5.1 Arquitectura, este SDK es el que tenía mejor encaje en el proyecto, ya que se había probado que funcionase y tenía más documentación que otras posibles soluciones.

El primer problema que ha presentado el SDK es la instalación de este en un proyecto de React Native. Siguiendo las instrucciones que se explicitan en la web de Thirdweb la instalación no se realiza satisfactoriamente por problemas relacionados con dependencias de desarrollo que requieren de una depuración minuciosa de cada uno de ellos hasta conseguir que el proyecto se ejecute. Esta tarea para desarrolladores más experimentados con el gestor de paquetes de Node *npm* quizás no es tan compleja, pero en el caso del desarrollador del proyecto que no ha liado nunca con esta problemática, de modo que solventar este problema ha supuesto varias horas más de las estimadas.

Otro problema que involucra al SDK es que tiene algunos fallos cuando se intenta vincular una billetera de criptomonedas a la aplicación (en el caso de este proyecto, a la aplicación de los asistentes) o se realiza una operación de escritura en la *blockchain*. Este problema aparece cuando tras realizar una de estas acciones se inicia la aplicación de billetera de criptomonedas y se precisa de firmar una transacción o aceptar alguna petición.

En el caso de vincular una billetera, algunas veces ocurre que tarda en aparecer el mensaje de vinculación en la aplicación de billetera. Por otro lado, cuando se requiere firmar una transacción, generalmente se muestra el mensaje de firma en el momento adecuado, pero tras firmar y volver a la aplicación de los asistentes ocurre que el componente de Thirdweb encargado de reaccionar a esta acción se queda en estado de carga indefinidamente.

No obstante, estos problemas no se han podido verificar que ocurran en otros dispositivos diferentes de los usados para el desarrollo del proyecto. Sin embargo, al ocurrir también en el dispositivo emulado (un dispositivo de Google) da a entender qué puede ocurrir en cualquier dispositivo. Otra posible causa de esta problemática es que no sea el SDK de Thirdweb el responsable, sino la aplicación de billetera que se conecta.

Por otro lado, otro problema que ha dificultado el desarrollo con Thirdweb es la falta de documentación precisa acerca de las funciones que proporciona el SDK y los posibles errores que pueden aparecer durante el desarrollo. Si bien tienen unos cuantos tutoriales publicados en Youtube, en su documentación de React Native para el SDK (versión 4) no hay apenas información de las funciones disponibles.

Más adelante, gracias a la comunidad de Discord de Thirdweb se ha podido encontrar la información que se precisaba para desarrollar correctamente algunas tareas.

6.4.2 Desestimación del uso del backend Engine de Thirdweb

Como ya se ha explicado en el capítulo 5.1 Arquitectura física, finalmente se decidió no utilizar el *backend* de Engine que proporciona Thirdweb para hacer llamadas API al *smartcontract* por los problemas que daba su configuración y el tiempo que estaba involucrando su resolución de errores. Viendo que ya el propio SDK de React Native presentaba problemas, era arriesgado continuar usando las herramientas de Thirdweb y por ello se ha optado por crear una API REST propia desde cero con Node.js.

6.4.3 Problemas con el despliegue de la API REST

Cuando se consideró implementar una API REST inicialmente la idea era hacerlo en Firebase Cloud Functions, ya que es gratuito y dispone de documentación suficiente [44]. El problema viene cuando a la hora de desplegar la API REST, una vez acabada, configurar el despliegue se convirtió en una tarea complicada, puesto que este es sencillo cuando creas el proyecto desde cero utilizando las funciones de Firebase Cloud Functions, pero no se debe que integrar en un proyecto ya existente. Básicamente, conectar las funcionalidades de Firebase Cloud Functions requiere reestructurar gran parte del proyecto *backend*.

Así que para evitar perder tiempo se ha considerado utilizar otra plataforma de alojamiento, en este caso Vercel. Esta plataforma es muy sencilla de integrar en proyectos de Node.js ya existentes y cumple perfectamente con los requisitos para el sistema que se ha desarrollado. Además de ello, Vercel dispone de herramientas de CI/CD. Permite vincular el proyecto con un repositorio de Github de modo que los cambios en la rama *main* (o la que uno especifique como rama de producción) van al despliegue de producción, mientras que las otras ramas se muestran en despliegues de previsualización. Otro punto a favor de Vercel frente a Firebase Cloud Functions es que no requiere de registrar un método de pago para su uso, basta con usar un correo electrónico.

6.4.4 Problemas con las librerías para el uso del NFC y HCE

Por último se presenta el problema que más ha puesto en jaque uno de los requisitos funcionales más interesantes del sistema: La validación de entradas por NFC.

La librería que se utiliza para el escaneo NFC en React Native funciona correctamente y las funcionalidades que requieren si están documentadas o se encuentran buscando en foros de programación. No obstante, el problema viene con la librería para el HCE. Si recapitulamos, el HCE nos permite emular tarjetas inteligentes en dispositivos móviles. De esta manera se consigue emular una tarjeta escaneable mediante NFC que tiene un mensaje con el código que necesita la aplicación de los validadores para validar la entrada. Cuando el usuario asistente quiere validar su entrada, su dispositivo activa el HCE y emula una tarjeta con dicho mensaje. Después, el validador, acerca su dispositivo y mediante NFC escanea esa tarjeta accediendo al contenido del mensaje.

El problema viene con la librería de HCE que emula el mensaje. Inicialmente, cuando se leía la tarjeta emulada con NFC esta nunca incluía el mensaje. Es por ello que se ha tenido que buscar acerca de este error entre la *Issues* de Github del repositorio del proyecto de la librería de HCE y otros foros de programación. Finalmente, se encontró en un foro de programación que esta librería contiene un error que afecta a los dispositivos Android con versiones superiores a Android 10. Y es que cuando construye el mensaje no dota de suficiente espacio (en kilobytes) para crear este mismo. Según lo que exponían algunos usuarios en los foros, este tamaño debía ser modificado para tener compatibilidad en dispositivos Android a 32KB.

Para solventar esta problemática ha sido preciso hacer un *fork* del proyecto de la librería y editar algunas partes del código. Después, una vez editado, se ha instalado en el proyecto de la aplicación de los asistentes.

No obstante, con lo que ya parecía una posible solución aún había problemas. En este momento es cuando se planteó quizás abandonar esta funcionalidad, pero tras probar los siguientes pasos la funcionalidad se ejecutaba perfectamente:

1. **Eliminar las aplicaciones de NFC del dispositivo que usase la aplicación de validadores con el HCE:** Por lo visto, estas aplicaciones, pese a no estar ejecutándose, interfieren con el HCE de esta librería.

2. **Cambiar el dispositivo de ejecución de la aplicación de asistentes al otro dispositivo y viceversa con la de los validadores:** Cabe destacar que los dispositivos físicos que se han utilizado tienen Android 12 en la aplicación para asistentes y Android 13 en la aplicación para validadores. Tras probar cambiar donde se instalaba cada se ha determinado que la dirección en la que se realiza el escaneo importa. El HCE funciona

correctamente cuando se instala en el dispositivo con Android 13 y se lee mediante NFC con el que dispone de Android 12 pero no al revés.

Evidentemente, en el alcance de este proyecto no está solucionar estos problemas de manera efectiva, de modo que basta con ejecutar las aplicaciones en los dispositivos correspondientes y usar la librería modificada.

7. Pruebas

En el siguiente capítulo se detallan las pruebas realizadas en este proyecto. La idea de estas es asegurar que las diferentes funcionalidades que se implementan funcionan correctamente ante diversas situaciones.

Antes de entrar de lleno en este asunto, cabe destacar que no se han implementado pruebas de UI automatizadas por el coste temporal que estas implican. Si bien son una herramienta muy útil, que en proyectos escalables y a largo plazo ahorran muchísimo tiempo a los desarrolladores probando la aplicación, para el contexto de este proyecto, no es algo en lo que valdría la pena invertir esfuerzos. Durante la primera semana de desarrollo se planteó la idea de incorporarlos usando la librería React Native Testing Library [45], pero tras indagar en el asunto se rechazó por el tiempo que podía consumir. No obstante, como se verá a continuación, sí se han implementado pruebas en otras partes del proyecto.

7.1 Pruebas al *smartcontract*

La base del proyecto es el *smartconctract*, es por ello que hay que asegurar que su comportamiento es el esperado. Además, probar el *smartconctract* antes de un despliegue cobra una importancia mayor cuando este proceso se ejecuta en *blockchains* públicas, puesto que una vez desplegado no se puede editar las funciones del mismo.

Para garantizar que el *smartcontract* funcione correctamente se ha utilizado Chai [46], un *framework* de pruebas de Javascript para Node.js. Además, esta herramienta ya viene incorporada al configurar Hardhat.

A continuación se detallan las diferentes pruebas que se han implementado. Estas pruebas están divididas según la lógica de negocio de cada uno de los métodos del *smartcontract*. Es por ello que por cada categoría se incluye al menos una prueba unitaria. Dependiendo de la funcionalidad, hay categorías que tienen implementadas pruebas de integración con otras funcionalidades.

Para identificar cada prueba, se le ha asignado un identificador correspondiente al nombre de la prueba en inglés.

Despliegue

D01 - *Should set the right owner*: Se verifica que el despliegue del contrato establece correctamente al propietario.

Listado de eventos

GE01 - *Should allow to get one listed event*: Se comprueba que se puede obtener un evento listado por su identificador.

GE02 - *Should allow to get all listed events*: Se comprueba que se pueden obtener todos los eventos listados.

Registro de un evento

LE01 - *Should allow owner to list an event*: Se verifica que solo el propietario puede listar un nuevo evento.

LE02 - *Should not allow non-owner to list an event*: Se comprueba que otros usuarios no pueden listar eventos.

***Minteo* de entradas**

MT01 - *Should allow users to mint tickets*: Se verifica que los usuarios pueden *mintear* entradas para eventos existentes.

MT02 - *Should not allow users to mint a ticket for an event they already assist*: Se comprueba que los usuarios no pueden *mintear* más de una entrada para el mismo evento.

MT03 - *Should not allow users to mint a ticket for an event that doesn't exist*: Se verifica que no se pueden *mintear* entradas para eventos no listados.

MT04 - *Should not allow users to mint more tickets than supply defines*: Se asegura que no se pueden *mintear* más entradas que el límite establecido para el evento.

Entradas de los usuarios

UT01 - *Should allow users to get their tickets*: Se verifica que los usuarios pueden obtener la lista de sus entradas.

Transferencia de entradas

TT01 - *Should allow user to send a ticket he owns*: Se comprueba que los usuarios pueden transferir entradas que poseen a otros usuarios.

TT02 - *Should not allow user to send a ticket he doesn't own*: Se asegura que los usuarios no pueden transferir entradas que no poseen.

TT03 - *Should not allow user to send a ticket to a user already has*: Se verifica que no se pueden enviar entradas a usuarios que ya poseen una entrada para el mismo evento.

TT04 - *Should not allow user to send a ticket for an event which is in validation*: Se comprueba que no se pueden transferir entradas de eventos en estado de validación.

TT05 - *Should not allow user to send a ticket for an event which is cancelled*: Se asegura que no se pueden transferir entradas de eventos cancelados.

Validación de entrada

VT01 - *Should allow user to validate a ticket*: Se verifica que los usuarios pueden validar entradas que poseen.

VT02 - *Should not allow user to validate a ticket from an event he doesn't assist*: Se asegura que los usuarios no pueden validar entradas de eventos a los que no asisten.

VT03 - *Should not allow user to validate a ticket that is already validated*: Se comprueba que no se pueden validar entradas ya validadas.

VT04 - *Should not allow user to validate a ticket which event is not in validation*: Se verifica que no se pueden validar entradas de eventos que no están en estado de validación.

VT05 - *Should not allow user to validate a ticket which event is cancelled*: Se asegura que no se pueden validar entradas de eventos cancelados.

Cancelación de un evento

CE01 - *Should allow owner to cancel an event*: Se verifica que solo el propietario puede cancelar eventos.

CE02 - *Should not allow to cancel an event that is already cancelled*: Se comprueba que no se pueden cancelar eventos ya cancelados.

CE03 - *Should not allow non-owner to cancel an event*: Se asegura que otros usuarios no pueden cancelar eventos.

CE04 - *Should not allow to cancel an event that doesn't exist*: Se verifica que no se pueden cancelar eventos no listados.

Devolución de fondos

RA01 - *Should refund event assistants*: Se verifica que los fondos se devuelven correctamente a los asistentes del evento cancelado.

RA02 - *Should not refund event assistants that owned a ticket but they transferred*: Se comprueba que no se devuelven fondos a usuarios que han transferido sus entradas.

Inicio de validación

SV01 - *Should allow owner to start event validation*: Se verifica que solo el propietario puede iniciar el estado de validación para un evento.

SV02 - *Should not allow to start event validation that is already started*: Se asegura que no se puede iniciar la validación de eventos que ya están en estado de validación.

SV03 - *Should not allow non-owner to start event validation*: Se comprueba que otros usuarios no pueden iniciar la validación de eventos.

SV04 - *Should not allow to start event validation that doesn't exist*: Se verifica que no se puede iniciar la validación de eventos no listados.

7.2 Pruebas en las aplicaciones

Como se ha comentado anteriormente, se descartó implementar pruebas automáticas de UI. Es por ello que ese tipo de pruebas, que hubiesen consistido en probar los casos de uso de las aplicaciones, se han realizado de forma manual. Para ello se ejecutaba la aplicación o bien en el emulador o en un dispositivo físico.

Para tener un registro de estas, se ha elaborado un documento con la batería de pruebas a ejecutar manualmente. De esta manera se tiene un informe claro de qué casos de uso funcionan correctamente, cuáles no y los errores detectados. Consultar el documento en el Anexo.

8. Ejecución del proyecto

El objetivo del siguiente capítulo es detallar como ha sido la ejecución del proyecto frente a la planificación inicial de este. En las próximas secciones se muestran los cambios tanto en la planificación como en presupuesto y el nivel de satisfacción de requisitos.

8.1 Cambios de planificación y presupuesto

En esta sección se explican los cambios en la planificación del proyecto y como estos afectan al presupuesto final.

Como tal, no ha habido grandes cambios respecto a la planificación inicial del proyecto. Es decir, las tareas a realizar en su mayoría se han mantenido intactas, salvo una en concreto que a mediados del proyecto se consideró descartar por una cuestión de requisitos de software del sistema, pero que más adelante se ha realizado para solventar otra problemática existente.

La idea inicial de proyecto era implementar las aplicaciones móviles de los asistentes y validadores y conectarlas a un *backend* para que fuese este quien interactuase con el *smartcontract*. Esta arquitectura separa claramente las piezas del proyecto, pero su implementación requiere o bien crear billeteras en el *backend* para cada usuario (una tarea bastante compleja que se aleja del alcance de este TFG, ya que requiere tener en cuenta otros factores de acerca de seguridad) o enviar las claves privadas de las billeteras de los asistentes hacia el *backend* para firmar las transacciones (una técnica poco recomendable por motivos de seguridad). De este modo se consideró descartar la tarea BC - Crear API *backend*, puesto que ahora las transacciones iban a hacerse todas directamente contra el *smartcontract*.

Sin embargo, para el caso de la aplicación de los validadores, tener que firmar cada validación de una entrada suponía un problema de agilidad con la aplicación. Este problema supuso reconsiderar la arquitectura del sistema. Como se ha detallado en capítulo 5 Arquitectura del sistema, para el caso de la aplicación de los validadores, al hacer uso de una única billetera, esta se puede alojar en el *backend* con sus claves privadas. De modo que finalmente la tarea de crear una API en el *backend* se materializó. La desviación en horas de esta tarea se ha considerado reflejarla en las tareas FI2 - Especificación de requisitos y FI4 - Preparación del entorno, ya que como tal crear la API no era una tarea de gran duración.

Otro imprevisto que apareció es el descrito en la sección 6.3 *Faucets* de criptomonedas. Como se explica en este apartado, se ha tenido que ir cambiando de *testnet* a lo largo del desarrollo. Realmente, el hacer el despliegue en una *testnet* distinta no produce una desviación de horas significativa (puesto que salvo casos concretos el código del *smartcontract* es el mismo). El problema fue que la *testnet* de Mumbai entró en desuso (de modo que dejó de funcionar) y hasta

pasados varios días no hubo consciencia de dicho suceso hasta ver una publicación relativa a este suceso por parte de Thirdweb [47]. Se perdieron alrededor de dos tardes de trabajo tratando de solucionar lo que se creía que era un error del SDK y no de la propia *testnet*. Más adelante también se tuvo que dedicar tiempo a buscar alternativas a esta *testnet* como se explica en esa misma sección del documento. Este imprevisto afecta a la tarea de SC - Crear el *smartcontract*

Otra problemática emergente es la descrita en la subsección 6.4.1 Problemas de instalación e inestabilidad del SDK de Thirdweb para React Native. Los problemas que ha presentado el SDK han aumentado las horas dedicadas, especialmente en la tarea de FI4 - Preparación del entorno.

Por último, una desviación a considerar respecto a la planificación inicial es que no se han ido realizando reuniones con la directora del proyecto. Si bien estas no se han realizado, ha bastado con llevar una comunicación continuada durante lo largo del proyecto para consultar dudas o explicar el estado de este. En este sentido, beneficia a los costes del proyecto.

A continuación, en la tabla 8.1, se indican las horas previstas a cada tarea, las horas reales dedicadas y cuál ha sido su desviación final. Las casillas marcadas en color verde representan que no ha habido desviación negativa o se ha realizado la tarea en menos tiempo del esperado. Las casillas en amarillo representan que esa tarea ha tenido una desviación positiva leve. Por último, las casillas en rojo indican que esa tarea ha tenido una desviación positiva significativa.

Se considera una desviación positiva significativa aquella que excede las 4 horas de desviación, ya que esa cantidad representa una tarde de dedicación al proyecto.

Código	Tarea	Horas previstas	Horas dedicadas	Desviación
GD1	Contextualización y alcance	25	25	0
GD2	Planificación temporal	8	8	0
GD3	Gestión económica y sostenibilidad	10	10	0
GD4	Integración del documento final	20	10	-10
GD5	Documentación del trabajo	80	96	+16
GD6	Elaboración presentación final	20	20	0
GD7	Reuniones con la directora	10	0	-10
FI1	Formación	100	100	0
FI2	Especificación de requisitos	15	22	+7
FI3	Diseño de las aplicaciones	10	8	-2
FI4	Preparación del entorno	25	39	+14
SC	Crear el *smartcontract*	20	36	+16
BC	Crear API *backend*	20	7	-13
AA1	Autenticar usuarios	5	10	+5
AA2	Consultar entradas en propiedad	15	20	+5
AA3	Consultar detalle entrada en propiedad	15	15	0
AA4	Transferir una entrada en propiedad	20	18	-2
AA5	Consultar eventos disponibles	20	22	+2
AA6	Adquirir una entrada	10	12	+2
AA7	Canjear entrada con código QR	20	8	-12
AA8	Canjear entrada con NFC	20	22	+2
AV1	Validar entrada con código QR	20	12	-8
AV2	Validar entrada con NFC	20	26	+6
TOTAL		528	557	+29

Tabla 8.1: Desviación en horas por tarea. Fuente: Elaboración propia

Una vez detallada la desviación de horas, se procede a realizar el desglose de esas horas entre los diferentes roles que las ejecutan. En la tabla 3.1, de la subsección 3.1.1 Recursos humanos, se detallan los costes por hora de los roles implicados. En la siguiente tabla 8.2, se muestra el coste real de estos recursos tras realizar el proyecto:

Código	Tarea	Horas dedicadas	Horas según rol					Coste
			JP	AP	D	PM	PB	
GD1	Contextualización y alcance	25	15	10				786,50 €
GD2	Planificación temporal	8	4	4				234,00 €
GD3	Gestión económica y sostenibilidad	10	10					403,00 €
GD4	Integración del documento final	10	10					403,00 €
GD5	Documentación del trabajo	96	20	76				2189,20 €
GD6	Elaboración presentación final	20	20					806,00 €
GD7	Reuniones con la directora	0	0					0,00 €
FI1	Formación	100				50	50	1.820,00 €
FI2	Especificación de requisitos	22	12	10				665,60 €
FI3	Diseño de las aplicaciones	8			8			124,80 €
FI4	Preparación del entorno	39	5			19	15	815,10 €
SC	Crear el *smartcontract*	36					36	702,00 €
BC	Crear API *backend*	7		7		7	0	127,40 €
AA1	Autenticar usuarios	10				10		169,00 €
AA2	Consultar entradas en propiedad	20				10	10	364,00 €
AA3	Consultar detalle entrada en propiedad	15				10	5	266,50 €
AA4	Transferir una entrada en propiedad	18				10	8	325,00 €
AA5	Consultar eventos disponibles	22				17	5	384,80 €
AA6	Adquirir una entrada	12				5	7	221,00 €
AA7	Canjear entrada con código QR	8				8	0	135,20 €
AA8	Canjear entrada con NFC	22				22	0	371,80 €
AV1	Validar entrada con código QR	12				12		202,80 €
AV2	Validar entrada con NFC	26				21	5	452,40 €
TOTAL		557h						11.969,10 €

Tabla 8.2: Desviación en horas por rol. Fuente: Elaboración propia

8.1.1 Indicadores de desviación

A continuación se procede a calcular las desviaciones de horas y costes para los recursos humanos del proyecto.

- **Desviación total de horas**: Esta ecuación permite calcular la diferencia total entre el tiempo que se estimó inicialmente para las tareas y el tiempo que realmente se invirtió en ellas. Una desviación positiva indica que las tareas necesitaron menos tiempo del esperado, mientras que una desviación negativa indica lo contrario.

$$Desviacion\ total\ de\ horas = Horas\ Reales - Horas\ Estimadas$$

Siendo el resultado de esta 557 - 528 = 29 horas

- **Desviación total de costes**: Esta fórmula calcula la diferencia entre los costes que se habían presupuestado para el proyecto y los costes reales incurridos. Al igual que con las horas, una desviación positiva significa un ahorro respecto a lo presupuestado, y una negativa, un sobrecoste.

$$Desviacion\ total\ de\ costes = Costes\ Reales - Costes\ Estimados$$

Siendo el resultado de esta 11.969,10 - 11.921,00 = 48,10 €

- **Desviación de costes imprevistos**: Por último, esta fórmula mide la diferencia entre los costes imprevistos que se estimaron al inicio del proyecto y los costes imprevistos que efectivamente se han presentado. Ayuda a evaluar la precisión de las estimaciones iniciales en relación con la realidad del proyecto.

$$Desviacion\ Costes\ Imprevistos = Costes\ Imprevistos\ Reales - Costes\ Imprevistos\ Estimados$$

Siendo el resultado de esta 48,10 - 910,00 = -861,9 €

8.1.2 Análisis de los resultados de los indicadores obtenidos

Los resultados obtenidos en las fórmulas de la subsección anterior denotan que la estimación de las horas a inicio del proyecto frente a las horas reales invertidas ha sido ligeramente superior pero muy dispar en cuanto a la dedicación de cada rol.

Es por este motivo que pese a que la desviación total en horas es de 29, la desviación total de costes es tan solo de 48,10. Muy por debajo de los costes por imprevistos estimados. Esto es debido a que ha habido tareas en las cuales se ha conseguido realizar el trabajo en mucho menos tiempo del esperado y otras donde se ha tardado más, pero los roles que las han ejecutado tenían

costes inferiores a los roles que se esperaba que las iban a ejecutar. En la tabla 8.3, se muestra la diferencia entre las horas estimadas para cada rol frente a las horas dedicadas de estos:

Rol	Horas previstas	Horas dedicadas	Diferencia
Jefe de proyecto [JP]	99	96	-3
Analista programador [AP]	94	107	+13
Diseñador UX/UI [D]	10	8	-2
Programador *Mobile* [PM]	175	201	+26
Programador *blockchain* [PB]	150	141	-9

Tabla 8.3: Roles del proyecto y sus horas estimadas frente a las previstas. Fuente: Elaboración propia

Como se puede observar, el rol de PM es el que más se ha desviado en horas de proyecto. Esto es debido a que las horas dedicadas por los problemas de la instalación del SDK de Thirdweb reflejados en la tarea FI4 - Preparación del entorno.
Por otro lado, el AP tiene también una desviación significativa por el hecho de que el exceso de horas de la tarea GD5 - Documentación se ha imputado a este rol.
Por último, se puede apreciar como los roles JP y PB (los más costosos económicamente) han necesitado menos horas de las previstas, de modo que eso hace que el coste de los recursos humanos del proyecto se equilibre.

8.1.3 Coste final del proyecto

En esta subsección se calcula el coste final del proyecto, habiendo calculado los costes reales de los recursos humanos de este. En la tabla 8.4, se pueden ver los costes finales.

	Coste final
Recursos humanos	11.969,10 €
Recursos materiales	115,54 €
Otros recursos	430,99 €
TOTAL	12.515,63 €

Tabla 8.4: Costes finales del proyecto por recursos. Fuente: Elaboración propia

Se concluye que el presupuesto final del proyecto ha sido inferior al estimado inicialmente. Siendo el estimado de 15.384,15 € y el final de 12.515,63 € dejando una diferencia entre ambos de 2868,52 €.

8.2 Grado de satisfacción de los requisitos funcionales y no funcionales

Para finalizar, se vuelven a presentar los requisitos funcionales y no funcionales del sistema. El objetivo de esta sección es cuantificar el grado de satisfacción de estos al finalizar el proyecto.

A continuación se irán mostrando diferentes tablas de los requisitos funcionales de las aplicaciones y los requisitos no funcionales, seguidos de un porcentaje de satisfacción y una explicación de por qué algunos requisitos no se consideran satisfechos al 100%.

Primero, en la tabla 8.5 se muestra la satisfacción de los requisitos funcionales de la aplicación de los asistentes:

Requisito funcional	Descripción	Grado de satisfacción
RFA1	La aplicación debe permitir vincular una billetera de criptomonedas e iniciar sesión con ella.	90%
RFA2	La aplicación debe permitir visualizar los eventos disponibles de los organizadores.	100%
RFA3	La aplicación debe permitir buscar un evento mediante su nombre como palabra clave.	100%
RFA4	La aplicación debe permitir visualizar la información de un evento.	100%
RFA5	La aplicación debe permitir comprar una entrada a un evento.	90%
RFA6	La aplicación debe permitir visualizar las entradas compradas por el usuario.	100%
RFA7	La aplicación debe permitir visualizar la información de cada entrada adquirida por el usuario.	100%
RFA8	La aplicación debe permitir buscar una entrada entre las adquiridas mediante el nombre del evento como palabra clave.	100%
RFA9	La aplicación debe permitir transferir una entrada a otra billetera.	90%
RFA10	La aplicación debe permitir validar una entrada mediante escaneo de código QR.	100%

RFA11	La aplicación debe permitir validar una entrada mediante escaneo NFC.	80%
RFA12	La aplicación debe permitir cerrar sesión con la billetera vinculada para poder conectar otra.	100%

Tabla 8.5: Satisfacción de los requisitos funcionales de la aplicación de asistentes. Fuente: Elaboración propia

RF1, RF5 y RF9: Se consideran completadas al 90%, ya que como se ha explicado en la subsección 6.4.1, probablemente por problemas del SDK de Thirdweb o las aplicaciones de billeteras, las firmas se quedan trabadas o tardan en sincronizar la respuesta entre aplicación de asistentes y billetera de criptomonedas.

RF11: Se considera al 80% por el motivo de que pese haber encontrado una solución para la librería y uso del HCE no funciona entre diferentes dispositivos dependiendo de que dirección se realice el escaneo como se detalla la subsección 6.4.4.

Seguidamente, en la tabla 8.6 se muestra la satisfacción de los requisitos funcionales de la aplicación de los validadores. Como se puede observar, para esta aplicación los requisitos se han cumplido en su totalidad.

Requisito funcional	Descripción	Grado de satisfacción
RFV1	La aplicación debe permitir seleccionar el evento para el cual se van a validar entradas.	100%
RFV2	La aplicación debe permitir validar una entrada mediante escaneo de código QR.	100%
RFV3	La aplicación debe permitir validar una entrada mediante escaneo NFC.	100%
RFV4	La aplicación debe permitir cambiar entre método de escaneo fácilmente.	100%
RFV5	La aplicación debe informar del estado de la validación de la entrada.	100%

Tabla 8.6: Satisfacción de los requisitos funcionales de la aplicación de validadores. Fuente: Elaboración propia

Por último, en la tabla 8.7, se muestra el grado de satisfacción, los requisitos no funcionales del sistema. Estos aplican en ambas aplicaciones.

Requisito funcional	Descripción	Grado de satisfacción
Usabilidad	La interfaz de usuario debe ser intuitiva y fácil de usar para personas con diferentes niveles de habilidad tecnológica.	100%
Rendimiento	La aplicación debe ser capaz de procesar rápidamente las solicitudes, especialmente las relacionadas con la compra y transferencia de entradas, para garantizar una experiencia fluida y sin demoras para los usuarios.	80%
Disponibilidad	La aplicación debe garantizar que los usuarios pueden acceder en cualquier momento y desde cualquier lugar.	100%

Tabla 8.7: Grado de satisfacción de los requisitos no funcionales. Fuente: Elaboración propia

Usabilidad: Se considera satisfecha al 100%, ya que al cumplir con criterios de UX/UI de Material 3 permite al usuario utilizar la aplicación sin tener que pensar que realiza cada elemento de la pantalla, puesto que estos tienen coherencia con los elementos de su sistema operativo.

Se ha probado con usuarios de dispositivos Android realizar las acciones que se describen en el requisito no funcional para cumplir la condición de satisfacción y estos cumplen incuso con menos tiempo del que se ha marcado como objetivo (menos de 5 minutos).

Rendimiento: A nivel de aplicación Android, el rendimiento es excelente. También cabe decir que no son aplicaciones que requieran de un uso intensivo de los recursos del sistema. Sin embargo, por la parte de *blockchain*, dependiendo del estado de la *testnet*, la velocidad de las transacciones se ha visto afectada en algunos momentos del desarrollo. No obstante, no es lo habitual y por ello es que se le ha asignado un porcentaje del 80%.

Disponibilidad: Este requisito funcional es complicado de validar si se cumple su condición de satisfacción, ya que para ello se precisa de un año natural para comprobarlo. Sin embargo, durante todo el desarrollo del proyecto, tanto el *smartcontract* como la API REST en Vercel han estado operativas sin ningún percance. Es por ello que se le asigna un 100%.

9. Aspectos legales

Para abordar los aspectos legales de este proyecto, es importante considerar las leyes aplicables sobre este y las licencias de *software* que se han utilizado. En esta sección se detallan las diferentes leyes aplicables y las licencias que se han tomado para su desarrollo.

9.1 Leyes aplicables al proyecto

Al tratarse de un proyecto basado en NFTs, hay varias leyes a tener en cuenta. Las leyes aplicables se separan entre las relacionadas con protección de datos y privacidad, la regulación de finanzas y criptomonedas y las de propiedad intelectual y derechos de autor. Cabe mencionar que, para el presente análisis, únicamente se han considerado leyes y regulaciones a nivel estatal (en el contexto de España) y a nivel europeo (en el contexto de la Unión Europea).

Protección de datos y privacidad

La ley que se expone a continuación no afecta como tal a los usuarios de las aplicaciones, pero sí a los organizadores de los eventos y artistas que registran su evento en el *smartcontract*. Esto es debido a que en el título y descripción del evento se pueden aportar datos de terceros. Si bien en este proyecto, al ser de carácter académico, no se ha considerado aportar una funcionalidad para que los organizadores puedan registrar eventos (se pueden listar directamente en el *smartcontract*, pero no se da esta capacidad a ningún usuario del sistema), en caso de ser así se debería tener en cuenta que toda información registrada en la *blockchain* es inmutable y por ende no puede ser eliminada. Esto es incompatible con las leyes de protección de datos, específicamente el Reglamento General de Protección de Datos (GDPR), que establece el derecho al olvido [48] . El GDPR requiere que los datos personales puedan ser borrados a petición del usuario, lo cual no es posible en una *blockchain* pública debido a su naturaleza inmutable. Esto sí, podría ser posible si se tratase de una *blockchain* permisionada o bien se guardaran los datos de los eventos fuera de la *blockchain*.

Regulación de finanzas y criptomonedas

Para este tipo de leyes se destacan dos en concreto. Primeramente, se destaca a nivel estatal la Ley de Prevención del Blanqueo de Capitales y de la Financiación del Terrorismo [49]. Esta ley afecta directamente a los proyectos de criptomonedas y NFTs, puesto que les obliga a implementar procedimientos de *Know Your Customer* (KYC) para verificar la identidad de los usuarios.

El KYC es un procedimiento fundamental, especialmente en empresas del sector financiero o que operan con criptomonedas. Su objetivo es verificar la identidad de sus clientes para así poder prevenir actividades ilegales [50]. Este procedimiento tiene cuatro distinguidas etapas:

1. **Recopilación de información personal:** Se solicita al cliente información básica como su nombre completo, dirección, fecha de nacimiento y nacionalidad. Esta información suele complementarse con datos de contacto como número de teléfono y correo electrónico.

2. **Verificación de identidad:** El cliente debe proporcionar documentos oficiales que prueben su identidad, como un pasaporte o documento nacional de identidad. En algunos casos, también se pueden requerir documentos adicionales que prueben la dirección, como facturas de servicios públicos o extractos bancarios.

3. **Análisis de riesgo:** La empresa evalúa el perfil del cliente para determinar el nivel de riesgo asociado. Esto puede incluir la verificación de antecedentes financieros y la búsqueda de posibles vínculos con actividades delictivas o listas de sanciones internacionales. La empresa puede denegar el servicio a ese cliente si no cumple

4. **Monitoreo continuo:** Después de la verificación inicial, las empresas realizan un seguimiento continuo de las actividades del cliente para detectar y reportar cualquier comportamiento sospechoso. Esto incluye la actualización periódica de la información del cliente y la revisión de transacciones.

No obstante, en el caso de este proyecto no existe dicho procedimiento al ser de carácter académico; sin embargo, sería indispensable considerarlo en uno real.

Propiedad intelectual y derechos de autor

Por último, es importante destacar las leyes relativas a la propiedad intelectual y derechos de autor. Aunque los NFTs del proyecto actúan como entradas, estos podrían incluir contenido protegido como puede ser imágenes, logos o marcas. No obstante, los NFTs que se *mintean* a través del *smartcontract* no contienen ningún tipo de atributo relativo al evento. Únicamente tienen un identificador. En el caso de tener contenido en los metadatos del NFT sí que aplicarían leyes como la Ley de Propiedad Intelectual en España [51].

9.2 Licencias

Para el desarrollo de todo el proyecto se ha ido haciendo uso de diferentes herramientas de *software*. La gran mayoría de herramientas, entre ellas React Native, Mobx, Ethers.js, Hardhat, Express.js hacen uso de MIT License [52]. Para el caso de Vercel se trata de Apache License 2.0 [53]. Ambas licencias son permisivas con el uso, modificación, distribución y comercialización del *software*.

Por otro lado, Infura dispone de unos términos de servicio específicos entre los cuales se establecen responsabilidades a los usuarios, de modo que son estos quienes deben asegurarse del cumplimiento de las leyes. También se acotan unos límites del uso del servicio (para evitar propósitos maliciosos o ilegales) además de restringirse el derecho de dar de baja las cuentas de los usuarios que incumplan con los términos del servicio [54].

10. Sostenibilidad y compromiso social

En este capítulo se analizará la sostenibilidad del proyecto mostrando su impacto a nivel económico, social y ambiental.

10.1. Dimensión económica

En el capítulo anterior ya se hizo una pequeña evaluación presupuestaria del proyecto referente a los costes de los recursos de este y los posibles costes añadidos debido a imprevistos.

Por otro lado, analizando la viabilidad del proyecto, se concluye que, pese a que el fin del proyecto es académico, la idea de este aborda un problema existente en el mundo de los eventos. Si bien es cierto que ya existen soluciones similares que hacen uso de la tecnología *blockchain*, el sistema se diferencia en algunos detalles como podrían ser el *marketplace* de entradas (los eventos listados con sus entradas) o la gratuidad del servicio en sí. No obstante, comparando con los sistemas tradicionales, este proyecto reduce significativamente los costes relacionados con la emisión de entradas (ya que *mintear* un NFT es a fecha de hoy del orden de céntimos de dólar americano en muchas redes *blockchain*).

10.2. Dimensión social

A nivel personal se destaca que el proyecto brinda al desarrollador la posibilidad de sumergirse en el desarrollo *blockchain* lo cual otorga experiencia técnica en este ámbito y le puede abrir puertas en un futuro para trabajar en proyectos reales con esta tecnología.

Si se analiza el impacto social a nivel general, se puede decir que, si fuese un proyecto real que saliese al mercado, sí podría tener cierto alcance por el tema de que erradica por completo el fraude de entradas y la especulación de estas. Como se ha visto en apartados anteriores, los *stakeholders* del proyecto van desde los propios asistentes hasta las autoridades gubernamentales. Sin embargo, tras estudiar el contexto del mundo de los eventos, no parece haber una necesidad inmediata de este tipo de soluciones (al menos *blockchain*), puesto que año tras año llevan ocurriendo las problemáticas planteadas.

10.3. Dimensión ambiental

Para analizar la dimensión ambiental, se pueden contemplar los recursos materiales que se utilizan para la realización de este, como pueden ser la energía, el ordenador y los dispositivos móviles. Para estos últimos se utilizan unos teléfonos Android en desuso.

Por otro lado, se debe que destacar que este sistema tiene una importante mejora en lo que a impacto ambiental se refiere, ya que al digitalizar el proceso de emisión y validación de entradas se consigue reducir significativamente el uso del papel.

Además de ello, es relevante mencionar que al haber optado por un *blockchain* como la de Polygon se tiene un impacto ambiental significativamente bajo respecto al de otras *blockchains* existentes. Polygon hace uso del mecanismo de consenso Proof of Stake (PoS) mientras que otras redes hacen uso de *PoF* como podría ser la de Bitcoin [55]. Estas últimas tienen un impacto ambiental notablemente superior, sin embargo, en las primeras ese impacto es inferior. También es interesante mencionar que, Polygon tiene un compromiso con el medioambiente. En 2022 se comprometió a convertirse en una *blockchain* neutral en carbono y es por ello que invirtió en organizaciones y proyectos ambientales para apoyar la sostenibilidad [56].

11. Conclusiones y trabajo futuro

Concluido el proyecto, a continuación se van a detallar las diferentes competencias técnicas trabajadas durante el desarrollo de este, así como la relación de este TFG con el grado y la especialidad. También se incluyen unas conclusiones personales y consideraciones a futuro.

11.1 Competencias técnicas trabajadas

- **CES1.1: Desarrollar, mantener y evaluar sistemas y servicios de software complejos y/o críticos.**

En este proyecto, esta competencia ha sido la más desarrollada, ya que el objetivo de este era construir un sistema completo. Para esta labor, como se ha ido viendo a lo largo del documento, se requería involucrar diferentes entornos de desarrollo para cada pieza del sistema. Como tal, el mantenimiento de estos no ha sido una tarea muy compleja.

- **CES1.2: Dar solución a problemas de integración en función de las estrategias, los estándares y las tecnologías disponibles.**

Durante el desarrollo del proyecto, a menudo se ha tenido que lidiar con diversos problemas para integrar las tecnologías que se habían planteado al inicio. Un ejemplo de esto es la integración del SDK de Thirdweb en el proyecto React Native de la aplicación de los asistentes. Como se ha comentado anteriormente, este SDK presenta bastantes problemas para configurarlo correctamente en el proyecto. Pese a ello, al ser la única solución que por lo general funcionaba correctamente, se ha puesto esfuerzo en tratar de integrar correctamente.

- **CES1.3: Identificar, evaluar y gestionar los riesgos potenciales asociados a la construcción de software que se pudieran presentar.**

Esta competencia se ha desarrollado de manera moderada en el proyecto, ya que al tratarse de un sistema con diferentes componentes, podían surgir diferentes problemas en cada uno de ellos. Es por este mismo motivo que en la sección 2.5 Gestión del riesgo se detallan algunos riesgos identificados al inicio del proyecto. No obstante, cabe decir que algunos de ellos no fueron identificados hasta entrar de lleno en la implementación del sistema por desconocimiento, sin embargo, han sido identificados y gestionados rápidamente.

- **CES1.4: Desarrollar, mantener y evaluar servicios y aplicaciones distribuidas con soporte de red.**

Este proyecto consiste en un sistema que implementa una solución basada en *blockchain*. Desde hace varios años esta tecnología se ha ido popularizando poco a poco; sin embargo, no es la más habitual entre los sistemas actuales (en cierto modo por los casos de uso a los que se puede

aplicar una solución blockchain). Es por ello que se han necesitado horas de estudio para el entendimiento de cómo esta funciona (en concreto la de Ethereum) para poder ir desarrollando el *smartcontract*.

También, debido a que se ha tenido que migrar el proyecto a diferentes *testnets* durante el desarrollo, ha sido necesario ir evaluando las diferentes *testnets* y como estas podían encajar en el proyecto. Ya que no todas proveían de *tokens* para las testnets, o bien estaban en desuso o no eran compatibles con según que funciones de Solidity.

- **CES1.7: Controlar la calidad y diseñar pruebas en la producción de software.**

Esta competencia también ha sido tratada a lo largo del proyecto. Se han ido realizando diferentes pruebas a lo largo del proyecto para así asegurar que las nuevas funcionalidades que se han ido implementando cumplen con ciertos requisitos de calidad y no dejan inoperativas funcionalidades ya implementadas. Aquí quedan incluidas las pruebas unitarias y de integración en el *smartcontract* y la API REST y las pruebas de sistema realizadas en las aplicaciones móviles.

- **CES2.1: Definir y gestionar los requisitos de un sistema de software.**

Al tratarse de un proyecto ideado por el alumno, se ha tenido que definir desde cero todos los requisitos del sistema. Para ello se han especificado requisitos funcionales y no funcionales, además de detallar los diferentes casos de uso de las aplicaciones del sistema.

La gestión de requisitos también ha tomado un papel importante en el sentido de que al ir presentándose inconvenientes con servicios externos planteados en un inicio ha sido necesario actuar a tiempo e irlos adaptando.

- **CES2.2: Diseñar soluciones apropiadas en uno o más dominios de aplicación, utilizando métodos de ingeniería del software que integren aspectos éticos, sociales, legales y económicos.**

La premisa sobre la que se sostiene la elaboración de este proyecto parte de que la industria de los eventos sufre de actividades poco éticas como el fraude de entradas y la reventa especulativa. Es por ello que la solución que se propone, basada en *blockchain*, tiene un impacto social, legal y económico sobre las partes interesadas del proyecto. Pese a tratarse de un proyecto de carácter académico, este trata de simular casos de uso reales que contribuyen en favor de la transparencia, la seguridad y la equidad en la venta y distribución de entradas.

Además, un sistema de este tipo, en un contexto de un producto de *software* real, contribuye al desarrollo de una infraestructura más robusta y confiable para la industria de los eventos, lo que ayuda a mejorar la experiencia del usuario final.

11.2 Relación del TFG con el grado y la especialidad

Este capítulo está dedicado a indicar las asignaturas cursadas durante el grado que han sido útiles para el desarrollo de este proyecto.

Especificación de requisitos, planificación temporal y gestión del proyecto

Las asignaturas más valiosas del grado para estas áreas han sido Enginyeria de Requisits (ER) junto con Gestio de Projectes de Software (GPS). Ambas asignaturas están enfocadas en todo aquello que no es la programación del proyecto, como puede ser el análisis de requisitos, planificación temporal y gestión del proyecto.

Los conocimientos de ER son especialmente útiles para realizar la contextualización y alcance del proyecto, además de la especificación de requisitos.

Por otro lado, en GPS se detallan diferentes metodologías de trabajo de la cual para este proyecto se ha escogido Scrum de Agile. En esta misma asignatura también se presentan algunas herramientas software para la gestión del proyecto, como puede ser Taiga, el cual se ha utilizado para la gestión de las tareas del proyecto. Otro aprendizaje de esta asignatura es la elaboración de diagramas de Gantt para la planificación del proyecto. Esto es algo imprescindible cuando se idea un proyecto desde las bases, ya que permite tener un mapa del desarrollo del proyecto.

No obstante, cabe mencionar que otras asignaturas como puede ser Projecte d'Enginyeria del Software (PES) también es de utilidad, puesto que recorre el desarrollo de un proyecto de inicio a fin, incluyendo las áreas que se han comentado en este apartado.

Utilización de Git como metodología de control

El proyecto se ha dividido en tres repositorios de GitHub. Uno para la aplicación de los asistentes, otro para la de los y un tercero para la API REST con el *smartcontract*. Sin un software de gestión de versiones es arriesgado adentrarse en cualquier proyecto de software. No solo porque guardar los ficheros en local se corre el riesgo de perderlos definitivamente, sino también porque ayuda a gestionar de manera eficiente las diferentes versiones del código. El uso de esta herramienta ha estado presente durante el grado en aquellas asignaturas donde se elaboraba un proyecto de software en equipo. Aunque en el caso de este proyecto se ha usado siendo un único desarrollador, esta herramienta es especialmente útil en proyectos colaborativos. Es por ello que asignaturas como Projecte de Programació (PROP), Aplicacions i Serveis Web (ASW) y PES han sido útiles para aprender a usar esta herramienta.

Patrones de arquitectura y de diseño

Los patrones de arquitectura y de diseño son piezas fundamentales en un proyecto de software para cumplir con los principios SOLID. Gracias a las técnicas aprendidas en asignaturas como Arquitectura del Software (AS), PROP y ASW se han podido aplicar patrones de arquitectura y diseño en diferentes partes del proyecto. Estas prácticas aseguran que el código cumple con unos requisitos mínimos de calidad.

Diseño *Frontend* y *Backend*

El diseño *Frontend* y *Backend* cobra especial importancia en el sistema construido en este proyecto. El primer contacto con tecnologías *Frontend* viene de asignaturas como ASW y PES. Para este proyecto en concreto PES, puesto que el grupo del desarrollador de este proyecto realizó su proyecto con React Native. También cabe destacar que, gracias a la asignatura de Interacció i Disseny d'Interfícies (IDI) se han aplicado técnicas de diseño UX/UI para garantizar la usabilidad de las aplicaciones y cumplir con unos estándares de diseño.

Por otro lado, para el *Backend* del sistema, las asignaturas que han sido más útiles han sido ASW y PES. Sobre todo en ASW donde se explican con más detalle cómo funcionan las APIs REST (implementación, despliegue, códigos de respuesta, etc).

Facilidad para adquirir conocimientos de diferentes ámbitos específicos del desarrollo

Puesto que el proyecto implementa tecnología *blockchain*, y esta apenas se menciona durante el grado y la especialidad, se ha requerido de un estudio previo de dicha tecnología para su comprensión. Es por este motivo que ha sido preciso adaptarse a esta e ir aprendiendo para poder integrar en el sistema. En este sentido, las asignaturas que más han contribuido a resolver problemas de forma autónoma y a aprender conocimientos en nuevas áreas han sido PES y ASW por ser asignaturas donde el alumno decide con qué *frameworks* trabajar y con ello se debe gestionar el mismo los percances que vayan surgiendo.

Conocimiento con las tecnologías utilizadas

Como se ha comentado en el párrafo anterior, en el grado y en la especialidad no se realiza un estudio exhaustivo en tecnología *blockchain*. No obstante, hay dos asignaturas donde se ha podido conocer algunos principios básicos. En concreto, estas son Gestió de la Ciberseguretat (GCS) y Xarxes de Computadors (XC).

Pruebas del sistema

Las pruebas al sistema son clave para validar que el sistema implementado funcione correctamente. Parte del conocimiento aplicado viene dado por asignaturas como AS, PROP y PES. En concreto hay que destacar AS donde se hace un análisis mucho más profundo de las diferentes pruebas que se pueden realizar al código.

Por otro lado, también hay que mencionar a IDI, ya que en esta asignatura se explican metodologías para probar que el software desarrollado cumplen con principios de diseño UX/UI. Estas han sido útiles para validar algunos requisitos no funcionales.

11.3 Conclusiones personales

El trabajo realizado en este proyecto ha sido una experiencia ideal para desenvolverme en un proyecto siendo autodidacta y en solitario. Si bien lo más común en la realidad laboral de un desarrollador de software suele ser la de trabajar en equipo, en esta ocasión he podido experimentar como es hacer un proyecto de pies a cabeza asumiendo todos los roles del proyecto, a excepción del de la directora de este.

Al ser un proyecto propuesto por mí mismo, le presto de especial valor, ya que es una idea que desde hacía meses quería desarrollar. El hecho de poder idear cómo va a ser, qué funciones tendrá, cómo se verán las pantallas es un proceso que requiere de tiempo de trabajo y esfuerzo, pero que, una vez terminado, da gusto ver el resultado final.

Durante este proyecto a nivel técnico hay dos materias en las que me he podido desarrollar más a fondo y por ello considero que este TFG me ha brindado una enriquecedora experiencia. La primera es acorde a desarrollo de *smartcontracts* y la segunda es respecto a desarrollo móvil con React Native. Aunque ya había trabajado con este *framework* en ocasiones anteriores, esta vez he podido aplicar mejores prácticas. Destaco sobre todo el hecho de trabajar con la librería para programación reactiva de Mobx, ya que permite tener el código mejor estructurado y la optimiza gestión de los estados.

Por otro lado, este proyecto me ha obligado a tener una gestión de mi tiempo bastante estricta. Esto es porque se ha realizado mientras cursaba una última asignatura optativa y trabajaba en la empresa donde desarrollo aplicaciones Android (en este caso en Kotlin Multiplatform). De modo que no ha habido semana que pudiese permitirme no trabajar en el proyecto.

De todos, la experiencia que me ha aportado realizar este TFG es sin duda positiva. Aunque me hubiese gustado poder dedicarle más tiempo para escalar las funcionalidades de la aplicación de los asistentes, creo que el resultado final del sistema es muy bueno.

11.4 Trabajo a futuro

Como se ha comentado en la sección anterior, el tiempo disponible para dedicar al proyecto era bastante justo. Es por ello que hay algunas partes de este que tienen un amplio margen de mejora.

En primer lugar, sería interesante poder desacoplar el uso de las billeteras de la aplicación de asistentes. Esto permitiría una experiencia mucho más cómoda a los usuarios sin necesidad de depender de una segunda aplicación para poder realizar cualquier gestión con las entradas.

Este detalle se podría haber conseguido pulir si la arquitectura del sistema hubiese sido la que aloja las billeteras de los usuarios en un servidor *backend*. Como se ha detallado en la sección 8.1 Cambios en la planificación y presupuesto, implementar esa arquitectura requería de más tiempo de desarrollo, puesto que hay varias consideraciones en materia de seguridad a tener en cuenta. Además, esta arquitectura permite no tener que hacer uso del SDK de Thirdweb, el cual da algunos problemas que la documentación que proporciona no termina de detallar el porqué. Otra mejora a alcanzar en desarrollos a futuro sería permitir a los usuarios autenticarse con Google (u otros servicios como Microsoft o Apple). Thirdweb permite autenticarse con Google, pero la idea de esto sería poder gestionarlo sin necesidad de hacer uso de este.

En segundo lugar, en un futuro, podría ampliarse el abanico de funcionalidades que proporciona la aplicación de los validadores. El estado con el que ha quedado tras la realización del proyecto es el de una aplicación donde comprar entradas para un evento, transferirlas y permitir su validación a través de dos métodos distintos. Algo con lo que se podría explotar el uso de la tecnología NFT es la posibilidad de obtener *royalties* por parte de los organizadores de los eventos. Es decir, si un usuario decide revender su entrada (dentro de un precio que no permita el abuso de la reventa como forma de lucrarse) el organizador del evento recibe una pequeña comisión por ello.

Otro punto de mejora que podría ser interesante a nivel de desarrollo de las aplicaciones, es crear *microfrontends* para poder reutilizar de forma eficiente los componentes, ya que tanto la aplicación de los asistentes como la de los validadores comparten el mismo tema y las funcionalidades de muchos de estos componentes son las mismas en ambas.

De cara al *smartcontract*, una mejora a considerar es este dividir sus métodos entre otros *smartcontracts* para así delegar más sutilmente las responsabilidades de este.

Como último punto de mejora, teniendo en cuenta que los usuarios se autentican mediante una cuenta para la propia aplicación, podría implementarse la posibilidad de añadir amigos a tu perfil. De este modo, la transferencia de entradas es más sencilla que insertando una dirección de una billetera de Ethereum.

12. Referencias

[1] R. Carranco, “Los Mossos detectan a más de 300 estafados en los conciertos de Coldplay: ‘Compré unas entradas a un chaval de Twitter que se llamaba Gabri,’” El País. Accessed: Feb. 26, 2024. [Online]. Available: https://elpais.com/espana/catalunya/2023-05-29/los-mossos-detectan-a-mas-de-300-estafados-en-coldplay-compre-unas-entradas-a-un-chaval-de-twitter-que-se-llamaba-gabri.htmlhttps://polygon.technology/sustainability

[2] “Blockchain,” *Wikipedia*. Feb. 19, 2024. Accessed: Feb. 23, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Blockchain&oldid=1209005465

[3] “Smart contract,” *Wikipedia*. Dec. 06, 2023. Accessed: Feb. 26, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Smart_contract&oldid=1188555829

[4] “Introducción a las DApps,” ethereum.org. Accessed: Feb. 26, 2024. [Online]. Available: https://ethereum.org/es/developers/docs/dapps

[5] “Non-fungible token,” *Wikipedia*. Feb. 24, 2024. Accessed: Feb. 26, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Non-fungible_token&oldid=1210067691

[6] “Intro to Ethereum,” ethereum.org. Accessed: Feb. 26, 2024. [Online]. Available: https://ethereum.org/developers/docs/intro-to-ethereum

[7] “Cryptocurrency wallet,” *Wikipedia*. Feb. 25, 2024. Accessed: Feb. 26, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Cryptocurrency_wallet&oldid=1210249124

[8] “Web3,” *Wikipedia*. Feb. 17, 2024. Accessed: Feb. 26, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Web3&oldid=1208312875

[9] “Código QR,” *Wikipedia, la enciclopedia libre*. Apr. 24, 2024. Accessed: May 16, 2024. [Online]. Available: https://es.wikipedia.org/w/index.php?title=C%C3%B3digo_QR&oldid=159671893

[10] “Comunicación de campo cercano,” *Wikipedia, la enciclopedia libre*. May 03, 2024. Accessed: May 16, 2024. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Comunicaci%C3%B3n_de_campo_cercano&oldid=159886557

[11] “Host card emulation,” *Wikipedia, la enciclopedia libre*. Feb. 09, 2023. Accessed: Jun. 21, 2024. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Host_card_emulation&oldid=149173445

[12] “Sell NFT tickets online with Oveit,” Oveit. Accessed: Feb. 26, 2024. [Online]. Available: https://oveit.com/nft-tickets/

[13] “What is Tokenization?,” Security. Accessed: Feb. 26, 2024. [Online]. Available: https://www.techtarget.com/searchsecurity/definition/tokenization

[14] “ENTRY NFT – TICKETS DIGITALES, SEGUROS Y TRAZABLES.” Accessed: Feb. 26, 2024. [Online]. Available: https://entrynft.net/

[15] R. Mafia and R. Labs, “Mr. Crypto | Private club,” Mr. Crypto. Accessed: Feb. 26, 2024. [Online]. Available: https://mr-crypto-landing.vercel.app/

[16] “Racks Labs - Agencia de desarrollo web3,” Racks Labs - Agencia de desarrollo web3. Accessed: Feb. 26, 2024. [Online]. Available: https://www.labs.racksmafia.com/

[17] “Expo,” Expo. Accessed: Jun. 21, 2024. [Online]. Available: https://expo.dev/
[18] “React Native · Learn once, write anywhere.” Accessed: Jun. 21, 2024. [Online]. Available: https://reactnative.dev/
[19] “Sidechains,” ethereum.org. Accessed: Feb. 26, 2024. [Online]. Available: https://ethereum.org/developers/docs/scaling/sidechains
[20] “Web3, Aggregated.” Accessed: Jun. 21, 2024. [Online]. Available: https://polygon.technology/
[21] “Optimism.” Accessed: Mar. 03, 2024. [Online]. Available: https://www.optimism.io/
[22] “The Value Layer of the Internet.” Accessed: Mar. 03, 2024. [Online]. Available: https://polygon.technology/
[23] “Introducing BNB Chain: The Evolution of Binance Smart Chain,” Binance Blog. Accessed: Mar. 03, 2024. [Online]. Available: https://www.binance.com/en/blog/ecosystem/introducing-bnb-chain-the-evolution-of-binance-smart-chain-421499824684903436
[24] “¿Cuánto paga Accenture por hora en 2024?,” Glassdoor. Accessed: Mar. 11, 2024. [Online]. Available: https://www.glassdoor.es/Salario-por-hora/Accenture-Salario-por-hora-E4138.htm
[25] “Vercel: Build and deploy the best web experiences with the Frontend Cloud,” Vercel. Accessed: Jun. 21, 2024. [Online]. Available: https://vercel.com/home
[26] “Engine.” Accessed: Jun. 21, 2024. [Online]. Available: https://portal.thirdweb.com/engine
[27] “Alchemy - the web3 development platform,” Alchemy. Accessed: May 16, 2024. [Online]. Available: https://www.alchemy.com/
[28] “Moralis Web3 - Enterprise-Grade Web3 APIs,” Moralis Web3 | Enterprise-Grade Web3 APIs. Accessed: May 16, 2024. [Online]. Available: https://moralis.io/
[29] “What is JSON-RPC in Ethereum?,” GeeksforGeeks. Accessed: May 17, 2024. [Online]. Available: https://www.geeksforgeeks.org/what-is-json-rpc-in-ethereum/
[30] “Clean Coder Blog.” Accessed: May 17, 2024. [Online]. Available: https://blog.cleancoder.com/uncle-bob/2011/09/30/Screaming-Architecture.html
[31] “About MobX · MobX 🇺🇦.” Accessed: May 17, 2024. [Online]. Available: https://mobx.js.org/index.html
[32] “Router, controllers and services | Ditsmod.” Accessed: Jun. 21, 2024. [Online]. Available: https://ditsmod.github.io/en/components-of-ditsmod-app/controllers-and-services/
[33] “Kotlin Multiplatform | Kotlin,” Kotlin Help. Accessed: May 29, 2024. [Online]. Available: https://kotlinlang.org/docs/multiplatform.html
[34] “The starting point for learning TypeScript.” Accessed: Jun. 21, 2024. [Online]. Available: https://www.typescriptlang.org/docs/
[35] “Getting started with Hardhat | Ethereum development environment for professionals by Nomic Foundation.” Accessed: Jun. 21, 2024. [Online]. Available: https://hardhat.org
[36] “Home - Truffle Suite.” Accessed: May 29, 2024. [Online]. Available: https://archive.trufflesuite.com/
[37] “Documentation.” Accessed: Jun. 21, 2024. [Online]. Available: https://docs.ethers.org/v5/
[38] “web3.js - Ethereum JavaScript API — web3.js 1.0.0 documentation.” Accessed: May 29, 2024. [Online]. Available: https://web3js.readthedocs.io/en/v1.10.0/
[39] “GitHub Copilot · Your AI pair programmer,” GitHub. Accessed: May 29, 2024. [Online]. Available:

https://github.com/features/copilot/

[40] “Proof of work,” *Wikipedia*. Feb. 03, 2024. Accessed: Mar. 11, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Proof_of_work&oldid=1202838008

[41] “OpenZeppelin.” Accessed: Jun. 21, 2024. [Online]. Available: https://www.openzeppelin.com

[42] “JSON Web Token,” *Wikipedia, la enciclopedia libre*. Apr. 30, 2024. Accessed: Jun. 21, 2024. [Online]. Available: https://es.wikipedia.org/w/index.php?title=JSON_Web_Token&oldid=159811805

[43] L. F. Zaguini, “zaguiini/react-native-pure-jwt.” Jun. 05, 2024. Accessed: Jun. 21, 2024. [Online]. Available: https://github.com/zaguiini/react-native-pure-jwt

[44] “Funciones en la nube para Firebase | Cloud Functions for Firebase,” Firebase. Accessed: Jun. 21, 2024. [Online]. Available: https://firebase.google.com/docs/functions?hl=es

[45] “React Native Testing Library.” Accessed: May 30, 2024. [Online]. Available: https://callstack.github.io/react-native-testing-library/

[46] “Chai.” Accessed: Jun. 16, 2024. [Online]. Available: https://www.chaijs.com/

[47] “Goodbye Mumbai, Hello Amoy!,” thirdweb. Accessed: Jun. 21, 2024. [Online]. Available: https://blog.thirdweb.com/goodbye-mumbai-hello-amoy/

[48] “Reglamento General de Protección de Datos,” *Wikipedia, la enciclopedia libre*. Apr. 30, 2024. Accessed: May 19, 2024. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Reglamento_General_de_Protecci%C3%B3n_de_Datos&oldid=159819851

[49] “Lucha contra el blanqueo de capitales y la financiación del terrorismo.” Accessed: May 19, 2024. [Online]. Available: https://www.consilium.europa.eu/es/policies/fight-against-terrorism/fight-against-terrorist-financing/

[50] “KYC: Qué es Know Your Customer. Análisis en profundidad.” Accessed: Jun. 19, 2024. [Online]. Available: https://www.tecalis.com/es/blog/kyc-know-your-customer

[51] C. E. Consulting, “NFTs: situación legal de los archivos digitales,” CE Consulting. Accessed: May 19, 2024. [Online]. Available: https://ceconsulting.es/blog-ceconsulting/nfts-situacion-legal-archivos-digitales/

[52] “The MIT License,” Open Source Initiative. Accessed: Jun. 21, 2024. [Online]. Available: https://opensource.org/license/mit

[53] “Apache License,” *Wikipedia, la enciclopedia libre*. May 16, 2024. Accessed: Jun. 21, 2024. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Apache_License&oldid=160149255

[54] “Terms of use,” Consensys. Accessed: May 19, 2024. [Online]. Available: https://consensys.io/terms-of-use

[55] “Bitcoin,” *Wikipedia*. Mar. 06, 2024. Accessed: Mar. 11, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Bitcoin&oldid=1212219544

[56] “Sustainability | Blockchain tech for the planet.” Accessed: Mar. 11, 2024. [Online]. Available: https://polygon.technology/sustainability

13. Índices de tablas

14. Índices de figuras

15. Anexo

Caso de uso	Descripción del caso de uso	Verificar	Estado	Descripción del problema
UCA01 - Inicio de sesion	1. El usuario vincula pulsa el boton de "Connect" 2. Se abre la aplicacion de billtera y se acepta la solicitud de vinculacion 3. Se regresa a la pantalla de inicio, debe aparecer la cartera vinculada y el boton de "Go to the App" 4. Si se pulsa se navega a la pantalla de eventos disponibles	1. Que se vincula la billetera seleccionada 2. Que aparece el boton de "Go the App" 3. Que se setea correctamente la red sobre la que se va a operar	Falla pocas veces	En la aplicación de billetera tarda en aparecer el mensaje para confirmar la vinculación o se toma mucho tiempo. Posible problema del SDK de Thirdweb o aplicaciones de billteras
UCA02 - Visualizar eventos disponibles	1. Se muestran una lista vertical escroleable con los eventos disponibles 2. Los eventos muestran su nombre, imagen, lugar, fecha y las entradas restantes disponibles para comprar 3. Si el evento ha vendido todas sus entradas, se muestra etiqueta de "Sold out"	1. Que se muestra un loader hasta que se cargan los eventos en pantalla 2. Que se muestran todos los eventos 3. Que los campos de los eventos son los correctos 4. Que la lista es scrolleable y se puede acceder a todos los eventos	Ok	
UCA03 - Busqueda	1. Si el usuario pulsa la lupa de busqueda, aparece un campo de texto para que pueda empezar a escribir 2. Si el usuario escribe, se muestra automaticamente aquellos eventos que coincidan en nombre que el contenido insertado en el campo de busqueda 3. Si el usuario navega hacia atras (con el boton del sistema) o pulsa el boton de cerrar de la busqueda esta se cierra y se vuelven a mostrar los eventos sin filtrar. 4. Si ningun evento coincide con la busqueda, se muestra mensaje indicandolo 5. Si el usuario cambia a otra pantalla, se cierra la busqueda	1. Que solo aparecen los eventos que coinciden parcialmente o totalmente en nombre con la busqueda 2. Que al cerrar la busqueda, ya sea por pulsar el botón o una navegación se cierra esta y se vuelven a mostrar todos los eventos 3. Que se muestre mensaje indicando que no hay eventos que coinciden con la busqueda	Ok	
UCA04 - Visualizacion de un evento	1. Si el usuario pulsa un evento se navega a la pantalla de detalle de este 2. Se muestra el nombre, imagen, lugar, fecha, hora, organizador y detalles 3. Se muestra un botón con el precio de la entrada 4. Si la entrada ya ha sido adquirida se deshabilita dicho botón 5. Si el evento ha sido cancelado se muestra un mensaje en la parte inferior 6. Si el usuario pulsa el botón de back se navega a la pantalla anterior	1. Que se muestra correctamente toda la información del evento y es escrolleable 2. Que el botón de compra indique el precio si no se ha comprado una entrada aún 3. Que el botón de compra indique que ya se ha adquirido la entrada si se ha comprado previamente 4. Que el botón de compra desaparezca y muestre el mensaje correspondiente si el evento ha sido cancelado	Ok	
UCA05 - Comprar una entrada	1. Si el usuario pulsa el boton de comprar y este no tiene una entrada se navega a la aplicación de billetera para firmar la transacción 2. Si se firma se regresa a la aplicación y se muestra un loader hasta que llegue la respuesta del smartcontract 3. Cuando se recibe la respuesta se navega a la pantalla de detalle de compra 4. Dentro de la pantalla de detalle de compra aparece la entrada con la información correspodiente y un botón de "See my tickets" para volver a la pantalla de las entradas del usuario. Debe aparecer la nueva entrada ahí 5. Si rechaza la transacción o ocurre un error inesperado, se muestra un popup informando al usuario	1. Que tras firmar la transacción en la billetera se regresa a la aplicación 2. Que tras realizarse el minteo se navega a la pantalla de detalles de compra y al pulsar el botón de "See my tickets" se navega a la pantalla de las entradas del usuario y aparece la entrada adquirida.	Falla pocas veces	En la aplicación de billetera tarda en aparecer el mensaje para confirmar la transacción o se firma pero no tarda en actualizarse en la aplicación. Posible problema del SDK de Thirdweb o aplicaciones de billteras
UCA06 - Visualizar entradas compradas	1. Se muestran todas las entradas compradas por el usuario. 2. Se muestran en este orden: Primero las entradas que están habilitadas para ser validadas, después las entradas que aún no están habilitadas para su validación, a continuación las que están validadas y por último aquellas entradas cuyos eventos han sido cancelados. 3. Si el usuario pulsa una entrada en estado de evento no cancelado se navega al detalle de la entrada 4. Si el usuario pulsa una entrada en estado de evento cancelado, se muestra un popup indicando que el evento ha sido cancelado y sus fondos devueltos.	1. Que se muestran todas las entradas del usurio en todo momento en una lista vertical escrolleable 2. Que se muestran siguiendo el orden descrito 3. Que se muestra el detalle de la entrada en caso de evento no cancelado 4. Que se muestra pop indicando que el evento ha sido cancelado en caso de ser así	Ok	
UCA07 - Visualizar detalle de entrada	1. Si el usuario selecciona una entrada se muestra el detalle de esta. Este es el nombre el id de la entrada, nombre del evento, fecha, hora y lugar 2. Si la entrada está disponible para ser validada se muestra un botón para empezar la validación 3. Si la entrada no está para ser validada se muestra un botón para transferirla además de un mensaje informativo indicando que no esta lista para ser validada	1. Que los detalles de la entrada se muestran correctamente 2. Que se muestra el botón de validar si el evento está en estado de validación 3. Que se muestra el botón de transferir si ele evento aún no está en estado de validación	Ok	
UCA08 - Transferir una entrada: Paso 1	1. Si el usuario pulsa el botón de transferir se navega al formulario para transferir la entrada 2. Si el usuario pulsa el campo del formulario se le abre el teclado para que pueda escribir 3. Si el usuario pulsa el botón de Pegar se pega automáticamente el contenido del portapapeles en el input del formulario 4. Si el usuario inserta una dirección válida de Ethereum se muestra un mensaje indicándolo y se habilita el botón de confirmación 5. Si el usuario inserta una dirección no válida de Ethereum se muestra un mensaje de error indicándolo 6. Si el usuario pulsa el botón de confirmación se navega a la pantalla de confirmación de la transferencia	1. Que se abre el teclado al pulsar el input del formulario 1. Que se pega el contenido en el input al pulsar el botón de pegar 2. Que el botón no se habilita hasta insertar una dirección valida de Ethereum 3. Que se muetra el mensaje de error o valdiación correcta según el input insertado	Ok	
UCA08 - Transferir una entrada: Paso 2	1. Se muestra la información de la entrada que se va a transferir 2. Se indica de qué billetere a qué billetera se va a transferir la entrada 3. Se habilita un checkbox para garantizar que el usuario es consciente de que va a transferir su entrada y puede no recuperarla 4. Si se se pulsa el botón de transferir se abre la billetera y si se firma la transacción se muestra un pop 5. Si la billetera de destino ya tiene una entrada para ese evento se muestra un popup indicandolo 6. Si hay un error se muestra un pop de error	1. Que la información de la entrada sea la correcta 2. Que las direcciones sean las correctas 3. Que solo pueda pulsar el botón de transferir si ha marcado el checkbox de confirmación 4. Que al pulsar el botón de transferir, se abra la aplicación de billetera y si se firma se retorna a la aplicación y se muestra un pop up indicando que se ha transferido correctamente 5. Que en caso de que ese usuario ya tenga una entrada para ese evento se decline la transacción y se muestra un pop up indicandolo 6. Que si hay un error se muestra un mensaje de error	Falla pocas veces	En la aplicación de billetera tarda en aparecer el mensaje para confirmar la transacción o se firma pero no tarda en actualizarse en la aplicación. Posible problema del SDK de Thirdweb o aplicaciones de billteras
UCA10 - Validar entrada mediante escaneo QR	1. Se muestra el QR de la entrada 2. Cuando este es validado, si lo hace satisfactoriamente se navega a la pantalla de entrada validada satisfactoriamente 3. El usuario puede navegar hacia atrás y escoger otro metodo de validación	1. Que el codigo QR se muestre correctamente 2. Que no se avance hasta la pantalla de entrada validada satisfactoriamente hasta que esta no lo esté 3. Que el usuario pueda navegar hacia atrás y escoger otro método de validación	Ok	
UCA11 - Validar entrada mediante escaneo NFC	1. Se muestran las indicacionnes en la pantalla y se activa el HCE 2. Cuando este es validado, si lo hace satisfactoriamente se navega a la pantalla de entrada validada satisfactoriamente 3. El usuario puede navegar hacia atrás y escoger otro metodo de validación	1. Que se active correctamente el HCE 2. Que no se avance hasta la pantalla de entrada validada satisfactoriamente hasta que esta no lo esté 3. Que el usuario pueda navegar hacia atrás y escoger otro método de validación	Ok	
UCV01 - Seleccioar un evento	1. El usuario aterriza en la aplicación 2. El usuario selecciona el evento del cual va a validar entradas 3. Se deben mostrar los eventos que están en "On going" en una lista vertical scrolleable con la información esencial de estos: nombre, imagen, hora, lugar y organizador 4. Si el usuario pulsa un evento se navega a la pantalla de selección de método de validación	1. Que se muestren correctamente los eventos que estan en "On going" 2. Que si el usuario pulsa un evento navega a la pantalla de selección de método de validación	Ok	
UCV02 - Validar mediante código QR	1. Si el usuario ha seleccionada validar con QR se muestra la pantalla de validación con QR 2. Se activa la cámara trasera del dispositivo 3. Se escanean los códigos QR que esten al alcance de la cámara 4. Inicialmente se muestra en "Ready" 5. Si se escanea se muestra "Checking ticket" 6. Tras finalizar la comprobación muestra "Ticket validated" si no hubo problemas" 7. Si la entrada ya se validó muestra "Ticket already validated" 8. Si se escanea un código QR que no es un entrada o ocurre un error se muestra "Error validating ticket" 9. Una vez terminado el escaneo de una entrada valida o invalida se retorna al estado de "Ready"	1. Que la cámara se inicie cuando el usuario llega esta pantalla 2. Que escanee los códigos QR 3. Que se actualice el estado de la validación correctamente 4. Que se puedan escanear varias entradas consecutivamente 5. Que si el usuario navega hacia atrás vuelva a la pantalla de selección de método de validación	Ok	
UCV03 - Validar mediante NFC	1. Si el usuario ha seleccionada validar con NFC se muestra la pantalla de validación con NFC 2. Seel NFC del dispositivo 3. Si el usuario pulsa el botón de escanear, se escanea la targeta NFC cerca (el dispositivo del asistente) 4. Inicialmente se muestra en "Tap to scan" 5. Si se escanea se muestra "Checking ticket" 6. Tras finalizar la comprobación muestra "Ticket validated" si no hubo problemas" 7. Si la entrada ya se validó muestra "Ticket already validated" 8. Si se escanea un código NFC que no es una entrada o ocurre un error se muestra "Error validating ticket" 9. Una vez terminado el escaneo de una entrada valida o invalida se retorna al estado de "Tap to scan"	1. Que se inicie la validación cuando el usuario pulsa el botón 2. Que escanee los targetas NFC 3. Que se actualice el estado de la validación correctamente 4. Que se puedan escanear varias entradas consecutivamente 5. Que si el usuario navega hacia atrás vuelva a la pantalla de selección de método de validación	Ok	
UCV04 - Cambiar de método de validación	1. Si el usuario esta en la pantalla de validación mediante código QR se navega a la de validación mediante NFC 2. Si el usuario esta en la pantalla de validación mediante NFC se navega a la de validación mediante código QR	1. Que estando en la pantalla validación mediante QR se navega a la de validación mediante NFC 2. Que estando en la pantalla validación mediante NFC se navega a la de validación mediante QR 3. Que al navegar atrás en cualquiera de las pantallas no se navega a la pantalla donde se ha estado anteriormente sino a la de selección de método de validación	Ok	