****

**Proyecto de Desarrollo de Aplicaciones Web**

**Módulo código IFC007 (antiguo 0616)**

**Índice**

[**1.** **Introducción** 3](#_Toc194233628)

[**2.** **Temporalización** 4](#_Toc194233629)

[**3.** **Life Cycle** 5](#_Toc194233630)

[**a.** **Análisis** 5](#_Toc194233631)

[**i.** **UML** 5](#_Toc194233632)

[**ii.** **Mockup** 6](#_Toc194233633)

[**b.** **Diseño** 13](#_Toc194233634)

[**i.** **Arquitectura** 13](#_Toc194233635)

[**ii.** **Diagrama E/R y Casos de uso** 14](#_Toc194233636)

[**iii.** **Diseño gráfico** 14](#_Toc194233637)

[**c.** **Implementación** 34](#_Toc194233638)

[**d.** **Testing** 43](#_Toc194233639)

[**e.** **Despliegue** 44](#_Toc194233640)

[**4.** **Manual de usuario** 45](#_Toc194233641)

[**5.** **Herramientas y tecnologías empleadas** 46](#_Toc194233642)

[**a.** **Miro** 46](#_Toc194233643)

[**b.** **Whimsical** 46](#_Toc194233644)

[**c.** **Adobe Photoshop** 47](#_Toc194233645)

[**d.** **Adobe Illustrator** 47](#_Toc194233646)

[**e.** **Node** 48](#_Toc194233647)

[**f.** **Express** 48](#_Toc194233648)

[**g.** **Nodemailer** 49](#_Toc194233649)

[**h.** **Bitcoin Core** 49](#_Toc194233650)

[**i.** **JWT (JSON Web Token)** 50](#_Toc194233651)

[**j.** **Socket.IO** 50](#_Toc194233652)

[**6.** **Propuestas de mejora** 60](#_Toc194233653)

[**7.** **Valoración personal** 61](#_Toc194233654)

[**8.** **Puntos a destacar del proyecto** 62](#_Toc194233655)

1. [**Introducción**](#Indice)

Wallacoin es una aplicación web que conecta a compradores con vendedores de bitcoin para que puedan intercambiar su criptomoneda por dinero fiduciario (dólares, euros, libras, yenes…) en efectivo, de manera anónima y sin esperas.

**¿Qué aporta realmente Wallacoin?**

Existen otras plataformas para intercambiar criptomonedas en efectivo; sin embargo, Wallacoin destaca al lograr que una transacción ordinaria dentro de la blockchain (tx onchain), que normalmente tardaría alrededor de 10 minutos por el proceso de minería, ¡sea inmediata!

Gracias a esto, los usuarios pueden intercambiar bitcoins entre particulares sin esperas, preservando totalmente el anonimato inherente al mundo de las criptomonedas.

**¡Créate una cuenta de correo**

**y**

**adquiere tus primeros bitcoins de manera anónima e instantánea!**

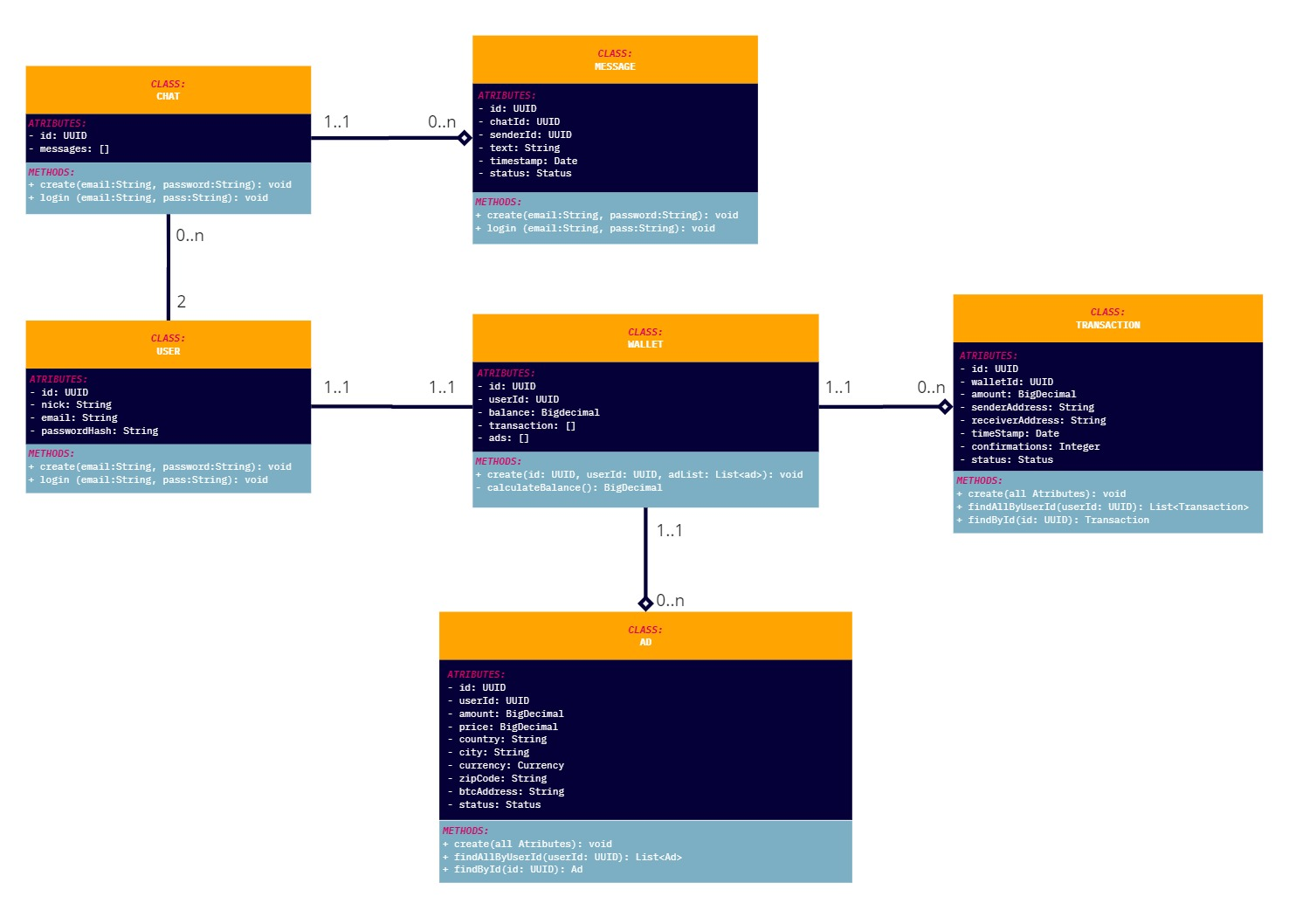
**¿Qué más se puede pedir?**

1. [**Temporalización**](#Indice)



1. **[Life Cycle](#Indice)**
   1. **Análisis**
      1. **UML**

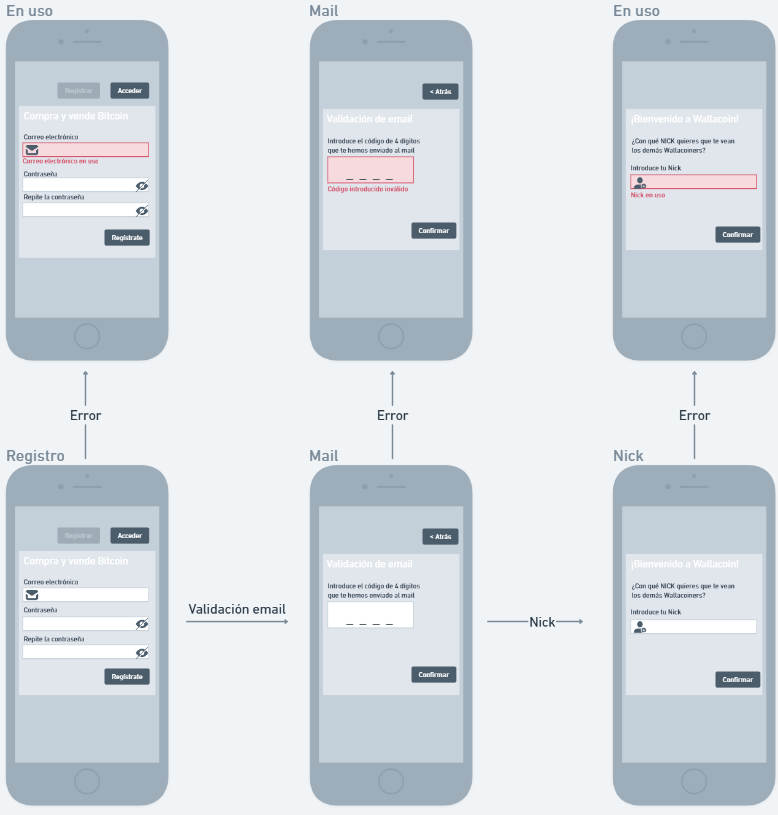
Comenzamos diseñando un diagrama UML para la disposición de las clases que vamos a utilizar en la aplicación:

* + - * User
      * Wallet
      * Transaction
      * Ad
      * Chat
      * ****Message
    1. [**Mockup**](#Analisis)

Para poder realizar un diseño en condiciones utilizamos Whimsical que nos permite desarrollar un mockup básico con apariencia de terminal móvil.  
A través de este link que he dejado público se puede ver el mockup completo:

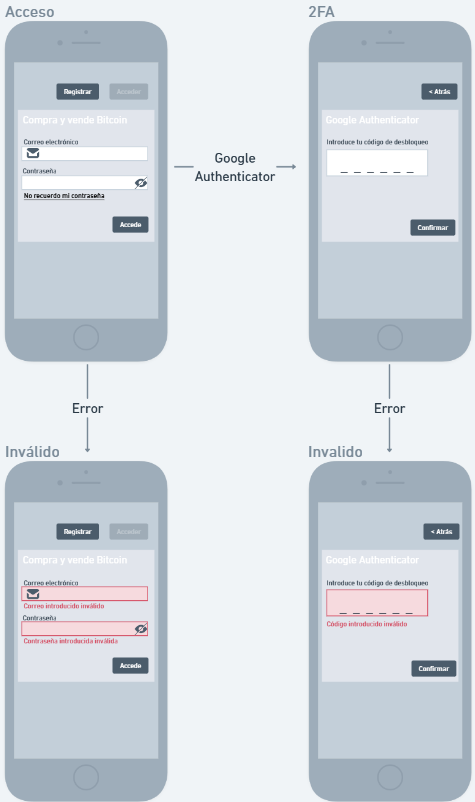
<https://whimsical.com/movil-S2obzAXGDSRYZPwhp6YoAZ>

**Checkin:**

****Un apartado donde se solicita tan solo un email y un password.

A modo de verificación de dicha dirección de correo se envía un número de 4 cifras que el usuario deberá introducir correctamente para finalizar el proceso de checkin.

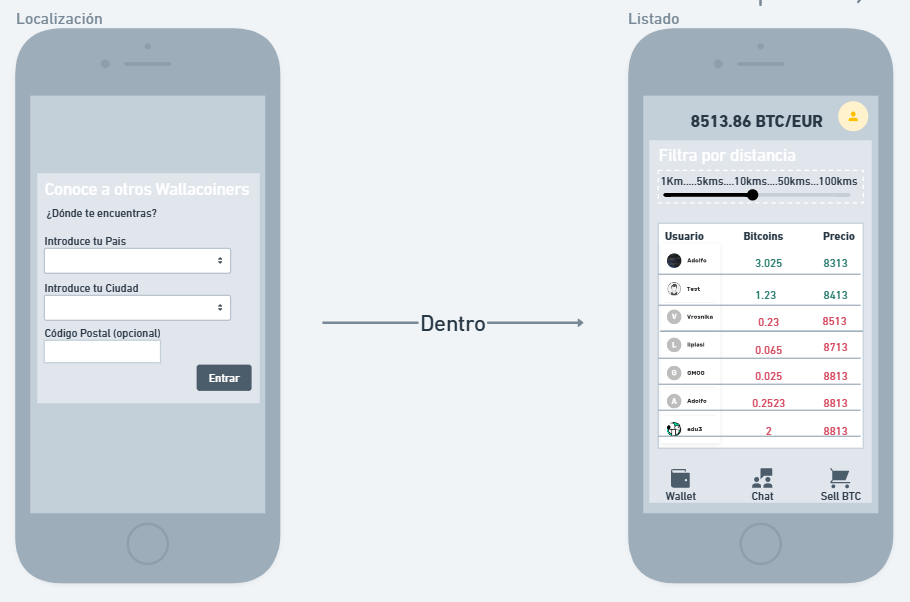
**Login:**

****El usuario introduce su email y su password.

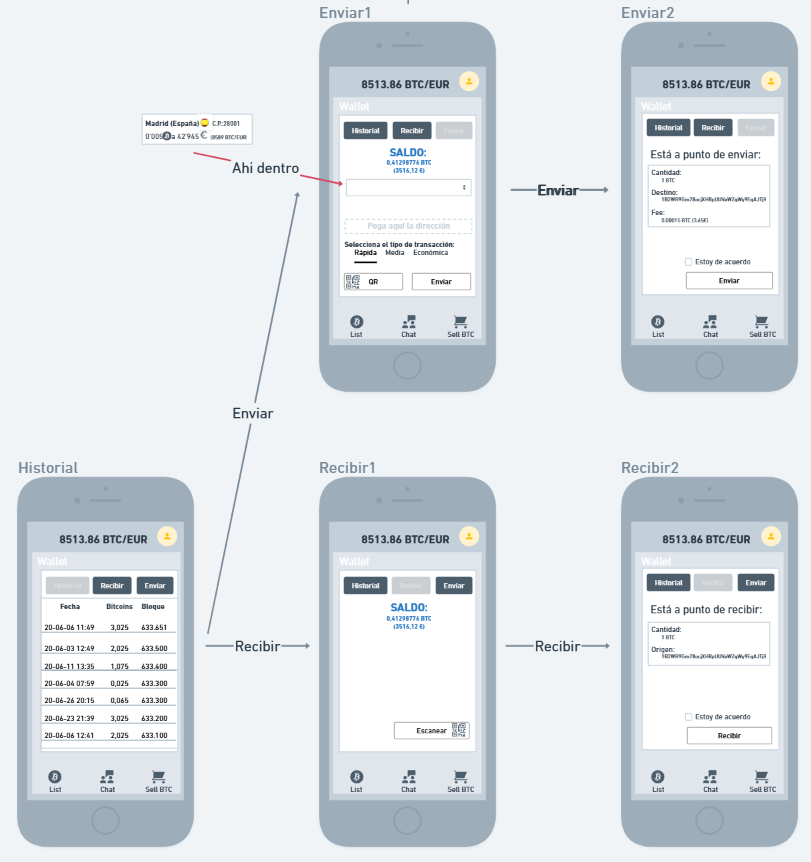
**Location y List:**

Cuando nos logueamos pasamos a una pantalla previa a ver el listado de los anuncios, la localización, donde indicamos country, state y city.

Una vez introducida la localización ya podemos visualizar el listado con todos los anuncios de venta de bitcoin por particulares de nuestra zona.

****

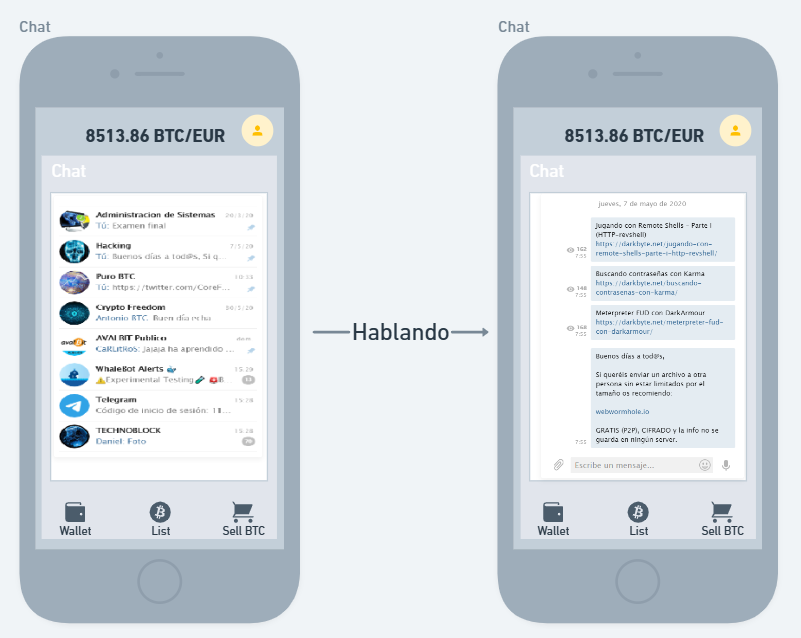
**Wallet:**

En este apartado se puede enviar y recibir bitcoin de cualquier usuario de una manera sumamente sencilla: Escaneando un QR****.

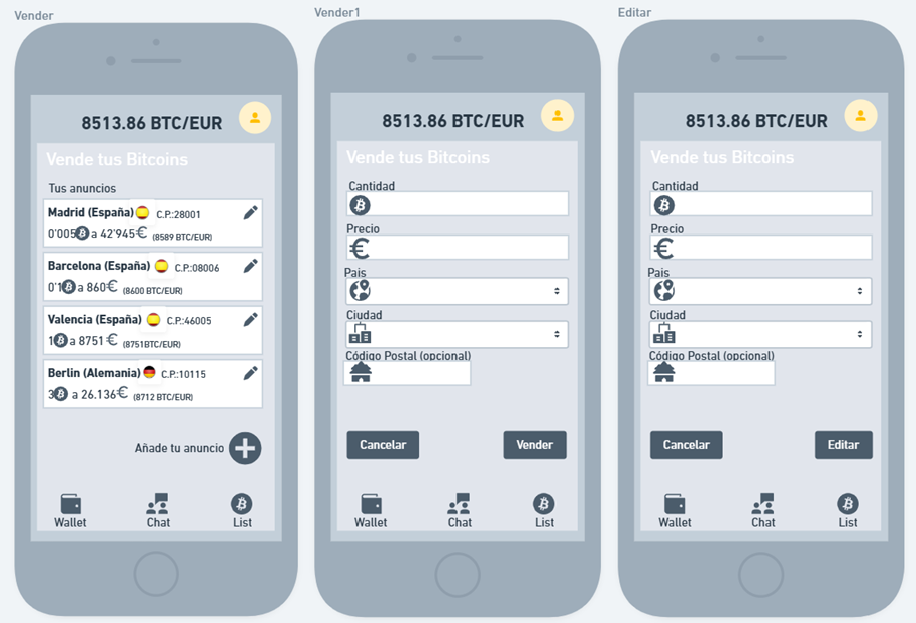
**Chat:**

Un chat muy similar a WhatsApp y Telegram, donde los usuarios pueden hablar entre ellos tan solo pinchando en el anuncio.

Los mensajes quedan almacenados en la base de datos para que los usuarios puedan consultar la conversación completa cada vez que lo deseen.

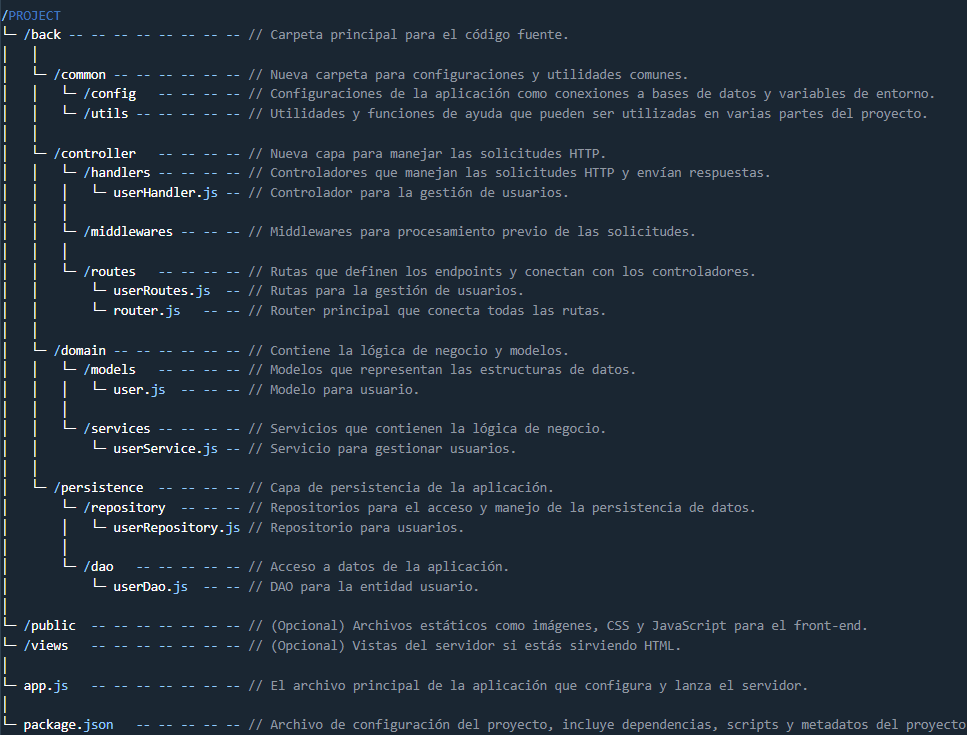


**Ads:**

Aquí podemos visualizar el listado de todos nuestros anuncios y con tan solo pinchar en uno de ellos lo podemos editar sin problemas.

*\*\* Existen algunas otras pantallas de menos relevancia como el panel del usuario para modificar algunos parámetros y las pantallas de error.  
Todas estas pantallas se pueden visualizar con más detalle en el* [*link*](https://whimsical.com/movil-S2obzAXGDSRYZPwhp6YoAZ) *facilitado.*

* 1. **[Diseño](#LifeCycle)**
     1. **Arquitectura**

****Lo primero que se ha diseñado, al tener clara la estructura de clases, es la arquitectura de la aplicación.

* + 1. [**Diagrama E/R y Casos de uso**](#Diseño)

El segundo punto del diseño que se ha realizado es el diagrama E/R y el de casos de uso.  
Como se trata de una aplicación que va a salir a producción real y en la que está basada este modelo de negocio este apartado no lo puedo hacer público.  
Durante la exposición enseñaré dichos puntos y realizaré las explicaciones pertinentes, pero como es lógico, no puedo dejar por escrito toda esa información tan vulnerable.

* + 1. [**Diseño gráfico**](#Diseño)

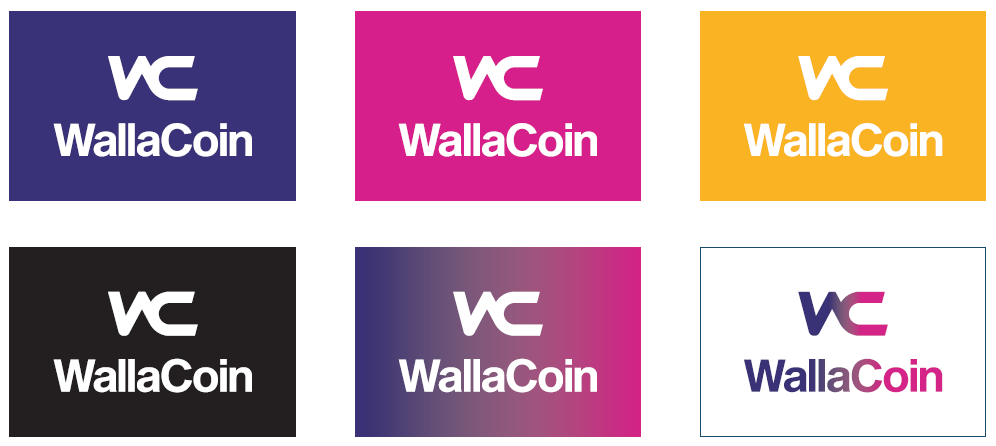
La tercera sección y quizás la más tediosa ha sido la referente a estilos y diseño gráfico.  
Esta parte ha sido encomendada a una empresa profesional de diseño gráfico que ha creado, partiendo del mockup, una versión bastante aproximada a la final, afinando la paleta de colores, fuentes, tamaños y disposición de los distintos elementos del frontend.

En este punto, aunque a nivel muy básico, me ha tocado experimentar tanto con Photoshop como con Illustrator, pues son totalmente necesarios para retocar parte de los diseños realizados por la empresa.

Como puntos principales de esta sección señalaremos

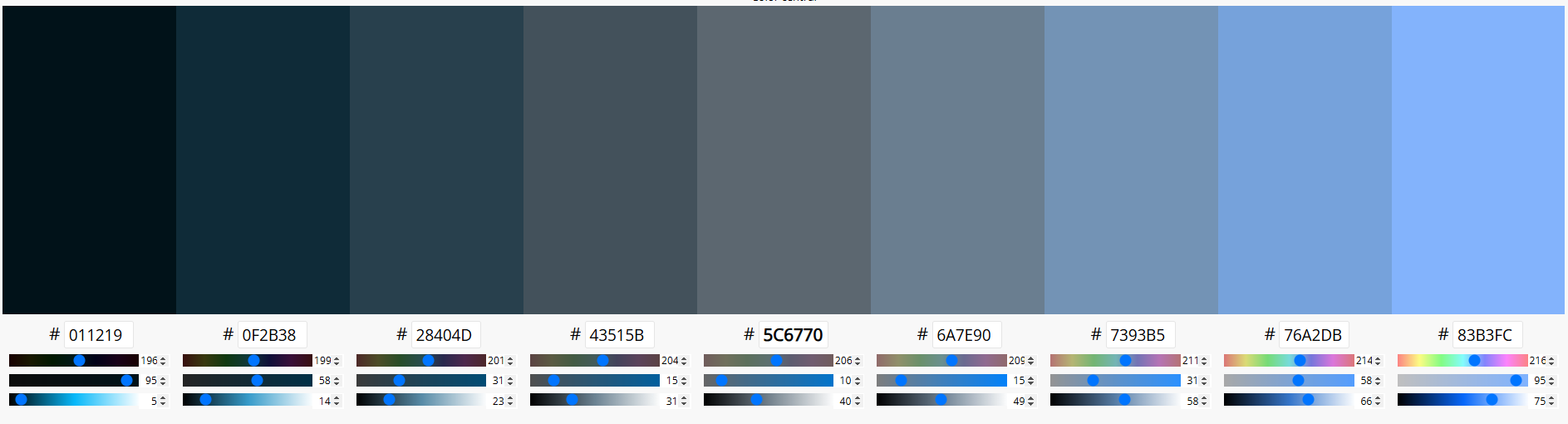
**Logo:**

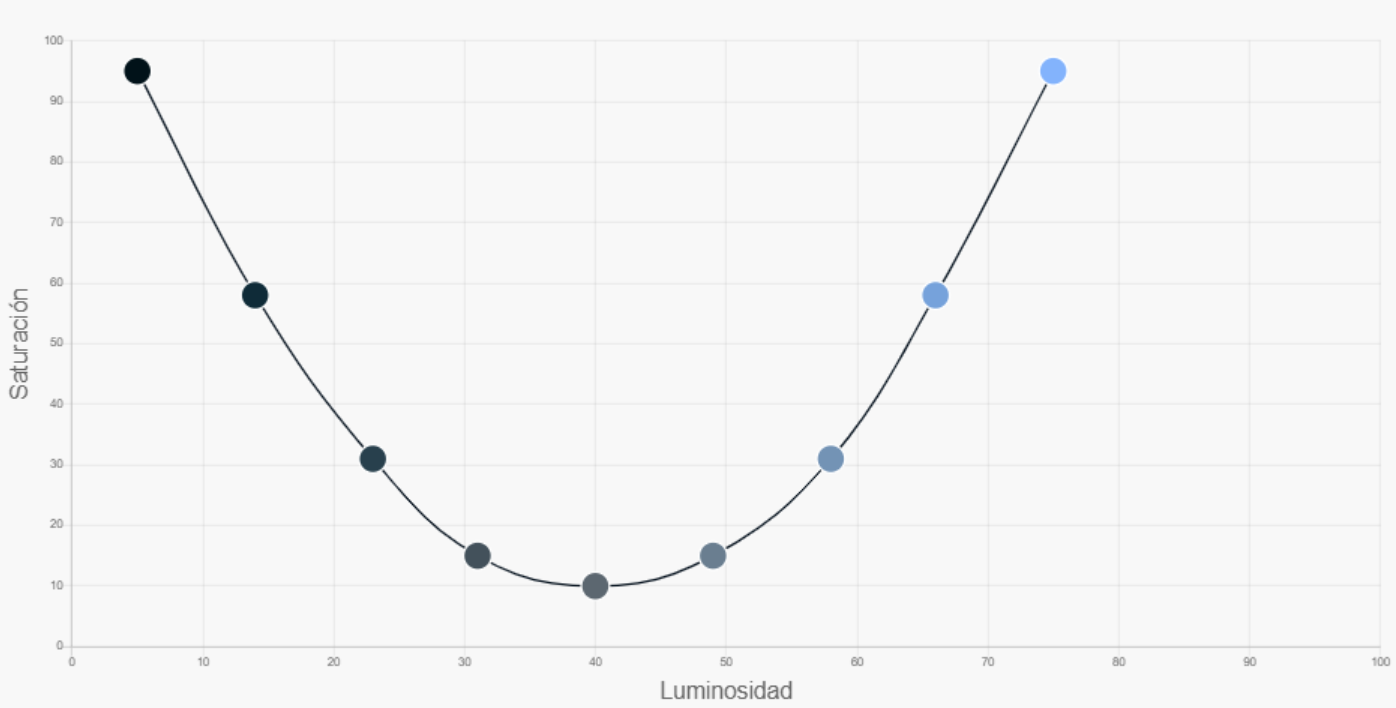


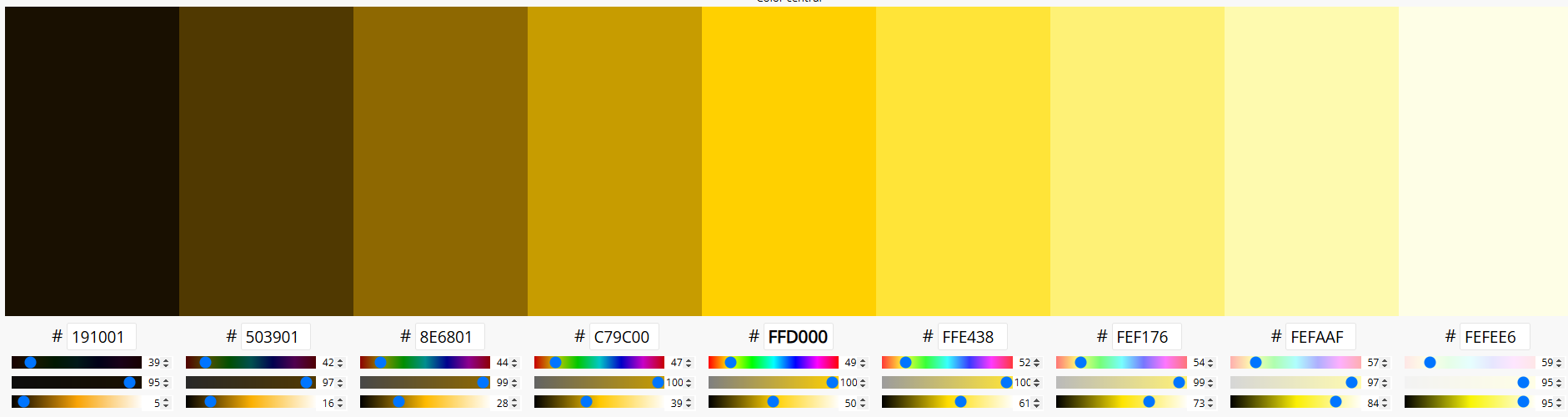
**Logo para diferentes plataformas y fondos:**

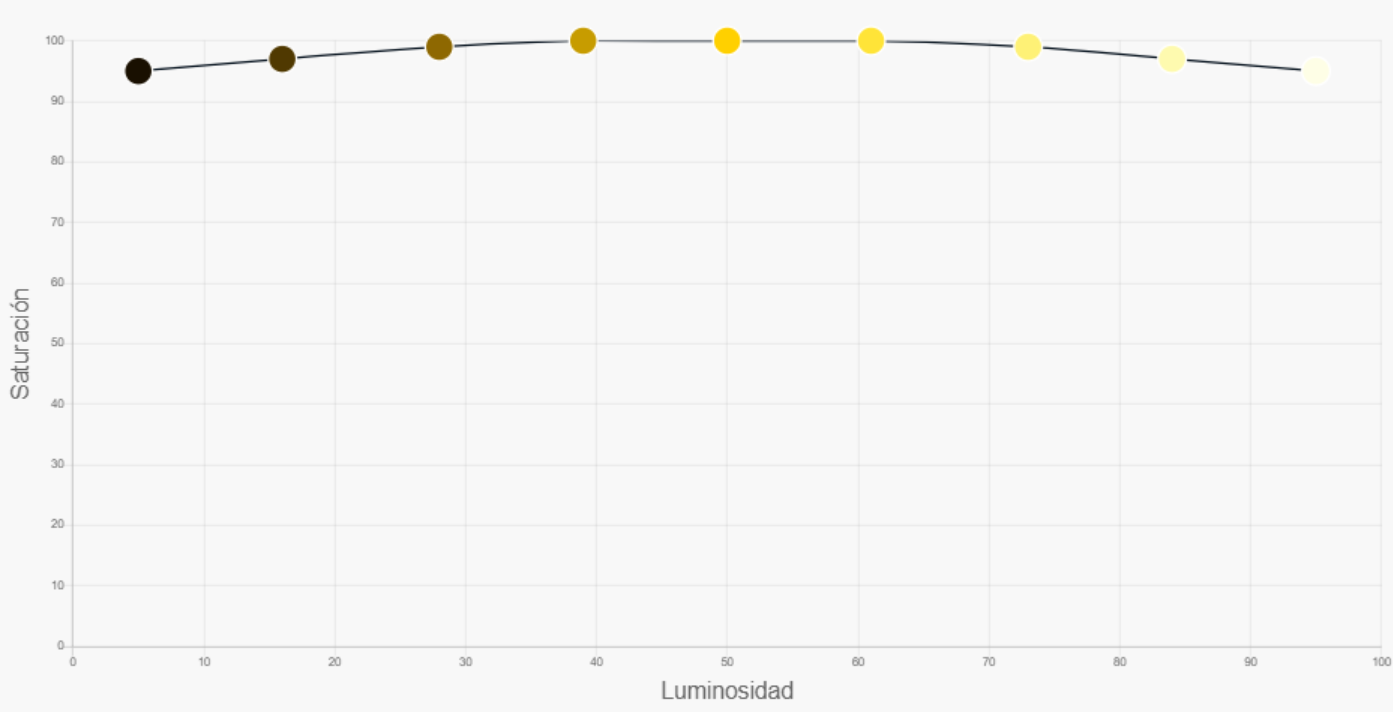
**Tipografía:**

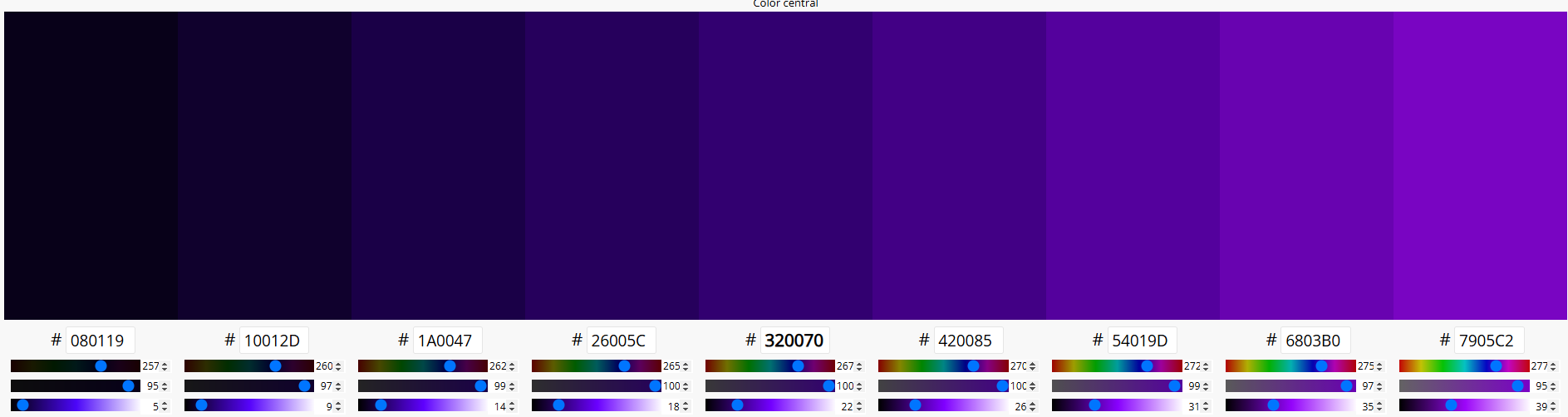
**Paleta de colores** (diseñada con la herramienta de [Lorenzo](https://logongas.github.io/color-schemes-generator/))

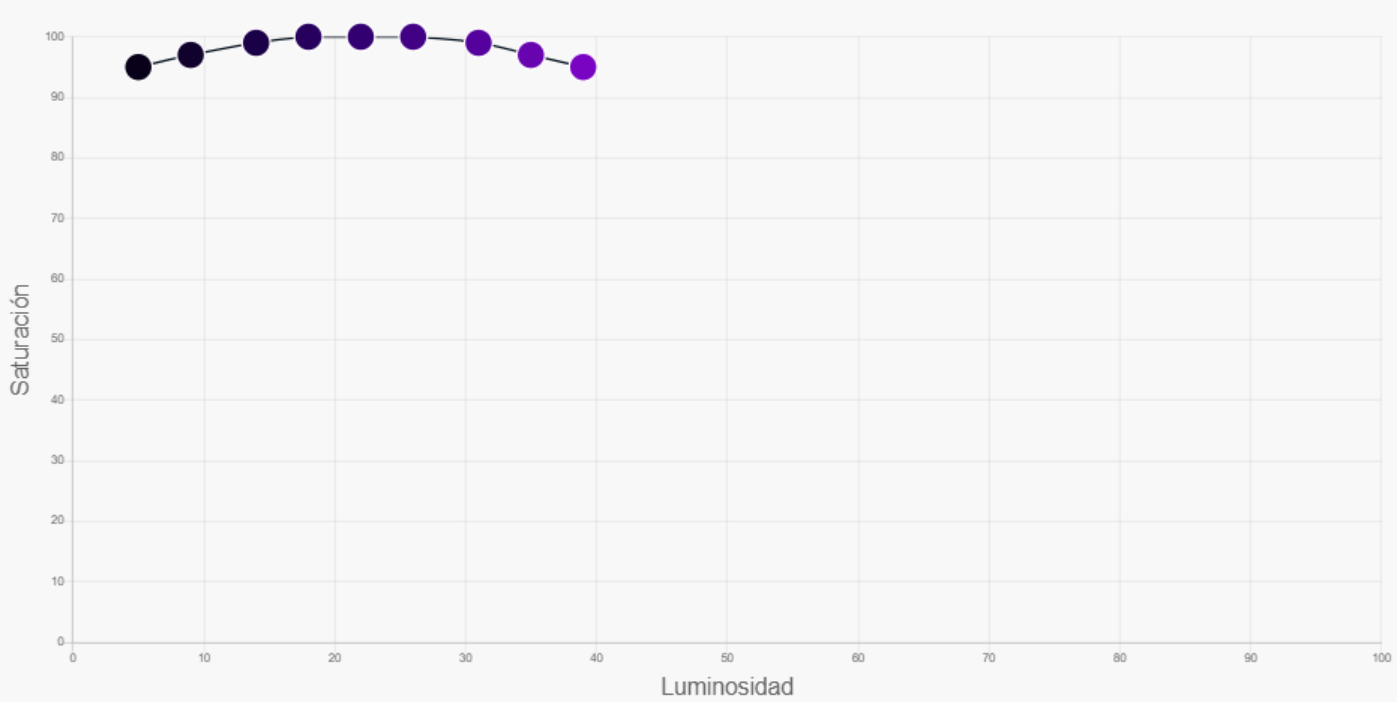


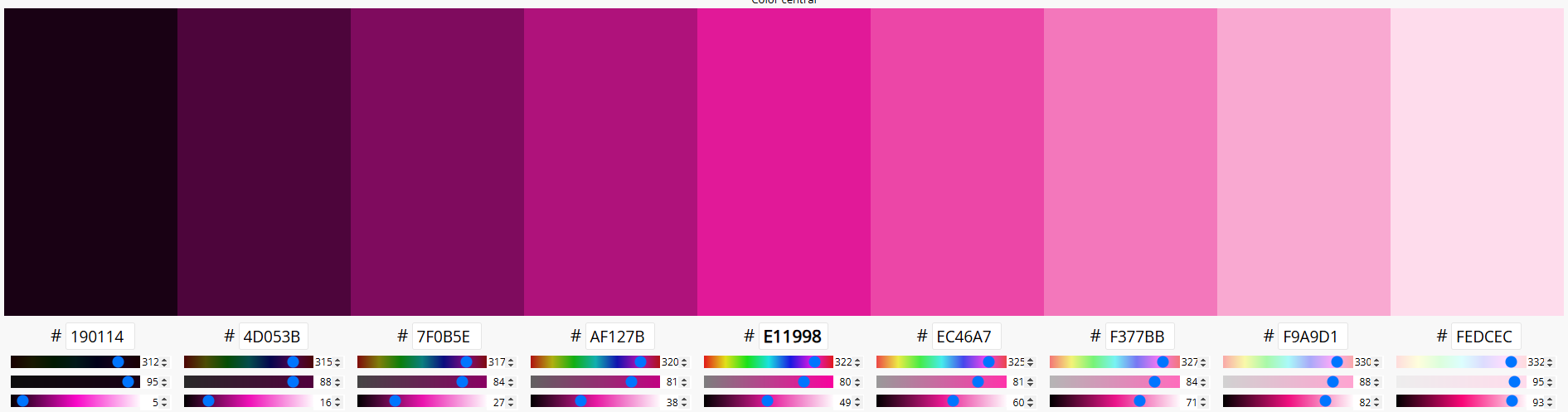


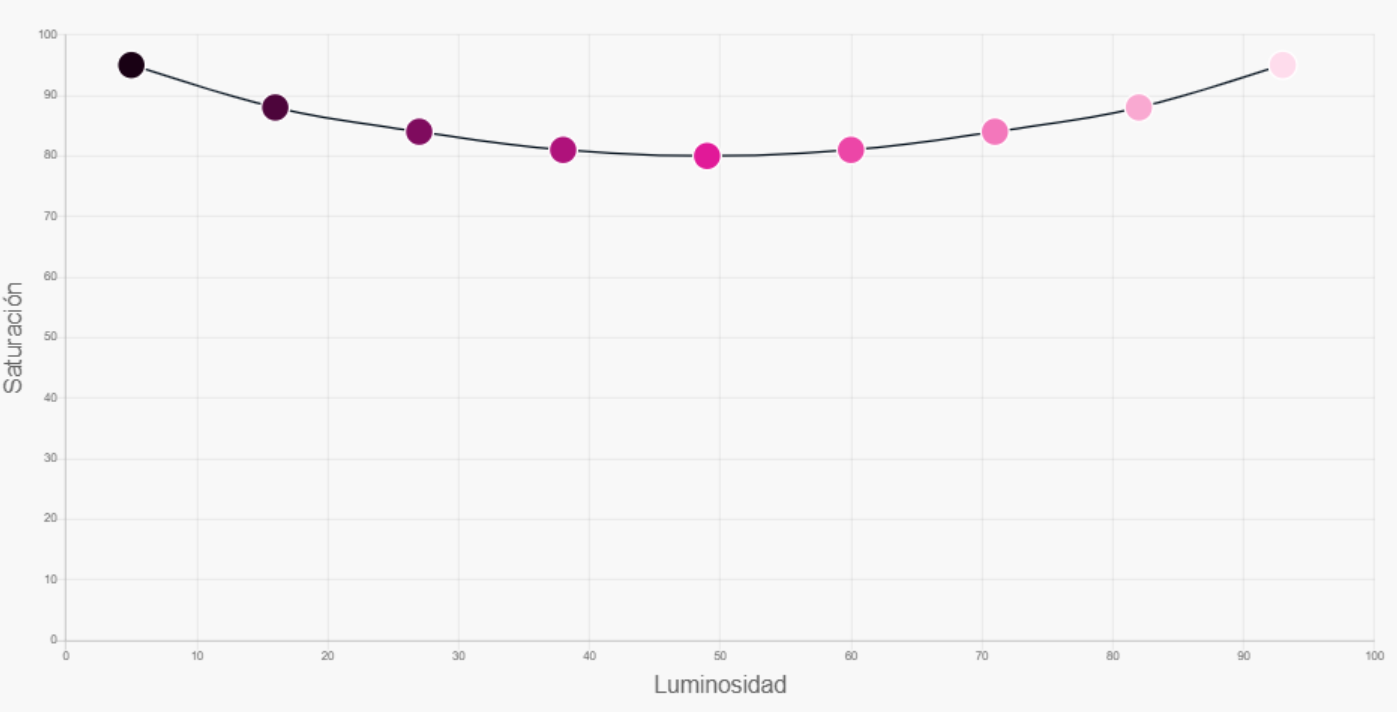












**Distribución de los elementos y mockup profesional:**

Este lo podemos marcar como hito principal de la aplicación, pues tras meses de mucho trabajo por fin vemos un producto “quasi” terminado de la app.  
Justo aquí ya podemos hacernos una idea de cómo va a ser la apariencia real de la aplicación y poder compartir con amigos y familiares para recibir cierto feedback e ir concretando últimos detalles de visualización.

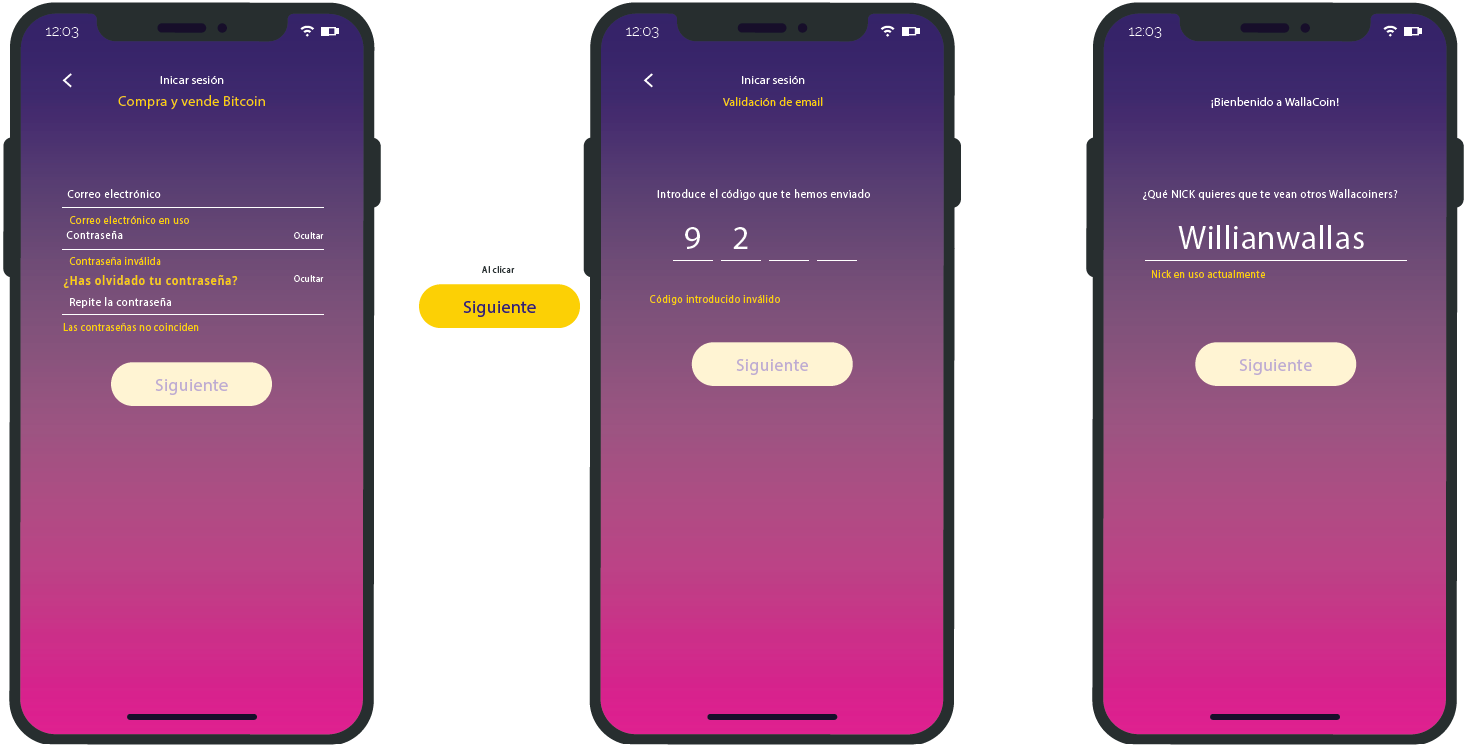
**Inicio y Home:**

Una pantalla blanca con el logo de colores para el INICIO y la inversa para el HOME, donde te puedes loguear en la aplicación o si aún no tienes cuenta, puedes crearte una.

**Checkin:**

El checkin está estructurado en 3 fases:

1. Introducción de credenciales (EMAIL y PASSWORD).
2. Verificar email (NÚMERO DE 4 CIFRAS que el usuario recibe en su correo).
3. Introducción de un NICK (para preservar el anonimato).



**Login:**

La empresa de diseño gráfico cobraba por unidad de pantallas diseñadas, así que se tomó la decisión de no diseñar el login porque suponía un coste innecesario al tratarse de pantallas tan similares a las del checkin.

A fin de cuentas, el mockup es una simple guía para que el desarrollador de frontend sepa exactamente qué tiene que hacer y de qué manera.

**Location y List:**

El usuario una vez dentro debe seleccionar country, state y city.  
Tal y como se puede apreciar en la imagen la idea inicial era país, ciudad y código postal, pero se decidió a posteriori modificar estas opciones.

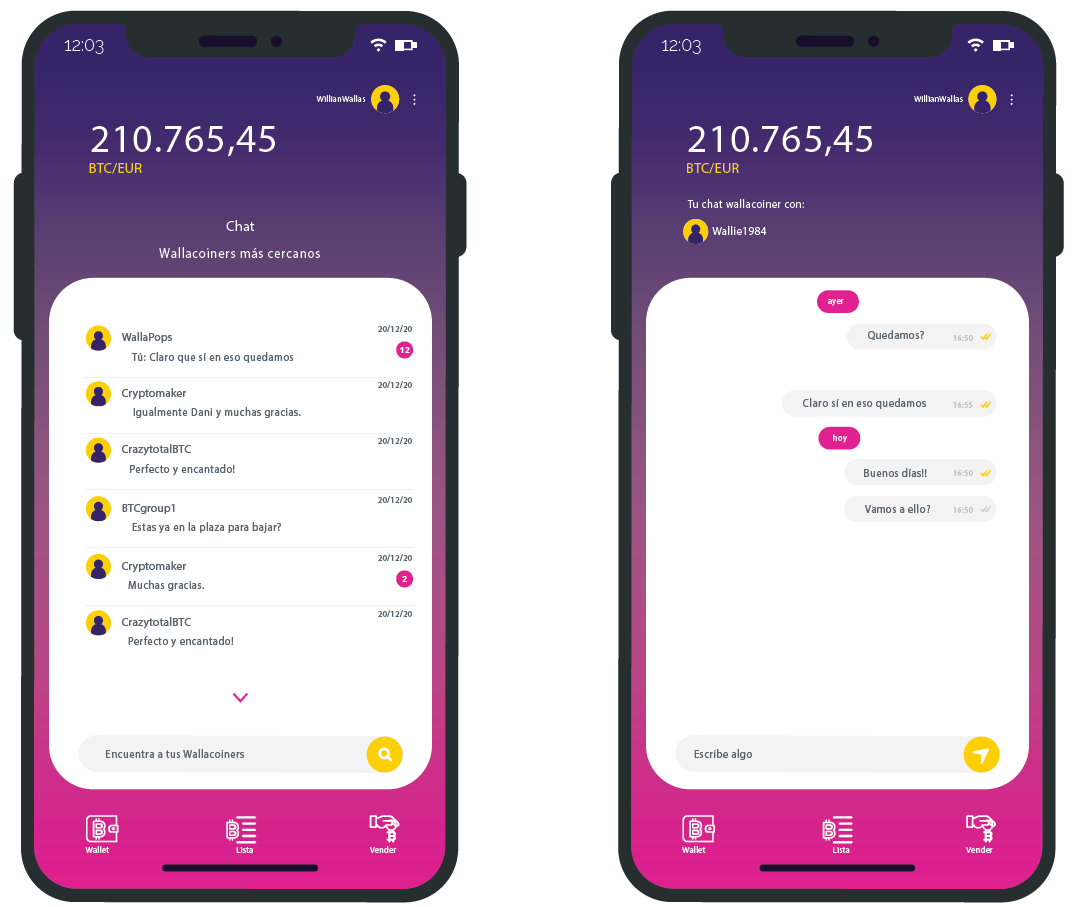
La pantalla de List muestra un listado con todos los anunciantes (vendedores) de bitcoins, que se pueden filtrar por distancia. No tiene ningún sentido ver a vendedores de Japón si se está en España.  
  
Si se pincha sobre cualquier usuario se abrirá automáticamente un chat con este, donde se podrá conversar para posteriormente quedar físicamente e intercambiar el bitcoin por la moneda solicitada.  
  
En este punto ya se puede visualizar el footer, que es una simple botonera con 3 opciones: wallet, chat, list.  
Este footer ha sido modificado a última hora con una opción más: ads, donde se puede ver el listado de anuncios del propio usuario



**Chat:**

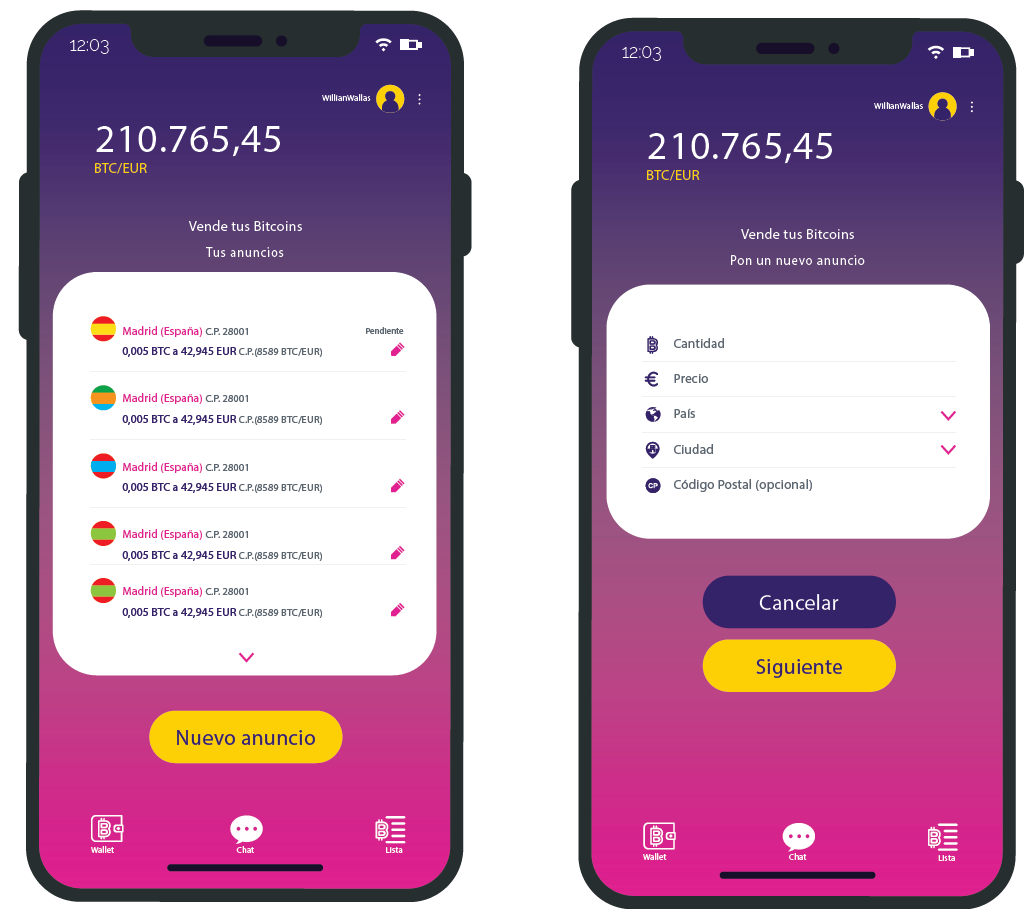
Tal y como se ha comentado, los usuarios pueden comenzar un chat con otro usuario con tan solo hacer click sobre cualquiera de los anunciantes que se visualizan en el panel general.

Si desean volver a ver todos sus chats tan solo tienen que presionar sobre el botón CHAT que se encuentra en el footer.



**Ads:**

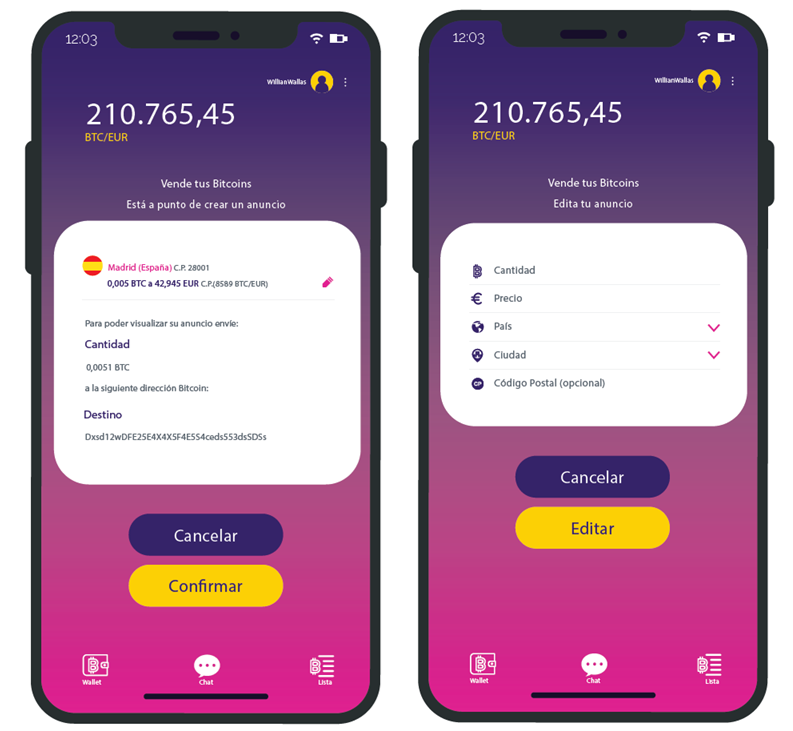
Los usuarios pueden vender sus bitcoins creando un anuncio para ello en la sección de ads.  
Lo primero que visualizará en dicha sección es un listado con todos sus anuncios y ahí mismo aparece también la opción de crear nuevo anuncio.



Para que los anuncios se puedan publicar deben contener saldo, así que deberán enviar la cantidad correspondiente de bitcoin a la dirección generada para ese anuncio concreto.

Dicho anuncio también puede ser editado, aunque no todas las opciones están habilitadas. Tan solo price, country, state y city.

El usuario puede modificar el precio al que desea vender y también la ubicación donde quiere realizar dicha venta, pero bajo ningún concepto puede modificar la cantidad, pues esos bitcoins ya se encuentran en la dirección asignada para el anuncio, en la blockchain de Bitcoin.

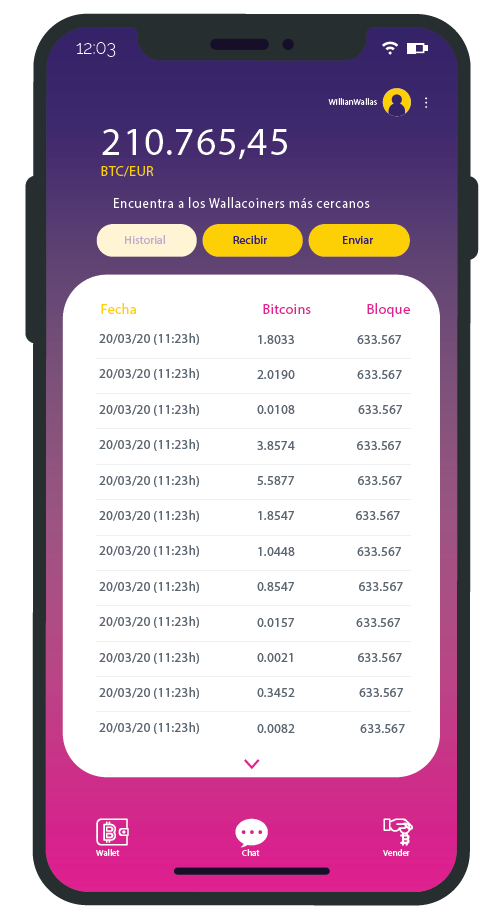


**Wallet:**

Una vez presionado el botón WALLET del footer se nos abre la cartera de Bitcoin que posee 3 opciones: history, receive y send.

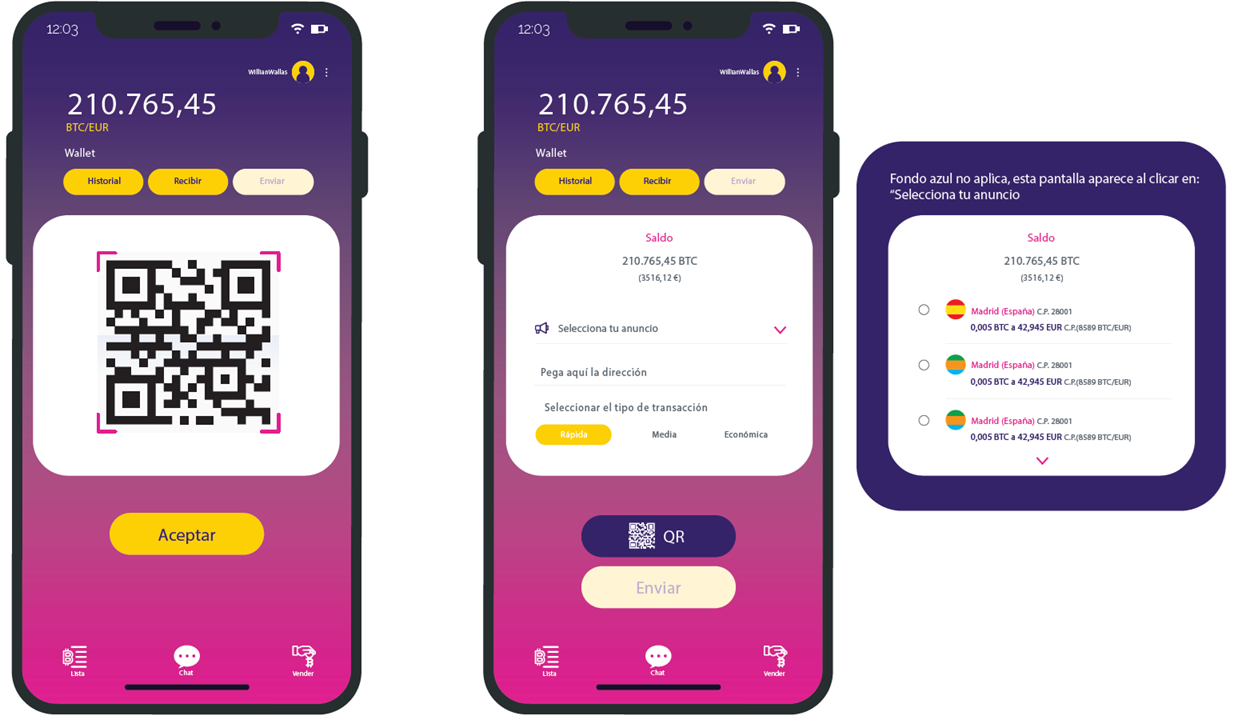
Lo primero que visualizará el usuario por defecto será el history, que muestra el historial de todas sus transacciones.

Si pincha sobre alguna de ellas le llevará directamente al explorador de la cadena de Bitcoin <https://www.blockchain.com/> donde podrá ver con detalle el estado de su transacción: saldo enviado, fecha y hora de la transacción, dirección a la que se envía, número de bloques minados desde ese momento…



El apartado de SEND permite al usuario enviar el bitcoin de 2 maneras: a través de un QR que el comprador escaneará o por la vía tradicional, que consiste en introducir la dirección de Bitcoin a la que se quiere enviar dicho bitcoin.

El vendedor tan solo selecciona su anuncio de un desplegable donde aparecen todos sus anuncios (si es que tuviera más de uno) e introduce la dirección donde quiere enviarlo, o en su defecto, genera un QR que el comprador escaneará y realizará este proceso “mágicamente”.



El usuario una vez generado el QR o enviado el anuncio a la dirección indicada recibe un mensaje de advertencia que debe corroborar con su email y su password a modo de seguridad.

*\*\*Nótese que durante todo el documento se hace mención a Bitcoin (con mayúsculas) y a bitcoin (con minúsculas).*

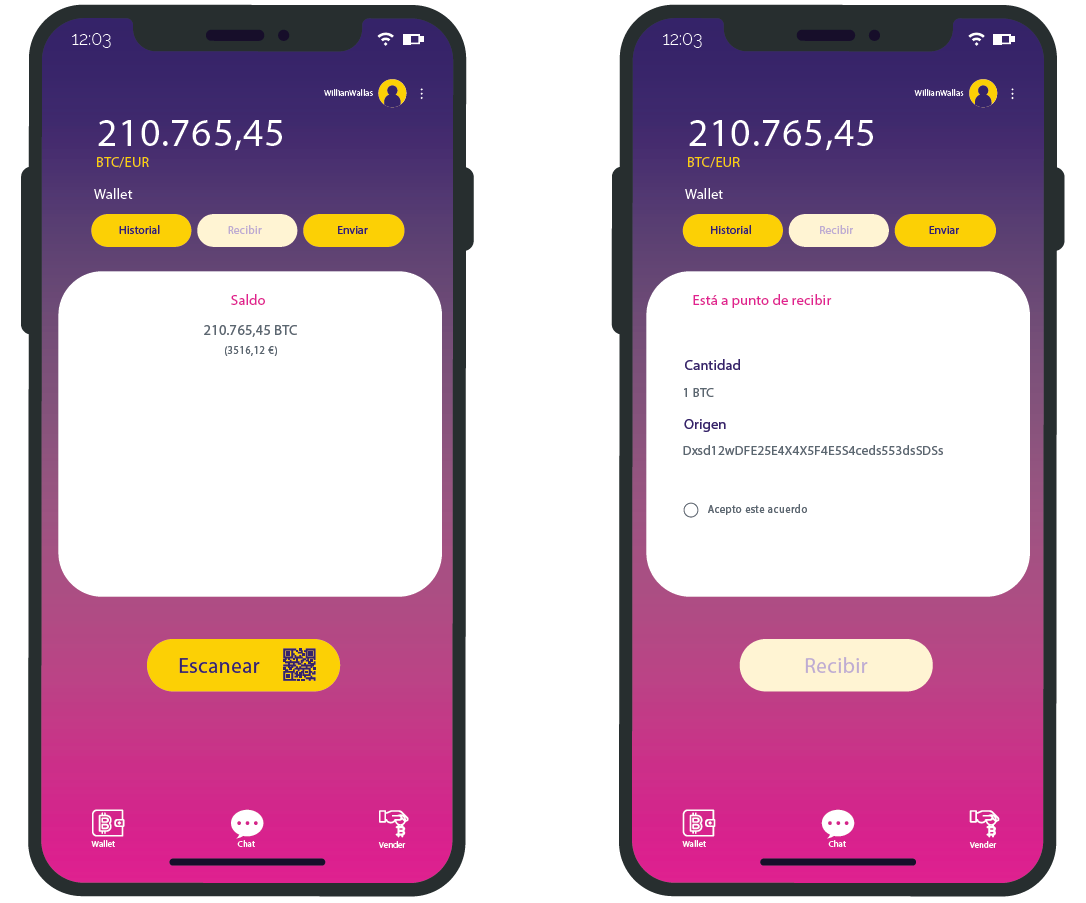
*Bitcoin se utiliza para referirse a la red y al protocolo, ambos en mayúsculas.*

*bitcoin es la criptomoneda, a veces abreviada con su ticker BTC.*

El apartado de RECEIVE está estructurado en 2 partes: escanear el QR y un mensaje de confirmación del saldo del vendedor, exactamente igual que en el paso del vendedor (SEND).

Cuando el usuario le pinche al botón de escanear se abrirá la cámara de su dispositivo y escaneará el QR del vendedor.

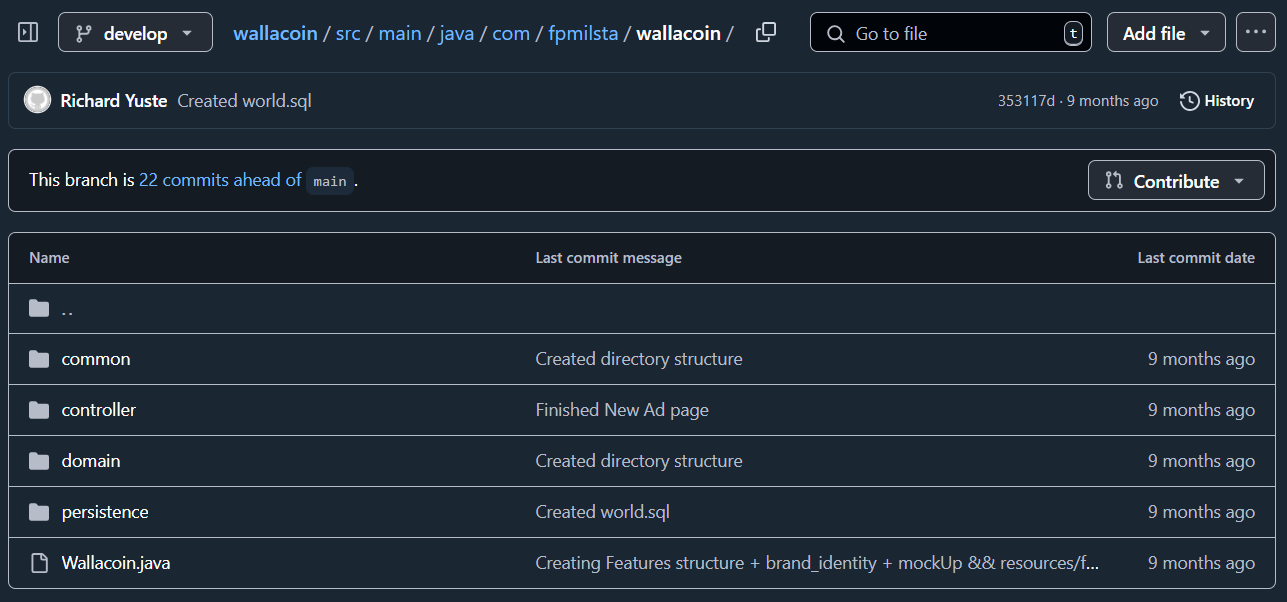
Acto seguido le llegará un mensaje de confirmación con todos los detalles de anuncio, donde tendrá que introducir sus credenciales (email y password) para verificar que todo está ok.

Al vendedor también le aparece este mensaje y también tiene que verificar con sus credenciales.

Por último, nos quedaría el **panel del usuario** y el **logout**, que por ahora no está habilitado ni diseñado pues ha sufrido muchos cambios desde el inicio de la aplicación.

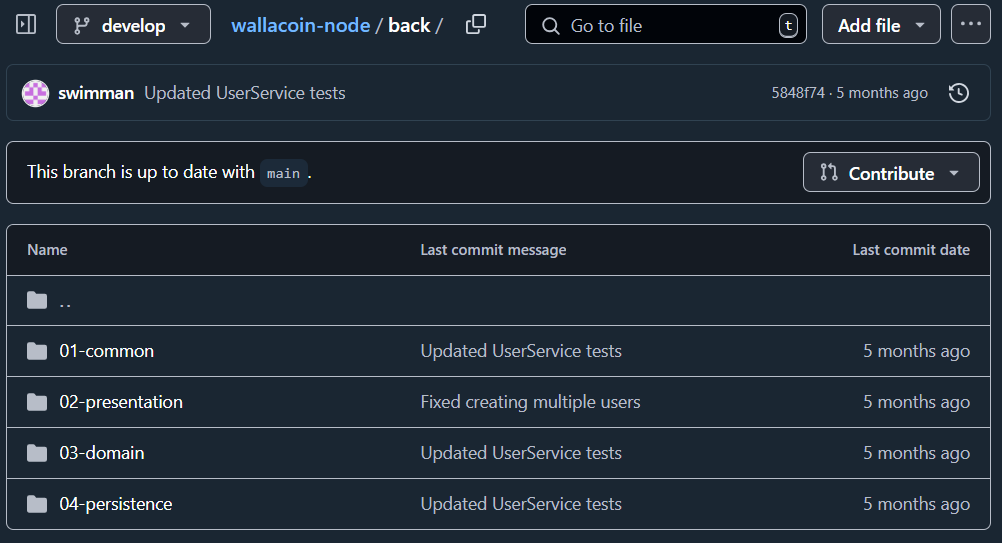
* 1. **[Implementación](#LifeCycle)**

**BackEnd:**

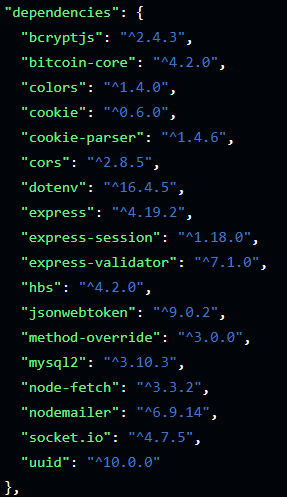
En un primer momento se desarrolló el backend en JAVA con Spring y con un sistema de plantillas llamado Tymeleaf, pero me encontré con el problema de que la mayoría de las librerías externas que utilizaba la aplicación referente a Bitcoin estaban hechas en JavaScript y solían ser APIs que devolvían JSON.

Este hecho conllevaba mucho más trabajo al tener que realizar mappers para cada una de las entidades que tenía en JAVA, así que busqué una solución que me facilitase dicha tarea.

Podemos decir que, en una segunda instancia, y digo segunda porque todavía hay una tercera, decidí desarrollar la app en NODE, un entorno para backend que utiliza JavaScript como lenguaje.

Para poder visualizar la aplicación en el frontend se utilizó un sistema de plantillas llamado Handlebars. A fin de cuentas, la idea era aprovechar todo lo que tenía de Java y que no tuviese que comenzar desde cero.

Como tercera y última decisión se concluyó que la mejor forma de realizar este proyecto era separar totalmente responsabilidades y dividir la aplicación en 2 subproyectos: backend y frontend.

De esta manera el back quedaría muchísimo más ligero y tan solo devolvería los datos que el front le solicitase, así que se pudo prescindir totalmente del sistema de plantillas utilizado.

El proyecto tal y como se encuentra en su última fase es una API que devuelve JSON y utiliza las siguientes tecnologías de la imagen de la izquierda, que se explican con mayor detalle en su [sección](#Herramientas) correspondiente.

Cabe destacar algunos puntos en esta sección como el uso de toda la criptografía y matemática de curvas elípticas que utiliza Bitcoin para realizar la creación de sus claves privadas.

Eso quizás ha sido lo que más tiempo me ha llevado aprender, pues llevo más de 5 años con este tema y todavía continúo aprendiendo cosas nuevas al respecto.

Durante el desarrollo de la aplicación he comprobado que existen librerías como BCRYPT que te facilitan mucho la tarea, pero existen funciones concretas que utiliza Wallacoin que no existían y he tenido que crear de cero.

Agradezco fielmente a la persona que ha creado CHATGPT, pues sin esta herramienta hubiera sido totalmente imposible llevar a cabo este proyecto.

Comentar que también he tenido que aprender como funciona Bitcoin como protocolo de red, pues además de la criptografía este también es un mundo inmenso con cantidad de información de gran importancia.

Esta idea de Wallacoin surgió en 2017 al ir a adquirir mis primeros BTC a través de un portal llamado <https://localbitcoins.com/es/>.

Me encontré con el problema de que la red se encontraba totalmente saturada y los BTC que había comprado tardaron 7h aproximadamente en llegar. Como estas transacciones eran P2P (entre particulares) y en efectivo se convierte en un escenario altamente tedioso cuando ambos participantes se quieren marchar, pero la transacción no se lleva a término por problemas en la propia red de Bitcoin. El vendedor ya ha enviado los BTC, pero el comprador no quiere abonar la cantidad en euros hasta que la transacción se finalice, pues corre el riesgo de pagarle y de que el vendedor cancele la transacción, ya que esta no ha concluido.

Wallacoin solventa este gran problema a través de la criptografía, pues la transacción se realiza antes de que los usuarios queden físicamente para intercambiarlos.

Para que se pueda crear el anuncio y el resto de usuarios pueda visualizarlo es requisito indispensable que el vendedor envíe los BTC a la dirección que genera la aplicación.

Una vez ha llegado el saldo los usuarios ya pueden interactuar a través del chat y quedar físicamente para realizar la transacción.

¿Cómo se realiza?

El vendedor entra en su **wallet** y en el apartado de **enviar** pulsa el único botón que hay y este genera un código QR, que es la clave privada (PK) que desbloquea el saldo en la blockchain de Bitcoin.

El comprador escanea dicho QR y esa PK se importa a su monedero dentro de Wallacoin.

Para los más entendidos en esta materia surgirá una duda sumamente importante en este punto concreto:

Si el vendedor tiene la PK y el comprador ahora también la tiene, ¿qué impide que ambos puedan sacar el dinero de esa dirección?

Este es el gran secreto de la aplicación y lo que Wallacoin ha conseguido resolver gracias a la matemática de las curvas elípticas.

Para entender el proceso imaginemos una caja cerrada con un candado y una única llave que abre dicho candado.

Esa llave es especial, pues se puede partir exactamente por la mitad. Hacen falta las 2 mitades para recomponer la llave y sin esa llave no se podrá abrir la caja.



**El funcionamiento de Wallacoin es el siguiente:**

1. El vendedor mete (envía) sus BTC dentro de la caja y genera media llave.
2. Wallacoin crea la otra media llave para que coincida exactamente con la cerradura del candado.
3. Nadie puede abrir el candado hasta este momento, pues el vendedor solo tiene media llave y Wallacoin la otra media.
4. El comprador adquiere (compra) esos BTC. Esto significa que tanto vendedor como Wallacoin le tienen que dar sus medias llaves.
5. Esto es lo que sucede “mágicamente” cuando escanea el QR

**¿Qué ocurre si el vendedor ya no quiere vender sus BTC y desea cancelar dicho anuncio?**

Que Wallacoin le envía la media llave que le falta para que pueda abrir el candado.

Quizás otro de los inconvenientes o problemas que ha surgido con el desarrollo de la aplicación, aunque en menor medida que los anteriormente comentados, ha sido el preservar variables entre métodos, pues era totalmente necesario almacenar valores de alguna forma entre algunos métodos que se llaman progresivamente.

Por poner un ejemplo:

Durante el proceso de checkin se tiene que crear un usuario que discurre en varias pantallas. En la primera se solicita email y password, en la segunda verificar el número de 4 cifras recibido en el correo y en la tercera introducir el nick.

Todos estos datos más la creación de la dirección de bitcoin para el usuario nuevo concurren en este proceso y no se quiere crear un usuario hasta que el proceso completo hay concluido.

¿Cómo se ha conseguido solventar?

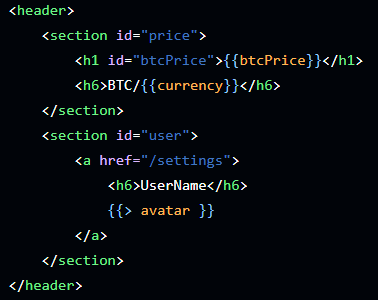
A través de cookies (almacenando valores no sensibles) y a través de variables de sesión (para los datos más sensibles).

En Express existe librerías como Cookies o Session que ayudan muchísimo con esta tarea.

[**FrontEnd**](#Implementación)**:**

Tal y como se ha comentado anteriormente se comenzó utilizando Tymeleaf como sistema de plantillas de Java y que se gestionaba desde el backend, se pasó a utilizar HandleBars cuando se migró el proyecto a Node, que es otro sistema de plantillas gestionado desde el back por el mismo Node y por último se decidió dejar el front en vanilla JavaScript.

Aquí podemos ver un ejemplo de cómo quedaban los partials en Handlebars:



*\*\* Los partials son fragmentos de código que se pueden reutilizar durante toda la aplicación.*

¿Entonces el front se encuentra escrito en vanilla JS?

Después de aprender Angular este último año y quedar sumamente enamorado de ese framework se ha decidido migrar nuevamente todo el front a Angular 19, que sinceramente es una maravilla.

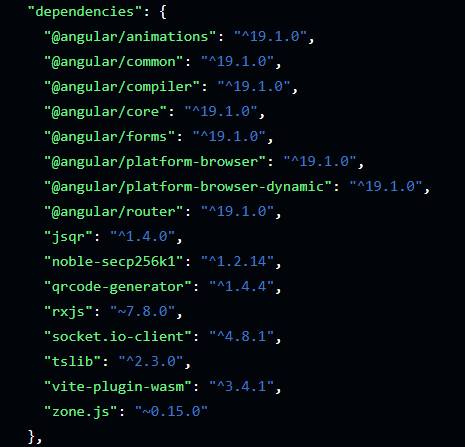
Así se encuentran en este instante ambos repositorios en GitHub:



Como se puede observar se han modificado en repetidas ocasiones las principales tecnologías (frameworks) con las que se está desarrollando la aplicación.

La finalidad no es otra que conseguir una aplicación robusta, limpia, organizada y con un código que sea fácil de leer y mantener, ya que la futura escalabilidad de esta dependerá en gran medida de estos factores.

En esta sección destacaremos las principales librerías utilizadas:



Empezaremos con jsQR y con QRcode-generator que son las utilizadas para leer y generar los QR respectivamente.

Dos tecnologías muy útiles, pero altamente complejas de utilizar pues, aunque parezca a priori que hacen bien poco, llevan muchísimos procesos internos para poder generar o leer los pertinentes códigos.

Para poder leer el QR necesitamos previamente que el HTML tenga un elemento **<video>** y otro **<canvas>**.

<video> para capturar el stream de la cámara:

* Es el único modo nativo de mostrar el flujo de la cámara en tiempo real en un navegador (usando getUserMedia).
* No se puede leer directamente los píxeles del video (sus frames son accesibles solo como un stream binario opaco).

<canvas> para extraer y analizar los frames:

* Actúa como un "lienzo" donde se copia temporalmente un trame estático del <video>.
* Una vez en el canvas, jsQR puede acceder a los píxeles via **getImageData()** y aplicar su algoritmo de decodificación.

El componente deberá tener 4 métodos obligatoriamente:

* **startCamera():** Inicia la cámara y llama a scanQRCode().
* **scanQRCode():** Prepara el canvas y ejecuta scan() por primera vez.
* **scan():** Analiza el frame actual.
  + Si no hay QR, se programa a sí misma para volver a ejecutarse (requestAnimationFrame).
  + Si detecta un QR, llama a stopScanning().
* **stopScanning():** Detiene la cámara.
  1. [**Testing**](#LifeCycle)
  2. [**Despliegue**](#LifeCycle)

1. [**Manual de usuario**](#Indice)
2. ****[**Herramientas y tecnologías empleadas**](#Indice)
   1. **Miro**
      1. **Logo:**
      2. **Web Site:** <https://miro.com/>
      3. **Descripción:**

Es una plataforma digital colaborativa que funciona como pizarra virtual, diseñada para crear diagramas, organizar ideas y trabajar en equipo en tiempo real.

Permite elaborar esquemas como diagramas UML, flujos de trabajo, wireframes y mapas conceptuales mediante herramientas de arrastrar y soltar.

Incluye plantillas predefinidas y soporta integraciones con otras herramientas como Slack, Jira y Google Drive.

Se usa en desarrollo de software, diseño de interfaces, planificación ágil (SCRUM/Kanban) y brainstorming, facilitando la visualización de proyectos de forma interactiva.

Es accesible desde navegadores sin necesidad de instalación.



* 1. **Whimsical**
     1. **Logo:**
     2. **Web Site:** <https://whimsical.com/>
     3. **Descripción:**

Es una herramienta colaborativa en línea para crear diagramas, esquemas y mapas visuales de forma rápida y sencilla.

Permite diseñar wireframes, diagramas de flujo, mapas mentales, tableros ágiles (como Kanban) y diagramas de sistemas usando elementos intuitivos de arrastrar y soltar. Ofrece plantillas predefinidas y permite trabajar en equipo en tiempo real con opciones para comentarios y edición compartida. Su interfaz minimalista y su enfoque en la simplicidad lo hacen útil para desarrollo de software, diseño de productos, planificación de proyectos y presentación de ideas**.**

* 1. ****[**Adobe Photoshop**](#herramientas)
     1. **Logo:**
     2. **Web Site:** <https://www.adobe.com/>
     3. **Descripción:**

Es un software profesional de edición y creación de imágenes basado en mapas de bits.

Se utiliza principalmente para retoque fotográfico, composición digital, diseño web (maquetación de interfaces) y creación de gráficos rasterizados.

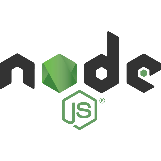
Ofrece herramientas avanzadas como capas, máscaras, filtros y ajustes de color, siendo estándar en fotografía, diseño publicitario y producción gráfica.

****

* 1. [**Adobe Illustrator**](#herramientas)
     1. **Logo:**
     2. **Web Site:** <https://www.adobe.com/>
     3. **Descripción:**

Es un programa de diseño vectorial profesional para crear ilustraciones, logotipos, iconos, tipografías y gráficos escalables sin pérdida de calidad.

Basado en curvas y vectores, incluye herramientas como plumas, pinceles y formas personalizables, siendo esencial para diseño de identidad visual, packaging, ilustración digital y material impreso.

* 1. [**Node**](#Herramientas)
     1. **Logo:**
     2. **Web Site:** <https://nodejs.org/es>
     3. **Descripción:**

Es un entorno de ejecución JavaScript basado en el motor V8 de Google Chrome, que permite ejecutar código JavaScript fuera del navegador, principalmente en servidores.

Node.js está orientado a la creación de aplicaciones web escalables y en tiempo real, utilizando programación asíncrona y orientada a eventos.

Es ampliamente usado en la construcción de APIs REST,

aplicaciones en tiempo real (chats, juegos), servicios backend y herramientas de automatización.

Incluye un ecosistema amplio (npm), que proporciona multitud de librerías y módulos reutilizables para desarrollo rápido y eficiente.

* 1. ****[**Express**](#Herramientas)
     1. **Logo:**
     2. **Web Site:** <https://expressjs.com/>
     3. **Descripción:**

Es un framework web ligero y flexible para Node.js, diseñado para desarrollar rápidamente aplicaciones web y APIs REST de forma sencilla y eficiente.

Express proporciona herramientas para manejar rutas (routing), solicitudes HTTP y middleware, permitiendo un control total sobre la estructura y comportamiento de las aplicaciones.

Es ampliamente usado en aplicaciones backend debido a su simplicidad, escalabilidad y facilidad para integrar bases de datos y servicios externos.

Es parte fundamental del ecosistema MEAN y MERN, facilitando la creación ágil de servicios web y aplicaciones basadas en JavaScript.

* 1. [**Nodemailer**](#Herramientas)
     1. **Logo:**
     2. **Web Site:** <https://nodemailer.com/>
     3. **Descripción:**

Es una biblioteca de Node.js que permite el envío sencillo de correos electrónicos desde aplicaciones o servicios web.

Proporciona soporte para múltiples protocolos y servicios de correo (SMTP, Gmail, Outlook, etc.) e incluye funcionalidades como envío de emails con archivos adjuntos, plantillas HTML personalizadas, cifrado seguro y manejo de errores detallado.

Se utiliza principalmente en aplicaciones backend para confirmaciones, notificaciones y gestión de comunicación automatizada vía email.

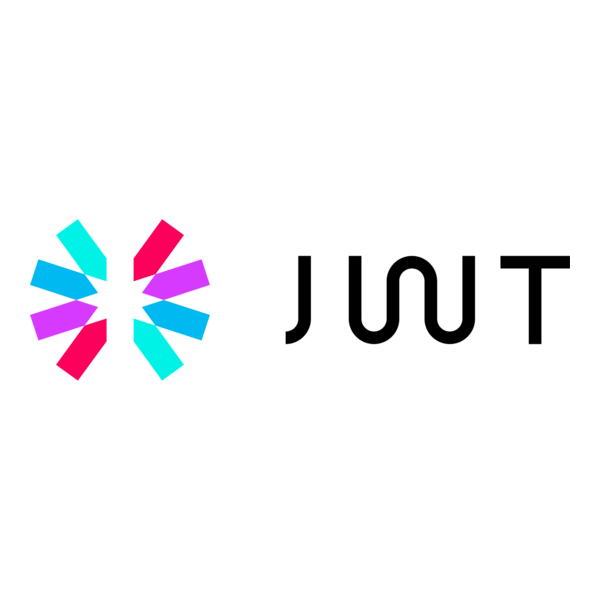


* 1. [**Bitcoin Core**](#Herramientas)
     1. **Logo:**
     2. **Web Site:** <https://bitcoincore.org/>
     3. **Descripción:**

Es el cliente original y principal implementación del protocolo Bitcoin, funcionando como software de código abierto que permite operar un nodo completo en la red Bitcoin.

Bitcoin Core facilita enviar y recibir bitcoins, verificar transacciones y mantener una copia completa de la cadena de bloques (blockchain).

Se utiliza principalmente para mantener la descentralización y seguridad de la red, ofreciendo funcionalidades avanzadas como la validación completa de transacciones, gestión segura de carteras (wallets), y soporte directo para nodos y mineros.

* 1. ****[**JWT (JSON Web Token)**](#Herramientas)
     1. **Logo:**
     2. **Web Site:** <https://jwt.io/>
     3. **Descripción:**

Es un estándar abierto (RFC 7519) que permite representar información de forma segura como un objeto JSON, facilitando la transmisión y autenticación segura entre dos partes.

JWT se utiliza principalmente en sistemas web y APIs para gestionar la autenticación y autorización de usuarios, asegurando la integridad de la información mediante firmas digitales.

Por su simplicidad, eficacia y compatibilidad, es ampliamente utilizado en aplicaciones web modernas, especialmente en escenarios donde se requiere autenticación basada en tokens, como sistemas RESTful y aplicaciones móviles.

* 1. ****[**Socket.IO**](#Herramientas)
     1. **Logo:**
     2. **Web Site:** <https://socket.io/>
     3. **Descripción:**

Es una biblioteca JavaScript que permite comunicación bidireccional en tiempo real entre cliente y servidor mediante websockets.

Socket.IO es conocido por su robustez, facilidad de implementación y soporte automático de múltiples protocolos (WebSocket, polling) para garantizar compatibilidad.

Es ampliamente utilizado en aplicaciones que requieren comunicación instantánea, como chats en vivo, notificaciones push, juegos multijugador y dashboards interactivos en tiempo real.

* 1. ****[**Angular**](#Herramientas)**:**
     1. **Logo:**
     2. **Web Site:** <https://angular.io/>
     3. **Descripción:**

Es un framework de desarrollo frontend creado por Google, basado en TypeScript, utilizado para construir aplicaciones web modernas, dinámicas y de alto rendimiento.

Angular facilita el desarrollo mediante componentes reutilizables, gestión eficiente de estados, enrutado (routing), formularios avanzados y comunicación HTTP con servicios backend.

Ofrece herramientas potentes como Angular CLI, que permite generar, compilar y probar fácilmente las aplicaciones, además de contar con un amplio ecosistema de bibliotecas y extensiones.

Es especialmente útil en aplicaciones empresariales, SPAs (Single Page Applications), dashboards interactivos y aplicaciones web progresivas (PWA).

* 1. **Asfasdz**
     1. **Logo:**
     2. **Web Site:** <https://socket.io/>
     3. **Descripción:**
     4. Es una biblioteca JavaScript que permite comunicación bidireccional en tiempo real entre cliente y servidor mediante
  2. **Fasdfa**
  3. **Afasdf**
  4. **Asfasdfa**
  5. **Fasdfa**
  6. **Fasfa**
  7. **Sf**
  8. **Asf**
  9. **Sa**
  10. **Fas**
  11. **F**
  12. **As**
  13. **F**
  14. **As**
  15. **f**

1. [**Propuestas de mejora**](#Indice)
2. [**Valoración personal**](#Indice)
3. [**Puntos a destacar del proyecto**](#Indice)