

FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

**SISTEMA INFORMÁTICO PARA EL MANEJO DE
BILLETERAS DE CRIPTOMONEDAS EN
DISPOSITIVOS MÓVILES PARA LA EMPRESA
SYNERGY VISION**

TRABAJO INSTRUMENTAL DE GRADO
Presentado ante la
UNIVERSIDAD CATÓLICA ANDRÉS BELLO
Como parte de los requisitos para optar al título de
INGENIERO EN INFORMÁTICA

REALIZADO POR
TUTOR EMPRESARIAL
TUTOR ACADÉMICO
FECHA

Br. Veronica Machado
Ing. Alexander Ramírez
Dr. Wilmer Pereira
03 de Julio de 2018

ÍNDICE DE CONTENIDOS

| | |
|--|----|
| ÍNDICE DE CONTENIDOS..... | 1 |
| ÍNDICE DE FIGURAS..... | 4 |
| ÍNDICE DE ANEXOS..... | 5 |
| SINOPSIS | 6 |
| CAPÍTULO I. PLANTEAMIENTO DEL PROBLEMA | 7 |
| 1.1. Necesidades de la Empresa | 7 |
| 1.2. Solución propuesta..... | 9 |
| 1.3. Objetivos | 11 |
| 1.3.1. Objetivo General | 11 |
| 1.3.2. Objetivos Específicos | 11 |
| 1.3.3. Aporte Funcional | 11 |
| 1.3.4. Aporte Tecnológico | 11 |
| 1.4. Alcance..... | 12 |
| 1.5. Limitaciones..... | 17 |
| 1.6. Justificación | 18 |
| CAPÍTULO II. MARCO REFERENCIAL | 19 |
| 2.1. Criptomonedas | 19 |
| 2.1.1. Fundamentos | 19 |
| 2.1.2. Funciones Criptográficas..... | 20 |
| 2.1.3. Criptografía Asimétrica..... | 21 |
| 2.1.4. Blockchain..... | 22 |
| 2.1.5. Billeteras | 23 |
| 2.1.6. Billeteras Determinísticas | 24 |
| 2.2. Herramientas de Desarrollo | 25 |
| 2.2.1. NodeJs..... | 25 |
| 2.2.2. Angular 4..... | 25 |
| 2.2.3. Ionic 3 | 26 |
| 2.2.4. Cordova 5 | 26 |
| 2.2.5. BlockCypher..... | 26 |
| CAPÍTULO III. MARCO METODOLÓGICO..... | 27 |
| CAPÍTULO IV. DESARROLLO DEL TRABAJO | 29 |

| | | |
|------------------------------|--|----|
| 4.1. | Fase de Exploración | 29 |
| 4.1.1. | Generación de base de conocimientos sobre criptomonedas | 30 |
| 4.1.2. | Reconocimiento de los servicios de la API REST de BlockCypher | 30 |
| 4.1.3. | Reconocimiento de los servicios de Firebase..... | 32 |
| 4.2. | Fase de Diseño de la Aplicación..... | 34 |
| 4.2.1. | Determinación de requerimientos funcionales y no funcionales ... | 34 |
| 4.2.2. | Diseño de las estructuras de datos | 35 |
| 4.2.3. | Configuración de Realtime Database | 35 |
| 4.2.4. | Planificación de las tareas..... | 36 |
| 4.3. | Fase de Implementación del Aplicativo Móvil | 37 |
| 4.3.1. | Adaptación de las vistas del FrontEnd | 38 |
| 4.3.2. | Adaptación de los modelos de datos del FrontEnd..... | 39 |
| 4.3.3. | Desarrollo del Módulo de Autenticación | 40 |
| 4.3.4. | Desarrollo del Módulo de Gestión de Contactos | 43 |
| 4.4. | Fase de Implementación de las Billeteras:..... | 44 |
| 4.4.1. | Adaptación de los servicios REST del FrontEnd | 45 |
| 4.4.2. | Implementación del Módulo de Consultas al Blockchain | 45 |
| 4.4.3. | Implementación del Módulo de Billeteras | 47 |
| 4.4.4. | Implementación del Módulo de Generación de Transacciones ... | 51 |
| 4.5. | Aporte Funcional | 54 |
| 4.6. | Fase de Implementación del BackEnd y Pruebas:..... | 58 |
| 4.6.1. | Implementación del BackEnd de 2FA..... | 58 |
| 4.6.2. | Pruebas de Funcionalidad en Dispositivos Móviles | 61 |
| 4.7. | Aporte Tecnológico..... | 63 |
| CAPÍTULO V. RESULTADOS | | 65 |
| 5.1. | Desarrollar las reglas necesarias para asegurar la persistencia de datos del aplicativo utilizando Firebase como BackEnd. | 65 |
| 5.2. | Desarrollar la aplicación web para poder activar el segundo factor de autenticación en el aplicativo móvil..... | 65 |
| 5.3. | Diseñar e implementar una aplicación móvil híbrida para el manejo de billeteras de criptomonedas..... | 65 |
| 5.4. | Desarrollar el módulo de Autenticación del usuario en el aplicativo móvil. | 66 |
| 5.5. | Desarrollar el módulo de Gestión de Contactos en el aplicativo móvil. | 66 |
| 5.6. | Desarrollar el módulo de Gestión de Billeteras en el aplicativo móvil.. | 67 |

| | |
|---|----|
| 5.7. Desarrollar el módulo de Consultas al Blockchain en el aplicativo móvil. | 67 |
| 5.8. Desarrollar el módulo de Generación de Transacciones en el aplicativo móvil. | 67 |
| 5.9. Aporte Funcional | 68 |
| 5.10. Aporte Tecnológico | 68 |
| CAPÍTULO VI. CONCLUSIONES Y RECOMENDACIONES | 70 |
| 6.1. Conclusiones | 70 |
| 6.2. Recomendaciones | 71 |
| REFERENCIAS BIBLIOGRÁFICAS..... | 73 |

ÍNDICE DE FIGURAS

| | |
|---|----|
| Tabla 1: Criptomonedas soportadas por BlockCypher | 31 |
| Tabla 2: Servicios ofrecidos por la API de BlockCypher | 32 |
| Tabla 3: Herramientas de Firebase utilizadas en el sistema informático | 33 |
| Tabla 4: Funciones del Servicio AuthService de la Aplicación Móvil | 42 |
| Tabla 5: Métodos del módulo de autenticación en FirebaseProvider | 42 |
| Tabla 6: Funciones del módulo de Gestión de Contactos en FirebaseProvider | 44 |
| Tabla 7: Funciones del Módulo de Consultas al Blockchain en RestService.... | 46 |
| Tabla 8: Funciones del Módulo de Billeteras en la clase KeyService | 47 |
| Tabla 9: Funciones del Módulo de Billeteras en la clase RestService..... | 50 |
| Tabla 10: Funciones del Módulo de billeteras en la clase FirebaseProvider | 50 |
| Tabla 11: Funciones del Módulo de Generación de Transacciones en RestService | 52 |
| Tabla 12: Funciones del Módulo de Generación de Transacciones en KeyService | 53 |
| Tabla 13: Rutas del Servidor de Segundo Factor de Autenticación del Sistema | 60 |
| Tabla 14: Funciones de la clase TwoFactorService de la Aplicación Móvil | 61 |
| Tabla 15: Comparación entre GulpJs y Webpack | 69 |

ÍNDICE DE ANEXOS

| | |
|--|-----|
| Anexo 1: Funcionamiento de las Billeteras HD | 75 |
| Anexo 2: Cronograma del Proyecto | 76 |
| Anexo 3: Modelos de Objetos Almacenados en Firebase Realtime Database . | 77 |
| 3.1. Modelo de Datos User | 77 |
| 3.2. Modelo de Datos Usuario (Firebase) | 79 |
| 3.3. Modelo de Datos User (List) | 81 |
| 3.4. Modelo de Datos AddressBook | 82 |
| 3.5. Modelo de Datos Wallet (User) | 83 |
| 3.6. Modelo de Datos Wallet (List)..... | 84 |
| 3.7. Modelo de Datos Keys..... | 85 |
| 3.8. Modelo de Datos Crypto | 86 |
| 3.9. Modelo de Datos Activities..... | 87 |
| 3.10. Modelo de Datos Token | 88 |
| Anexo 4: Ejemplo de la estructura de datos JSON en Realtime Database | 89 |
| Anexo 5: Descripción de las tareas de Kanban | 90 |
| Anexo 6: Tablero Kanban Semana por Semana..... | 103 |
| Anexo 7: Clases y Servicios creados durante la adaptación de las vistas del FrontEnd..... | 108 |
| Anexo 8: Manejo de Errores dentro de la aplicación móvil..... | 110 |
| Anexo 9: Estructuras de Datos de la aplicación móvil: Modelos | 111 |
| Anexo 10: Estructuras de Datos de la aplicación móvil: Interfaces | 113 |
| Anexo 11: Generación de la información criptográfica de una Billetera HD ... | 116 |
| Anexo 12: Proceso de creación y envío de una transacción de criptomonedas. | 117 |
| Anexo 13: Diagrama de Secuencia de la Activación del Segundo Factor de Autenticación | 118 |
| Anexo 14: Pantallas de la Aplicación Móvil..... | 119 |
| Anexo 15: Reglas de Seguridad establecidas en Firebase Realtime Database | 123 |

SINOPSIS

Synergy Vision es una empresa que ofrece soluciones y asesorías relacionadas a las áreas de finanza, inversión, economía y trading. Se tiene interés en poder agregar a la oferta de servicios, nuevas herramientas para manejar tecnologías emergentes, específicamente en el ámbito relacionado con las criptomonedas. El presente Trabajo Instrumental de Grado consiste en el desarrollo de un sistema informático para dispositivos móviles, cuyo objetivo es ser un manejador de billeteras de criptomonedas. El producto final es una aplicación instanciable y altamente configurable, el cual puede ser ofrecido a los clientes de la empresa para que este se adapte a las necesidades específicas de cada proyecto.

La aplicación está compuesta por una aplicación móvil híbrida, lo que permite un mayor grado de flexibilidad y portabilidad a diferentes dispositivos móviles con menos esfuerzo que una aplicación nativa. Las herramientas seleccionadas ofrecen a su vez una arquitectura modular y flexible, basada en patrones de programación que facilitan la configuración y abstracción del producto final sin necesidad de realizar una especificación o adaptación que requiera de un trabajo exhaustivo. Esto permite que el producto se pueda adaptar a necesidades futuras de los clientes de la empresa.

El desarrollo de la aplicación fue llevado utilizando técnicas de metodologías ágiles, específicamente a través del método de Kanban. Usando principios ágiles, se llevó un desarrollo adaptativo y flexible a los requerimientos del proyecto, además de permitir una gestión del sistema informático iterativa e incremental. A su vez, el desarrollo se separó en cinco etapas, con el fin de poder determinar las actividades necesarias para el funcionamiento de la aplicación.

CAPÍTULO I. PLANTEAMIENTO DEL PROBLEMA

1.1. Necesidades de la Empresa

Las aplicaciones basadas en “Blockchain” o cadenas de bloques, son un mercado emergente y novedoso, de especial interés en Venezuela y con gran nivel de actividad en el mundo. Son sistemas distribuidos y públicos, con transparencia absoluta sobre las actividades que se realizan en ellos. Todos los nodos del sistema son libres de participar en el mantenimiento y validez del aplicativo, recibiendo como incentivo un producto derivado conocido como Tokens o Monedas. Estos, son activos digitales que presentan una gran variedad de usos y funcionalidades dependiendo de la implementación y características del sistema específico.

El caso más conocido es el Bitcoin, que funciona como una moneda digital y usa conceptos criptográficos. No obstante, existen otras alternativas que aportan diferentes funcionalidades, tales como Ethereum, que implementa contratos inteligentes, o Ripple, que actúa con transacciones “peer to peer” [1].

Por otro lado, las plataformas basadas en cadenas de bloques no solo poseen un atractivo tecnológico importante, también presentan numerosas ventajas al facilitar el manejo de activos digitales. Si bien estas poseen cierto nivel de volatilidad, permiten tener ahorros no sujetos a devaluación, inflación u otras situaciones irregulares causados por la economía de un país. El hecho de que las mismas no dependan de una entidad central, organización o gobierno, sino de un sistema público, abierto y distribuido, genera un alto nivel de interés para individuos y empresas.

Al mismo tiempo, una de las problemáticas presentes en la adaptación de sistemas basados en Blockchain es la base de la tecnología, la cual funciona utilizando criptografía. Esta área puede llegar a ser confusa o difícil de

comprender para usuarios inexpertos, en especial en el área referente a criptografía asimétrica. El sistema, pseudo-anónimo y altamente técnico, complica entender por completo el funcionamiento de estas aplicaciones distribuidas. Por esta razón, se idea el uso de monederos virtuales, también conocidos como billeteras, cuyo objetivo es facilitar al usuario manejar adecuadamente el uso de llaves públicas y privadas y la realización de transacciones sin necesidad de descargarse la plataforma o aplicación distribuida de la moneda a usar, o de entender los conceptos bases de criptografía para poder participar en el sistema.

En este orden de ideas, se han desarrollado variadas aplicaciones cuyo objetivo es facilitar el manejo de criptomonedas, sin embargo, en Venezuela no existe suficiente desarrollo en el área, puesto que la mayoría de los sistemas actuales depende del uso de billeteras o plataformas extranjeras, las cuales suelen presentar problemas al momento de ser utilizadas. Para comprar monedas, por ejemplo, los servicios más populares y seguros usualmente requieren del uso de herramientas de redes tales como VPNs y Proxies, o no aceptan Bolívars Fuertes (BsF). Un ejemplo de esto son Coinbase, BitStamp y Kraken, que no brindan soporte a usuarios o clientes en Venezuela, y en casos de aplicaciones venezolanas, como MonkeyCoin, no se poseen aplicaciones móviles.

Las criptomonedas son entonces un mercado emergente, con alto nivel de potencial de explotación a nivel comercial e industrial, pero que, debido a su base tecnológica de alto nivel técnico, puede generar incertidumbre en terceros al momento de evaluar la realización de una inversión en criptomonedas. Es por ello que Synergy Vision desea tomar la oportunidad presente y captar parte de este mercado informático.

1.2. Solución propuesta

Synergy Vision desea ofrecer un nuevo aplicativo a sus clientes, cuyo objetivo no es el de ser un producto final para el mercado, sino más bien, un sistema funcional genérico, el cual deba poder ser instanciado y configurado de manera fácil y rápida, ajustándose a requerimientos futuros, funcionales y no funcionales, específicos a los clientes de la empresa. En otras palabras, es un producto que, a través de la ejecución de tareas específicas, herramientas diversas y archivos de configuración, debe generar un producto personalizado con alto nivel de calidad y versatilidad. Este aplicativo móvil, a desarrollar para la empresa, presentará todas las características y funcionalidades de un producto final, pero no tendrá elementos distintivos de una marca o producto.

Tomando en consideración estos requerimientos, el sistema presenta una serie de módulos estandarizados, los cuales están diseñados, implementados e integrados para poder adaptar, agregar o eliminar componentes de manera dinámica, sin que afecten considerablemente el tiempo de desarrollo. La propuesta es la siguiente:

Aplicación Móvil de Billetera: La misma realiza llamadas a servicios REST API para ofrecer las diferentes funcionalidades básicas de una billetera de criptomonedas. A su vez se encuentra dividida en una serie de módulos, presentados a continuación:

1. **Gestión de Billeteras:** Permite al cliente crear y manejar diferentes billeteras, una por cada criptomoneda disponible.
2. **Gestión de Contactos:** Permite al usuario realizar operaciones CRUD relacionadas con otros usuarios del aplicativo.

3. **Generación de Transacciones:** Se encarga de la generación de direcciones de billeteras para el recibo de criptomonedas, además de la creación de la estructura de la transacción, y firmado de la misma para envío de dinero a otros usuarios de la aplicación o terceros.
4. **Módulo de Autenticación:** se encargará de la persistencia del estado de autenticación del usuario en el aplicativo, además de recuperación de la contraseña, verificación del correo electrónico e inicio de sesión y registro.
5. **Consulta al Blockchain:** Permite consultar la información pública presente en el Blockchain, tal como registro de transacciones y último bloque propagado.

Aplicación Web: Su funcionalidad es la de ofrecer un servicio adicional a la billetera, a través de la activación de un segundo factor de autenticación en el aplicativo móvil.

BackEnd: Se encarga de la persistencia de los datos del aplicativo, recuperación de contraseñas y cuentas, además de proporcionar metadata acerca de la persistencia de sesión de los usuarios.

Estos componentes funcionan de manera interconectada, ofreciendo un servicio móvil de billeteras o monederos de criptomonedas.

1.3. Objetivos

1.3.1. Objetivo General

Desarrollar un sistema de manejo de billeteras para criptomonedas en dispositivos móviles para la empresa Synergy Vision.

1.3.2. Objetivos Específicos

1. Desarrollar las reglas necesarias para asegurar la persistencia de datos del aplicativo utilizando Firebase como BackEnd.
2. Desarrollar la aplicación web para poder activar el segundo factor de autenticación en el aplicativo móvil.
3. Diseñar e implementar una aplicación móvil híbrida para el manejo de billeteras de criptomonedas.
4. Desarrollar el módulo de Autenticación del usuario en el aplicativo móvil.
5. Desarrollar el módulo de Gestión de Contactos en el aplicativo móvil.
6. Desarrollar el módulo de Gestión de Billeteras en el aplicativo móvil.
7. Desarrollar el módulo de Consultas al Blockchain en el aplicativo móvil.
8. Desarrollar el módulo de Generación de Transacciones en el aplicativo móvil.

1.3.3. Aporte Funcional

9. Diseñar el proceso de gestión de riesgos de seguridad de las billeteras de criptomonedas para asegurar la integridad de los datos en el aplicativo móvil.

1.3.4. Aporte Tecnológico

10. Evaluar dos herramientas para automatización de tareas en aplicaciones

web desarrolladas en ambientes NodeJS.

1.4. Alcance

1. Desarrollar las reglas necesarias para asegurar la persistencia de datos del aplicativo utilizando Firebase como BackEnd.

La base de datos a utilizarse será Firebase Realtime Database, la cual permite tener persistencia de datos en la nube utilizando llamadas REST. Es un sistema NoSQL, utilizando datos JSON y una serie de reglas declarativas que permiten definir la estructura, índices y manejo de datos, incluyendo permisos de lectura y escritura. Posee numerosas ventajas, tales como capacidades fuera de línea en aplicativos webs, manejo de latencias y sincronización en tiempo real. Se almacena la información de las billeteras de cada cliente, además de registro de libreta de contactos y preferencias de usuario.

2. Desarrollar la aplicación web para poder activar el segundo factor de autenticación en el aplicativo móvil.

El aplicativo web es un componente básico del sistema informático completo, puesto que ofrecerá el servicio de autenticación de dos factores al aplicativo móvil. Esta aplicación web funciona como un BackEnd, donde se encargará de verificar la identidad del usuario previo al envío de dinero. Una vez que el usuario se encuentre debidamente identificado, otorgará permiso a la billetera móvil de enviar la transacción generada al Blockchain de la criptomoneda seleccionada. Este funcionamiento es opcional, siendo activada por el usuario a través de la opción de autenticación de dos factores.

3. Diseñar e implementar una aplicación móvil híbrida para el manejo de billeteras de criptomonedas.

La empresa posee un aplicativo de vistas previo, el cual cuenta con parte de las vistas y de las hojas de estilo. Este se encuentra construido sobre el ambiente Angular 4 [2] + Ionic 3 [3] y Cordova [4], resultando en una aplicación web híbrida adaptada a teléfonos inteligentes Android o IOS. Se deberá modificar este aplicativo, el cual es un esqueleto funcional visual, para manejar datos dinámicos, recibidos por un BackEnd, en vez de datos estáticos; agregar nuevos componentes y generar manejo de errores apropiados. El sistema también debe ser adaptado para que permita mostrar y modificar múltiples billeteras.

Se modificarán las estructuras de datos antiguos para que acepten los objetos o archivos enviados a través de llamadas REST que se van a consumir, además de adaptar las vistas y servicios a los nuevos requerimientos, siempre considerando factores de usabilidad propios de los dispositivos móviles.

4. Desarrollar el módulo de Autenticación del usuario en el aplicativo móvil

Este módulo se utiliza como soporte a el aplicativo de autenticación del BackEnd en la nube de Firebase [5], el cual permite integrar diferentes proveedores de autenticación, manejo de usuarios, recuperación de cuenta y contraseña y establecimiento de permisos. Este servicio, conocido como Firebase Auth, permite acceder y administrar al usuario de manera segura a través de llamadas REST utilizando la API de Firebase. Adicional a esto, se establece la opción de utilizar Autenticación de dos factores al momento de realizar pagos, lo que permite agregar un factor adicional de seguridad al aplicativo.

5. Desarrollar el módulo de Gestión de Contactos en el aplicativo móvil

Este módulo permite realizar operaciones CRUD (consultar, registrar, actualizar y eliminar) en los contactos del aplicativo del usuario registrado. Estos

contactos deben de poseer una cuenta en la aplicación, factor que se determinará a través del uso del correo electrónico al momento de iniciar sesión o registrarse. Este módulo también se encarga de la generación automática de direcciones de recibo de criptomonedas en el caso de que se realicen transacciones o pagos entre dos usuarios registrados, evitando tener que generar y enviar direcciones de manera manual.

6. Desarrollar el módulo de Gestión de Billeteras en el aplicativo móvil

Se diseñará e implementará este módulo, que incluye la creación de monederos para diferentes criptomonedas. El aplicativo utilizará la API REST de BlockCypher, el cual incluye apoyo a las siguientes monedas: Ethereum, Bitcoin, Litecoin, Dash y Dogecoin, además de dos versiones de Testnet de Bitcoin y una de Ethereum. Se podrá crear una o más billeteras por moneda y realizar operaciones, como generación de direcciones y configuración de exportación y respaldo de billeteras. Se utilizarán billeteras HD (Hierarchical Deterministic) según los estándares BIP 32 [6] y BIP39 [7] en todas las implementaciones de cada criptomoneda, con excepción de Ethereum y su respectivo Testnet.

7. Desarrollar el módulo de Consulta del Blockchain en el aplicativo móvil

El Módulo de consulta del Blockchain permitirá obtener datos específicos sobre información disponible en la cadena de bloques de varias criptomonedas. Este, al ser un sistema distribuido, permite que cualquier nodo pueda consultar datos sobre los bloques confirmados, transacciones realizadas y balance de cualquier dirección válida en la red. A través de llamadas REST a la API de BlockCypher, el usuario podrá observar estos datos. Adicionalmente, permitirá consultar billeteras y direcciones de cada cliente, requerimiento necesario para el adecuado funcionamiento del recibo de dinero y generar los balances de cada monedero.

8. Desarrollar el módulo de Generación de Transacciones en el aplicativo móvil

El mempool es un elemento funcional de todas las criptomonedas elegidas, donde las transacciones enviadas por los nodos del sistema esperan a ser incorporadas a un bloque por los mineros. De modo que, todas las transacciones que se vayan a realizar deben pasar primero por este componente. El módulo de generación de transacciones se encargará de enviar información, tal como transacciones, al mempool del Blockchain de la moneda de la billetera seleccionada. Además, permitirá configurar diversos parámetros tales como tipo de transacción, monto y prioridad, y seleccionará automáticamente las direcciones apropiadas para cada pago, manejando las firmas y hashes de las transacciones. Para ello, hará uso de la API de Transacciones de BlockCypher.

Aporte Funcional

9. Diseñar el proceso de gestión de riesgos de seguridad de las billeteras de criptomonedas para asegurar la integridad de los datos en el aplicativo móvil.

En este proyecto se desean evaluar los factores que puedan generar riesgos de seguridad en la aplicación, puesto que se manejará información sensible y activos digitales de terceros. Es importante asegurar la integridad y confidencialidad de los datos que puedan ser vulnerable, como las llaves privadas generadas por criptografía asimétrica en las criptomonedas. Se consideran los elementos con mayor factor de riesgo de la aplicación, para que las precauciones necesarias sean tomadas al momento de instanciar el producto.

Uno de los análisis se realizará a la herramienta Firebase, la plataforma BackEnd REST de Google, puesto que se utilizará Realtime Database para el almacenamiento y persistencia de los datos del usuario y el componente de

Autenticación de la misma, para manejar los estados de sesión y recuperación de la cuenta en la aplicación sin necesidad de almacenar contraseñas. Debido a que se debe transmitir datos a través de internet, y que en la base de datos se almacenarán llaves privadas, se analizará si la selección de Firebase es adecuada para el tipo de proyecto, y si se puede utilizar en las futuras instancias del aplicativo finalizado.

Aporte Tecnológico

10. Evaluar herramientas para automatización de tareas en aplicaciones web desarrolladas en ambientes NodeJS.

La aplicación desarrollada debe ser instanciable, por lo que se busca automatizar lo más posible la generación de los archivos de configuración, cambios de estilo y realización de tareas de construcción de las instancias. Para ello, se realiza una comparación entre dos herramientas para aplicativos construidos para la plataforma NodeJS: GulpJS y Webpack. Ambas herramientas permiten la generación de tareas que son ejecutadas a través de línea comando, con el fin de poder modificar componentes del programa de manera automática.

Este tipo de herramientas poseen un alto nivel de utilidad para las empresas que ofrecen suites de servicios o productos, puesto que agilizan el proceso de desarrollo y configuración de las aplicaciones. Se realiza un análisis en ambas herramientas para determinar cuál es más apropiada para configurar el producto final de manera fácil, sin necesidad de codificación extensiva.

La comparación consiste en implementar las tareas asociadas a la configuración de las pantallas de Inicio, Menú Lateral, Registro e Inicio de Sesión. Una vez ambas se puedan ejecutar de manera exitosa, se analiza el tiempo de ejecución utilizando la consola de Ionic y se comparan las vistas generadas para analizar el resultado final. Se toma en cuenta también la dificultad de configuración desde el punto de vista del desarrollador.

1.5. Limitaciones

1. El aplicativo se probará en un mínimo de dos dispositivos Android, por lo cual no se asegura la portabilidad completa del mismo en todas las versiones de sistemas operativos o teléfonos inteligentes.
2. El aplicativo web se implementará en NodeJS v8.9.1.
3. La aplicación móvil se implementará en NodeJS v8.9.1, Angular 4, Ionic 3 y Cordova 5.
4. Las criptomonedas aceptadas por el aplicativo están limitadas a aquellas que se soportadas por el servicio REST de BlockCypher, el cual soporta Bitcoin, Dogecoin, Dash, Ethereum y Litecoin, además de soporte a tres Testnets.
5. Las pruebas de que las transacciones se realicen de manera adecuada solo serán realizadas en los Testnets de BlockCypher (Testnet nativo de Bitcoin, Testnet de BlockCypher de Bitcoin y Testnet de Ethereum).
6. La estabilidad de los servicios Ethereum depende de los servicios de la API de Ethereum de BlockCypher, la cual se encuentra en estado BETA.
7. Ethereum y su respectivo Testnet no serán implementados como billeteras, sino más bien como dirección criptográfica normal debido a la ausencia de una API de Billeteras en BlockCypher para estas criptomonedas.
8. El número máximo de solicitudes HTTP al servicio REST está limitado al del plan gratuito de BlockCypher, con hasta 5 solicitudes por segundo y un máximo de 600 solicitudes por hora [8].
9. La aplicación requiere de acceso a internet.
10. El código del sistema será exclusivo de la empresa.
11. El BackEnd de la aplicación se desarrollará utilizando Firebase, específicamente Firebase Realtime Database y Firebase Auth.

1.6. Justificación

Existe en la actualidad un alto interés en la temática de las criptomonedas y el Blockchain, demostrada en el nivel de adaptación entre el público que han obtenidos algunas aplicaciones distribuidas tales como Bitcoin, Ethereum y Dash. A nivel financiero y económico, la adaptación de diversas criptomonedas como medios de pago ha ido en aumento en los últimos años, con monedas cuyo valor ha incrementado de manera dramática, resultando en oportunidades de inversión y trading. Adicionalmente, las características innatas al Blockchain tales como descentralización e inmutabilidad de registro, ofrecen una plataforma novedosa y transparente a numerosas industrias, servicios y empresas para poder llevar diferentes aspectos asociados a la realización de tareas y actividades diarias, manejo de activos y control de productos.

Considerando entonces las oportunidades relacionadas a esta área, es importante que las empresas de tecnología posean productos competitivos y actualizados a las tecnologías emergentes. Desarrollar una aplicación de este estilo, permite a la empresa Synergy Vision mostrar un portafolio que demuestre el manejo de esta área computacional emergente. Además, el hecho de que el sistema final sea instanciable sin que se requiera de una codificación extensiva facilita ofrecer un producto atractivo, modular y moderno.

CAPÍTULO II. MARCO REFERENCIAL

2.1. Criptomonedas

2.1.1. Fundamentos

Las criptomonedas es un término utilizado para describir proyectos informáticos caracterizados por la implementación de activos digitales en redes “peer-to-peer” no centralizadas como medio de intercambio económico [9]. Construidos bajo premisas criptográficas, son sistemas inmutables con alto nivel de seguridad, los cuales muestran la información de las transacciones realizadas en un sistema de bloques de registros públicos, conocido comúnmente como cadena de bloques o Blockchain.

La primera criptomoneda, el Bitcoin, fue creada en el 2009 por un desarrollador o grupo de desarrolladores conocido solamente por el alias de Satoshi Nakamoto. A partir de este punto, el crecimiento y desarrollo del mercado de las criptomonedas ha sido errático y veloz, con alrededor de 1,900 criptomonedas en existencia para junio de 2018 [10]. Es un tema con mucho grado de interés a nivel global, involucrando diferentes áreas tales como derecho, económica, relaciones internacionales y ciencias de la computación y con la capacidad actual de adaptar la arquitectura base a procesos de la industria internacional.

Debido a que el valor de las monedas depende del libre mercado y la oferta y la demanda, poseen alto grado de inversión por parte de individuos y empresas. Esto ha ocasionado que no solamente se haya desarrollado un mercado en base a la creación de nuevas criptomonedas, sino también en sistemas y aplicaciones que faciliten la gestión de los activos digitales y la realización de transacciones.

Al no existir un ente centralizado actualizando el registro de transacciones realizadas, esta responsabilidad recae en los nodos del sistema descentralizado,

conocidos como mineros. Estos, a través del proceso de minería, mantienen el contenido de la cadena de bloques consistente y completa, verificando y recolectando constantemente los movimientos nuevos que estén ocurriendo en el sistema. Cada criptomoneda establece un método específico para aceptar los nuevos bloques al sistema, denominado como método de consenso.

Ethereum, es un caso utilizado en este proyecto que difiere un poco de las demás criptomonedas. Es una plataforma que permite realizar contratos inteligentes, el cual es una implementación de acuerdos entre dos partes aplicada a Blockchain. Posee una moneda, conocida como “Ether”, que se comporta de manera semejante a otras criptomonedas, puesto que posee un valor monetario, pero es realmente una recompensa al sistema de Ethereum y no la base del mismo. No se utiliza el sistema de contratos inteligentes en este sistema, por lo que al hablar de Ethereum, nos referimos a la red por la cual se pueden realizar transacciones de Ether.

La mayoría de las criptomonedas poseen un número finito de monedas, y las mismas determinan la cantidad que existe en circulación. Lo más común en la implementación de estos sistemas distribuidos es que se generen nuevas monedas cada cierto tiempo, específicamente durante la creación de nuevos bloques durante la minería.

2.1.2. Funciones Criptográficas

Los hashes, son funciones criptográficas basadas en operaciones matemáticas que permiten, a partir de una entrada digital cualquiera, generar una cadena de caracteres que identifica los datos originales de una manera inequívoca. Esta operación asegura la integridad de los datos: la probabilidad de que dos entradas diferentes que generen el mismo hash es casi inexistente y por lo tanto despreciable, y permite verificar que no hay discrepancia entre los datos enviados y los recibidos. Son utilizados por la arquitectura de las criptomonedas

en algunos métodos de consenso, para identificar a los usuarios y realizar transacciones.

Uno de los algoritmos utilizado por las monedas implementadas en esta aplicación es el de SHA-256, el cual es una serie de funciones criptográficas generadas por la Agencia de Seguridad Nacional de los Estados Unidos, o NSA [11] para generar hashes. Otro algoritmo comúnmente utilizado es ECDSA, el cual es un algoritmo utilizado comúnmente para generar elementos que permiten verificar a los miembros involucrados en una operación propagada en una red (autenticación) y a la vez, asegurar la integridad de los datos enviados.

2.1.3. Criptografía Asimétrica

Para el recibo y envío de criptomonedas, es necesario el uso de conceptos básicos de criptografía asimétrica, y de un elemento específico a las criptomonedas llamada dirección. La criptografía asimétrica se basa en el uso de un par de llaves, una privada, que verifica la integridad de la información enviada y una pública, que se envía a través de la red de la criptomoneda para la autenticación de los recipientes de la transacción.

Todas las transacciones realizadas en una red de una criptomoneda deben ser válidas. Además de ser registros que poseen información sobre la operación que se desea realizar, el que desea emitir la transacción debe incluir la dirección de criptomonedas, obtenida de aplicar SHA-256 a la llave pública y firmar los datos utilizando su llave privada. Esto permite a otros nodos validar la información, al poder autenticar correctamente al individuo, y asegurar la integridad de los datos de la transacción [12]. Una vez que una transacción es validada, esta será incluida en un siguiente bloque del Blockchain, efectivamente realizándose. Si la misma no cumple con los criterios de identidad o integridad, esta es descartada.

2.1.4. Blockchain

La cadena de bloques, o Blockchain, es parte fundamental de la arquitectura de una criptomoneda. Funciona como un libro mayor en sistemas descentralizados y está compuesto por uno o más registros inmutables denominados bloques, los cuales se encuentran enlazados o encadenados entre ellos. Estos almacenan la información de las transacciones entre nodos del sistema, por lo cual es factible validar y verificar la localización actual de una criptomoneda en cualquier punto de su historia. Estos sistemas son completamente públicos y cada vez que se agrega un bloque, mediante el método de consenso de la criptomoneda, este es propagado por la red, de manera de que cada nodo posea la misma información.

Existen diversos algoritmos establecidos por los cuales las criptomonedas establecen un consenso entre los nodos del sistema para aceptar o rechazar un bloque nuevo. Todas las criptomonedas utilizadas en este sistema informático utilizan el algoritmo de “Proof of Work”. Cada nodo crea un bloque con la información transmitida en la red, la cual se encuentra en un espacio global conocido como mempool. Seguido de esto, a los datos del bloque, se les aplica SHA-256 junto con un número aleatorio, denominado “nonce”. El resultado de esta operación, debe ser un hash con un valor menor que la dificultad del sistema. Este sistema es fácil de comprobar, pero difícil de computar, puesto que cada nodo debe intentar con una alta cantidad de números “nonce” antes de encontrar un hash que cumpla con las características establecidas.

Cada bloque nuevo posee una serie de datos, incluyendo un hash, el cual es generado mediante el proceso de minería, y una referencia al hash del bloque previo. Esto posibilita poder seguir la cadena bloque tras bloque, y poder observar todas las transacciones realizadas en una red de criptomonedas determinada. Debido a que la generación de bloques debe ser controlada, cada red implementa una serie de políticas, la cual se encarga de graduar el tiempo

mínimo en el cual se crea y agrega un bloque nuevo al Blockchain. Las criptomonedas soportadas por el sistema informático desarrollado utilizan un número denominado dificultad, el cual corresponde a un valor en hexadecimal que se autoajusta cada cierto tiempo según la velocidad de generación de bloques nuevos.

Los Blockchains son registros básicamente inmutables, ya que utilizan principios de firma digital y un enlazamiento a bloques anteriores. Cualquier cambio dentro de la información de un bloque ya validado daría resultado información incongruente o alterada en el hash del mismo, ocasionado que el bloque entero sea descartado. Si bien se pueden generar divisiones en el Blockchain, con dos versiones de la cadena siendo aprobados al mismo tiempo, ya que cada nodo del sistema posee una copia del registro, y estos tienden a aceptar siempre la cadena más larga que sea propagada por la red, se utiliza un tiempo de espera o confirmación dentro del Blockchain para afirmar que un bloque ha sido aprobado. El éxito del ataque dependerá entonces de que cantidad de nodos del sistema estén controlados por el atacante.

2.1.5. Billeteras

Una billetera, o monedero de criptomonedas, es cualquier tipo de mecanismo que permita almacenar una colección de claves públicas y privadas de una criptomoneda específica [13]. Es común el uso de software especializado para el manejo de estos datos, en especial para la realización del firmado de las transacciones. Se utilizan llaves criptográficas generadas aleatoriamente agregando un mayor nivel de seguridad al momento de realizar transacciones, puesto que se debe utilizar una dirección nueva en cada transacción, conocido como pseudo-anónimo. La desventaja de este sistema es que se deben respaldar las direcciones utilizadas para tener control sobre el balance de la billetera, por lo que se debe ofrecer un sistema que permita al usuario generar una copia de la información almacenada. En general una billetera solo puede

poseer divisas en una criptomoneda, pero una aplicación puede poseer soporte a múltiples billeteras de diversas criptomonedas

Existen varios tipos de billeteras, esto depende de si el sistema posee una copia completa de la cadena de bloques, es decir, es un nodo o si envían las transacciones mediante nodos de confianza, utilizado comúnmente en billeteras móviles o dispositivos con almacenamiento limitado. Además, se pueden dividir en: “hot wallets” o billeteras online, donde la información criptográfica esta almacenada en sistema conectado a una red o “cold wallets” las cuales almacenan evitan el uso de una conexión a internet por mayor seguridad. Otra cateogria depende de si la información criptografía esta almacenada en un servidor o bajo el control de terceros, o si solamente el usuario puede acceder a ella. Este sistema implementa billeteras online con la información criptográfica almacenada en un servidor.

2.1.6. Billeteras Determinísticas o HD

Una billetera “Hierarchical Deterministic”, o HD, por sus siglas en inglés, es un tipo de billetera que utiliza una semilla o “seed” para la generación de las llaves criptográficas en vez de ser generadas de manera aleatoria. Este es una especificación propuesta por el Bitcoin Improvement Proposal 32 [6] y soportado por numerosos servicios y aplicaciones de criptomonedas. Las billeteras HD utilizan el algoritmo ECDSA de criptografía de curva elíptica para generar un par maestro de llaves, de las cuales se derivan las claves privadas y públicas a utilizarse para la realización de transacciones. En la sección de anexos se incluye una imagen sobre el proceso de derivación de las llaves de una billetera, en *Anexo 1: Funcionamiento de las Billeteras HD*.

La ventaja de este tipo de implementación es que facilita totalizar el balance de la billetera, ya que todas direcciones generadas a partir de la misma semilla sin parte de la misma billetera. No es necesario tampoco realizar respaldos

frecuentes de la información, puesto que todas las direcciones utilizadas pueden ser reconstruidas, facilitando poder importar y exportar las billeteras a otros sistemas. Suelen ir implementadas en conjunto con otros estándares, tales como el BIP 39, el cual establece la generación de la semilla de la billetera utilizando frases de recuperación o “mnemonics”, los cuales representan un grupo de 12 o más palabras en idioma común. Esta permite exportar la información de la billetera en otros servicios y debe ser respaldados por el usuario en un tercer medio, de manera de poder recuperar la llave privada en caso de falla en el sistema.

2.2. Herramientas de Desarrollo

2.2.1. NodeJs

Es un ambiente de desarrollo multiplataforma de código abierto para el desarrollo de código JavaScript. Permite a desarrolladores implementar aplicativos de cualquier tipo del lado del servidor, en vez de netamente en el navegador. Es un proyecto orientado a objetos, que facilita la creación de sistemas completos web en un solo lenguaje. Se complementa con el uso de npm, el cual es un manejador de paquetes para librerías de JavaScript, lo que permite organizar las dependencias del proyecto.

2.2.2. Angular 4

Angular es un framework de desarrollo para aplicaciones web creada por Google. Utiliza TypeScript, el cual es un lenguaje compilado a JavaScript que agrega funcionalidades adicionales, incluyendo orientación a objetos y tipeo estático. Esto permite que se puedan definir clases e interfaces de datos a ser reutilizadas por toda la aplicación, aumentando la abstracción y el reúso de código y componentes de manera global. Se utiliza la versión 4, la cual posee compatibilidad con Ionic 3.

2.2.3. Ionic 3

Orientado a facilitar el desarrollo de aplicaciones web híbridadas, Ionic es un “Software Development Kit” construido sobre Angular. Facilita la integración de servicios y herramientas para el desarrollo en dispositivos móviles mediante Cordova. Incluye una documentación extensiva y componentes visuales diseñados para la usabilidad en teléfonos inteligentes. La versión 3 es la más reciente en ser distribuida.

2.2.4. Cordova 5

Apache Cordova es un framework para el desarrollo de aplicaciones en dispositivos móviles de Adobe. Permite construir aplicaciones con soporte a código en CSS, HTML y JavaScript, caracterizándolas como híbridadas. Adicionalmente permite al sistema en desarrollo, mediante el uso de plugins, acceder a elementos nativos de los dispositivos móviles, tales como la cámara, GPS y otros componentes del hardware.

2.2.5. BlockCypher

El aplicativo utiliza la Interfaz de Programación de Aplicaciones (API, por sus siglas en inglés) de servicios web de BlockCypher. Este es un nodo servidor, conectado a diferentes cadenas de bloques, de manera de proveer de la infraestructura necesaria, para poder crear y enviar solicitudes REST sin necesidad de instalar las aplicaciones distribuidas específicas a cada moneda. Como nodo de cada sistema, posee una alta disponibilidad en las criptomonedas soportadas, las cuales son Bitcoin, Litecoin, Dogecoin, Ethereum y Dash. Además, ofrece servicios de Testnet, los cuales son redes distribuidas utilizadas por los desarrolladores como ambientes de prueba sin los riesgos asociados al manejo de criptodivisas.

CAPÍTULO III. MARCO METODOLÓGICO

Considerando la ausencia de un cliente o un producto definido, y el alto énfasis en obtener un producto con alto nivel de usabilidad, se utilizará una metodología ágil basada en Kanban [14], adaptado al tamaño del equipo, el cual es de un solo desarrollador. Este tipo de estructura de trabajo permite desarrollar un producto completo puesto que los principios de Kanban generan entregas parciales y gracias a la estructura y arquitectura de las herramientas elegidas para el desarrollo, se puede observar fácilmente si los resultados están cumpliendo las necesidades planteadas. Para este proyecto, se utilizó la aplicación de Google, Kanbanchi, la cual permite llevar un tablero o pizarrón Kanban de manera digital.

Kanban como se caracteriza por un flujo de trabajo balanceado y adaptado a las capacidades del equipo de trabajo, con un alto énfasis en la visualización de las tareas y el progreso realizado a través de un tablero. Es una metodología ideal para equipos con prioridades cambiantes y proyectos que no requieren una jerarquía de trabajo compleja o una organización elaborada. Es flexible y con alto nivel de retroalimentación, donde se busca amplificar los beneficios del equipo y desarrollar sin roles predeterminados. Acepta que los requerimientos, e inclusive el producto final, estén sujetos a cambios durante todo el ciclo de desarrollo. Se utilizan ciclos cortos de planificación, con énfasis en entregas rápidas y, gracias a su énfasis incremental e iterativo.

En este caso, la metodología de desarrollo viene definida por el planteamiento de tareas pequeñas de manera semanal. Cada tarea posee cinco fases: por comenzar, planeamiento, desarrollo, pruebas y listas. Estas tareas se definen utilizando como base el progreso en las tareas de la semana anterior, con el fin de que no se acumulen más de cinco en la fase de desarrollo y evitar así perder el orden de prioridad de los requerimientos. Se realizaron reuniones informales con el tutor empresarial de manera regular, donde se discute el avance realizado,

dificultades presentadas y cambios o soluciones propuestas, con el fin de generar observaciones o indicar cambios, si son necesarios, antes de continuar con el desarrollo del sistema. Con el fin de dividir los objetivos y construir el aplicativo de manera gradual, este proyecto se divide en cinco grandes fases, representadas en *Anexo 2: Cronograma del Proyecto*.

Exploración: Consiste en la recolección de la información y conocimientos necesarios sobre criptomonedas, sus usos, las características y funcionamiento del Blockchain, especificaciones de las monedas seleccionadas, reconocimiento del API de BlockCypher y de las herramientas de Firebase a través de herramientas REST.

Diseño de la Aplicación: Estructuración de los árboles de datos del Realtime Database de Firebase, y generación de los requerimientos funcionales y no funcionales de cada módulo y del aplicativo final. En base a esta información, se dividen los requerimientos en las tareas necesarias para generar cada una. Al finalizar, se configura el Realtime Database de Firebase.

Implementación del Aplicativo Móvil: Donde se desarrollan los módulos de autenticación, gestión de contactos y la adaptación de las vistas y modelos de datos del FrontEnd.

Implementación de las Billeteras: Incluye la creación y manejo de billeteras, implementación de los servicios REST necesarios al API de BlockCypher, la creación de transacciones, consultas al Blockchain y el manejo de operaciones criptográficas.

Implementación del BackEnd y Pruebas: Se desarrolla el sistema de Autenticación de dos factores (2AUF) y se realizan las pruebas de funcionalidad en dispositivos móviles.

CAPÍTULO IV. DESARROLLO DEL TRABAJO

Se debe realizar una aclaración referente a la terminología a utilizar durante el desarrollo del proyecto. Si bien se utiliza el término Billetera para englobar todas las criptomonedas soportadas por la API de BlockCypher, cabe destacar que existen dos excepciones. Debido a que BlockCypher agregó recientemente a Ethereum y Testnet Ethereum a su servicio, y que las mismas no poseen soporte a la API de Billeteras, estas deben ser manejadas como direcciones de criptomonedas comunes, es decir, un par de llaves criptográficas sin posibilidad de agregar más direcciones a la billetera, mientras que las demás criptomonedas si se encuentran implementadas como Billeteras HD.

Ya que ambas estructuras comparten un gran número de características a nivel de manejo interno de la aplicación, muchos de los métodos o funciones creadas durante el desarrollo de la aplicación móvil manejan ambas estructuras de datos de manera idéntica, y la diferencia recae en el manejo con el servidor de BlockCypher. Por esta razón, en el siguiente proyecto, cuando se mencione el termino Billetera, este engloba a todas las criptomonedas, mientras que si existen diferencias o especificaciones a nivel de las funciones o métodos desarrollados para el manejo de las estructuras de datos se referencia específicamente si son Billeteras HD o direcciones de Ethereum o Ethereum Testnet.

4.1. Fase de Exploración

Esta fase tuvo una duración estimada de una semana, durante la cual se realizaron tres actividades. El objetivo fue la generación de los conocimientos necesarios para comenzar el desarrollo de la aplicación móvil. Se dividieron las actividades realizadas de la siguiente manera:

4.1.1. Generación de base de conocimientos sobre criptomonedas

Para comenzar a realizar la división de tareas y estructurar los datos, se debe conocer en detalle el funcionamiento de la tecnología base del sistema informática, las criptomonedas. Para ello, se realizó una lectura profunda sobre el funcionamiento y la arquitectura del Blockchain usando “Bitcoin: Un Sistema de Efectivo Electrónico Usuario-a-Usuario” [15], de Satoshi Nakamoto, además de otras lecturas referencias en el CAPITULO II. MARCO REFERENCIAL, con el fin de entender los diferentes elementos que constituyen el Blockchain: direcciones, bloques, las billeteras y sus estándares, las diversas maneras en las que se puede obtener consenso en los nodos del sistema distribuido y finalmente, el flujo de acciones que deben ocurrir para realizar transacciones directamente en la cadena de bloques.

4.1.2. Reconocimiento de los servicios de la API REST de BlockCypher

BlockCypher fue la API escogida para realizar la conexión con las diversas cadenas de bloques. Esta posee un gran número de funcionalidades, por lo que fue necesario analizar cuales servicios o funciones provee, las características del servicio, además de la estructura y formato de las solicitudes y respuestas esperadas. Para especificar la compatibilidad del servicio, se presenta una tabla con las plataformas aceptadas por la API Web de BlockCypher, incluyendo el nombre, si es una cadena de bloques principal o de prueba y el URL al recurso.

| Blockchain | Cadena | Recurso (URL) |
|------------|----------|----------------------------------|
| Bitcoin | Main | api.blockcypher.com/v1/btc/main |
| 1Bitcoin | Testnet3 | api.blockcypher.com/v1/btc/test3 |
| Dogecoin | Main | api.blockcypher.com/v1/doge/main |

| Blockchain | Cadena | Recurso (URL) |
|------------------|--------|----------------------------------|
| Dash | Main | api.blockcypher.com/v1/dash/main |
| Litecoin | Main | api.blockcypher.com/v1/ltc/main |
| BlockCypher | Test | api.blockcypher.com/v1/bcy/test |
| Ethereum | Main | api.blockcypher.com/v1/eth/main |
| Ethereum Testnet | Test | api.blockcypher.com/v1/beth/test |

Tabla 1: Criptomonedas soportadas por BlockCypher

En la siguiente tabla, se detallan las funcionalidades ofrecidas por BlockCypher y cuáles se utilizaron en el desarrollo del aplicativo. Se debe destacar que todos los servicios son compatibles con las siguientes criptomonedas: Bitcoin, Bitcoin Testnet 3, Dogecoin, Litecoin, Dash y el Testnet de BlockCypher. En el caso de Ethereum y su Testnet, no poseen soporte a todas las funcionalidades o recursos, debido a que BlockCypher agregó soporte a esta criptomoneda recientemente, por lo que, para la fecha actual, se encuentra en estado BETA. En la tabla también se especifican que servicios poseen soporte a Ethereum y Ethereum Testnet.

| Nombre de la API | Características | Se utilizó | Soporte a Ethereum |
|----------------------|---|------------|--------------------|
| API de Blockchain | Permite realizar consultas generales a diferentes cadenas de bloques y sus respectivos bloques. | Si | Si |
| API de Direcciones | Permite consultar información correspondiente a direcciones y generar direcciones. | Si | Si |
| API de Billeteras | Permite generar, modificar, consultar y eliminar billeteras de criptomonedas. Posee soporte a billeteras HD y billeteras normales | Si | No |
| API de Transacciones | Permite consultar transacciones realizadas, además de ofrecer una alternativa conveniente para generar y | Si | Si |

| | | | |
|-----------------------------|--|----|----|
| | enviar transacciones creadas desde el lado del cliente. | | |
| API de Metadata | Permite introducir datos a bloques, direcciones o transacciones específicas, de manera pública o privada. | Si | No |
| API de Micro transacciones | Permite enviar micro transacciones, las cuales son transacciones de montos y tarifas de mineros bajas. Se encuentra obsoleta en la red de Bitcoin debido al alto congestionamiento de la misma. | No | No |
| Factor de Confianza | Servicio adicional que permite medir la posibilidad de que exista un doble gasto en una transacción no confirmada | No | No |
| API de Analítica | Permite correr consultas personalizadas con el fin de extraer datos y generar análisis sobre el comportamiento de una cadena de bloques. En estado Beta, por lo cual es inestable. | No | No |
| API de Activos | Permite crear y manejar activos digitales en cadenas de bloques públicas. | No | No |
| Redireccionamiento de Pagos | Ofrece un servicio para generar direcciones de un solo uso, las cuales luego transfieren sus fondos a una dirección o billetera específica. Pensada para vendedores que acepten pagos en criptomonedas. Servicio exclusivo de/ planes BlockCypher pagos. | No | No |
| Eventos y Hooks | Servicio API que permite recibir notificaciones a eventos especificados por el usuario, tales como transacciones no confirmadas a direcciones específicas, creación de nuevos bloques, entre otras, a través del uso de WebSockets o WebHooks. | No | No |

Tabla 2: Servicios ofrecidos por la API de BlockCypher

4.1.3. Reconocimiento de los servicios de Firebase

Firebase es una plataforma de desarrollo perteneciente a Google, la cual ofrece una suite de herramientas para aplicaciones webs y móviles. En la actualidad, cuenta con un gran número de soluciones para desarrolladores, con

el fin de que estos se puedan integrar de manera fácil a proyectos previos o nuevos. Para el desarrollo de este proyecto, fue necesario aprender de manera adecuada el funcionamiento de la plataforma, de forma de decidir cuales herramientas se adaptaban a las necesidades del sistema informático. En la siguiente tabla, se caracterizan las herramientas seleccionadas:

| Nombre | Descripción | Beneficios |
|----------------------------|--|--|
| Firebase Auth | Permite la autenticación de usuarios a la aplicación. Posee soporte a diversos proveedores de autenticación, tales como correo y contraseña, Google, Facebook, GitHub y Twitter. | Manejo de autenticación de los usuarios de la aplicación sin necesidad de crear un BackEnd. Manejo de casos tales como recuperación de cuenta y reenvío de contraseña. Manejo de privilegios a usuarios a otros productos de Firebase. Evita a el sistema informático almacenar información sensible, tales como contraseñas. |
| Firebase Realtime Database | Servicio de base de datos NoSQL en la nube a través de llamadas a un servicio API Rest. Utiliza arboles de datos en formato JSON para estructurar la información. | Permite establecer reglas de lectura y escritura a los usuarios de la aplicación. Sincronización en tiempo real a todos los usuarios. Utilización de un SDK que permite acceso a datos aun cuando no existe conexión a internet. Elimina la necesidad de un servidor o BackEnd. |
| Firebase Cloud Storage | Permite almacenar y manejar objetos y archivos tales como imágenes y videos en la nube. | Subida y descarga de archivos de manera robusta y tolerante a errores en la red. |

Tabla 3: Herramientas de Firebase utilizadas en el sistema informático

Al utilizar Firebase como plataforma de desarrollo, se presenta también la posibilidad de agregar o adaptar otros productos que son ofrecidos por la suite de servicios en un futuro, lo cual ofrece un gran número de opciones a nivel de desarrollo o manejo de aplicaciones dependiendo de las necesidades de los clientes. Se utilizaron dos proveedores de autenticación de Firebase Auth para

llevar el control de los usuarios del sistema: correo electrónico y contraseña y cuenta Google.

4.2. Fase de Diseño de la Aplicación

En esta fase, el objetivo fue estructurar los árboles de datos del Realtime Database de Firebase, y generación de los requerimientos funcionales y no funcionales de cada módulo y del aplicativo final. En base a esta información, se generan los diagramas de los componentes del sistema y la división de las tareas necesarias para generar cada una. Al finalizar, se configura, Realtime Database de Firebase, se establecen las reglas de escritura y lectura y se crean las tareas determinadas en el Kanban.

4.2.1. Determinación de requerimientos funcionales y no funcionales

Para generar la estructura de datos a almacenar, es necesario determinar los requerimientos y las necesidades que debe satisfacer una aplicación móvil de billeteras de criptomonedas. Debido a la naturaleza delicada que poseen las criptomonedas, y en general, cualquier activo digital con valor monetario, es indispensable que la aplicación sea segura y a la vez, al ser un sistema desarrollado para dispositivos móviles, se debe realizar un enfoque a la usabilidad de la aplicación.

Tomando en consideración estas limitantes, se decidió llevar el proyecto con un énfasis en diseño orientado al usuario. En este tipo de filosofía de trabajo, se orienta la funcionalidad del producto a la usabilidad desde el punto de vista de un usuario promedio, sus capacidades y expectativas. Se analiza que acciones puede realizar y cómo lograr esas acciones de manera de utilizar una base de conocimientos previo y beneficios propios de las aplicaciones móviles. Se considera la diversidad de dispositivos que existe en el mercado, lo que genera un alto grado de heterogeneidad tanto en componentes físicos como en sistemas

operativos. Se busca realizar diseños dinámicos y apropiados a diversos tamaños de pantallas.

4.2.2. Diseño de las estructuras de datos

Una vez determinados los requerimientos y funcionalidades esperadas por un usuario promedio de una billetera de criptomonedas, se determinan qué factores son importantes almacenar para poder ofrecer un servicio completo. Para ello, se generaron los modelos de datos a ser almacenados en la base de datos, los cuales se encuentran descritos en el *Anexo 3: Modelos de Objetos Almacenados en Firebase Realtime Database*.

La estructura de datos fue pensada tomando en cuenta que Realtime Database es una base de datos no SQL, por lo que el almacenamiento y forma de los datos queda anidada utilizando estructuras de árboles. En *Anexo 4: Ejemplo de la estructura de datos JSON en Realtime Database*, se observa el archivo JSON generado por Firebase una vez la aplicación empieza a almacenar datos de manera dinámica.

4.2.3. Configuración de Realtime Database

El acceso a la información del sistema es controlado por Firebase, ya que esta se encarga de funcionar como BackEnd de la aplicación móvil. Realtime Database posee una serie de herramientas para manejar la seguridad de los datos almacenados en la misma, en especial los permisos de escritura y lectura, con el fin de evitar que un usuario no registrado pueda acceder a información para la cual no posee privilegios. La configuración de la misma es establecida en la consola de Firebase para proyectos en desarrollo. De manera semejante a la base de datos como tal, las reglas se establecen a través de un archivo JSON.

La forma en la que se acceden a los datos en Realtime Database, es diferente a una base de datos relacional. Al utilizar árboles de datos, Firebase siempre recuperará todos los nodos hijos almacenados en una dirección específica. Esto implica que se deben establecer reglas de lectura y escritura para evitar que los usuarios puedan acceder a información no relevante. Es necesario generar listas de objetos con datos repetidos solamente para realizar acciones de consulta a otros miembros de la aplicación. Por ejemplo, el árbol de datos User List, descrito en *Anexo 3.3: Modelo de Datos User (List)* permite a un usuario conocer si otra persona se encuentra registrada en la aplicación.

Existen tres tipos de reglas: escritura, lectura y validación. Las dos primeras definen cuando un usuario puede escribir o leer datos en Realtime Database, respectivamente. En el caso de la aplicación, queremos establecer que la escritura de datos solo puede suceder si el usuario se encuentra registrado y con una sesión activa. El tercer tipo de regla, asegura que los datos se ajusten a estilos y tipos, generando consistencia. Por ello, se debe configurar las reglas de validación que sean necesarias para evitar problemas con datos incorrectos. Durante esta fase, y modificada nuevamente durante el desarrollo del Aporte Funcional, descrito en el *CAPITULO IV.5 Aporte Funcional*.

4.2.4. Planificación de las tareas

El desarrollo del proyecto fue organizado a de Kanban, con el fin de poder dividir de manera adecuada los requerimientos de cada módulo en tareas de bajo nivel de complejidad. Se creó y configuró el tablero Kanban, herramienta que permite mantener un control de las actividades que se deben realizar durante el ciclo de desarrollo del sistema informático. Para ello, se utilizó Kanbanchi, el cual es una herramienta diseñada para el set de aplicaciones Google, disponible gratuitamente a través de la tienda de Google Chrome. Esta permite crear un tablero adaptado a la metodología ágil y generar tareas y actividades, de manera de poder gestionar el proyecto de una forma visual, atractiva y simple.

Se dividieron las tareas por módulos, asignándole a cada una un color identificativo. También se establecieron las cinco fases por las cuales debe pasar cada actividad, las cuales son Por Comenzar, Planeando, Desarrollo, Probando y Completadas. En los *Anexo 5: Descripción de las tareas de Kanban* y *Anexo 6: Tablero Kanban Semana por Semana* se encuentran ilustradas las tareas realizadas y el estado del tablero, semana por semana.

En la fase Por Comenzar, las tareas no se han comenzado a realizar. Una vez que pasan a Planeando, las mismas están siendo analizadas, de manera de determinar (si existen) que tareas dependen de su finalización, cuales elementos afectan su desarrollo y como se debe implementar de manera de ajustarse a los que ya se encuentra realizado. Al ser determinados estos requerimientos, la actividad se mueve a la fase de Desarrollo, donde se realiza la tarea.

Una vez que la tarea está implementada, esta se mueve a la fase de Probando, donde se realizan pruebas directamente en la aplicación para observar si la actividad funciona de manera adecuada y se corrigen errores si existen. Cuando la tarea se encuentra probada adecuadamente, esta pasa a la fase de Completadas, indicando que la misma se encuentra finalizada.

4.3. Fase de Implementación del Aplicativo Móvil

Durante esta fase, comienza el desarrollo del aplicativo móvil del sistema informático. Para poder ofrecer servicios de billetera de criptomonedas, primero es necesario adaptar el FrontEnd previo, de manera de ajustarlo a las estructuras de datos que se adapten a los objetos y clases almacenados en Firebase. También se deben implementar los módulos de gestión de contactos y de autenticación antes de poder implementar los módulos de gestión de billeteras, generación de transacciones y de consulta al Blockchain.

4.3.1. Adaptación de las vistas del FrontEnd

Uno de los objetivos de mayor alcance, fue ajustar las vistas previas del FrontEnd para que estas se adaptaran a los nuevos requerimientos. Esto incluye presentación y modificación de datos dinámicos, manejo de los diversos tipos de errores que se puedan presentar y adaptar las vistas a un diseño responsivo e interactivo para los usuarios. Debido a la complejidad de algunas estructuras de datos que se deben manejar por la aplicación, también es necesario la creación de nuevas vistas, de manera de evitar sobrecargar los componentes antiguos con un exceso de información, con el objetivo de mantener un diseño minimalista, simple, usable y amigable. Debido a que este objetivo estuvo directamente relacionado con la implementación de los módulos del aplicativo, se detallará en cada módulo los cambios a las vistas correspondientes

Durante el desarrollo del sistema, las vistas previas del FrontEnd pasaron por varias modificaciones de diseño. Originalmente se planteó alterar el FrontEnd previo, propiedad de la empresa, lo menos posible, pero una vez fue generada una base de conocimientos sobre criptomonedas, se observó y analizó que efectivamente los cambios serían más extensos de lo planificado, debido a que estas tenían que mostrar en las vistas más información u opciones de lo pensado.

Estos cambios ocasionaron que la aplicación requiriera de cambios en el estilo, incluyendo agregar nuevos componentes interactivos, pantallas de espera y mensajes de información al usuario. Además, se realizó un trabajo extensivo en el manejo de errores de manera de que estos fueran transparentes para el usuario. La aplicación utiliza un gran número de llamadas asíncronas a servicios REST API, lo que involucro la creación de estructuras nuevas en la aplicación para poder interceptar errores relacionados con los mismos.

Para realizar las llamadas a servicios externos o compartir información entre componentes, se utilizan componentes conocidos como Servicios. En la sección

de anexos, *Anexo 7: Clases y Servicios creados durante la adaptación de las vistas del FrontEnd* se detallan los mismos.

Adicionalmente, la aplicación utilizó la librería de npm, ngx-translate, para traducir el contenido en español e inglés. Esta dependencia utiliza archivos JSON, específicamente llave-valor para poder traducir mensajes o textos en Angular. La misma también se utiliza para el manejo de errores dentro de la aplicación, al llegar el código a los componentes visuales, el servicio busca el mensaje a mostrar en los archivos JSON de la aplicación. En *Anexo 8: Manejo de Errores dentro de la aplicación móvil*, se muestra esta estructura.

4.3.2. Adaptación de los modelos de datos del FrontEnd

El FrontEnd previo posee estructuras de datos bastante simples, debido a que solo muestra datos estáticos. Por ello se debían adaptar y crear los nuevos objetos según lo establecido en los requerimientos del proyecto. La aplicación móvil cuenta con dos tipos de estructura de datos: Interfaces y Clases. La diferencia entre ambos, es que una interfaz es una clase abstracta, por lo que no posee funciones o métodos inherentes. Se utilizan para asegurar que los mensajes de envío o de respuesta a servicios externos posean una estructura determinada. Las clases son modelos de datos utilizados de la misma manera que en otros lenguajes orientados a objetos y fuertemente tipados, poseen un constructor y funciones específicas.

Para facilitar el mantenimiento y adaptación de la aplicación, se crearon dos directorios dentro de la raíz del proyecto. En una, llamada Models, se almacenaron todas las Clases que utiliza el aplicativo. Estas se encuentran definidas en *Anexo 9: Estructuras de Datos de la aplicación móvil: Modelos*. En ella observamos constructores a los objetos inherentes a la aplicación, siendo ellas las mismas estructuras de datos que se encuentran especificadas en la

Fase de Diseño de la Aplicación, específicamente en *Anexo 3: Modelos de Objetos Almacenados en Firebase Realtime Database*.

Cabe destacar que algunos de los modelos de datos creados, poseen un atributo llamado key. Esta referencia es el id del objeto en Firebase. Al momento de construir los objetos la generación de los id únicos es realizado automáticamente por Firebase al insertar o almacenar por primera vez. Debido a como se encuentra estructurada la información en las bases de datos NoSQL, es necesario conocer esta llave o índice para poder modificar un nodo hijo del árbol desde la aplicación móvil.

El segundo directorio creado para facilitar la el manejo de las respuestas de los servicios REST y APIs se llama Interfaces. Las clases creadas se encuentran detalladas en *Anexo 10: Estructuras de Datos de la aplicación móvil: Interfaces*.

Se crea un servicio general a la aplicación, llamado SharedService, cuyo objetivo principal es redireccionar a otros servicios y mantener la información en un estado consistente entre componentes. Manipula los datos previos a enviar a otros servicios, almacena el perfil del usuario y permite recuperar datos específicos según las necesidades de la aplicación. Durante el desarrollo de la aplicación, se trata de redirigir a este servicio para evitar que los componentes manipulen datos de manera directa.

También se creó un servicio, llamado FirebaseProvider, el cual tiene como único objetivo la realización de las llamadas REST a Firebase, con el fin de agregar, modificar, consultar y eliminar objetos almacenados en la base de datos.

4.3.3. Desarrollo del Módulo de Autenticación

Paralelamente se desarrolló del Módulo de Gestión de Contactos junto con el Módulo de Autenticación. Este último, tiene como objetivo el manejo de los

estados de sesión de los usuarios de la aplicación, lo que incluye registro de nuevos usuarios a través de dos proveedores distintos: correo electrónico y contraseña y Google, inicio de sesión, envío de tokens a el servidor web que maneja el segundo factor de autenticación, recuperación de contraseña, verificación de correo electrónico y recuperación de metadata asociada al usuario almacenada por Firebase Auth.

Para simplificar el manejo de las funciones y atributos específicos de Firebase Auth, se creó un Servicio en la aplicación móvil solamente para la realización de las llamadas a la API REST de Firebase y mantener consistente el estado de sesión del usuario dentro de la aplicación. Esta, se mantiene activa hasta que se cierre la aplicación, se finalice sesión, o la misma expire según lo establecido por Firebase Auth. Los detalles de esta nueva clase se encuentran detallados a continuación:

| Método | Descripción |
|-------------|---|
| signup | Registra a un nuevo usuario dentro de la aplicación móvil utilizando el proveedor de correo electrónico y contraseña de Firebase Auth. Una vez que recibe la respuesta de la API, almacena la información del nuevo usuario dentro de Realtime Database. Devuelve un objeto User en caso exitoso, o un código de error en caso de falla. |
| login | Permite a un usuario ya registrado iniciar sesión dentro de la aplicación móvil utilizando el proveedor de correo electrónico y contraseña de Firebase Auth. Si el usuario logra iniciar sesión, cambia el estado de navegación a la pantalla de Inicio o de Crear Billetera, si el usuario no posee una billetera asociada. En caso de error, devuelve un código de error. |
| loginGoogle | Provee manejo de sesiones (registro e inicio de sesión) de usuarios que decidan utilizar el proveedor de Google para entrar a la aplicación. Si el usuario logra iniciar sesión, cambia el estado de navegación a la pantalla de Inicio o de Crear Billetera, si el usuario no posee una billetera asociada., o un código de error en caso de falla. |
| logout | Cierra la sesión del usuario activo. Genera un cambio de navegación a la pantalla de Inicio de Sesión. |

| | |
|-----------------------|--|
| getLoggedInUser | Devuelve la información almacenada por Firebase Auth del usuario que posee una sesión activa dentro de la aplicación móvil en ese momento. |
| sendVerificationEmail | Envía un correo electrónico al usuario para que este verifique que la cuenta efectivamente es propiedad de él. |
| recoverPassword | Envía un correo electrónico al usuario para que este pueda recuperar su contraseña, en caso de haber iniciado sesión a través del proveedor correo electrónico y contraseña. |
| setProfilePicture | Cambia la foto de perfil del usuario que posee una sesión activa dentro de la aplicación móvil en ese instante dentro del objeto almacenado por Firebase Auth. |
| getIdToken | Devuelve el Token del usuario activo dentro de la aplicación móvil. Esto es utilizado por otro servicio para los métodos de segundo factor de autenticación. |

Tabla 4: Funciones del Servicio AuthService de la Aplicación Móvil
Se implementaron también diversos métodos en la clase FirebaseProvider:

| Función | Descripción |
|-------------------|--|
| addUser | Método que inserta un usuario en el directorio /user/ de Realtime Database. Incluye la información por defecto del Objeto User, como imagen de perfil, moneda local y el objeto Token. |
| getProfilePicture | Devuelve el URL donde se encuentra la imagen de perfil del usuario. |
| getToken | Devuelve el objeto Token del usuario con la sesión activa en la aplicación móvil. |
| getCurrency | Devuelve la moneda local del usuario con la sesión activa en la aplicación móvil. |
| updateCurrency | Actualiza el valor de la moneda local del usuario con la sesión activa en la aplicación móvil. |

Tabla 5: Métodos del módulo de autenticación en FirebaseProvider

Al momento de ser almacenados los usuarios, se genera una ubicación en Firebase Storage con el fin de poder almacenar una imagen de perfil personalizada, en caso de que el usuario lo desee. Esto permite compartir y recuperar las imágenes entre los usuarios de la aplicación.

Se adaptaron algunas de las vistas del FrontEnd previo, específicamente home, account, y el app o menú lateral, para mostrar los datos del usuario una vez este ha iniciado sesión. Se agregaron nuevos componentes visuales a login

y register, con el fin de mostrar adecuadamente las validaciones de los campos de los formularios y mostrar mensajes de error. Se crearon dos componentes nuevos, uno dedicado a mostrar un mensaje de confirmación de registro, con instrucciones para verificar el correo electrónico, nombrada confirm-email y una página con las preferencias de usuario, llamada account-security.

Por último, se realizaron las llamadas pertinentes en las capas lógicas de cada componente para que accedieran a la información del usuario a través del servicio SharedService y se actualizaron los componentes visuales a través de la modificación de los archivos CSS de la aplicación.

4.3.4. Desarrollo del Módulo de Gestión de Contactos

El producto principal de este módulo es la Libreta de Contactos, la cual permite a un usuario registrado guardar a otros usuarios de la aplicación utilizando solamente el correo electrónico asociado. Este servicio permite realizar transacciones entre miembros del sistema informático sin necesidad de tener que escanear o introducir manualmente una dirección de recibo, ya que esta automáticamente selecciona la billetera del recipiente correspondiente a la criptomoneda con la cual se desea enviar el pago. Se implementaron los siguientes métodos en la clase FirebaseProvider:

| Función | Descripción |
|------------------------------|--|
| getAddressBook | Recupera la Libreta de Contactos de un usuario registrado. Se encuentra asociado al id único (\$uid) del mismo. |
| addAddressToAddressBook | Inserta un contacto en la Libreta de Contactos utilizando el método push, el cual asigna un id único dentro de la lista al nuevo contacto. |
| removeAddressFromAddressBook | Elimina un contacto en la Libreta de Contactos utilizando el método remove. Es necesario conocer el id único del contacto. |
| editAddressFromAddressBook | Modifica la información de un contacto dentro de la Libreta de Contactos utilizando el método update. |

| | |
|---------------------------|---|
| getUserByEmail | Busca dentro del directorio /user/ de Realtime Database si existe un usuario registrado bajo un correo electrónico específico. Si este es el caso, devuelve el uid e imagen del mismo. |
| getAddressFromAddressBook | Recupera la información de un contacto utilizando el correo electrónico de la Libreta de Contactos. Se utiliza para evitar registros duplicados, si el método devuelve null, el contacto nuevo no se encuentra registrado en la Libreta de Contactos. |
| getWalletByEmail | Recupera el nombre de una billetera, en caso de billeteras HD, o la dirección, en caso de que la criptomoneda sea Ethereum; de un usuario utilizando solamente el correo electrónico. Para ello busca dentro del directorio /user/ de Realtime Database usando el uid perteneciente al contacto almacenado dentro de la Libreta de Contactos. |

Tabla 6: Funciones del módulo de Gestión de Contactos en FirebaseProvider

Adicionalmente se agregaron los métodos correspondientes para poder acceder a la información de la Libreta de Contactos a través de la clase SharedService. También se realizaron cambios a los componentes correspondientes a este módulo, los cuales se encuentran localizados en el directorio /src/pages, cada una bajo una carpeta personalizada. Estos fueron address, address-book y edit-address, lo que incluyó la modificación de los archivos TypeScript, HTML y CSS de cada uno para poder recuperar y mostrar adecuadamente la información referente al módulo de Gestión de Contactos.

4.4. Fase de Implementación de las Billeteras:

Durante esta fase, las actividades de desarrollo estaban orientadas a todas aquellas tareas que estuvieran relacionadas de manera directa e indirecta con el manejo de las billeteras, incluyendo la creación y manejo de billeteras, implementación de los servicios REST necesarios al API de BlockCypher, la creación de transacciones, consultas al Blockchain y el manejo de operaciones criptográficas.

4.4.1. Adaptación de los servicios REST del FrontEnd

El FrontEnd previo no se encontraba adaptado para realizar solicitudes a servicios REST de una manera estándar. Las llamadas se realizaban directamente desde los componentes de las vistas. Para mejorar y adaptar esta solución, se creó un servicio, llamado RestService, con el objetivo de englobar todas las llamadas a la API de BlockCypher. Estas llamadas, al igual que las de otros servicios de la aplicación móvil, pasan por la clase `HttpErrorInterceptor` para el manejo de código de errores internamente.

La API de BlockCypher requiere de un Token o identificador, asociado a una cuenta dentro de la página, para la realización de ciertas solicitudes, en especial aquellas que requirieran transmitir datos a través de llamadas HTTPS, es decir, todos los métodos POST y DELETE, y algunos GET. Se crearon los métodos dentro del servicio que formatean los datos antes de ser enviados a través de archivos JSON a la API de BlockCypher.

Durante las siguientes semanas, se agregaron de manera constante a este servicio las funciones necesarias para el correcto funcionamiento de los módulos restantes por desarrollar, es decir, aquellos que requerían de llamadas a la API de BlockCypher para ofrecer los servicios de billeteras a la aplicación móvil. Estos cambios serán documentados en la sección correspondiente a cada módulo.

4.4.2. Implementación del Módulo de Consultas al Blockchain

El Módulo de Consultas al Blockchain, como es indicado por su nombre, se encarga de la realización de las llamadas asociadas a consultas a la API de BlockCypher. Esto incluye obtener información a partir de las cadenas de bloque de las diversas criptomonedas ofrecidas por la API y recuperar los datos sobre una transacción o dirección específica.

Existen diversos usos a este módulo, la mayoría interna a la aplicación. La consulta de transacciones permite mostrar información sobre monto transferido, direcciones de envío y recibo, tipo de pago, valor de la tarifa de mineros pagada, fecha de recibo en el Blockchain, y número de confirmaciones, información referencial para que los usuarios puedan consultar en cualquier momento los datos de una transacción en la que se encuentre involucrada una billetera o dirección propia.

Se crearon las siguientes funciones en la clase RestService con el objetivo de realizar consultas al Blockchain:

| Nombre | Descripción | Método |
|-------------------|--|--------|
| getBlockChain | Devuelve la información del Blockchain de una criptomoneda específica, en un objeto IBlockchain. | GET |
| getBlock | Devuelve la información de un bloque del Blockchain de una criptomoneda específica en un objeto IBlock. | GET |
| getBalanceAddress | Devuelve la información específica de una dirección de una criptomoneda, incluyendo balance de la misma y transacciones realizadas, en un objeto IAddress. | GET |
| getTransaction | Devuelve la información específica de una transacción mediante su Hash, incluyendo direcciones de envío y recibo y balance de la operación, en un objeto ITransaction. | GET |

Tabla 7: Funciones del Módulo de Consultas al Blockchain en RestService

Además de la generación de estos métodos, se generó la vista blockchain, la cual es un componente que se utiliza para que los usuarios de la aplicación puedan realizar consultas a las cadenas de bloques disponibles, o a bloques específicos si conocen el hash que los identifica. Finalmente, se agregaron las llamadas a las nuevas funciones a través de SharedService, de manera de manejar el acceso de los componentes a los datos mediante este servicio.

4.4.3. Implementación del Módulo de Billeteras

Este módulo se encarga del manejo de las billeteras de criptomonedas. Una vez que un usuario inicia sesión, una de las primeras acciones que se realizan dentro del flujo de actividades es recuperar las billeteras almacenadas en el árbol del usuario en Realtime Database. Esta información es un arreglo de objetos de tipo Wallet en una variable de acceso global en el servicio SharedService. Aprovechando de las ventajas de las arquitecturas de Angular, esta clase cumple con los patrones de un singleton, por lo que se asegura la consistencia de la información en toda la aplicación.

Se implementó una nueva clase, la cual se encarga de manejar todas las funciones asociadas a la generación y validación de los datos criptográficos de una billetera. Esta clase, nombrada KeyService, posee una serie de métodos específicos a la creación y manejo de las llaves públicas y privadas de las billeteras del usuario, los cuales se encuentran descritos en la siguiente tabla:

| Nombre | Descripción |
|-----------------------|---|
| createKeys | Genera de manera aleatorio las llaves de una billetera de cualquier criptomoneda. Devuelve un objeto Keys, el cual es almacenado en Firebase Realtime Database. |
| validateMnemonic | Verifica que una clave de recuperación efectivamente cumple con los requerimientos mínimos, es decir, que sean doce palabras que se encuentren en el diccionario del estándar Bip 39. |
| generateAddress | Genera una dirección para una billetera de Ethereum o Testnet Ethereum según la llave privada de la misma. |
| importWalletMnemonics | Permite importar una billetera utilizando la clave de recuperación y su contraseña (si existe). Función solo válida para importar billeteras HD en las criptomonedas soportadas. |

Tabla 8: Funciones del Módulo de Billeteras en la clase KeyService

Se utilizaron tres librerías de npm para la creación de la información criptográfica de una billetera:

- Bip39: librería que permite generar una semilla en forma de clave de recuperación. Esta está compuesta por doce palabras aleatorias en inglés, las cuales, junto con una contraseña escogida por el usuario, permiten importar la billetera HD en otras aplicaciones que soporten este formato.
- BitcoinJs: Librería que maneja la generación de llaves públicas y privadas para las criptomonedas con soporte a billeteras HD. Adicionalmente posee métodos y objetos que permiten firmar y manejar el proceso de derivación de direcciones desde una llave privada extendida. En el desarrollo de la aplicación se utiliza con frecuencia una clase de esta librería:
 - HDNode: es una clase, que a partir de una semilla en formato hexadecimal (generada por bip39, por ejemplo) o una llave privada extendida (en caso de ser una billetera preexistente) permite manejar el proceso de derivación y firmado de una billetera HD. Permite también exportar las llaves de la billetera a una cadena de caracteres en base 58.
- EthereumJS: Librería que maneja la generación de llaves públicas y privadas para Ethereum y Ethereum Testnet, incluyendo mostrar la dirección de Ethereum correspondiente en un formato Hexadecimal.

El proceso de generación de la información criptográfica de una billetera se encuentra descrita en *Anexo 11: Generación de la información criptográfica de una Billetera HD*.

Es necesario para poder acceder a la vista del home de la aplicación que el usuario con la sesión activa posea al menos una billetera registrada en Realtime Database. Si el usuario no posee ninguna, la aplicación cambia el flujo de navegación, con el objetivo de redirigir a un componente de creación de Billeteras.

Una vez que el usuario selecciona la criptomoneda de la nueva billetera, se genera la información criptográfica de la misma. Seguido, si la nueva billetera es de tipo HD, se registra en la API de BlockCypher bajo el Token de la aplicación y al recibir la respuesta positiva del servicio REST, se almacena la información en Firebase y se redirige al componente de home. En caso contrario, si es una dirección de Ethereum o Ethereum Testnet, ya que estas criptomonedas no poseen soporte de parte de la API de Billeteras, no es necesario realizar el registro a través del servicio de BlockCypher, por lo que se almacenan directamente en Realtime Database.

Se agregaron nuevos métodos a la clase RestService. Debido a las limitaciones existentes en el plan gratis de BlockCypher, no es posible tomar ventaja de funcionalidades como realizar llamadas con múltiples parámetros de una sola vez, por lo que es necesario realizar las consultas a cada billetera de manera individual.

| Nombre | Descripción | Método |
|--------------------------|---|--------|
| createWallet | Utilizando una llave publica extendida (xpub), se crea un registro de una Billetera HD bajo un nombre único en el servicio REST de la API de BlockCypher. Devuelve un objeto IHDWallet. | POST |
| getWalletBalance | Devuelve la información de una Billetera HD o dirección de Ethereum y Testnet Ethereum específica, utilizando el nombre de la misma. Incluye balance de la billetera, número de transacciones y ultimo nivel de profundidad utilizado, mediante un objeto IBalance. | POST |
| deriveAddress | Genera una dirección nueva utilizando curva elíptica en una billetera HD, con el fin de ser utilizada para generar direcciones de recibo. | POST |
| getUnusedAddressesWallet | Devuelve una dirección asociada a una billetera que no tenga transacciones realizadas. Utilizada para el envío de | GET |

| | | |
|--|--|--|
| | criptomonedas a otros usuarios de la aplicación. | |
|--|--|--|

Tabla 9: Funciones del Módulo de Billeteras en la clase RestService

Adicionalmente, se crearon los siguientes métodos en el proveedor FirebaseProvider, con el fin de generar persistencia de las billeteras, y recuperar la información necesaria para el funcionamiento del módulo:

| Nombre | Descripción |
|--------------------|---|
| addWallet | Almacena un objeto Wallet en la base de datos, bajo el identificador único del usuario cuya sesión esta activa. |
| getWallets | Devuelve la información de una todas las billeteras almacenadas bajo el identificador único del usuario cuya sesión esta activa. Esto devuelve una lista de objetos Wallet con la excepción del objeto Keys. |
| getWalletsKeys | Devuelve la información criptográfica de una billetera específica. Esta es la otra ocasión, luego de la creación de la billetera, donde las llaves públicas y privadas de una billetera se encuentran en la aplicación móvil. |
| updateWalletCrypto | Devuelve una dirección asociada a una billetera que no tenga transacciones realizadas. Utilizada para el envío de criptomonedas a otros usuarios de la aplicación. |

Tabla 10: Funciones del Módulo de billeteras en la clase FirebaseProvider

Se desarrollo una clase global llamada ExchangeService, cuya función es la de consultar la tasa de cambio de las diversas criptomonedas a la moneda local de la aplicación. Esta clase realiza una llamada HTTP a la API gratuita de Cryptocompare, recibiendo como respuesta un archivo JSON con el valor del cambio. Esta información, junto con la almacenada en el objeto Crypto de cada Billetera, nos permite mostrar una aproximación del valor del balance de la billetera o de una transacción en diferentes componentes del aplicativo.

Se crearon dos componentes nuevos:

- create-wallet, la cual genera un formulario para que el usuario seleccione las opciones de configuración de la nueva billetera.
- show-mnmemonics, que muestra al usuario las doce palabras de recuperación asociadas a una billetera.

Para la generación de direcciones de recibo dentro de la aplicación móvil, se modificó el componente receive o de recibo para que mostrara una dirección de generada por la aplicación. El flujo de acciones a tomar para la misma depende del tipo de billetera; en caso de ser una dirección de Ethereum o Ethereum Testnet, solo muestra la información almacenada dentro del atributo address o dirección de la billetera seleccionada.

En el caso de ser una billetera HD, el proceso es más complejo. Para seguir buenas prácticas de seguridad en criptomonedas, la dirección a utilizarse para el recibo de dinero no debe tener ninguna transacción previa. Por lo tanto, se realiza una llamada a la API de BlockCypher para consultar la billetera seleccionada, específicamente la última dirección generada en el último nivel de profundidad de derivación. En base a esta información, se verifica que esta no posea transacciones asociadas para poder ser mostrada en el componente de Ionic, o se genera una nueva mediante el método deriveAddress, en caso de existir una transacción previa. Una vez que la dirección de recibo llega al componente, se genera un código QR para ser escaneado por terceros.

Otras actividades realizadas durante el desarrollo de este módulo estuvieron involucradas con la modificación y adaptación de los componentes del FrontEnd para poder mostrar adecuadamente la información de las billeteras, la creación de los métodos y funciones intermediarias desde la clase global SharedService y el manejo en tiempo real de la información de las billeteras a través de llamadas asíncronas.

4.4.4. Implementación del Módulo de Generación de Transacciones

El desarrollo de este módulo involucra la realización de las actividades relacionadas con la consulta, generación y envío de transacciones en las billeteras del usuario. Es en este punto del ciclo de desarrollo donde se genera

un problema de seguridad relacionado con el manejo de las llaves públicas y privadas, expuesto en el *CAPITULO IV.5 Aporte Funcional*.

Es necesario describir el funcionamiento y objetos de las dos librerías utilizadas dentro del proceso de firmado. BitcoinJs permite, utilizando una semilla o directamente una llave privada, generar un objeto llamado HDNode, para la realización de actividades criptográficas. Se crearon las siguientes funciones en la clase RestService, descritos a continuación:

| Nombre | Descripción | Método |
|-----------------------|--|--------|
| getWalletTransactions | Devuelve la información específica de todas las transacciones asociadas a una Billetera HD o dirección de Ethereum y Testnet Ethereum específica, a través de un objeto IAddress. | GET |
| createPayment | Genera un esqueleto de la transacción (ITransactionSke) a ser enviada a través de BlockCypher a la cadena de bloques correspondiente a una criptomoneda. Esta transacción necesita ser modificada y firmada para que sea válida. | POST |
| sendPayment | Envía a BlockCypher una transacción ya firmada para que esta sea empujada a la cadena de bloques correspondiente. Devuelve un objeto ITransaction con la información de la nueva transacción | POST |

Tabla 11: Funciones del Módulo de Generación de Transacciones en RestService

Al momento de realizar una transacción, se deben recuperar las claves criptográficas de la billetera que desea realizar la operación. Una vez recuperada la información criptográfica de la billetera, esta es referida a KeyService, donde se realiza la firma de los datos relevantes de la transacción. En la siguiente tabla, se encuentran descritos los métodos creados durante el desarrollo de este módulo.

| Nombre | Descripción |
|-------------------------------|---|
| derivePrivateKeys | Método que dada la información criptográfica de una billetera y un arreglo de N IInputs (donde N representa el número de direcciones origen), genera un arreglo de objetos HDNode de la librería BitcoinJS utilizando la llave privada extendida (xpriv) y derivando N veces. El resultado será utilizado para firmar el arreglo tosign del objeto ITransactionSke. |
| signTransaction | Método que recibe una billetera y un objeto de tipo ITransactionSke, y dependiendo del tipo de la misma, redirecciona a signWithDerivedPrivateKey o signWithPrivateKey. |
| signWith DerivedPrivateKey | Función que realiza el firmado de los datos relevantes de un objeto ITransactionKey si la billetera es de tipo HD. Obtiene el resultado de derivePrivateKeys, y realiza en un ciclo las firmas dentro de la transacción, además de almacenar dentro del objeto todas las llaves públicas correspondientes a las direcciones de envío de dinero. Devuelve el objeto ITransactionSke. |
| signWithPrivateKey | Función que realiza el firmado de los datos de un objeto ITransactionKey si es una billetera Ethereum o Ethereum Testnet. Genera un objeto EthereumJS Wallet en base a la llave privada de la dirección y firma. Devuelve el objeto ITransactionSke. |

Tabla 12: Funciones del Módulo de Generación de Transacciones en KeyService

Una vez firmado los datos, se envía la transacción modificada a la API de BlockCypher. Este recibe la información, y si la operación es exitosa, devuelve un objeto de tipo ITransaction con la información de la transacción realizada. Cabe destacar, que debido a que la aplicación móvil utiliza la API de BlockCypher, esta se encarga de empujar la transacción directamente. La billetera no tiene control sobre el proceso de aprobación o verificación de la transacción, ni mucho menos la capacidad de revertir un pago una vez realizado. El proceso completo de creación de transacciones se encuentra ilustrado en *Anexo 12: Proceso de creación y envío de una transacción de criptomonedas*.

Se crearon tres componentes nuevos dentro del FrontEnd del aplicativo

- transactions; muestra la información de todas las transacciones asociadas a una billetera. Este componente utiliza la llamada getWalletsTransactions de la clase RestService para obtener la información a ser consultada.
- transaction-confirmation: componente de consulta, que muestra en detalle la información de una sola transacción. Es llamado cuando se envía una transacción con éxito a la API de BlockCypher, o se desea obtener más detalles de una de las transacciones listadas en transactions.
- send-confirm: este componente complementa la creación de una transacción, y permite escoger el monto y la tarifa o preferencia de la transacción a realizarse, además permitir que el usuario se asegure de que los datos estén correctos previo a realizar la transacción.

Adicionalmente, se modificó la vista del componente send o de envío, para poder permitir al usuario escoger un recipiente para la transacción, sea dirección o contacto registrado. Se agregó una funcionalidad adicional al componente de address-book o libreta de contactos, para poder enviar la información al componente send o de envío, del contacto que fue seleccionado.

Se agregaron los métodos y variables correspondientes a el módulo dentro de la clase SharedService para acceder a los datos de las demás clases dentro de los componentes de Ionic y se agregaron los códigos pertinentes de errores dentro de los archivos de traducción.

4.5. Aporte Funcional

Diseñar el proceso de gestión de riesgos de seguridad de las billeteras de criptomonedas para asegurar la integridad de los datos en el aplicativo móvil.

El tipo de billetera implementada durante este proyecto fue “hot wallet” con la información criptográfica alojada en un servidor. Ya que la información del usuario almacenada dentro de la estructura de Realtime Database es data no sensible, con excepción del nodo de billeteras, la cual puede ocasionar perdidas

monetarias importantes si es accedida por terceros maliciosos o sobrescrita por accidente. Al momento de recuperar una billetera, esta se almacena en una variable dentro de SharedService, pero esta información no incluye el objeto Keys por razones de seguridad.

Es necesario recuperar la información criptográfica de las billeteras cada vez que se desee realizar una transacción, ya que la firma se debe realizar desde la aplicación móvil. Esto es debido a que no existe un BackEnd tradicional al cual se le pueda relegar el manejo de las operaciones más sensible. Existe entonces una posible vulnerabilidad a ser explotada, debido a que los datos que se encuentran dentro de una aplicación móvil pueden ser extraídos por terceros maliciosos siempre y cuando la sesión del usuario se encuentre activo. Además de eso, la transmisión de la información criptográfica de una billetera a través de una red abierta como lo es el internet posee un factor de riesgo adicional.

El uso de Firebase y sus herramientas ayuda a disminuir, mas no evitar por completo, algunos de los riesgos de seguridad descritos. Firebase Auth elimina la necesidad de tener que manejar los estados de sesión y contraseñas de los usuarios de la aplicación móvil, evitando que exista un listado de contraseñas que pueda ser vulnerado por terceros. Adicionalmente, el uso obligatorio de HTTPS, junto con la consola de desarrollo de Firebase, la cual restringe el acceso a las herramientas e información del producto, permiten que las credenciales de la aplicación, si bien se encuentren visibles dentro de los archivos de configuración de la aplicación móvil, no puedan ser accedidos por todo el mundo.

El mayor riesgo radica entonces en el acceso de lectura y escritura a la Realtime Database. Como fue descrito en el *CAPITULO IV.2.3*, las operaciones de la base de datos son negadas o permitidas dependiendo de si se cumplen las reglas establecidas en la consola para poder acceder a los nodos del árbol de datos. Si un nodo padre posee tan solo un hijo para el cual no se cumple una regla, Firebase no permite la operación. Es por ello que se decidió desarrollar

estructuras de datos con información no sensible los cuales puedan ser accedidos por cualquier usuario, mientras que la información específica e individual de un usuario solo puede ser accedida por el mismo.

Todas estas decisiones disminuyen desde varios ángulos los riesgos de seguridad de la aplicación y existen diversas alternativas que se pueden plantear para el producto final. Una de ella, involucra no almacenar las llaves privadas extendidas en la base de datos, sino en los dispositivos físicos. Si el usuario desinstala la aplicación, necesitaría volver a importar su billetera mediante el uso de la clave de recuperación más contraseña, por lo que esto puede generar molestias en los usuarios finales y, por lo tanto, conllevar a un producto menos usable. Lo importante de este tipo de consideración, es que el servidor nunca tendría acceso a la información criptográfica de las billeteras.

Otra alternativa es realizar las firmas directamente desde un BackEnd seguro, de manera de que la aplicación nunca posea las llaves privadas de las billeteras, evitando realizar operaciones sensibles dentro del dispositivo móvil. Esta opción se considera más simple de implementar para proyecto, pero se siguen almacenando las llaves privadas, factor que sigue generando inseguridad, ya que son datos de carácter extremadamente sensible. Se debe asegurar la integridad y seguridad de este BackEnd, no solamente de terceros maliciosos, sino de internos. Además, el usuario debe poder acceder a sus claves privadas si lo desea, por lo que sería necesario el envío de la información criptográfica vía red.

Se considera que, al momento de crear el producto, es necesario realizar un análisis a el cliente del mismo, para determinar y establecer un código de seguridad adaptado a la naturaleza y las reglas del negocio, la arquitectura, sistemas informáticos previos y las limitaciones de equipos y personal si es aplicable. Se considera analizar si el uso de Firebase Functions es adecuado para este BackEnd, ya que integrar esta herramienta al sistema completo es sencillo considerando que Firebase ya está instanciado dentro del producto.

Firebase como suite de herramientas es bastante poderosa y, dado el ambiente adecuado, puede ser tan segura como la aplicación la amerite. En este caso particular, se considera que Firebase Auth complementado con Realtime Database son adaptables a la mayoría de los clientes, pero se debe considerar la firma digital de las transacciones y el manejo de la información criptográfica, información se considera no debe estar almacenada en Realtime Database.

La estructura de datos Keys, presentado en el *CAPITULO IV.2*, es bastante insegura, puesto que no es necesario almacenar todos los atributos del objeto.

- El dato seed es redundante, ya que se puede obtener fácilmente utilizando la librería bip39.
- El campo passphrase puede evitar ser almacenado. Este campo, se utiliza para la el cifrado de la semilla de la billetera junto con la clave de recuperación y es opcional.
- El campo xpub solo es utilizado al momento de generar las transacciones o de registrar una billetera HD en BlockCypher, pero esta es derivada de la llave privada extendida.
- El campo xpriv es generable a partir del seed o semilla hexadecimal, o de la clave de recuperación más el passphrase.
- El campo mnemonics o clave de recuperación puede no ser almacenado, siempre y cuando se le dé oportunidad al usuario de respaldar esta información.

En base a las dos alternativas propuestas anteriormente:

- Si se almacena la llave privada de la aplicación, con opción de recuperación de la billetera en caso, se considera que no es necesario el desarrollo de un BackEnd. En cambio, se propone cambiar la estructura de datos criptográficos almacenados en Realtime Database, borrando toda la información almacenada en

el objeto Keys de cada billetera con excepción de la clave pública. Este valor permite verificar que efectivamente la llave privada o llave privada extendida almacenada en el dispositivo corresponde a la billetera seleccionada. Este dato no compromete la seguridad, puesto que la llave pública solo se utiliza para autenticar. Esto involucraría agregar mayor seguridad al manejo de almacenamiento interno del dispositivo.

- Si se almacena en una base de datos la información criptográfica del usuario, además de trasladar los servicios de firmado a un BackEnd, es necesario implementar nuevas políticas de seguridad en este servidor, de manera de garantizar disponibilidad de los servicios de firmado de transacciones y evitar fuga de información sensible a terceros. En este caso, ya que es necesario utilizar algún tipo de base de datos o sistema de almacenamiento, se debe considerar el uso de Realtime Database y trasladar todas las estructuras al nuevo sistema de almacenamiento. Se pueden mantener los servicios de Firebase según los requerimientos del cliente.

4.6. Fase de Implementación del BackEnd y Pruebas:

Se realizaron las tareas finales del ciclo de desarrollo de la aplicación. Para ello, es necesario integrar completamente la aplicación móvil para que use las ventajas de Cordova, agregando funcionalidades nativas de dispositivos móviles. También se desarrolla el servidor BackEnd, el cual se encarga de ofrecer el servicio opcional de segundo factor de autenticación o 2FA a los usuarios.

4.6.1. Implementación del BackEnd de 2FA

Para esta tarea, se desarrolló un servidor NodeJs, con el fin de recibir solicitudes por parte de la aplicación móvil. El tipo de autenticación escogida para este componente del sistema completo fue de autenticación móvil de dos factores mediante el algoritmo de contraseña de un solo uso, o OTP (One Time Password). Esta es una implementación de seguridad que requiere de tres partes: un servidor de autenticación, la aplicación móvil y una tercera aplicación que genere las claves o contraseñas de un solo uso, como Google Authenticator o Authy. Es necesario que el usuario active la opción desde la aplicación móvil, la cual envía una solicitud al servidor NodeJS con el token de sesión del cliente. Una vez el mensaje es recibido por el BackEnd, este genera una clave secreta de 80 bits mediante el algoritmo SHA512, la cual es almacenada en la base de datos y enviada de vuelta al usuario, junto con un código QR para ser escaneada por la tercera aplicación.

Cuando la aplicación móvil requiera confirmar la identidad del usuario con la sesión activa, le pedirá al mismo que inserte el código generado por la tercera aplicación. Este código tiene un tiempo de expiración corto, de unos 30 segundos, que es enviada al servidor NodeJs de autenticación, el cual, a través del uso del algoritmo de Time-based One-Time Password (TOTP), verifica que la contraseña es válida para la clave secreta del usuario almacenada en la base de datos. Debido a que el tiempo de expiración de este tipo de claves es muy corto, se agregó un margen donde los últimos 5 códigos (2 minutos y medio) son considerados válidos por el servidor NodeJs. El sistema de activación se encuentra descrito en *Anexo 13: Diagrama de Secuencia de la Activación del Segundo Factor de Autenticación*

La aplicación web fue implementada utilizando express para la generación de los servicios REST y las librerías de npm de nodemailer para el envío de correos electrónicos al usuario, speakeasy, que permite generar las claves de un solo uso y finalmente QRCode, que genera los códigos QR que son enviados al usuario. Además, cuenta con una conexión de tipo administradora de Firebase,

lo que le permite escribir y leer datos dentro de la base de datos. En la siguiente tabla, se describen las rutas implementadas por el servidor BackEnd de autenticación:

| Ruta | Método | Descripción |
|-----------------------|--------|--|
| /setup /enable | POST | Ruta que maneja la solicitud de activación del servicio de segundo factor de autenticación. Recibe como entrada el token del usuario, el cual es verificado utilizando Firebase. Luego, genera el secreto del usuario, lo almacena en la base de datos y envía un correo electrónico al usuario con el código QR correspondiente a su token. Devuelve a la aplicación móvil un mensaje con el código asociado a transacción exitosa, o con el mensaje de error. |
| /setup /deactivate | POST | Ruta que desactiva el servicio de segundo factor de autenticación. Es necesario enviar el token generado por Firebase del usuario, para verificar la identidad del mismo. Sobrescribe los datos del token del usuario, cambiando los valores de los atributos enabled y activated a false del objeto Token. Devuelve a la aplicación móvil un mensaje con el código asociado a transacción exitosa, o con el mensaje de error. |
| /setup /verify | POST | Ruta que verifica que el usuario introdujo correctamente el código QR generado a partir de su secreto en una aplicación de autenticación. Acepta como entrada el token del usuario generado por Firebase, y la contraseña de un solo uso. Si esta es válida, devuelve a la aplicación móvil un mensaje con el código asociado a transacción exitosa, y sobrescribe el valor del atributo activated a true del objeto Token del usuario almacenado en la base de datos. En caso contrario, regresa el mensaje de error respectivo. |
| /verify | POST | Ruta que maneja la autenticación de un usuario que haya activado (/setup/enable) y verificado (/setup/verify) el servicio de segundo factor de autenticación. Recibe como entrada el token generado por Firebase del usuario y la contraseña de un solo uso. Compara el valor de esta con el campo secret almacenado en Realtime Database y verifica que es válida utilizando el algoritmo de TOTP. Devuelve un mensaje con el código asociado a transacción exitosa o de error, si el código no es válido o existen problemas verificando al usuario. |

Tabla 13: Rutas del Servidor de Segundo Factor de Autenticación del Sistema

Desde el lado de la aplicación móvil, se implementó una clase nueva, llamada TwoFactorService, responsable de realizar las llamadas al servidor NodeJS desarrollado. Se crearon una serie de métodos, con el objetivo de enviar la información al servidor de manera correcta, y recibir la respuesta de forma de mostrar las respuestas adecuadas al usuario dentro del aplicativo. En la siguiente tabla se muestran las funciones creadas en esta clase.

| Nombre | Descripción |
|----------------|---|
| Activate2FAU | Método que activa la opción de segundo factor de autenticación para el usuario con la sesión activa dentro de la aplicación móvil. Realiza una llamada al servidor NodeJS con el token generado por Firebase del usuario. |
| Validate2FAU | Método que confirma la activación del segundo factor de autenticación para el usuario con la sesión activa dentro de la aplicación móvil. Envía el token generado por Firebase del usuario y la contraseña de un solo uso introducida por el usuario. |
| Verify2FAU | Método que confirma que una contraseña de solo uso corresponde a un usuario, efectivamente verificando la identidad del usuario. Envía el token generado por Firebase del usuario y la contraseña de un solo uso introducida por el usuario. |
| Deactivate2FAU | Método que desactiva el segundo factor de autenticación del usuario dentro del aplicativo móvil. Realiza una llamada al servidor NodeJS con el token generado por Firebase del usuario. |

Tabla 14: Funciones de la clase TwoFactorService de la Aplicación Móvil

Se crearon los métodos globales dentro de la clase SharedService, desde la cual los componentes account-security o ajustes de la seguridad de la cuenta y send o recibo, redirigen las solicitudes TwoFactorService.

4.6.2. Pruebas de Funcionalidad en Dispositivos Móviles

La integración de Cordova a la aplicación móvil, permite agregar las funcionalidades asociadas a componentes nativos de dispositivos móviles a el sistema final. Fue necesario la instalación de numerosos componentes de Cordova, en especial para la autenticación con Firebase, redireccionamiento a la aplicación, manejo de vistas desde el navegador, acceso a la cámara nativa

del dispositivo, y la habilidad de poder escanear códigos QR. La mayoría de estos métodos se encontraban implementados, mas no probados, y en algunos casos se tuvo que realizar ajustes a la lógica de los mismos con el fin de que el funcionamiento en dispositivos reales fuera robusto. Se modificaron los archivos de hojas de estilo para utilizar variables globales, con el objetivo de mantener consistencia entre los diferentes componentes de la aplicación móvil y verificar que el diseño fuera responsivo y adaptativo a diferentes dispositivos.

Las pruebas en dispositivos móviles fueron realizadas utilizando dispositivos Android, ya que el equipo de desarrollo otorgado por la empresa fue una computadora de escritorio con sistema operativo Windows 10, limitante al momento de probar en sistemas iOS. Se descargaron las versiones de Android 6.0 y 7.0, y se realizaron pruebas en tres dispositivos diferentes: un LG G6 con Android 7.0, un Xperia Z5 Dual con Android 6.0 y un Samsung S7 con Android 7.1.1.

En esta fase también se verifica que efectivamente la aplicación móvil maneja correctamente los errores, se agregan las alertas o pantallas de carga necesarias para incluir interactividad a las pantallas mientras se realizan operaciones asíncronas, tales como recuperación o envío de datos a los servicios REST. Se realiza la traducción de la aplicación de manera de mostrar todos textos tanto en español como en inglés, se refactoriza información innecesaria o duplicada dentro de los componentes y se busca generalizar en lo más posible los datos con el fin de evitar complicaciones innecesarias al momento de la instanciación del producto.

Se generaron archivos de configuración, con el fin de que inyectar o actualizar datos a la aplicación se realice desde archivos estáticos. Estos serán modificados al momento de la realización o generación de la instancia del producto.

4.7. Aporte Tecnológico

Evaluar dos herramientas para automatización de tareas en aplicaciones web desarrolladas en ambientes NodeJS.

Al instanciar un producto, es necesario realiza cambios de variables de configuración y de hojas de estilo, con el fin de adaptar y personalizar lo más posible la aplicación móvil a los requerimientos del cliente. Este tipo de trabajo puede llegar a consumir bastante tiempo de desarrollo, ya que es una labor extensa mas no compleja. Para evitar realizar estas tareas a mano, existen herramientas de construcción, las cuales permiten simplificar prepara los ambientes de aplicaciones web para ser ejecutadas. Estas herramientas permiten automatizar tareas, realizar análisis de código, modificar y crear archivos, entre otras cosas. Se compararon dos herramientas que permiten ofrecer funcionalidades semejantes:

- GulpJS es un kit de herramientas o “toolkit”, desarrollada para ambientes web que trabajen en NodeJS. Permite definir y ejecutar tareas mediante el uso de “pipes” o tuberías, las cuales leen datos de un directorio, las modifican y las escriben en otro directorio. Este sistema, sencillo de configurar, permite modificar archivos de manera fácil, con alto rendimiento.
- Webpack es una herramienta de empaquetamiento, con un alto número de funcionalidades avanzadas para la compilación de aplicaciones web. Permite construir módulos de JavaScript con el contenido de las dependencias utilizadas en el mismo archivo, modificar recursos y mejorar el rendimiento de la aplicación al momento de ser ejecutada.

Ionic 3 por defecto usa Webpack para la generación de archivos estáticos, cuando en versiones anteriores utilizaba GulpJs, por lo que no es necesario integrar Webpack al proyecto. La comparación consistió en implementar las

tareas asociadas a la inyección de recursos visuales en las pantallas de Inicio, Menú Lateral, Registro e Inicio de Sesión, específicamente los iconos del menú, imágenes de fondo y diferentes segmentos, además de tareas asociadas con la generación de archivos estáticos y compresión de código. Esto es un simulacro que sirve de ejemplo de los cambios que se deben realizar dentro de un sistema informático para ser instanciado. También se realizaron cambios referentes a las traducciones ofrecidas.

Se estableció un directorio con los recursos visuales a insertar dentro de la aplicación, y mediante scripts de tareas de ambas herramientas, introducirlas a la aplicación móvil. Al correr los scripts utilizando la consola de Ionic, se generó un directorio /www con el resultado. Ambas herramientas produjeron el mismo resultado a nivel de las pantallas generadas, por lo que no existe una diferencia en el resultado esperado. La tabla comparativa se encuentra en *CAPITULO V.10 Aporte Tecnológico*.

Se considera que se puede migrar de GulpJS a Webpack para el proceso de instanciación, o incluso utilizar ambas herramientas al mismo tiempo debido a la factibilidad de uso y previas experiencias. Configurar GulpJS para ser utilizado en Ionic no es complejo, pero WebPack logra realizar el mismo producto final y ya se encuentra integrado para el mismo proceso de empaquetamiento del aplicativo final.

CAPÍTULO V. RESULTADOS

5.1. Desarrollar las reglas necesarias para asegurar la persistencia de datos del aplicativo utilizando Firebase como BackEnd.

Se obtuvieron las estructuras a almacenar en el aplicativo, incluyendo reglas de validación y tipos de datos. Se configuró adecuadamente Firebase Realtime Database a través de la consola de proyectos, para establecer las reglas de escritura y lectura de los nodos de la base de datos de la aplicación. Las estructuras de datos se encuentran detalladas en *Anexo 3: Modelos de Objetos Almacenados en Firebase Realtime Database*.

Las reglas de seguridad forman parte de los resultados del aporte funcional, presentados en *CAPITULO V.9. Aporte Funcional: Diseñar el proceso de gestión de riesgos de seguridad de las billeteras de criptomonedas para asegurar la integridad de los datos en el aplicativo móvil*.

5.2. Desarrollar la aplicación web para poder activar el segundo factor de autenticación en el aplicativo móvil.

Se desarrolló un servidor NodeJS con el objetivo de ofrecer el servicio de segundo factor de autenticación a la aplicación móvil, manejando solicitudes por parte de la aplicación móvil con el fin de otorgar permisos a un usuario debidamente identificado para realizar transacciones dentro del sistema. Es un elemento de seguridad adicional, con el fin de evitar riesgos asociados a transacciones no deseadas, complementando la aplicación móvil. La descripción del desarrollo de este servicio, incluyendo los métodos y operaciones realizadas se encuentra descrito en el *CAPITULO IV.6.1. Implementación del BackEnd de 2FA*.

5.3. Diseñar e implementar una aplicación móvil híbrida para el manejo

de billeteras de criptomonedas.

Se obtuvo una aplicación móvil híbrida desarrollada en ambiente Angular 4, Ionic 3 y Cordova 5, con cinco módulos interconectados. La descripción del desarrollo de la aplicación móvil se encuentra descrito en el *CAPITULO IV.2. Fase de Diseño de la Aplicación, IV.3. Fase de Implementación del Aplicativo Móvil, IV.4. Fase de Implementación de las Billeteras y IV.4.6.2 Pruebas de Funcionalidad en Dispositivos Móviles*. Las interfaces se encuentran representadas en *Anexo 14: Pantallas de la Aplicación Móvil*.

5.4. Desarrollar el módulo de Autenticación del usuario en el aplicativo móvil.

El producto derivado obtenido de completar este objetivo, fue un módulo de la aplicación móvil, el cual permite manejar la autorización y autenticación de los usuarios mediante el uso del servicio Firebase Auth. Ofrece numerosos servicios relacionados con las credenciales utilizados por los usuarios, el manejo de claves y estados de sesión sin necesidad de codificar un BackEnd específicamente para esto. El desarrollo de este módulo se encuentra especificado en el *CAPITULO IV.3.4 Desarrollo del Módulo de Autenticación*.

5.5. Desarrollar el módulo de Gestión de Contactos en el aplicativo móvil.

El producto derivado obtenido de completar este objetivo, fue un módulo de la aplicación móvil, el cual se encarga de la gestión de los contactos del usuario del sistema, cumpliendo con las operaciones de consultar, agregar, eliminar y editar la información almacenada en Realtime Database. Realiza también las validaciones necesarias para el correcto funcionamiento de la aplicación. El desarrollo de este módulo se encuentra especificado en el *CAPITULO IV.3.3 Desarrollo del Módulo de Gestión de Contactos*.

5.6. Desarrollar el módulo de Gestión de Billeteras en el aplicativo móvil.

El producto derivado obtenido de completar este objetivo, fue un módulo de la aplicación móvil, el cual se encarga de la gestión de las billeteras del usuario del sistema, de manera de permitir la creación de billeteras en cinco plataformas, las cuales son Bitcoin, Dogecoin, Dash, Litecoin y Ethereum; además de tres cadenas de bloques de prueba, el Testnet de Bitcoin, y los Testnets de BlockCypher para Bitcoin y Ethereum. Genera y almacena la información criptográfica y de las billeteras en Realtime Database y utiliza la API de BlockCypher para el registro de las billeteras HD y la consulta de información de las mismas. Realiza también las validaciones necesarias para el correcto funcionamiento de la aplicación. El desarrollo de este módulo se encuentra especificado en el *CAPITULO IV. 4.3 Implementación del Módulo de Billetera.*

5.7. Desarrollar el módulo de Consultas al Blockchain en el aplicativo móvil.

El producto derivado obtenido de completar este objetivo, fue un módulo de la aplicación móvil, el cual se encarga de la realizar consultas a la API de BlockCypher con el fin de obtener información de una cadena de bloques. Esta información se utiliza para el funcionamiento interno de la aplicación, o para la consulta de datos específicos por el usuario. Realiza también las validaciones necesarias para el correcto funcionamiento de la aplicación. El desarrollo de este módulo se encuentra especificado en el *CAPITULO IV.4.2 Implementación del Módulo de Consultas al Blockchain.*

5.8. Desarrollar el módulo de Generación de Transacciones en el aplicativo móvil.

El producto derivado obtenido de completar este objetivo, fue un módulo de la aplicación móvil, el cual se encarga de la generación de transacciones de las

billetteras del usuario del sistema, de manera de permitir el envío de criptomonedas a terceros. Maneja las operaciones de firmado y envía la información a la API de BlockCypher para el registro y la consulta de información de las transacciones. Realiza también las validaciones necesarias para el correcto funcionamiento de la aplicación. El desarrollo de este módulo se encuentra especificado en el *CAPITULO IV.4.4 Implementación del Módulo de Generación de Transacciones*.

5.9. Aporte Funcional

Diseñar el proceso de gestión de riesgos de seguridad de las billetteras de criptomonedas para asegurar la integridad de los datos en el aplicativo móvil.

Como parte de este proceso, se obtuvo el archivo de configuración de las reglas de seguridad de Firebase Realtime Database, presentado en *Anexo 15: Reglas de Seguridad establecidas en Firebase Realtime Database*. El análisis realizado para la seguridad del producto se encuentra descrito en *CAPITULO IV.5 Aporte Funcional*.

5.10. Aporte Tecnológico

Evaluar dos herramientas para automatización de tareas en aplicaciones web desarrolladas en ambientes NodeJS.

Se realizó una comparación entre GulpJS y Webpack, descrita en el *CAPITULO IV.7 Aporte Tecnológico*. En la siguiente tabla se muestra los resultados de la comparación realizada:

| Comparación | GulpJS | WebPack |
|---------------------|----------------------|---------------|
| Tipo de herramienta | Kit de herramientas. | Empaquetador. |

| | | |
|--|---|--|
| Archivo de configuración | gulpfile.js | webpack.config.js |
| Componentes Adicionales Instalados Para la prueba | Jshint Concat Minify css Rename Sass Shelljs Gutil Angular-templatecache | html-webpack-plugin html-loader url-loader img-loader file-loader css-loader sass-loader postcss-loader |
| Facilidad de configuración | Medio | Medio |
| Tiempo de compilación | 328ms (solo recursos) | 381s (empaquetado completo) |
| Ejemplo de Pantalla generada |  |  |

Tabla 15: Comparación entre GulpJs y Webpack

Nota: Pruebas realizadas en laptop Asus i7 2.4Ghz, 8GB de RAM, sistema operativo Windows 10.

CAPÍTULO VI. CONCLUSIONES Y RECOMENDACIONES

6.1. Conclusiones

- La metodología ágil Kanban es muy buena para el desarrollo de sistemas informáticos, ya que permite construir software de manera gradual, ajustándose a tiempos cortos y requerimientos variables. Al establecerse una fase de pruebas durante los ciclos de las tareas, también se producen aplicativos de alto grado de calidad, con prioridades establecidas según los requerimientos del proyecto y el tamaño y habilidades del equipo.
- El uso de Angular 4 permite establecer proyectos con múltiples componentes interconectados, facilitando el desarrollo y puesta en producción de aplicaciones modulares e instanciables, con alto nivel de abstracción. El uso de TypeScript, que facilita el desarrollo orientado a objetos y clases, ofrece una ventaja también para agregar estructura de datos directamente relacionada con las respuestas que se obtienen desde un servidor REST. Angular es además un framework con mucho soporte por parte de los usuarios y soporta numerosos tipos de aplicaciones web, bajo diversos estándares y patrones de programación.
- El conjunto de herramientas Ionic 2 + Cordova 5 es recomendado para aplicaciones híbridas, puesto que Ionic posee una curva de aprendizaje baja, con documentación extensa y múltiples componentes visuales que simulan de manera adecuada elementos visuales nativos de teléfonos inteligentes. Al ser una aplicación web, no es necesario generar instancias diferentes para aplicaciones IOS o Android, y trasladar los estilos visuales a diferentes dispositivos es menos complejo por poseer soporte a hojas de estilo CSS.
- El uso del set de herramientas de Firebase facilita la creación de productos instanciables móviles para la empresa, puesto que se evita el desarrollo de un BackEnd solamente para el producto, disminuyendo el tiempo de implementación, además, se adapta adecuadamente a las necesidades

de la aplicación móvil, la cual requiere de un BackEnd con alta disponibilidad y robusto.

- El uso de la API de BlockCypher para el manejo de las billeteras disminuyó notablemente el uso de información que tuvo que ser almacenada y la cantidad de funciones que tuvieron que ser desarrolladas para el funcionamiento adecuado de la aplicación móvil.
- El uso de la API de BlockCypher evito tener que desarrollar un servidor que actuara como nodo de numerosas cadenas de bloques para el envío de transacciones. El tener que implementar este tipo de estructura habría sido costoso para la empresa, puesto que, por ejemplo, el Blockchain de Bitcoin tiene un tamaño de varios cientos de gigabytes, y requiere de una conexión estable a internet y alto poder de computo, con alta disponibilidad para recibir y enviar respuestas a la aplicación móvil, lo que probablemente implicaría la compra adicional de equipos.
- El desarrollo de un módulo de gestión de contactos dentro de la aplicación móvil que automáticamente permite enviar dinero a otro usuario registrado sin necesidad de tener que escribir o escanear la dirección de recibo de dinero aumenta la usabilidad y seguridad del sistema completo.
- La implementación de un sistema con la opción de activar un segundo factor de autenticación, aumenta la seguridad de la aplicación, la cual posee una naturaleza delicada, ya que maneja activos digitales con valor económico, y es necesario establecer políticas de seguridad que disminuyan los riesgos a los mismos.

6.2. Recomendaciones

- Es necesario el análisis de los requerimientos del cliente previo la especificación del sistema informático desarrollado, puesto que la suite de herramientas de Firebase puede no ajustarse adecuadamente a las fortalezas y limitaciones del cliente. Se considera que Firebase posee un gran número de funcionalidades para clientes que no posean una base de

datos previa que quieran utilizar, o si no tienen desarrollado un servidor para el manejo de solicitudes.

- Se recomienda analizar que otros componentes de la suite de Firebase pueden ser integrados a la aplicación móvil, puesto que, usada correctamente, Firebase Functions podría realizar operaciones desde un servidor seguro sin necesidad de tener que desarrollar un BackEnd.
- Es necesario analizar los riesgos de seguridad que se puedan generar si el sistema será integrado a una plataforma existente o como un componente adicional a los servicios ofrecidos por el cliente.
- Implementar una clase dentro de la aplicación móvil que utilice la API de Eventos de BlockCypher, con el fin de desarrollar un módulo de notificaciones de transacciones recibidas.
- Como Billetera de tipo online o “hot wallet” una vez que el producto se instancie se debe considerar agregar alguna funcionalidad que permite exportar la billetera de manera mas segura y cómoda, independientemente de si se almacenan las claves criptográficas en un servidor BackEnd o en el dispositivo. Se debe evitar mostrar la frase de recuperación de la billetera por razones de seguridad.
- Si las características del cliente final del producto lo permiten, almacenar la información criptográfica dentro del dispositivo final, y no en un BackEnd.

REFERENCIAS BIBLIOGRÁFICAS

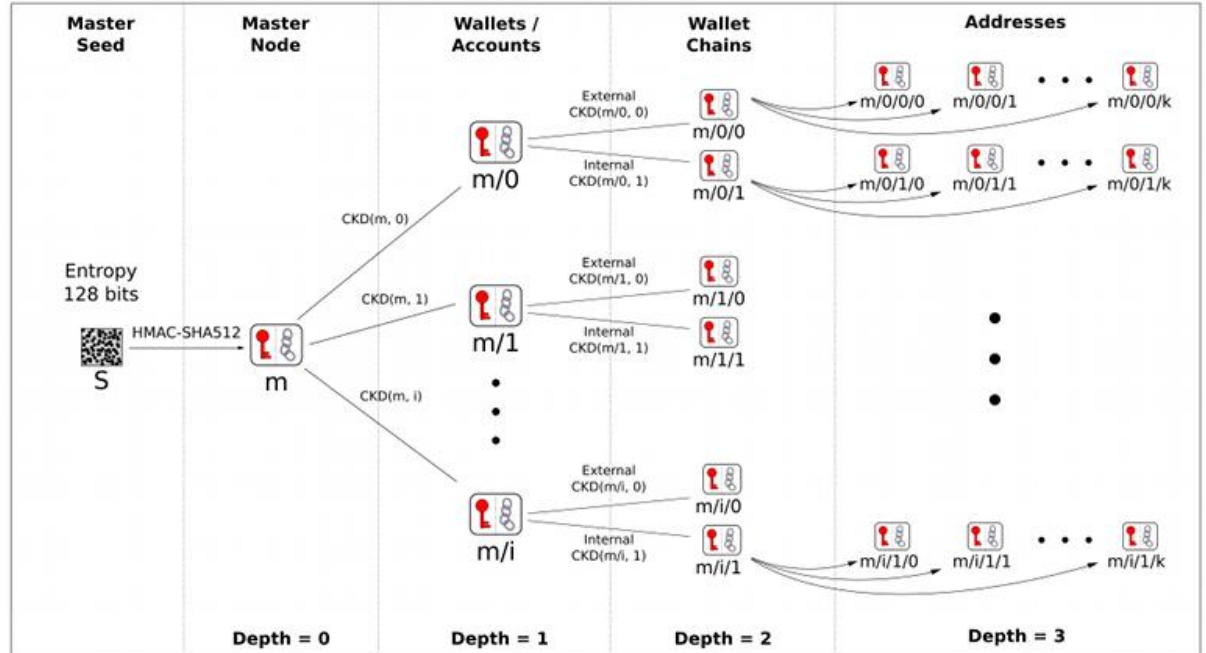
1. Desjardins, Jeff. (2017). *Comparing Bitcoin, Ethereum, and Other Cryptos*. Visual Capitalist. Disponible en: <http://www.visualcapitalist.com/comparing-bitcoin-ethereum-cryptos/> Consultado el 01 de junio de 2018.
2. Google. (2018). *Angular Framework Architecture Overview*. Disponible en: <https://angular.io/guide/architecture>. Consultado el 01 de junio de 2018
3. Ionic Team. (2018). *Ionic Framework*. Disponible en: <https://ionicframework.comr>. Consultado el 01 de marzo de 2018.
4. Apache Software Foundation. (2015). *Cordova Introduction*. Disponible en: <https://cordova.apache.org/docs/en/latest/>. Consultado el 12 de marzo de 2018
5. Google. (2018). *Firebase Console*. Disponible en: <https://firebase.google.com>. Consultado el 01 de marzo de 2018
6. Wuille, Pieter. (2011). *Hierarchical Deterministic Wallets*. BIP 32. Disponible en: <https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki>. Consultado el 28 de marzo de 2018.
7. Palatinus, Marek, Pavol, Rusnak, et at. (2013). *Mnemonic code for generating deterministic keys*. BIP 39. Disponible en: <https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki>. Consultado el 28 de marzo de 2018.
8. BlockCypher Inc. (2016). *BlockCypher API Documentation*. Disponible en: <https://blockcypher.github.io/documentation/>. Consultado el 15 de junio dede 2018
9. Chohan, Usman (2017). *Cryptocurrencies: A Brief Thematic Review*. UNSW Business School. Disponible en: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3024330 Consultado el 20 de junio de 2018.
10. Cornish, Chloe (2018). *Growing number of cryptocurrencies spark*

concerns. Financial Times. Disponible en:
<https://www.ft.com/content/a6b90a8c-f4b7-11e7-8715-e94187b3017e>
Consultado el 20 de junio de 2018.

11. National Institute of Standards and Technology. *Descriptions of SHA-256, SHA-384 and SHAD-512.* Disponible en:
<https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/example-values#aHashing>. Consultado el 20 de junio de 2018
12. Kumar, Toshendra. *How Does Blockchain use public key Cryptography.* (2017). Blockchain Council. Disponible en: <https://www.blockchain-council.org/blockchain/how-does-blockchain-use-public-key-cryptography/>. Consultado el 19 de junio de 2018.
13. Vault0x. Hierarchically Deterministic Wallets, The Concepts (2017). Disponible en: <https://medium.com/vault0x/hierarchically-deterministic-wallets-the-concepts-3aa487e71235>. Consultado el 21 de junio de 2018.
14. Laptick, Sergey. *How & Why to Use the Kanban Methodology for Software Development.* (2016). SitePoint. Disponible en:
<https://www.sitepoint.com/how-why-to-use-the-kanban-methodology-for-software-development/satosji>. Consultado el 23 de marzo de 2018.
15. Nakamoto, Satoshi. *Bitcoin: A Peer-to-Peer Electronic Cash System.* (2008). Bitcoin. Disponible en: <https://bitcoin.org/bitcoin.pdf>. Consultado el 19 de marzo de 2018.

Anexo 1: Funcionamiento de las Billeteras HD

BIP 32 - Hierarchical Deterministic Wallets



Child Key Derivation Function \sim $CKD(x,n) = \text{HMAC-SHA512}(x_{\text{Chain}}, x_{\text{PubKey}} || n)$

Imagen extraída de <https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki>

Anexo 2: Cronograma del Proyecto

Cronograma del Proyecto

| Actividad | Fecha Inicio | Fecha Fin | Duración (Semanas) | Abril | | | | | | | Mayo | | | Junio | | | Julio | | | |
|--|--------------|------------|--------------------|-------|---|---|---|---|---|---|------|---|----|-------|----|----|-------|----|----|----|
| | | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 14 | 13 | 15 | 16 | 17 |
| Exploración | | | | | | | | | | | | | | | | | | | | |
| Generación de base de conocimientos sobre criptomonedas | 19/03/2018 | 23/03/2018 | 1 | | | | | | | | | | | | | | | | | |
| Reconocimiento de los servicios REST de BlockCypher | 19/03/2018 | 23/03/2018 | 1 | | | | | | | | | | | | | | | | | |
| Reconocimiento de los servicios de Firebase | 19/03/2018 | 23/03/2018 | 1 | | | | | | | | | | | | | | | | | |
| Diseño de la Aplicación | | | | | | | | | | | | | | | | | | | | |
| Diseño de las estructuras de datos | 19/03/2018 | 23/03/2019 | 1 | | | | | | | | | | | | | | | | | |
| Determinación de Requerimientos Funcionales y no Funcionales | 19/03/2018 | 30/03/2018 | 2 | | | | | | | | | | | | | | | | | |
| Configuración de Realtime Database | 26/03/2018 | 30/03/2018 | 1 | | | | | | | | | | | | | | | | | |
| Planificación de las tareas | 26/03/2018 | 30/03/2018 | 1 | | | | | | | | | | | | | | | | | |
| Implementación del Aplicativo Móvil | | | | | | | | | | | | | | | | | | | | |
| Adaptación de las Vistas del FrontEnd | 02/04/2018 | 18/05/2018 | 7 | | | | | | | | | | | | | | | | | |
| Adaptación de los Modelos de datos del FrontEnd | 02/04/2018 | 13/04/2018 | 2 | | | | | | | | | | | | | | | | | |
| Desarrollo del Módulo de Gestión de Contactos | 09/04/2018 | 20/04/2018 | 2 | | | | | | | | | | | | | | | | | |
| Desarrollo del Módulo de Autenticación | 16/04/2018 | 30/04/2018 | 3 | | | | | | | | | | | | | | | | | |
| Implementación de las Billeteras | | | | | | | | | | | | | | | | | | | | |
| Adaptación de los servicios REST del FrontEnd | 23/04/2018 | 27/04/2018 | 1 | | | | | | | | | | | | | | | | | |
| Implementación del Módulo de Consultas al BlockChain | 30/04/2018 | 26/05/2018 | 4 | | | | | | | | | | | | | | | | | |
| Implementación del Módulo de Billeteras | 07/05/2018 | 15/06/2018 | 6 | | | | | | | | | | | | | | | | | |
| Implementación del Módulo de Generación de Transacciones | 14/05/2018 | 15/06/2018 | 5 | | | | | | | | | | | | | | | | | |
| Implementación del BackEnd y Pruebas | | | | | | | | | | | | | | | | | | | | |
| Implementación del 2FA | 11/06/2018 | 22/06/2018 | 2 | | | | | | | | | | | | | | | | | |
| Pruebas de Funcionalidad en Dispositivos Móviles | 11/06/2018 | 29/06/2018 | 3 | | | | | | | | | | | | | | | | | |

Anexo 3: Modelos de Objetos Almacenados en Firebase Realtime Database

3.1. Modelo de Datos User

| | | | | |
|--|--------|--|-----------|-------|
| Objeto | User | | | |
| Descripción | | | | |
| Objeto que almacena información relevante de un usuario de la aplicación, incluyendo preferencias de usuario, datos de cada billetera y su libreta de contactos. | | | | |
| Atributos | | | | |
| Nombre | Tipo | Descripción | Requerido | Único |
| userEmail | String | Correo electrónico del usuario. | Si | Si |
| Img | String | Dirección a la imagen de perfil del usuario. Si el proveedor es Google, este es su imagen del correo electrónico. En caso contrario, es una imagen almacenada en Firebase Storage. | Si | No |
| currency | String | Abreviación de la moneda local que el usuario utiliza. Permite generar una referencia al valor de una criptomoneda o del saldo de una billetera. Valor por defecto es USD. | Si | No |
| token | Token | Cada usuario tiene asociado un objeto de tipo Token, el cual permite manejar la información referente al | Si | Si |

| | | | | |
|-------------|---------------------|--|----|----|
| | | segundo factor de autenticación | | |
| wallet | Árbol de Billeteras | Lista de todas las billeteras asociadas al usuario. Cada vez que se inserta una billetera nueva, esta se encuentra debajo de un id único. | No | No |
| addressBook | Árbol de Contactos | Lista de todos los contactos de la libreta de contactos del usuario. Cada vez que se inserta un contacto nuevo, esta se encuentra debajo de un id único. | No | No |

3.2. Modelo de Datos Usuario (Firebase)

| | | | | |
|--|-------------------------|---|-----------|-------|
| Interfaz | Usuario (Firebase Atuh) | | | |
| Descripción | | | | |
| Interfaz de la clase usuario de Firebase Auth. Posee información relevante a un usuario registrado en la aplicación. | | | | |
| Atributos | | | | |
| Nombre | Tipo | Descripción | Requerido | Único |
| uid | String | Cadena de caracteres única generada por Firebase Auth que identifica al usuario. | Si | Si |
| email | String | Correo electrónico del usuario. | Si | Si |
| emailVerified | Boolean | Indica si el usuario confirmo su correo electrónico a través de Firebase Auth. Si el usuario inicia sesión a través de Google, esto es True. | Si | No |
| photoURL | String | Dirección a la imagen de perfil del usuario. Si el proveedor es Google, este es su imagen del correo electrónico. En caso contrario, es una imagen almacenada en Firebase Storage | Si | Si |
| displayName | String | El nombre o alias del usuario. Si este accede a través de Google, es el nombre asociado al correo electrónico. En caso contrario es el correo electrónico. | No | No |

| | | | | |
|--------------|---------|--|----|----|
| providerData | Objeto | Información sobre el proveedor del usuario | No | No |
| providerID | String | ID del proveedor del usuario. | Si | No |
| phoneNumber | String | Número de teléfono asociado al usuario. | No | No |
| metadata | Objeto | Metadata sobre el usuario, tal como último acceso, si es un usuario nuevo y coordenadas, | No | No |
| isAnonymous | Boolean | Indica si el usuario se encuentra anónimo | Si | No |

3.3. Modelo de Datos User (List)

| | | | | |
|--|-------------|---|-----------|-------|
| Objeto | User (List) | | | |
| Descripción | | | | |
| Objeto que almacena información relevante de un usuario de la aplicación, incluyendo su imagen de para consulta por parte de otros usuarios. | | | | |
| Atributos | | | | |
| Nombre | Tipo | Descripción | Requerido | Único |
| Email | String | Correo electrónico del usuario. | Si | Si |
| Img | String | Dirección a la imagen de perfil del usuario. Si el proveedor es Google, este es su imagen del correo electrónico. En caso contrario, es una imagen almacenada en Firebase Storage | Si | No |
| Email | String | Abreviación de la moneda local que el usuario utiliza. Permite generar una referencia al valor de una criptomoneda o del saldo de una billetera. Valor por defecto es USD. | Si | No |

3.4. Modelo de Datos AddressBook

| | | | | |
|--|-------------|---|-----------|-------|
| Objeto | AddressBook | | | |
| Descripción: Almacena una lista de contactos que el usuario haya guardado en su libreta de contactos. Los contactos son otros usuarios de la aplicación. | | | | |
| Atributos | | | | |
| Nombre | Tipo | Descripción | Requerido | Único |
| Alias | String | Nombre que el usuario coloca al contacto. | Si | No |
| Email | String | Correo electrónico del contacto. | Si | Si |
| Img | String | Dirección a la imagen de perfil del contacto. Si el proveedor es Google, este es su imagen del correo electrónico. En caso contrario, es una imagen almacenada en Firebase Storage. | No | Si |
| Uid | String | Cadena de caracteres única generada por Firebase Auth que identifica al contacto. | Si | No |

3.5. Modelo de Datos Wallet (User)

| | | | | |
|--|---------------|---|-----------|-------|
| Objeto | Wallet (User) | | | |
| Descripción | | | | |
| Objeto que almacena la información de una billetera. Incluye los datos de las llaves públicas y privadas, el nombre de la criptomoneda y preferencias del usuario. | | | | |
| Atributos | | | | |
| Nombre | Tipo | Descripción | Requerido | Único |
| Name | String | Nombre de la billetera. Este valor es utilizado por BlockCypher para identificar cada billetera dentro de un Blockchain. | Si | Si |
| Keys | Keys | Objeto que almacena la información criptográfica de la billetera, necesario para realizar transacciones y generar nuevas direcciones | Si | Si |
| Crypto | Crypto | Objeto que almacena la información asociada a la criptomoneda seleccionada y las preferencias del usuario respecto a las unidades. | Si | No |
| address | String | Dirección de envió y recibo de criptomonedas. Solo si la criptomoneda de la billetera es Ethereum. Es generado a través del Keys de la billetera. | No | Si |

3.6. Modelo de Datos Wallet (List)

| | | | | |
|---|---------------|---|-----------|-------|
| Objeto | Wallet (List) | | | |
| Descripción | | | | |
| Lista que almacena información no sensible. Utilizada por la aplicación para poder manejar ciertas funciones dentro de la aplicación. | | | | |
| Atributos | | | | |
| Nombre | Tipo | Descripción | Requerido | Único |
| Name | String | Nombre de la billetera. Este valor es utilizado por BlockCypher para identificar cada billetera dentro de un Blockchain. | Si | Si |
| User | String | Correo electrónico del usuario que es dueño de la billetera. | Si | No |
| Crypto | String | Abreviación del nombre de la criptomoneda y si es cadena principal o de prueba. Valores: bcy (BlockCypher Testnet), btc (Bitcoin), tes (Bitcoin Testnet), dog (Dogecoin), ltc (Litecoin), eth (Ethereum), tet (Ethereum Testnet), das (Dash). | Si | No |
| address | String | Dirección de envió y recibo de criptomonedas. Solo si la criptomoneda de la billetera es Ethereum. Es generado a través del Keys de la billetera. | No | Si |

3.7. Modelo de Datos Keys

| | | | | |
|---|--------|--|-----------|-------|
| Objeto | Keys | | | |
| Descripción | | | | |
| Información criptográfica de una billetera. | | | | |
| Atributos | | | | |
| Nombre | Tipo | Descripción | Requerido | Único |
| mnemonics | String | Frase de recuperación de la Billetera, generada a través de BIP 39. | Si | Si |
| passphrase | String | Clave de encriptación utilizada al momento de generar la semilla de la frase de recuperación, es un segundo factor de seguridad. Necesario junto con la clave de recuperación para poder importar una billetera. | Si | No |
| seed | String | Valor de la semilla hexadecimal utilizada para generar la llave privada extendida. Es la clave de recuperación más la clave de encriptación (si esta existe) | Si | Si |
| xpriv | String | Llave privada extendida de la billetera, o llave privada en caso de que la criptomoneda sea Ethereum. | Si | Si |
| xpub | String | Llave pública extendida de la billetera, o llave publica en caso de que la criptomoneda sea Ethereum. | Si | Si |

3.8. Modelo de Datos Crypto

| | | | | |
|---|--------|---|-----------|-------|
| Objeto | Crypto | | | |
| Descripción | | | | |
| Objeto que almacena información sobre la criptomoneda de una billetera y las preferencias de usuario de la misma. | | | | |
| Atributos | | | | |
| Nombre | Tipo | Descripción | Requerido | Único |
| value | String | Abreviación del nombre de la criptomoneda y si es cadena principal o de prueba. Valores: bcy (BlockCypher Testnet), btc (Bitcoin), tes (Bitcoin Testnet), dog (Dogecoin), ltc (Litecoin), eth (Ethereum), tet (Ethereum Testnet), das (Dash). | Si | No |
| coin | String | Abreviación del nombre de la criptomoneda. Valores: BTC, DOGE, LTC, DASH. | Si | No |
| difference | number | Número que representa la diferencia entre una criptomoneda y su unidad más pequeña no divisible. (Ej.: 1 BTC = 100,000,000 satoshis) | Si | No |
| name | String | Nombre completo de la moneda. | Si | No |
| units | Objeto | Objeto que guarda la preferencia del usuario, con el nombre de la unidad elegida y su diferencia con la unidad más pequeña no divisible | Si | No |

3.9. Modelo de Datos Activities

| | | | | |
|---|------------|--|-----------|-------|
| Objeto | Activities | | | |
| Descripción | | | | |
| Almacena una lista de notificaciones referente a actividades que ha realizado el usuario. | | | | |
| Atributos | | | | |
| Nombre | Tipo | Descripción | Requerido | Único |
| date | String | Fecha en la que ocurrió la actividad. | Si | No |
| description | String | Cadena de caracteres con información adicional sobre una | No | No |
| name | String | Nombre de la actividad. | Si | No |

3.10. Modelo de Datos Token

| | | | | |
|---|---------|---|-----------|-------|
| Objeto | Token | | | |
| Descripción | | | | |
| Almacena la información única del segundo facto de autenticación del usuario. | | | | |
| Atributos | | | | |
| Nombre | Tipo | Descripción | Requerido | Único |
| activated | boolean | Indica que la opción de segundo factor de autenticación fue activada y verificada. | Si | No |
| enabled | boolean | Indica que la opción de segundo factor de autenticación fue activada, mas no necesariamente verificada. | Si | No |
| otpURL | String | Descripción del url que provee de información a la aplicación de autenticación | No | Si |
| dataURL | String | Valor en Base32 del código QR utilizado por la aplicación de autenticación | No | Si |
| secret | String | Identificador único del usuario, utilizado para autenticarlo | No | Si |

Anexo 4: Ejemplo de la estructura de datos JSON en Realtime Database



Anexo 5: Descripción de las tareas de Kanban

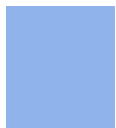
5.1. Descripción de las tareas de Adaptación de las vistas del FrontEnd

| | | | |
|---|---|---|-----------|
| División | | | Color |
| Adaptación de las vistas del FrontEnd | | | |
| Tareas asociadas a la actualización del FrontEnd para mostrar información dinámica, manejo de errores y cambios de estilo | | | |
| N | Nombre | Descripción | Prioridad |
| F-1 | Desarrollo Servicio Global Compartido AlertService | Creación del servicio e implementación de los métodos encargados de mostrar errores y mensajes informativos al usuario a través del uso del componente Alert de Ionic 3. | Baja |
| F-2 | Desarrollo Servicio Global Compartido LoaderService | Creación del servicio e implementación de los métodos encargados de mostrar pantallas de espera a través del uso del componente Loader de Ionic 3. | Baja |
| F-3 | Desarrollo Interceptor HttpErrorInterceptor | Creación de un interceptor de errores mediante el uso de la clase HttpClient de Angular, con el fin de manejar los errores retornados por los servidores REST y las APIs. | Media |
| F-4 | Traducción de la Aplicación Móvil | Uso de la librería de npm ngx-translate y archivos JSON para traducir la aplicación mediante el uso de un servicio clave – valor. | Baja |
| F-5 | Implementación de las vistas del | Modificación de los componentes de Ionic – Angular que permiten al | Media |


| | | | |
|------|--|---|------|
| | módulo de Gestión de Contactos | usuario interactuar con el módulo de Gestión de Contactos: libreta de contactos, agregar contacto y editar contacto. | |
| F-6 | Implementación de las vistas del módulo de Autenticación | Generación y modificación de los componentes de Ionic – Angular que permiten al usuario interactuar con el módulo de Autenticación: inicio de sesión, registro, configuración de cuenta y confirmar correo. | Alta |
| F-7 | Implementación de las vistas del módulo de Consultas al Blockchain | Modificación del componente de Ionic – Angular que permiten al usuario interactuar con el módulo de Consultas al Blockchain: explorador del Blockchain. | Baja |
| F-8 | Implementación de las vistas del módulo de Gestión de Billeteras | Generación y modificación de los componentes de Ionic – Angular que permiten al usuario interactuar con el módulo de Consultas al Gestión de Billeteras: inicio de sesión, ajustes de billeteras, creación de billeteras y mostrar frase de recuperación. | Alta |
| F-9 | Implementación de las vistas del módulo de Generación de Transacciones | Generación y modificación de los componentes de Ionic – Angular que permiten al usuario interactuar con el módulo de Generación de Transacciones: recibir, enviar, confirmación de envió transacciones y detalles de transacción. | Alta |
| F-10 | Cambios a vistas para agregar | Modificación y generación de los componentes de Ionic – Angular que | Alta |

| | | | |
|------|---|---|-------|
| | elementos de 2FA. | permiten al usuario interactuar con las opciones de segundo factor de autenticación: enviar y ajustes de seguridad. | |
| F-11 | Manejo de errores HTTP | Mostrar errores de manera adecuada utilizando AlertService, LoaderService, HttpErrorInterceptor y ngx-translator. | Medio |
| F-12 | Implementación de clase global ExchangeService | Creación de la clase ExchangeService y métodos necesarios para mostrar el valor actualizado del balance de una billetera de criptomonedas en otras divisas. | Bajo |
| F-13 | Aplicación de estilo final a las vistas del aplicativo. | Aplicación de hojas de estilo y recursos visuales a los componentes de Ionic según el diseño determinado por la empresa. | Medio |

5.2. Descripción de las tareas de Adaptación de la capa lógica del FrontEnd

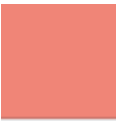
| | | | |
|---|--|--|---|
| División | | | Color |
| Adaptación de la capa lógica del FrontEnd | | |  |
| Tareas asociadas a la actualización del FrontEnd previo para ajustarse a los servicios REST a utilizarse y las nuevas estructuras de datos. | | | |
| N | Nombre | Descripción | Prioridad |
| L-1 | Configuración de Reglas de Firebase | Creación y configuración desde la consola de Firebase para los servicios utilizados dentro del aplicativo. | Baja |
| L-2 | Cambio a las estructuras de Datos de la aplicación móvil | Creación de nuevos modelos de datos y actualización de los preexistentes según lo establecido en la sección de Anexo 9 y Anexo 10 dentro del aplicativo móvil. | Alta |
| L-3 | Creación Servicio Global Compartido SharedService | Generación y construcción de la clase SharedService a ser utilizado de manera global por los demás componentes y servicios dentro del aplicativo móvil. | Alta |
| L-4 | Creación Proveedor Compartido FirebaseProvider | Generación y construcción de la clase FirebaseProvider, encargada de la realización de las llamadas asíncronas a Firebase Realtime Database con el fin de realizar operaciones CRUD en los modelos de datos establecidos en Anexo 3. | Alta |
| L-5 | Creación Servicio Global Compartido RestService | Generación y construcción de la clase RestService, encargada de las llamadas asíncronas a la API de BlockCypher. | Alta |

5.3. Descripción de las tareas del Módulo de Autenticación


| División Módulo de Autenticación | | | Color  |
|--|-------------------------------------|---|--|
| Tareas asociadas a la autenticación, manejo de sesión y registro de los usuarios de la aplicación móvil. | | | |
| N | Nombre | Descripción | Prioridad |
| A-1 | Creación de clase AuthService | Generación de la clase AuthService donde se implementan los métodos y asociados al módulo de autenticación. | Alta |
| A-2 | Registro de usuarios | Registro de nuevos usuarios dentro del sistema utilizando correo electrónico y contraseña. | Media |
| A-3 | Autenticación del usuario. | Proceso asociado a verificar al usuario registrado dentro de la aplicación móvil. | Alta |
| A-4 | Inicio de Sesión | Inicio de sesión de usuarios ya registrados dentro del sistema utilizando correo electrónico y contraseña. | Media |
| A-5 | Verificación del Correo Electrónico | Verifica que el correo electrónico que un usuario utilizo para registrarse en el sistema es de su propiedad al enviar un mensaje de verificación. | Baja |
| A-6 | Recuperación de Contraseña | Envía un mensaje al correo electrónico asociado a un usuario registrado, para restablecer la contraseña utilizada para iniciar sesión | Baja |
| A-7 | Manejo de Estado de Sesión | Manejo del estado de sesión del usuario dentro del aplicativo, incluyendo cierre de sesión y acciones asociadas al inicio de sesión | Alta |
| A-8 | Autenticación de | Registro e inicio de sesión de usuarios al | Media |

| | | | |
|--|----------------------------------|---|--|
| | usuario (Proveedor Google) | sistema utilizando sus cuentas Google. Manejo de redireccionamiento y credenciales asociados. | |
|--|----------------------------------|---|--|


5.4. Descripción de las tareas del Módulo de Gestión de Contactos

| División Módulo de Gestión de Contactos | | | Color |
|---|---|---|---|
| Tareas asociadas a el manejo de la libreta de contactos de el aplicativo. | | |  |
| N | Nombre | Descripción | Prioridad |
| C-1 | Agregar Contactos | Permite a un usuario con sesión activa, agregar nuevos registros dentro de la libreta de contactos de su aplicación. | Media |
| C-2 | Eliminar Contactos | Permite a un usuario con sesión activa, eliminar nuevos registros dentro de la libreta de contactos de su aplicación. | Baja |
| C-3 | Consultar Contactos | Recupera la información de la libreta de contactos de un usuario con sesión activa dentro de la aplicación. | Media |
| C-4 | Editar Contactos | Permite editar la información de un registro almacenado en la libreta de contactos de un usuario con sesión activa dentro de la aplicación. | Baja |
| C-5 | Buscar Contactos por Correo Electrónico | Verifica que un contacto que se vaya a registrar dentro de la libreta de contactos se encuentre registrado en el sistema. | Media |

5.5. Descripción de las tareas del Módulo de Consultas al BlockChain

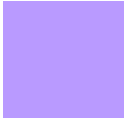
| División | | | Color |
|--|-------------------------------------|---|---|
| Módulo de Consultas al BlockChain | | |  |
| Tareas asociadas a las consultas de información relacionada con las distintas cadenas de bloques, direcciones y bloques. | | | |
| N | Nombre | Descripción | Prioridad |
| B-1 | Consultas de Blockchains | Permite consultar el estado mas reciente de un Blockchain especifico dentro de la aplicación móvil. | Bajo |
| B-2 | Consulta de Bloques | Permite consultar un bloque en específico a partir de su hash. | Bajo |
| B-3 | Consulta de Balances de Direcciones | Consulta el balance de una dirección de criptomonedas. | Media |
| B-4 | Consultas de Transacciones | Consulta la información de una transacción en especifico a partir de su hash. | Media |

5.6. Descripción de las tareas del Módulo de Gestión de Billeteras


| | | | |
|--|--|--|---|
| División | | | Color |
| Módulo de Gestión de Billeteras | | |  |
| Tareas asociadas con la creación y gestión de las billeteras de criptomonedas del usuario. | | | |
| N | Nombre | Descripción | Prioridad |
| W-1 | Desarrollo de clase global compartida KeyService | Creación de la clase KeyService, que permite manejar la información criptográfica de una billetera y realizar el firmado de las transacciones. | Alta |
| W-2 | Creación de Billeteras HD | Implementación de las funciones asociadas a la creación y almacenamiento de una billetera HD y su registro en BlockCypher. | Alta |
| W-3 | Creación de Direcciones Ethereum | Implementación de las funciones asociadas a la creación y almacenamiento de una dirección de Ethereum y Ethereum Testnet. | Alta |
| W-4 | Consulta de Balances de Billeteras HD | Permite consultar el balance, incluyendo número de transacciones, de una billetera HD. | Alta |
| W-5 | Consulta de Balances de Direcciones Ethereum | Permite consultar el balance, incluyendo número de transacciones de una dirección de Ethereum y Ethereum Testnet. | Alta |
| W-6 | Generación de Direcciones de Recibo de Criptomonedas | Genera y muestra, una dirección para el recibo de criptomonedas. | Alta |
| W-7 | Mostrar Frase de | Muestra la frase de recuperación de | Baja |

| | | | |
|-----|--|--|------|
| | Recuperación | una billetera HD dentro de la interfaz de la aplicación móvil. | |
| W-8 | Importar Billeteras HD | Permite al usuario importar billeteras HD a la aplicación móvil a partir de una clave de recuperación. | Baja |
| W-9 | Cambiar preferencias de las Billeteras | Permite al usuario configurar las opciones de unidades de criptomonedas | Baja |


5.7. Descripción de las tareas del Módulo de Generación de Transacciones

| División Módulo de Generación de Transacciones | | | Color |
|---|--|--|---|
| Tareas asociadas con la generación de transacciones de criptomonedas. | | |  |
| N | Nombre | Descripción | Prioridad |
| T-1 | Generación de Transacciones | Implementación de los métodos necesarios para permitir que un usuario genere y modifique la información de una transacción de criptomonedas. | Alta |
| T-2 | Firmado de Transacciones de Billeteras HD | Implementación de las funciones necesarias para poder realizar el firmado de las transacciones de billeteras HD. | Alta |
| T-3 | Firmado de Transacciones de Direcciones Ethereum | Implementación de las funciones necesarias para poder realizar el firmado de las transacciones de direcciones Ethereum y Ethereum Testnet. | Alta |
| T-4 | Envío de Transacciones | Implementación de las funciones necesarias que permiten enviar transacciones firmadas a BlockCypher. | Alta |

5.8. Descripción de las tareas de Implementación del servidor Backend de 2FA

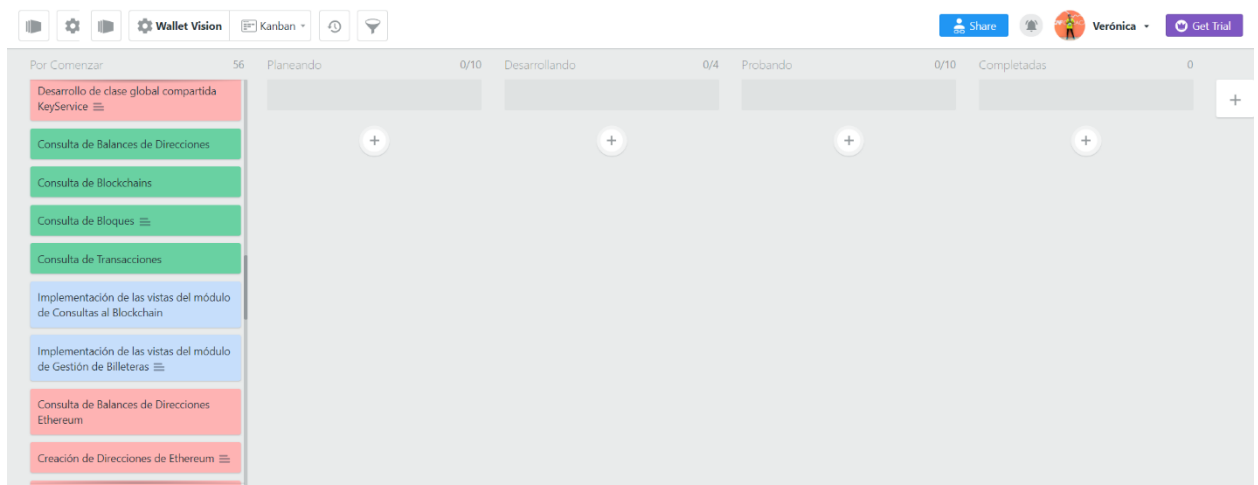
| División Implementación del servidor Backend de 2FA | | | Color  |
|--|---|---|--|
| Tareas asociadas con la creación y gestión de las billeteras de criptomonedas del usuario. | | | |
| N | Nombre | Descripción | Prioridad |
| N-1 | Creación y configuración de servidor 2FA | Creación y configuración inicial de un servidor NodeJS para recibir solicitudes por parte de la aplicación móvil. | Alta |
| N-2 | Implementación de ruta para activar opción de 2FA en el servidor. | Desarrollo de la ruta para poder activar la opción de 2FA en el servidor, incluyendo almacenar el token del usuario y envío de mensaje al correo del usuario con instrucciones. | Alta |
| N-3 | Implementación de ruta para desactivar opción de 2FA en el servidor. | Desarrollo de la ruta que permite a un usuario desactivar la opción de 2FA en el servidor, incluyendo la actualización de los datos del token del usuario. | Media |
| N-4 | Implementación de ruta para verificar identidad del usuario en el servidor. | Desarrollo de la ruta que permite a un usuario ser autenticado siempre y cuando posea la opción de 2FA activada en el servidor e introduzca la clave de un solo uso o OTP. | Alta |
| N-5 | Implementación de clase global TwoFactorService en aplicación móvil. | Generación e implementación de un servicio de Angular que permite a la aplicación móvil manejar las solicitudes al servidor de 2FA. | Alta |

5.9. Descripciones de las tareas de Pruebas y Trabajos finales del aplicativo

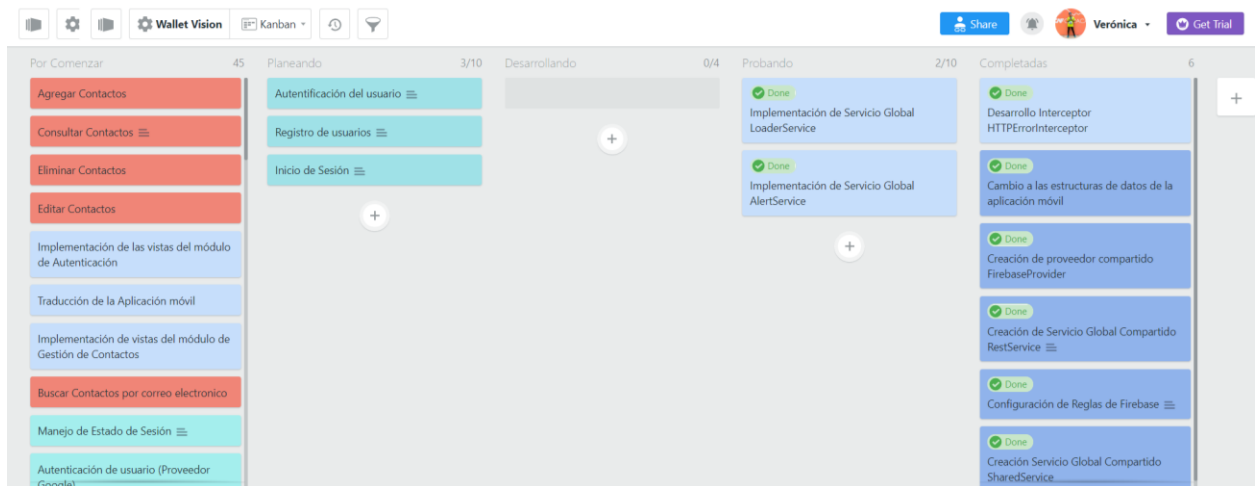
| | | | |
|---|---|---|---|
| División | | | Color |
| Pruebas y Trabajos finales del aplicativo | | |  |
| Tareas asociadas con las pruebas de funcionalidad en dispositivos móviles y requerimientos finales. | | | |
| N | Nombre | Descripción | Prioridad |
| M-1 | Integración con Cordova. | Agregar a la aplicación móvil todas las extensiones que sea requeridas para agregar soporte nativo en dispositivos móviles. | Media |
| M-2 | Pruebas en Dispositivos Móviles | Pruebas del aplicativo móvil y servidor 2AF en dispositivos reales, verificando que cumple con los requerimientos establecidos. | Media |
| M-3 | Generación de archivos de configuración. | Generación de archivos de configuración para facilitar el proceso de instanciación del aplicativo móvil. | Baja |
| S-1 | Proceso de Gestión de Riesgos de Seguridad de la Aplicación | Análisis de los factores vulnerables de la aplicación móvil, incluyendo estructuras de datos, procesos internos del aplicativo, firmado de las transacciones y seguridad general. | Alta |
| S-2 | Comparación entre GulpJS y WebPack | Realizar una comparación entre dos herramientas para automatización de tareas en aplicaciones web NodeJs para realizar instancias del producto. | Alta |

Anexo 6: Tablero Kanban Semana por Semana

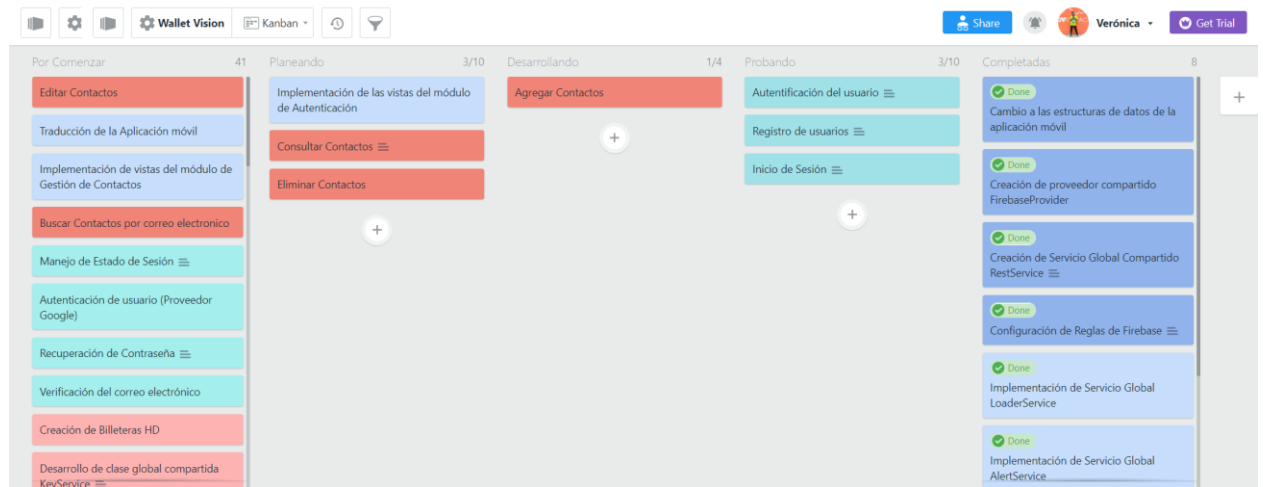
Semana 1



Semana 2



Semana 3



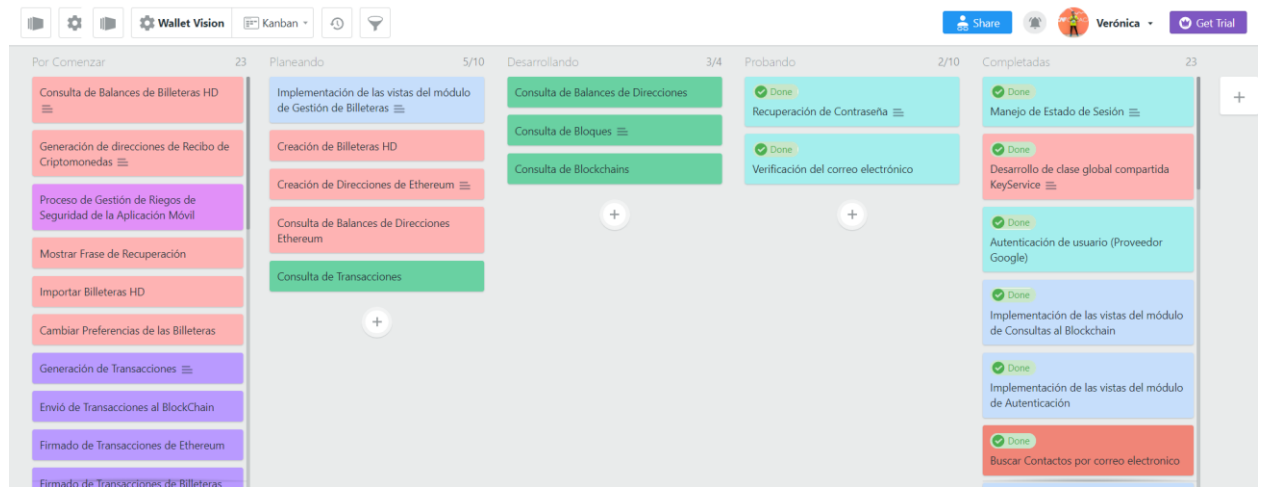
Semana 4



Semana 5



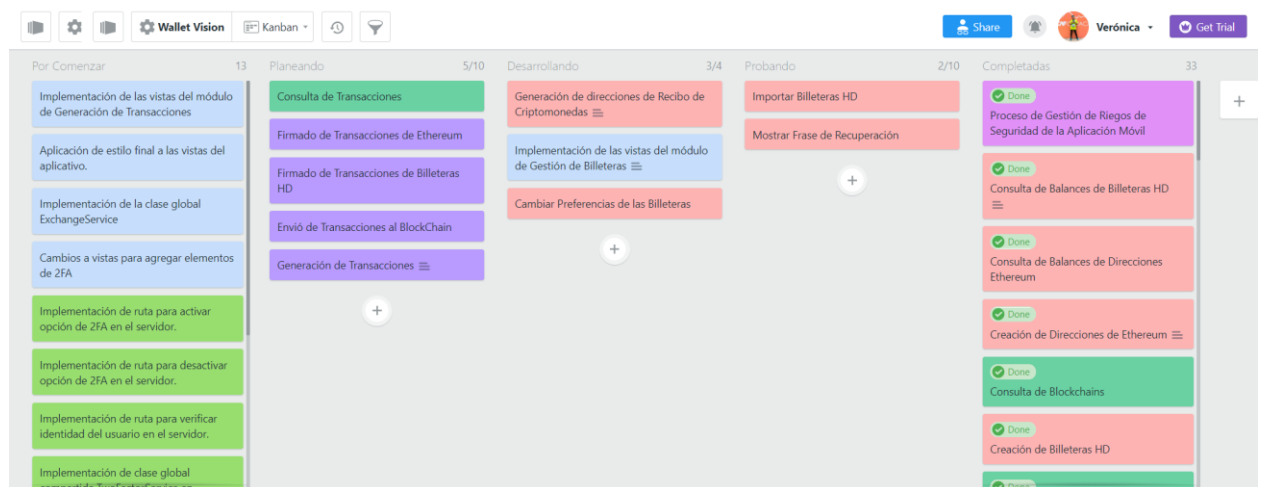
Semana 6



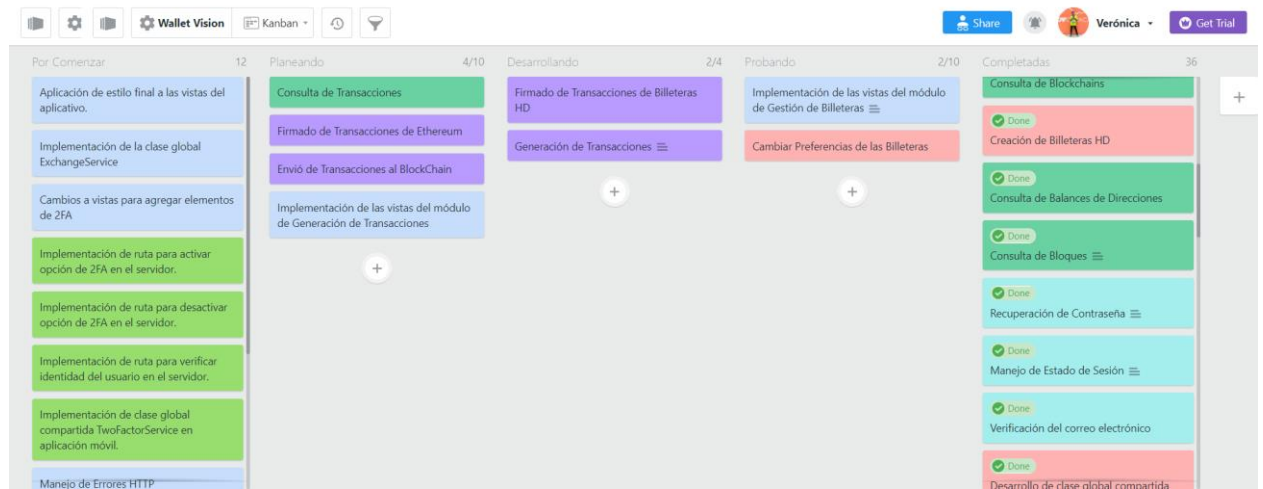
Semana 7



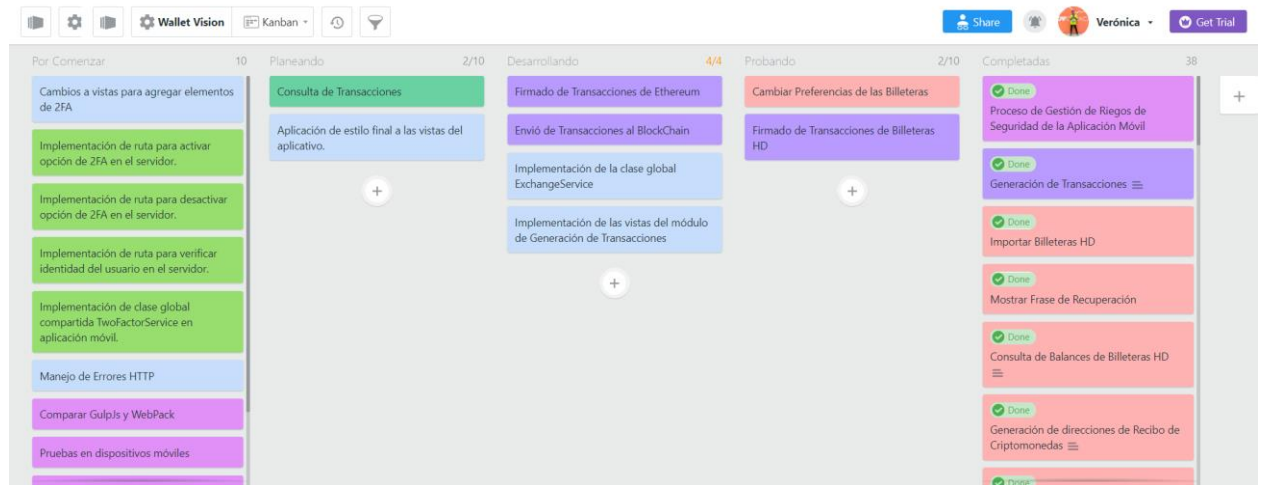
Semana 8



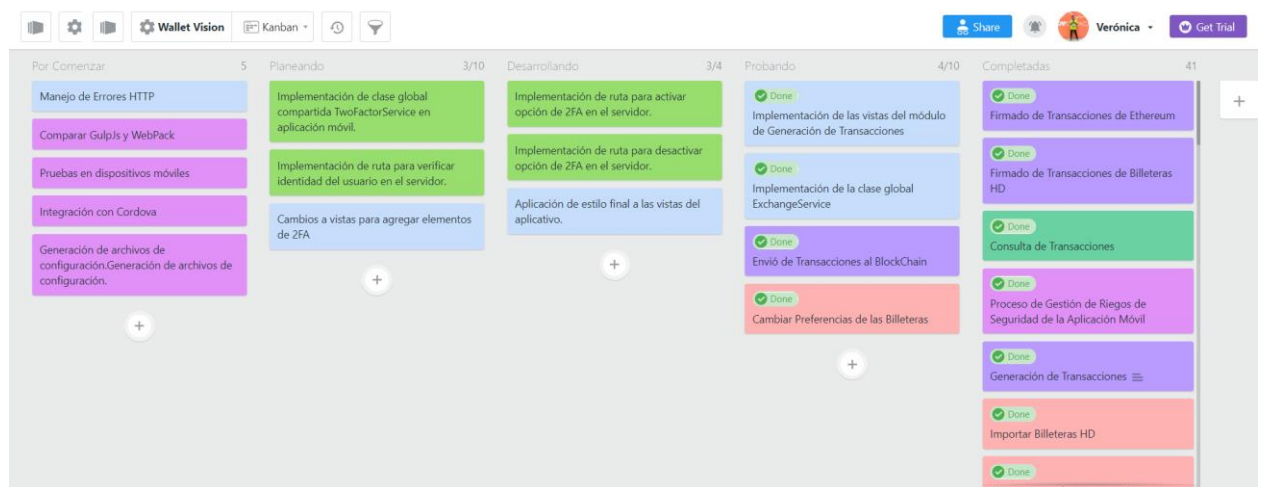
Semana 9



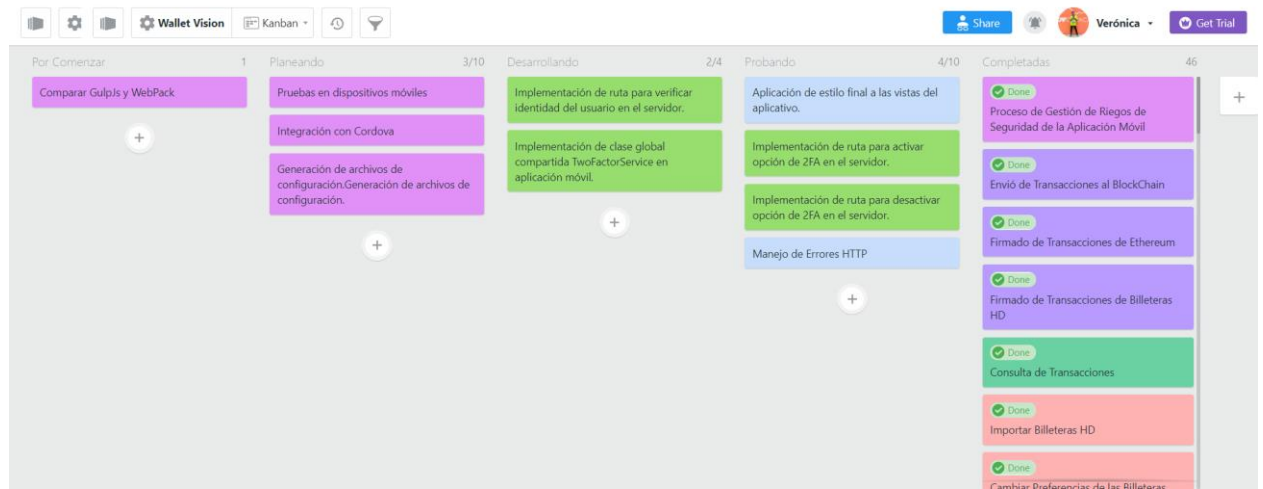
Semana 10



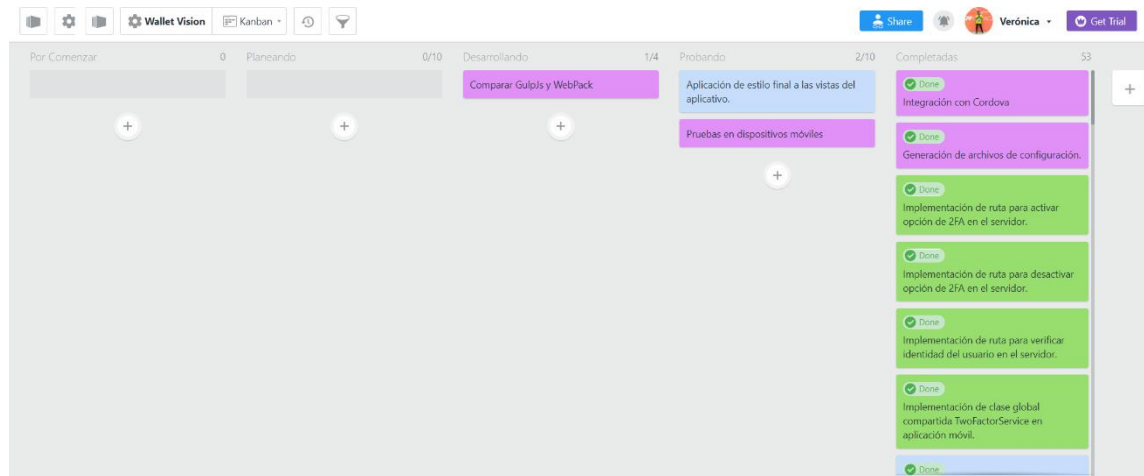
Semana 11



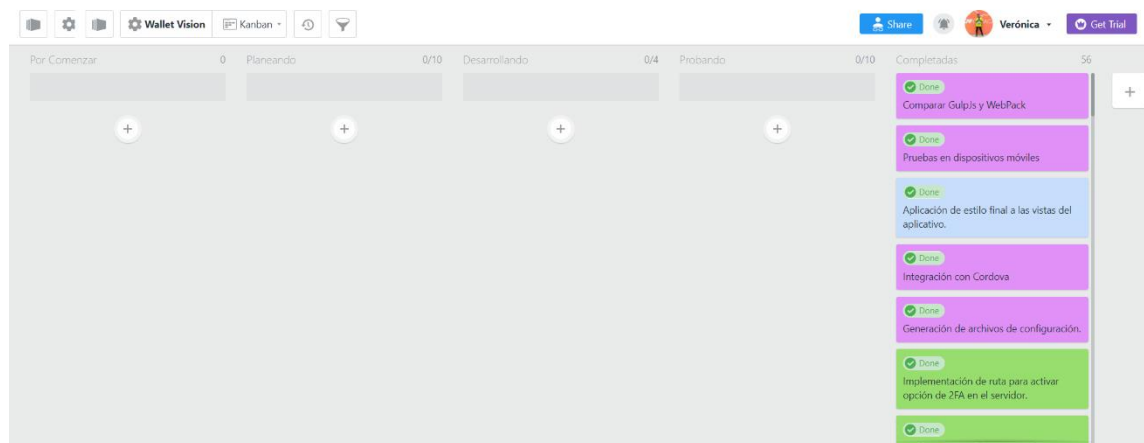
Semana 12



Semana 13



Semana 14

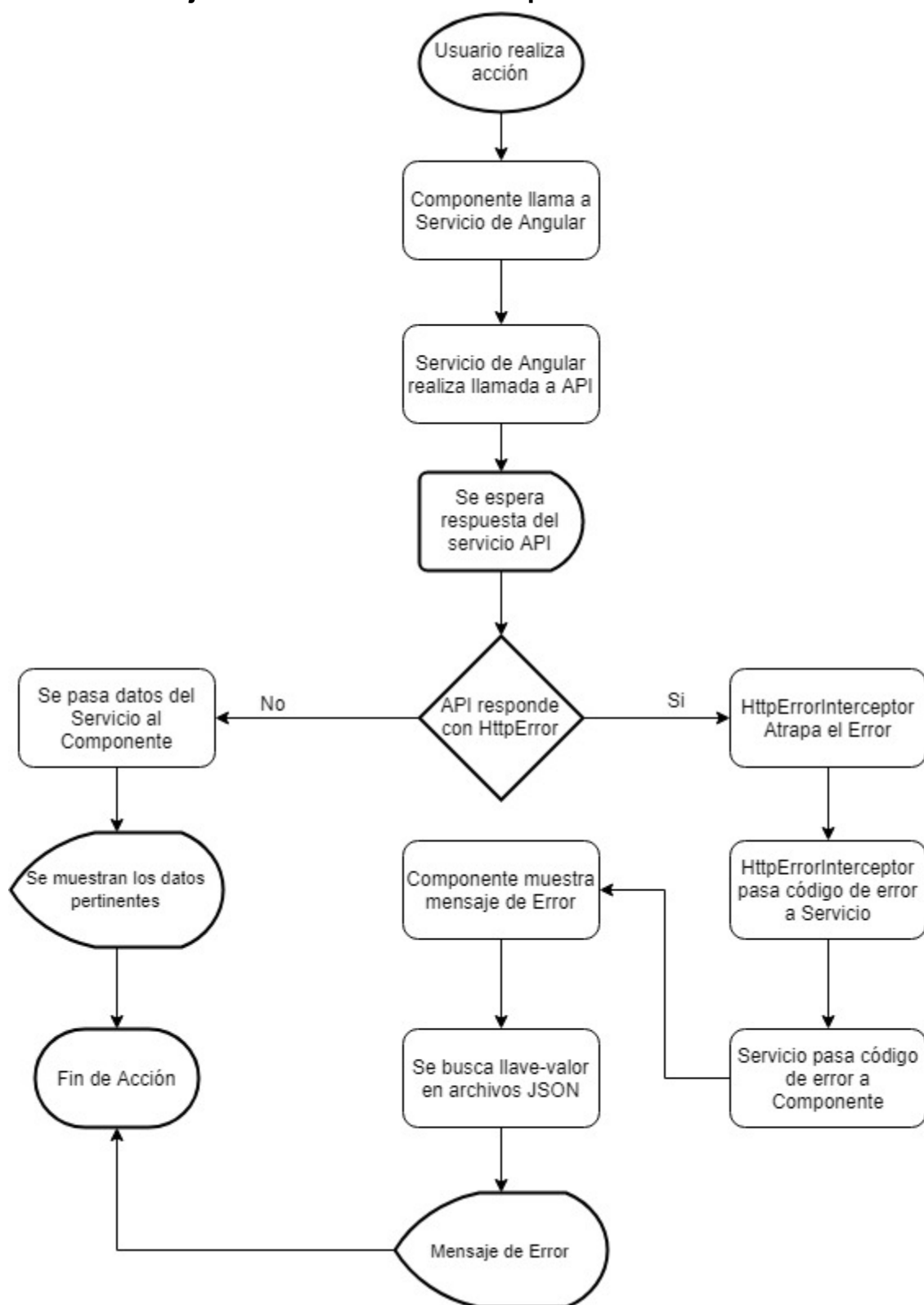


Anexo 7: Clases y Servicios creados durante la adaptación de las vistas del FrontEnd

| Nombre | Descripción | Métodos |
|----------------------------------|---|--|
| HttpError Interceptor | Clase que intercepta las solicitudes y respuestas a las APIS que utiliza la aplicación, con el fin de manejar adecuadamente los códigos de errores que se puedan recibir, sea por solicitudes incorrectas o factores externos (problemas con la conexión) | <ul style="list-style-type: none"> - intercept: función que se encarga de modificar las solicitudes a servicios REST, modificarlas si es necesario, y redirigir las respuestas del servidor que sean de tipo <code>HttpErrorResponse</code> al método <code>handleError</code> - handleError: se encarga de asignar los códigos específicos de cada API a un código interno de la aplicación, con el fin de que este se maneje adecuadamente en otros servicios o componentes. |
| LoaderService | Servicio de Angular inyectable, que permite a través de una serie de métodos, presentar en la interfaz un “loader” o mensaje de carga, con el fin de indicarle al usuario que la aplicación está recuperando datos o realizando una acción en el fondo. Utiliza como base el componente de carga nativo de Ionic 3. | <ul style="list-style-type: none"> - showLoader: muestra un componente de carga pequeña. Se pasa como parámetro el código de un mensaje a mostrar. - showSpinner: muestra un componente de carga pequeña con un “spinner” o icono giratorio. - showFullLoader: muestra un componente de pantalla completa. Se pasa como parámetro el código de un mensaje a mostrar. - dismissLoader: eliminar la instancia del componente que este activa en una vista. |
| AlertService | Servicio que muestra al usuario mensajes de tipo alertas, los cuales | <ul style="list-style-type: none"> - showError: muestra una alerta con un mensaje de error y un botón para |

| | | |
|------------------------|---|--|
| | <p>poseen un mensaje de retroalimentación a una acción que el usuario desea realizar o que indican que un error ha ocurrido. Suelen ser acompañados de cambios en la navegación de la aplicación. Utiliza como base el componente de alertas nativo de Ionic 3.</p> | <p>continuar. Acepta como parámetro un código del error a mostrar.</p> <ul style="list-style-type: none"> - showAlert: muestra una alerta con un título y un mensaje, y un botón para continuar. Acepta como parámetros un código para el título y uno del mensaje a mostrar. - showFullError: equivalente al método showError, pero oculta toda la pantalla. - showFullAlert: equivalente al método showAlert, pero oculta toda la pantalla. |
| ActivityService | <p>Servicio que maneja mostrar notificaciones al usuario relacionada con actividad registrada en la aplicación. Este servicio se encuentra desarrollado para aceptar objetos de tipo Activityby mostrar la información de manera atractiva. Las notificaciones</p> | <ul style="list-style-type: none"> - |

Anexo 8: Manejo de Errores dentro de la aplicación móvil



Anexo 9: Estructuras de Datos de la aplicación móvil: Modelos

| Nombre | Descripción | Atributos | Opcional | Métodos |
|-----------------|---|--|--|---|
| Activity | Modelo de datos que almacena las actividades del usuario | <ul style="list-style-type: none"> - key: String - name: String - date: String - description: Any (Objeto o String) | <ul style="list-style-type: none"> - Si - No - No - Si | <ul style="list-style-type: none"> - constructor - setName - getDate |
| Address | Modelo de datos que almacena los contactos de la libreta de direcciones. | <ul style="list-style-type: none"> - key: String - email: String - alias: String - uid: String - img: String | <ul style="list-style-type: none"> - Si - No - No - Si - Si | <ul style="list-style-type: none"> - constructor - setUID |
| Crypto | Modelo de datos que almacena la información de la criptomoneda de una billetera, y la preferencia del usuario | <ul style="list-style-type: none"> - name: String - value: String - difference: Number - coin: String - units: Objeto: { name: String, exchange: Number } | <ul style="list-style-type: none"> - No - No - No - No - Si | <ul style="list-style-type: none"> - constructor - setUnits - getUnits |
| Keys | Modelo de datos que almacena la información criptográfica | <ul style="list-style-type: none"> - mnemonics: String - seed: String - xpub: String - xpriv: String | <ul style="list-style-type: none"> - No - No - No - No - Si | <ul style="list-style-type: none"> - constructor - setPassphrase |

| | | | | |
|---------------|--|---|--|---|
| | de una billetera | - passphrase: String | | |
| Token | Modelo de datos que almacena las preferencias del usuario respecto a | - activated: boolean - enabled: boolean | - No - No | - Constructor |
| User | Modelo de datos que almacena los datos del usuario que posee la sesión activa dentro de la billetera | - uid: String - email: String - emailVerified: boolean - photoURL: String - displayName: String - token: Token | - No - No - No - Si - Si - Si | - constructor - setPhotoURL - getDisplayName - createToken - getToken |
| Wallet | Modelo de datos que almacena los datos de una billetera perteneciente al usuario. | - key: String - name: String - keys: Keys - crypto: Crypto - address: String | - Si - No - No - No - Si | - constructor - setKeys - changeUnits |

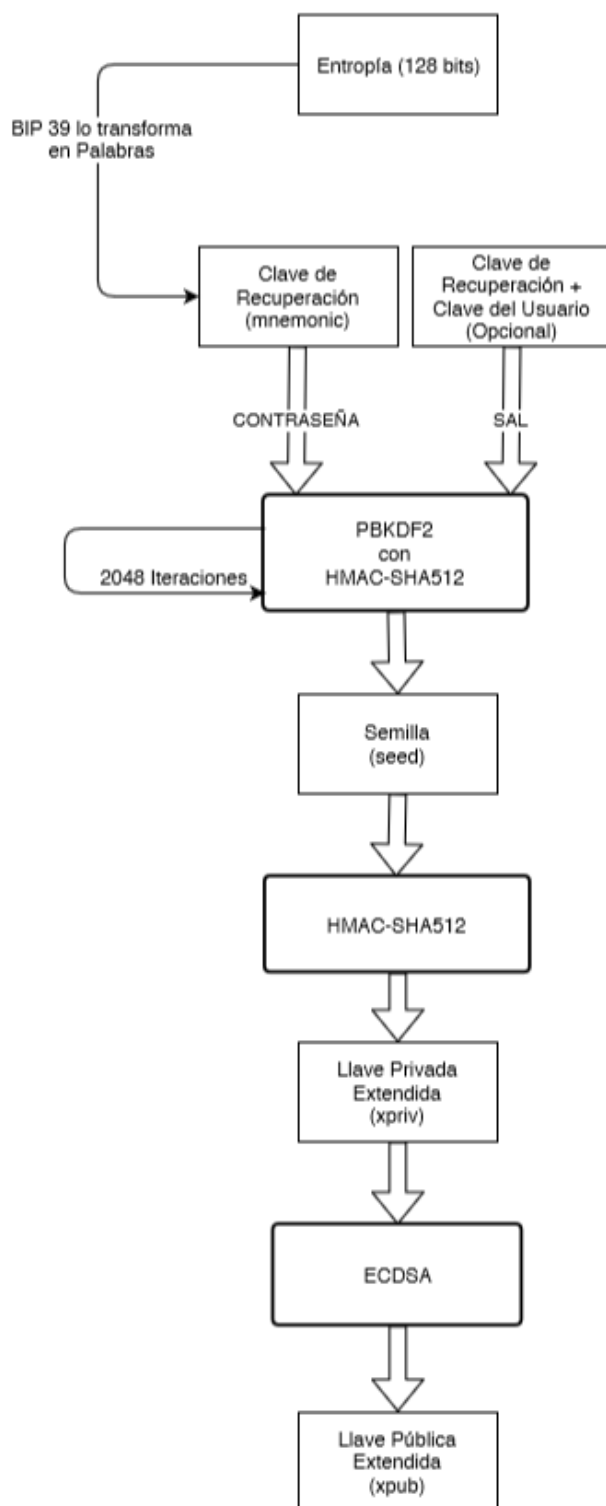
Anexo 10: Estructuras de Datos de la aplicación móvil: Interfaces

| Nombre | Descripción | Recurso | Método |
|--------------------|---|---|--------|
| IAddress | Representa una dirección pública de un Blockchain específico, incluye información sobre su balance y transacciones realizadas. | https://api.blockcypher.com/v1/\$MONEDA/\$CADENA/addr/\$ADDRESS | GET |
| | | https://api.blockcypher.com/v1/\$MONEDA/\$CADENA/addr/\$ADDRESS/full | GET |
| IBalance | Versión alternativa de IAddress con menos atributos, ya que no incluye datos de transacción. | https://api.blockcypher.com/v1/\$MONEDA/\$CADENA/addr/\$ADDRESS/balance | GET |
| IBlock | Objeto que muestra los datos de un bloque particular del Blockchain, tal como el número de transacciones y el volumen en criptomonedas. | https://api.blockcypher.com/v1/\$MONEDA/\$CADENA/blocks/\$HASH | GET |
| | | https://api.blockcypher.com/v1/\$MONEDA/\$CADENA/blocks/\$BLOCK_HEIGHT | GET |
| IBlockchain | Objeto que representa el estado más actualizado de un Blockchain particular. Muestra información general, referente a el tamaño, última actualización y el hash del bloque más nuevo. | https://api.blockcypher.com/v1/\$MONEDA/\$CADENA/ | GET |

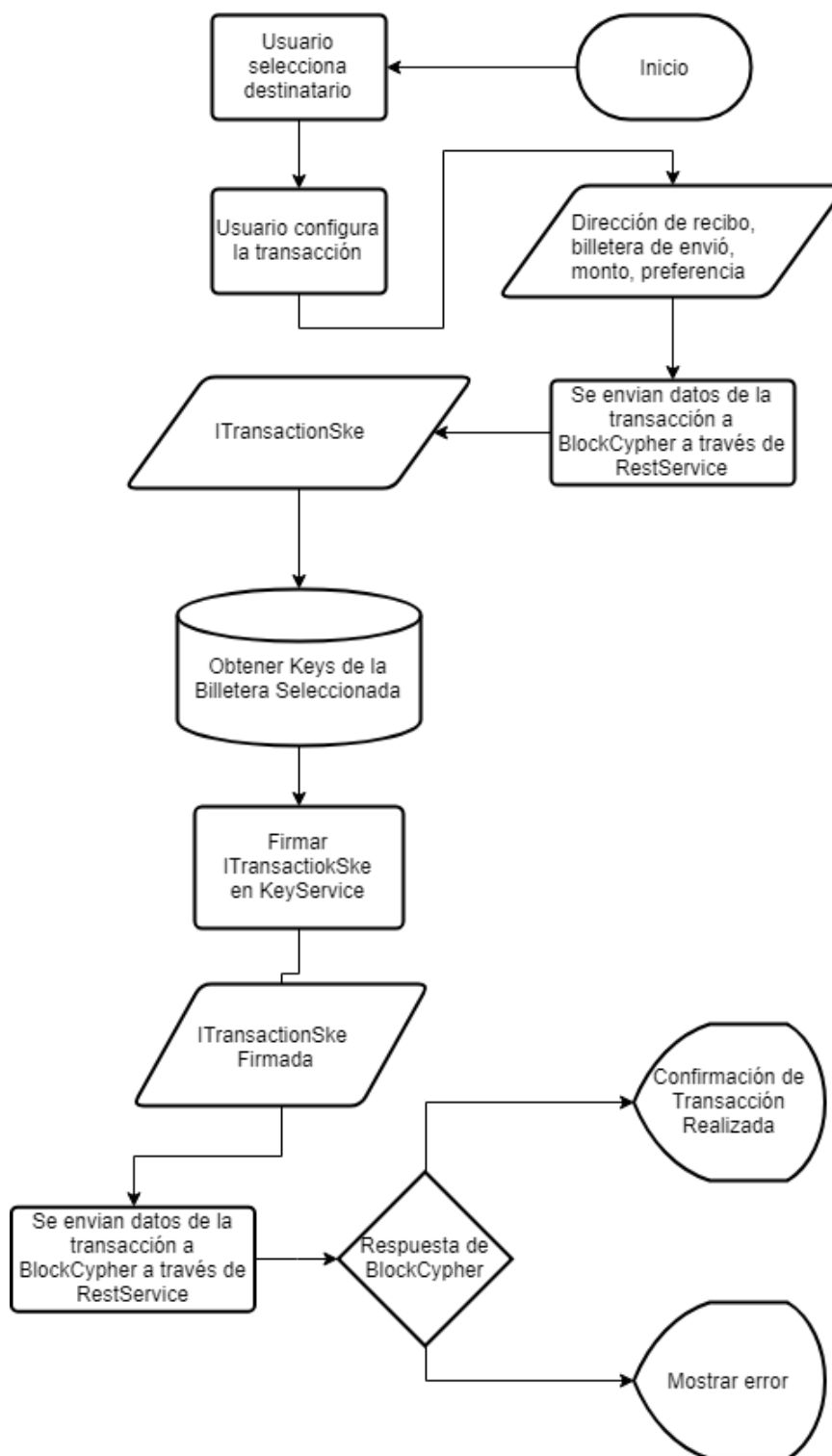
| | | | |
|-------------------|--|---|---------------------------------------|
| IExchange | Objeto utilizado para obtener la tasa de cambio entre criptomonedas y otras monedas. | min- api.cryptocompare.com/d ata/ price?fsys=\$CRYPTOCO INLIST&tsym=\$CURREN CYLIST | GET |
| IHDAddress | Estructura que almacena la información de una dirección específica de una Billetera HD, incluyendo su profundidad. | https://api.blockcypher.co m/v1/\$MONEDA/\$CADE NA/wallets/hd/\$NAME/ad dresses/derive | POST |
| IHDChain | Estructura que almacena la información de las cadenas y sus respectivas direcciones de una Billetera HD. Por defecto se utiliza una sola cadena. | Objeto anidado dentro de IHDWallet. | |
| IHDWallet | Objeto que almacena la información sobre una billetera HD, incluyendo la profundidad actual de la cadena de derivación. | https://api.blockcypher.co m/v1/\$MONEDA/\$CADE NA/wallets /hd https://api.blockcypher.co m/v1/\$MONEDA/\$CADE NA/wallets /hd/\$NAME https://api.blockcypher.co m/v1/\$MONEDA/\$CADE NA/wallets /hd/\$NAME/addresses | GET / POST GET / GET |

| | | | |
|------------------------|---|--|------------------|
| ITInput | Una entrada consumida dentro de una transacción, es decir, información referente a la dirección de la cual originan los fondos. | Objeto típicamente anidado dentro de otras respuestas, particularmente ITransaction y ITransactionSke. | GET |
| ITOutput | Representa la salida creada por una transacción, incluyendo monto en criptomonedas y direcciones de recibo. | Objeto típicamente anidado dentro de otras respuestas, particularmente ITransaction y ITransactionSke. | GET |
| ITransaction | Objeto con la información correspondiente a una transacción, independientemente del estado de confirmación. | https://api.blockcypher.com/v1/\$MONEDA/\$CADENA/txs https://api.blockcypher.com/v1/\$MONEDA/\$CADENA/txs/\$TX_HASH | GET GET |
| ITransactionSke | Estructura utilizada para la creación y envío de nuevas transacciones, incluyendo la data a ser firmada, la cual se encuentra almacenada en un atributo llamado tosign. | https://api.blockcypher.com/v1/\$MONEDA/\$CADENA/txs/new https://api.blockcypher.com/v1/\$MONEDA/\$CADENA/txs/send | POST POST |

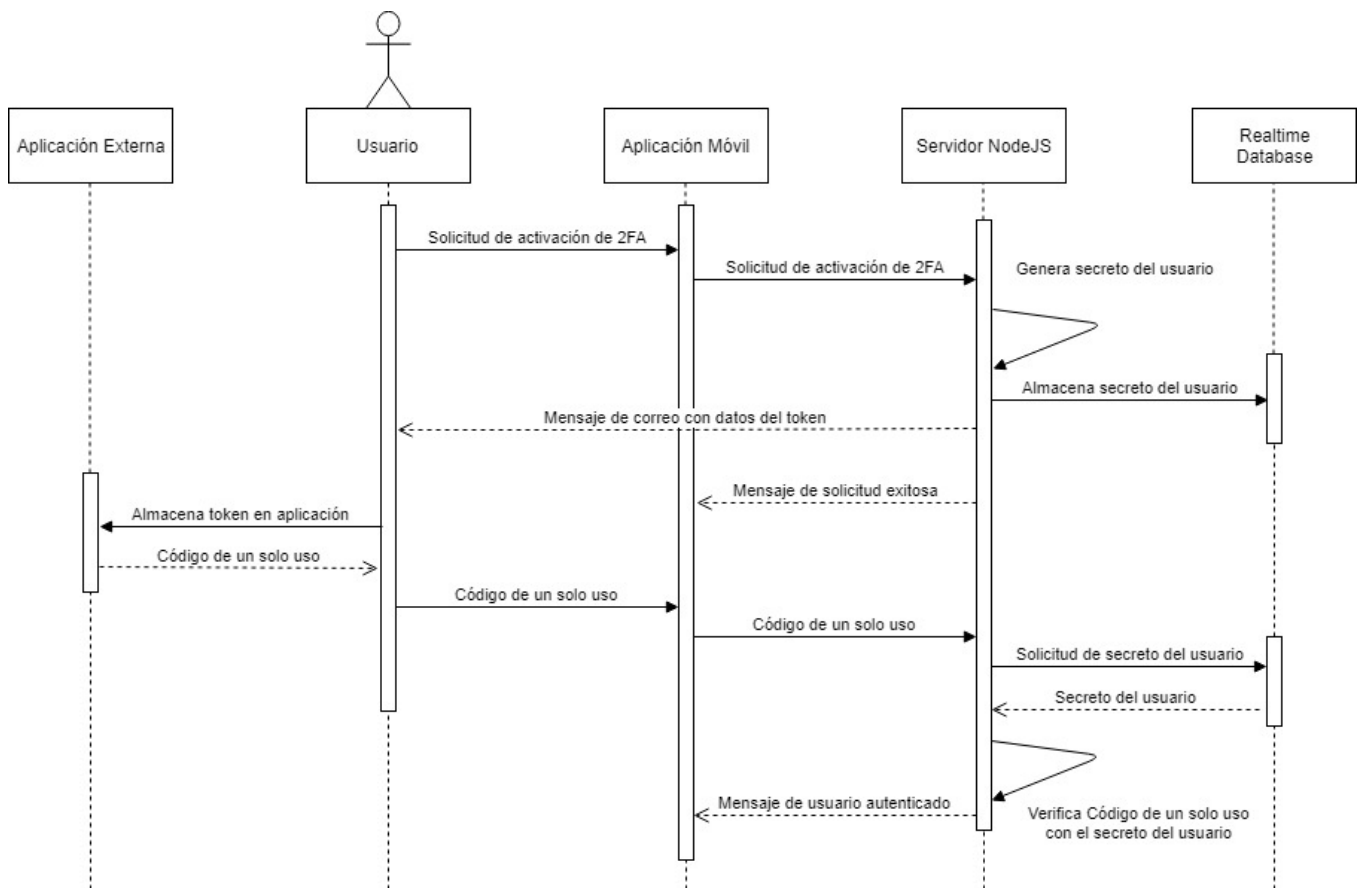
Anexo 11: Generación de la información criptográfica de una Billetera HD



Anexo 12: Proceso de creación y envío de una transacción de criptomonedas.



Anexo 13: Diagrama de Secuencia de la Activación del Segundo Factor de Autenticación



Anexo 14: Pantallas de la Aplicación Móvil



Figura 14.1 Pantalla de Inicio de Sesión

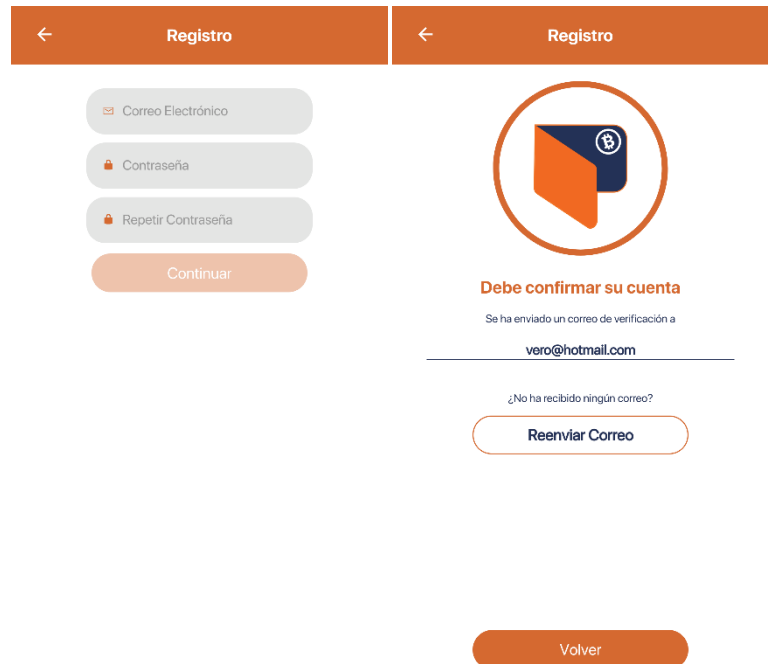


Figura 14.2 Pantalla de Registro

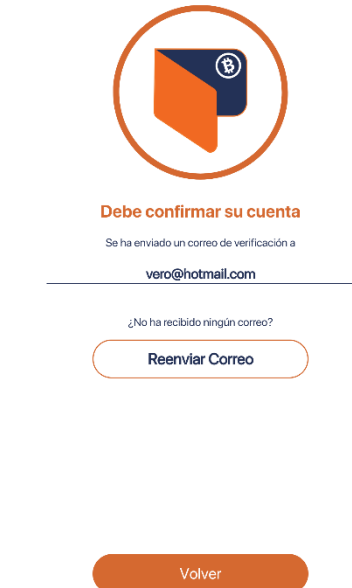


Figura 14.3 Pantalla de Confirmar Correo

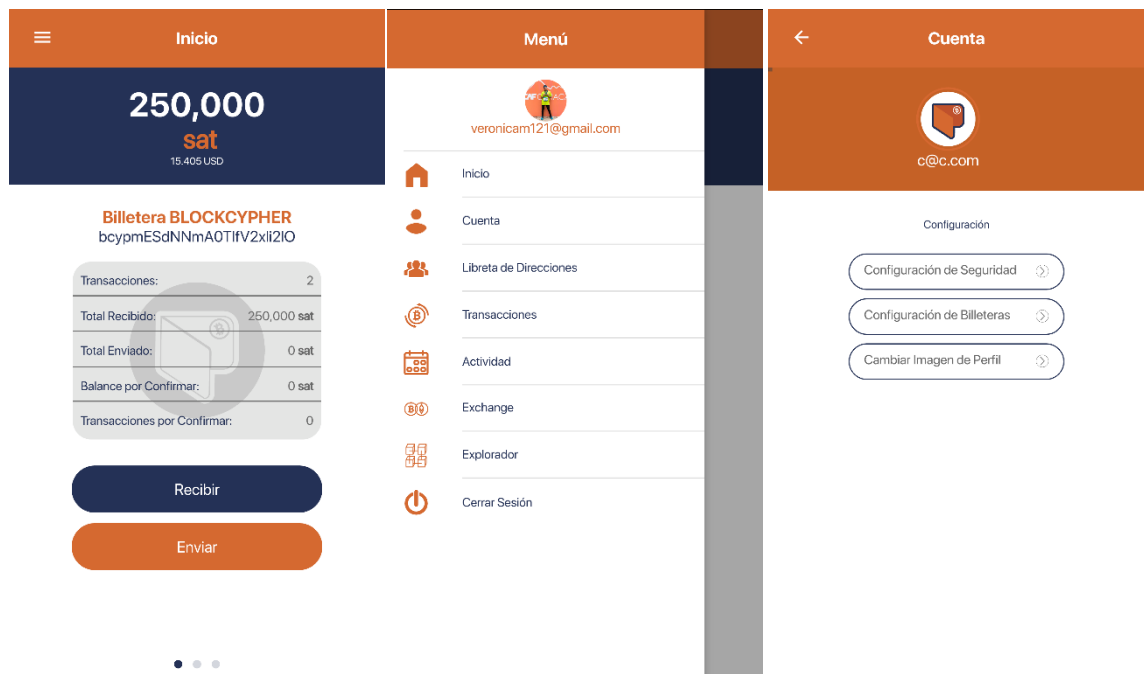
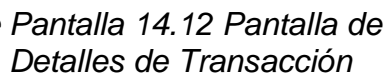
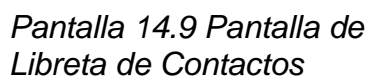


Figura 14.4 Pantalla de Inicio

Figura 14.5 Pantalla de Menú Lateral

Pantalla 14.6 Pantalla de Configuración de la Cuenta





Pantalla 14.19 Pantalla para mostrar Frase de Recuperación

Anexo 15: Reglas de Seguridad establecidas en Firebase Realtime Database

```

rules = {
  "wallet": {
    ".indexOn": "user",
    ".write": "auth != null",
    ".read": "auth != null"
  },
  "userList": {
    ".indexOn": "email",
    ".write": "auth != null",
    ".read": "auth != null"
  },
  "user": {
    "$user_id": {
      ".indexOn": "userEmail",
      ".write": "auth != null",
      "userEmail": {
        ".read": "auth != null",
        ".validate": "newData.isString() && newData.val().contains('@)"
      },
      "img": {
        ".read": "auth != null",
        ".validate": "newData.isString()"
      },
      "currency": {
        ".read": "auth != null && auth.uid == $user_id",
        ".validate": "newData.isString() && newData.val().length === 3"
      },
      "token": {
        ".read": "auth != null"
      },
      "wallet": {
        ".read": "auth != null && auth.uid == $user_id",
        "$wallet_id": {
          "name": {
            ".validate": "!data.exists() && newData.isString() && newData.val().length >= 24"
          },
          "keys": {
            ".validate": "!data.exists() && newData.hasChildren(['mnemonics', 'seed', 'xpub', 'x
          ],
          "crypto": {
            "coin": {
              ".validate": "newData.isString() && newData.val().length === 3"
            },
            "difference": {
              ".validate": "newData.isNumber()"
            },
            "name": {
              ".validate": "newData.isString()"
            },
            "value": {
              ".validate": "newData.isString()"
            },
            "units": {
              ".validate": "newData.hasChildren(['exchange', 'name']) && newData.child('exchange
            },
          },
        },
      },
    },
  },
  "addressBook": {
    ".indexOn": "email",
    ".read": "auth != null && auth.uid == $user_id",
    "$addressBook_id": {
      ".validate": "newData.hasChildren(['alias', 'email', 'uid', 'img'])",
      "alias": {
        ".validate": "newData.isString()",
      },
      "img": {
        ".validate": "newData.isString()",
      },
      "email": {
        ".validate": "newData.isString() && newData.val().contains('@)"
      },
      "uid": {
        ".validate": "newData.isString() && newData.val() != $user_id",
      },
    },
  },
}

```