UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DISEÑO E IMPLEMENTACIÓN DE APLICACIÓN MÓVIL DE LA PLATAFORMA CRYPTOMARKET

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

JAIME IGNACIO CAPPONI ZERENÉ

PROFESORA GUÍA:
JOCELYN SIMMONDS WAGEMANN

MIEMBROS DE LA COMISIÓN:
ALEJANDRO HEVIA ANGULO
PATRICIO INOSTROZA FAJARDIN

SANTIAGO DE CHILE
2018

Resumen

CryptoMarket es una empresa chilena que nace con el objetivo de acercar a las personas al mundo de las criptomonedas de manera sencilla, dándole la capacidad a sus usuarios de transar en diversas criptomonedas en su moneda local.

Actualmente en CryptoMarket, cerca del 40% de sus usuarios de un total de más de 40.000, acceden a la plataforma web por su dispositivo móvil. Esta plataforma si bien tiene diversas adaptaciones para utilizarse mediante un navegador móvil, no está totalmente adaptada para usarse desde dispositivos móviles.

Al inicio de esta memoria, CryptoMarket contaba con una aplicación móvil en que solo se podían ver los precios de sus mercados y los gráficos de precios correspondientes a cada mercado.

El objetivo general del trabajo realizado fue desarrollar una solución móvil para la empresa CryptoMarket, en la cual los usuarios pueden hacer diversas acciones de *trading* de criptomonedas descargando la aplicación. Esta aplicación debe ser capaz de entregar una buena experiencia para los usuarios, dándoles la comodidad que esperan al tener una aplicación móvil, pero también la simplicidad y seguridad que acostumbran tener en la plataforma web.

Se logró desarrollar la aplicación móvil, que a día de hoy está lanzada en la App Store de iOS de diversos países. La aplicación posee inicio de sesión, evitando que el usuario tenga que ingresar su correo electrónico y contraseña de CryptoMarket cada vez que quiera usar la plataforma. El usuario puede realizar todas las operaciones básicas de *trading* de criptomonedas como lo son; ver los precios de las criptomonedas en los diferentes mercados, vender y comprar criptomonedas, ver gráficos y datos, historiales de transacciones, cancelar órdenes, etc. Todas estas funcionalidades se lograron conectándose a un servidor ya existente en la empresa mediante *API* y *WebSockets*.

Se hizo un estudio de usabilidad en donde se encontraron algunos problemas, como la dificultad para encontrar las órdenes de mercado abiertas. Actualmente se está viendo cómo resolver estos problemas en una futura actualización de la aplicación.

Desde que se lanzó la aplicación ya se cuenta con cerca de 4.000 usuarios los cuales están agradecidos por entregarles esta solución, como también otorgan *feedback* para las futuras funcionalidades y pequeños errores que encuentran.

Agradecimientos

Agradezco a mi familia ya que sin ellos no estaría donde estoy ni sería quien soy.
Agradezco a mis amigos de Los Ángeles que siempre han estado conmigo.
Agradezco a mis amigos de la Universidad de Chile que gracias a ellos el camino ha sido menos difícil de lo que realmente pudo haber sido.
Agradezco a mi profesora guía Jocelyn Simmonds por darme una oportunidad de ser su alumno y ayudarme con esta memoria.
Agradezco a CryptoMarket por darme la oportunidad de realizar esta memoria con ellos.
Agradezco a mis amigos de CryptoMarket que sin ellos sería mucho más difícil seguir.

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

Capítulo 1

Introducción

1 INTRODUCCIÓN

En este capítulo se presenta el contexto en el que se enmarca el trabajo de memoria. Además, se describe el problema general, la solución propuesta a dicho problema, una breve descripción de los resultados, el alcance del trabajo, y los objetivos tanto generales como específicos.

1.1 Contexto

Con el paso del tiempo cada vez hay más interés de parte de las personas por el mundo de las criptomonedas [7]. Las criptomonedas son una forma de intercambiar valor distinto al físico [8], basándose en una tecnología llamada "*blockchain*" o cadena de bloques [1]. Este sistema busca garantizar la seguridad e integridad de las transacciones y cuentas mediante el uso de métodos de criptografía y ayuda colectiva de otros computadores, denominados mineros, que usan su poder computacional para validar las transacciones.

CryptoMarket (http://www.cryptomkt.com) es una empresa chilena que nace con el objetivo de acercar a las personas al mundo de las criptomonedas de manera sencilla, dándole la capacidad a sus usuarios de transar en diversas criptomonedas en su moneda local. Al día de hoy, CryptoMarket está en Chile, Argentina, España, Brasil y México.

El modelo de negocio de CryptoMarket está basado en cobrar una comisión de 0.68% en cada transacción de compra o venta que realiza un usuario. Por ejemplo, si un usuario compra 100 Bitcoin, CryptoMarket se queda con 0.68% de estas Bitcoin. Si un usuario vende, CryptoMarket se queda con el mismo porcentaje en la moneda en que se haga la venta (pesos argentinos, reales, euros, etc). Para CryptoMarket es importante entregar el mejor servicio posible para atraer una mayor cantidad de usuarios a su plataforma, así como también entregar nuevas funcionalidades y herramientas para diferenciarse de su competencia.

Al tener sus usuarios la posibilidad de poder comprar criptomonedas con su divisa local, se puede lograr una transferencia de dinero entre dos personas de diferentes países. La primera persona compra cierto monto de dinero en criptomonedas, y luego las transfiere desde su billetera hacia la billetera de la segunda persona. Finalmente la segunda persona vende esas criptomonedas por su moneda local y posteriormente puede recibir el dinero en su cuenta bancaria de su país. Con esto se tiene un modo de transferir valor entre personas con una comisión mucho menor, en relación a otros

métodos tradicionales de envío de dinero. Logrando poder transferir dinero entre personas de diferentes países de manera rápida, segura y con una muy baja comisión utilizando las criptomonedas como método de envío.

1.2 Problema

Actualmente en CryptoMarket, cerca del 40% de sus usuarios de un total de más de 40.000, acceden a la plataforma web por su dispositivo móvil. Esta plataforma si bien tiene diversas adaptaciones para utilizarse mediante un navegador móvil, no está totalmente enfocada en su uso desde estos dispositivos.

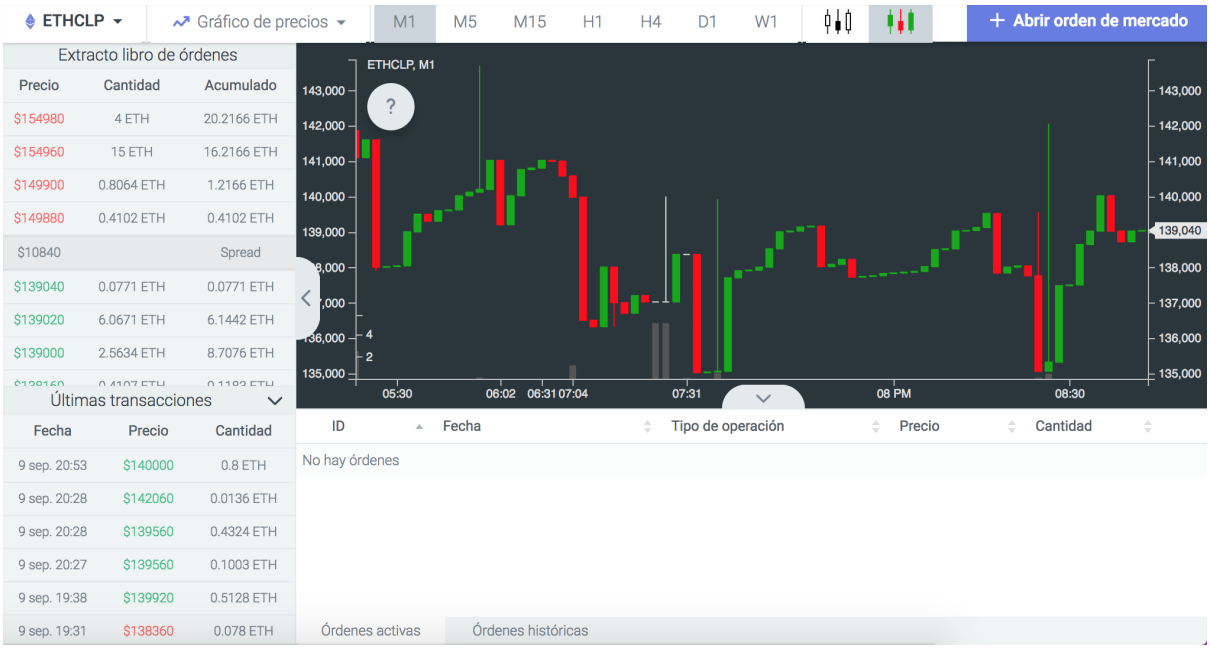


figura 1: vista de mercado en la plataforma web

figura 2: plataforma web visualizada desde un dispositivo móvil

Como se puede apreciar en la *figura 1* y la *figura 2*, la vista móvil de la plataforma CryptoMarket presenta problemas de visualización de la información. A su vez, durante mucho tiempo los usuarios de CryptoMarket han solicitado constantemente una aplicación móvil con la que puedan acceder con mayor comodidad a la plataforma y recibir notificaciones varias. CryptoMarket posee una aplicación móvil, pero solo muestra los precios de las diversas criptomonedas, por lo que no se considera una plataforma móvil de CryptoMarket.

CryptoMarket busca poder aumentar la cantidad de usuarios que posee diferenciándose de su competencia y entregando más y mejores herramientas a sus usuarios. Además, CryptoMarket busca entregar un servicio que provea la mejor experiencia de usuario posible. Por esto que nace la idea de desarrollar una solución móvil nativa, la cual logre adaptar muchas capacidades de la página web de una manera fácil, cómoda, familiar, rápida y segura para los usuarios. Con esto CryptoMarket espera lograr diferenciarse de la competencia positivamente, así como entregar una mejor herramienta para los usuarios móviles.

Al tener una versión móvil se favorece con una serie de ventajas por sobre una versión web responsiva, como por ejemplo un sistema de notificaciones o alertas, comodidad y familiaridad en el uso de la plataforma, rapidez al acceso de información de balance y transacciones de los usuarios, entre otras.

1.3 Solución escogida

Como solución escogida se propone desarrollar una aplicación nativa para iOS en el lenguaje de programación *Swift* de Apple. Si bién existen *frameworks* como *Ionic* [14] o *Xamarin* [15], que permiten desarrollar solo una aplicación y luego es compilada para los dos más utilizados sistemas operativos móviles (iOS y Android) CryptoMarket, por seguridad, prefiere mantener el mayor control posible de todo el flujo de la información. Además, estos *frameworks* disponibles a día de hoy no entregan la robustez y performance de una aplicación nativa [9].

La aplicación debe ser capaz de entregar una buena experiencia para los usuarios, dándoles la comodidad que esperan al tener una aplicación móvil, pero también la simplicidad y seguridad que acostumbran tener en la plataforma web.

Los usuarios deben ser capaces de ver los mercados que CryptoMarket ofrece, capacidad de iniciar sesión, abrir órdenes de compra y órdenes de venta, ver el estado de estas órdenes y a su vez poder cancelarlas, recibir notificaciones, entre otros. Todo esto se logra comunicándose con los servidores ya existentes de CryptoMarket mediante comunicación con una *API Rest* [16] cuando es necesario, pero también mediante *WebSockets* para disminuir la carga al servidor y a su vez obtener datos de manera mucho más rápida y en tiempo real .

Cabe mencionar que la memoria se desarrolló en iOS por experiencia de quien la realiza, pero también se desarrolló la versión de Android posteriormente siguiendo las mismas líneas de diseño y funcionalidades de la versión de iOS.

1.4 Objetivos

Los objetivos de este trabajo son los que se describen a continuación.

1.4.1 Objetivo general

Entregar una solución que facilite la experiencia de usuarios móviles, mediante una aplicación nativa que mejora el acceso desde dispositivos móviles a los servicios de *trading* de criptomonedas ofrecidos por CryptoMarket.

1.4.2 Objetivos específicos

- Diseñar el acceso a la plataforma de CryptoMarket mediante una aplicación móvil que permita al usuario realizar las diversas acciones de la plataforma: iniciar sesión, ver los diversos mercados de criptomonedas en tiempo real, realizar ordenes de compra y venta, etc. Además la aplicación debe permitir la definición de diversas alertas que ofrecen información oportuna al usuario independiente de la ubicación.

- Desarrollar dicha aplicación móvil para iOS de manera nativa, soportando diversos tamaños de pantalla y tipos de dispositivos.

- Validar la solución desarrollada mediante un estudio de usabilidad de la aplicación móvil, con usuarios conocedores de criptomonedas y que utilizan la plataforma iOS.

1.5 Alcances

El trabajo realizado corresponde a:

- El diseño y desarrollo de una aplicación móvil que permita el acceso a los usuarios CryptoMarket desde su celular de manera sencilla, segura y familiar.

- Hacer una validación mediante un estudio de usabilidad.

El trabajo no corresponderá a realizar desarrollos del lado del servidor pues ya están creadas todas las vías de comunicación necesarias (*API Rest* y *WebSockets*) para el correcto funcionamiento de una aplicación móvil de la plataforma CryptoMarket.

En cuanto al diseño gráfico se contó con José Laguna, diseñador con experiencia en desarrollo móvil y en conjunto con él se fue decidiendo la parte gráfica y de usabilidad de la versión móvil. José era quien generaba los recursos visuales (imágenes, colores, diagramación, entre otros) para la aplicación y los mockups que posteriormente fueron implementadas en la aplicación.

El desarrollo de la aplicación móvil fue hecho en el IDE de Apple *XCode 10* [17] con el lenguaje de programación *Swift* 4. Realizada para los sistemas operativos iOS 11 en adelante, por lo que la aplicación tiene soporte para diferentes tamaños de pantalla y celulares iPhone desde el iPhone 5S (año 2013) en adelante. También cabe decir que se utilizaron diferentes *frameworks* como Alamofire para las consultas HTTP [18], *Realm* para la base de datos móvil [19], Socket.IO para la comunicación por *sockets* [20], y Charts para la realización de gráficos de mercado [21].

Capítulo 2

Marco Teórico

2 MARCO TEÓRICO

2.1 Cadena de bloques y Bitcoin

La tecnología denominada cadena de bloques (*blockchain*) nace de una propuesta realizada por Satoshi Nakamoto quien propuso una forma de intercambiar valor de manera electrónica mediante un sistema *peer-to-peer* en donde no se requiere el paso a través de una institución financiera. Este nuevo sistema no requiere confianza en este tipo de instituciones (como los bancos). A esta tecnología la llamó Bitcoin [1]. Esta criptomoneda se basa en la utilización de la *blockchain* utilizada para ir registrando todas las transacciones entre las billeteras de Bitcoin. A medida que se van haciendo transacciones desde una billetera a otra, dichas transacciones se van registrando en un bloque de información. Cuando el bloque se completa comienza el proceso de validación del bloque para guardarlo y comenzar con uno nuevo que está enlazado al anterior.

La seguridad de este sistema se basa en que, si algún *hacker* pretende cambiar la información en los bloques, debe de ser capaz de vulnerar a más de la mitad de los computadores que participan de la validación de la *blockchain* (la información correcta es la que existe en la mayoría de los computadores). Esta acción es muy difícil computacionalmente de lograr [1].

Luego de varios años, luego de hacerse más famosa la criptomoneda, comenzaron a aparecer las diversas falencias de este sistema. Por ejemplo, la muy baja cantidad de transacciones por segundo y el aumento de complejidad computacional para validar los bloques, logrando un elevado precio por transacción (en este momento entre 1 USD a 30 USD) y un alto tiempo de espera para validar una transacción (que pueden llegar a ser varias horas).

Estas grandes falencias no logran posicionar a la Bitcoin como la mejor criptomoneda para intercambiar valor, ni como un mejor método de pago en el futuro.

2.2 Otras criptomonedas

Es por esto que nacen otros tipos de sistemas con sus propias criptomonedas basadas en esta tecnología *blockchain* pero que pretenden ser más ambiciosas y tener otro tipo de utilidades. Es el caso de la red Ethereum que mejora en gran medida con respecto a Bitcoin los costos y tiempos de transacciones. Ethereum también utiliza su *blockchain* para implementar los denominados *smart contracts* los cuales son programas que se ejecutan en la *blockchain* en vez de un servidor o computador (obteniendo beneficios como la integridad de la base de datos, pero con un costo monetario mayor a un servidor tradicional). A modo de ejemplo, la empresa Consensys está desarrollando una aplicación denominada uPort en donde utilizan la *blockchain* de Ethereum y los *smart contracts* para guardar toda la información de identidad de las personas de manera muy segura (en vez de en papel, o computadores centralizados como se hace actualmente, se guardarán en la *blockchain* de Ethereum) [2].

Si bien la red Ethereum fue un gran avance con respecto a Bitcoin, aún sus tiempos de transacción (varios minutos) y costos (0.1 USD o más) son algo elevados y siempre se está en busca de un mejor sistema.

Posteriormente nacen las denominadas criptomonedas de tercera generación. Una de ellas el Lumen, de la *blockchain* Stellar, quien logra costos por transacción muy bajos (100.000 transacciones en Stellar por 1 USD) pero según ellos necesarios para evitar ataques *DDoS* a la red. La red Stellar también logra una muy alta velocidad de transacción la cual está entre 2 a 5 segundos [3] (mucho más rápido que los minutos y horas de Bitcoin y Ethereum) .

Al día de hoy no se sabe cuál será la moneda o tecnología elegida por la sociedad para usar a futuro para el intercambio de valor entre las personas. Se está en búsqueda de las criptomonedas de cuarta generación donde buscan que los costos por transacción sean cero y las transferencias instantáneas, manteniendo la seguridad del sistema, entre otras mejoras.

2.3 Aplicaciones de otros exchanges chilenos

Por lo mencionado anteriormente, nacen los denominados *exchanges*. Los cuales dan la capacidad a sus usuarios de poder transar en sus mercados las diversas criptomonedas, así como también transferir éstas a cualquier parte del mundo con una muy baja comisión por envío.

Actualmente en Chile existen tres empresas que dan acceso a las criptomonedas a sus usuarios: Buda, CryptoMarket y Orionx. CryptoMarket apunta a ser la primera que ofrezca una experiencia completa en una aplicación móvil.

En cuanto a aplicaciones móviles, Buda posee una aplicación móvil [10] en la cual solo se puede ver el portafolio. Ahi se muestran todas las criptomonedas que se poseen, y su valor aproximado en pesos de un usuario (figura 3).

figura 3: captura de aplicación Buda

CryptoMarket posee una aplicación móvil (figura 4) en la cual solo se puede ver información de sus mercados y noticias, aplicación que se cambiará y se hará desde cero en esta memoria.

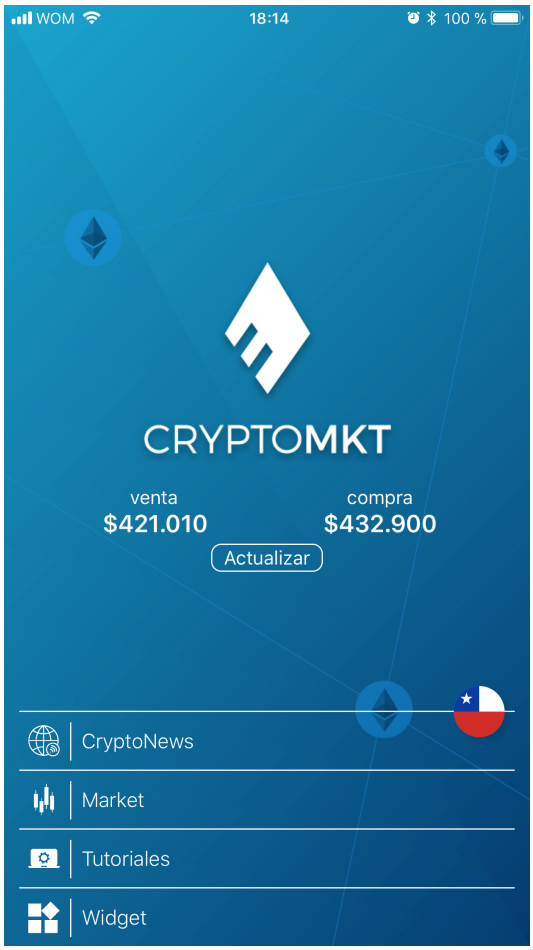


figura 4: captura de la aplicación antigua de CryptoMarket

Orionx [10] no posee aplicación móvil aún.

2.4 Aplicaciones de otros exchanges extranjeros

En otros países hay *exchanges* más grandes que sí poseen aplicaciones móviles. Es el caso de Binance y KuCoin. KuCoin [12] en este momento tiene una aplicación móvil sólo disponible en Android que simplemente es un *webview* (mostrar una página web dentro de una sección de una aplicación) de su plataforma web pero que está repleta de publicidad invasiva. Mientras que Binance [5] posee una aplicación completa pero

compleja de usar por la alta concentración de información en pantallas pequeñas (figura 5).

figura 5: captura de la aplicación de Binance

2.5 Estado del Arte en desarrollo móvil

En cuanto a desarrollo móvil la tendencia que se está dando en los últimos años es a utilizar frameworks de desarrollo multiplataforma como *Xamarin*, *React* e *Ionic* que permiten desarrollar una sola aplicación y compilarla para ambos sistemas operativos iOS y Android. Sin embargo, CryptoMarket piensa que lo mejor es desarrollar la aplicación nativamente con el IDE *Xcode* que Apple ofrece ya que esto permite mucha más facilidad en el desarrollo, mejor rendimiento (por ejemplo en algunas aplicaciones desarrolladas con *frameworks* multiplataforma hacer *scrolling* es menos fluido que en las aplicaciones nativas*)*, mayor acceso a librerías de terceros e información, sacar mejor provecho las capacidades del sistema operativo y de los dispositivos, etc.

En cuanto a la base de datos móvil, existen 4 formas nativas de guardar datos en una aplicación de iOS: 1) *UserDefaults* usado para guardar pequeñas cantidades de información; 2) *Keychain* para guardar datos sensibles de manera encriptada, como las contraseñas o secret keys; 3) Codificación y decodificación de información en archivos de texto internos; 4) *Core Data* la cual es la base de datos más poderosa y compleja que ofrece Apple para iOS.
Existen también opciones de terceros como *Realm* la cual es una base de datos muy rápida y poderosa que al hacer una consulta entrega directamente objetos, lo cual es muy cómodo de usar, ya que se pueden usar inmediatamente después de hacer una consulta. Existen además bases de datos online como *Firebase* [22].

Para este desarrollo se decidió usar *UserDefaults* para guardar pequeñas cantidades de información que deben ser necesarias de obtener fácilmente en cualquier parte de la aplicación, por ejemplo el estado del usuario si ha iniciado sesion o no, el país y criptomoneda seleccionada, etc.

Se utilizó *Keychain* para guardar datos sensibles como la secret key del usuario, necesaria para la comunicación con las *API Rest*.

Se utilizó *Realm* versión *Swift* [19] para hacer un pequeño modelo de datos interno para, por ejemplo, guardar la información de los mercados que existen en nuestra plataforma (se descargan al comenzar la aplicación y se guardan en *Realm* para acceder a ellos cuando se necesite).

Muchos de los datos de la aplicación son dinámicos (datos entregados por el servidor en tiempo real). Por esto, muchos de los datos que llegan a la aplicación se cargan en memoria RAM en vez de ser guardados en una base de datos, pues cambian muy rápidamente y no son necesarios de guardar (como datos de órdenes de mercados, precios históricos minuto a minuto, órdenes abiertas del usuario, balances o saldos de la cuenta del usuario, etc)

Para la comunicación con el servidor de manera segura se utilizan dos grandes métodos muy utilizados; *API Rest* y *WebSockets*. Las *API Rest* se utilizan para todo tipo de petición sensible y que requiera un mayor grado de seguridad, como lo es el iniciar sesión y recibir las llaves públicas y privadas necesarias para el uso de las *API* que sean privadas (también se usan algunas públicas, sin necesidad de identificación alguna) como también enviar tokens de 2FA (*two factor authentication*), autorización de nuevo dispositivo, abrir órdenes de mercado y cancelarlas, comprar en el mercado instantáneo (instant market), etc. Se utilizan *sockets* para la comunicación en tiempo real con el servidor, para obtener los precios actualizados de los mercados, información en tiempo real del libro de órdenes e historial de órdenes, gráficos de profundidad y gráfico de

precios, balance actualizado del usuario, etc. Para las *API* Rest se decidió utilizar una librería de terceros llamada *Alamofire* [18] muy conocida en el desarrollo iOS que permite hacer peticiones a la *API* con parámetros, headers, etc. Para *sockets* se utilizó Socket.io [20] que es la librería más conocida multiplataforma para la implementación de *sockets* en tanto aplicaciones web como aplicaciones iOS y Android.

2.6 Usabilidad

En cuanto a usabilidad, la aplicación debe otorgar a sus usuarios una experiencia familiar, también debe ser fácil de usar.

Para esto, luego de desarrollar la aplicación esta se analizará desde los atributos de usabilidad, una referencia para poder medir el diseño y desarrollo de software [6]. Los atributos a medir son:

- Eficiencia: Se refiere a la rapidez con que es posible acceder a las distintas funcionalidades de la aplicación una vez que el usuario ya ha aprendido a utilizarla.

- Errores: Se refiere a la facilidad o frecuencia con la que los usuarios cometen diversos errores en una aplicación y también la capacidad de los usuarios de darse cuenta que cometieron un error y poder corregirlos.

Se medirán mediante un estudio de usabilidad a 5 personas, las cuales realizarán una serie de tareas en que se medirá el tiempo de ejecución de cada una, si logró o no completar cada tarea, e *issues* de usabilidad.

2.7 Seguridad

El tema de la seguridad es fundamental desde los inicios de CryptoMarket ya que tienen la confianza de los usuarios que usan la plataforma y ellos esperan que sus datos, fondos en criptomonedas y dinero de diversos países no se vean alterados nunca por terceros. Es por esto que CryptoMarket tiene altos estándares de seguridad en cuanto al acceso a sus servidores.

Como medida de seguridad inicial, los usuarios tienen que autorizar el dispositivo desde donde están ingresando a su cuenta de CryptoMarket cada vez que inician sesión en un nuevo dispositivo, o cuando su IP cambia por estar en un lugar diferente al de donde acceden comúnmente a la plataforma. También se da la posibilidad y se fomenta utilizar autenticación de dos pasos (2FA) para proteger sus cuentas. La aplicación también implementa el inicio de sesión junto a autorización de dispositivo y 2FA.

Para llamar a las *API* que no son públicas, se requiere utilizar una llave pública y llave privada que se otorgan al iniciar sesión en la aplicación (guardadas en la base de datos del teléfono) para poder crear una firma diferente cada vez que se hace una petición a la API.

Para crear la firma se utiliza un mensaje que es la concatenación de la llave pública, los valores de los parámetros de la petición concatenados lexicográficamente y el timestamp en segundos en el momento que se genera la petición. Todo esto luego se firma utilizando la llave privada y el algoritmo *hashed message authentication code* (HMAC) SHA384 generando un código *hash* que se envía como uno de los tantos parámetros de los headers de la petición a la API.

El servidor, utilizando la llave privada del usuario, es capaz de validar si esa petición fue hecha por el usuario y luego la ejecuta. El *timestamp* es necesario también ya que el servidor solo ejecuta una acción dentro de 30 segundos del *timestamp* enviado por la petición, esto para evitar que si un tercero está escuchando la petición, no pueda volver a ejecutarla ya que esta se invalida a los 30 segundos.

En cuanto a la seguridad de los *sockets*, se utiliza una *API* privada con la seguridad previamente mencionada para obtener los diversos IDs para la conexión con los *sockets*, así como también la dirección del *socket* con el cual se hará la comunicación. Además, los IDs de conexión caducan cada 24 horas, lo cual gatilla un evento de desconexión del *socket* (o si ha pasado más de 24 horas y el usuario entra a la aplicación) se deben de volver a pedir los datos del *socket*.

Cuando la aplicación detecta peticiones erróneas a la API por no tener permiso para hacerlas (o cuando el usuario cierra sesión) se borran todas los datos de seguridad del usuario (sus claves públicas y privadas, credenciales de conexión con *sockets*, estado de inicio de sesión) y se fuerza al usuario a volver a iniciar sesión para renovar la sesión en la aplicación.

Capítulo 3

Diseño y Análisis de la solución

3 DISEÑO Y ANÁLISIS DE LA SOLUCIÓN

En este capítulo se presenta una descripción del problema a solucionar, así como también los requisitos de la solución, arquitectura y diseño de esta.

3.1 Descripción del problema

Como se mencionó en la introducción, en CryptoMarket cerca del 40% de sus usuarios acceden a la plataforma web utilizando su dispositivo móvil. La plataforma web cuenta con una versión adaptada para dispositivos móviles que si bien tiene diversas optimizaciones para usar la plataforma en un celular, no está totalmente enfocada en su uso desde estos dispositivos, ocurriendo que en ocasiones no se vean de buena forma varias secciones de la plataforma, los gráficos son gráficos simples y poco informativos, errores diversos visuales al navegar por la web, etc.

Estos problemas llegan como quejas directamente a servicio al cliente, usando del tiempo de los trabajadores en solucionar problemas que podrían ser evitados.

Fruto de que CryptoMarket siempre busca entregar un servicio que provea la mejor experiencia de usuario posible, nace la idea de desarrollar una solución móvil nativa la cual logre adaptar muchas capacidades de la página web de una manera fácil, cómoda, familiar, rápida y segura para los usuarios.

Esta aplicación móvil ha sido muy solicitada por los usuarios que están en constante observación de los mercados, esperando el momento oportuno para poder hacer órdenes de mercado. Esto no lo pueden hacer ni rápida ni cómodamente desde un dispositivo móvil en la plataforma web.

También estos usuarios piden la capacidad de tener notificaciones o alertas ya que tienen que estar constantemente ingresando a la plataforma web para ver el estado de sus órdenes, precios de mercados, confirmación de transferencias de criptomonedas o dinero, etc.

Al tener una versión móvil se puede favorecer con una serie de ventajas por sobre una versión web responsiva, como un sistema de notificaciones o alertas, comodidad y familiaridad en el uso de la plataforma, rapidez al acceso de información de balance y transacciones de los usuarios, visualización cómoda de gráficos y datos de mercados, entre otras cosas.

3.2 Características de la solución

Los requisitos para la realización de la aplicación son los siguientes:

3.2.1 Requisitos de usuario

- El usuario debe de poder ver la información de los mercados; precios de compra y venta y cambio en 24 horas.
- El usuario debe poder iniciar sesión, usar 2FA y autorizar su dispositivo.
- El usuario debe poder abrir órdenes de mercado y poder cancelar las órdenes de mercado aún no ejecutadas.
- El usuario debe poder decidir el porcentaje de su balance a transar.
- El usuario debe poder cambiar de criptomoneda.
- El usuario debe poder cambiar de divisa.
- El usuario debe poder ver el libro de órdenes de un mercado.
- El usuario debe poder ver las últimas transacciones de un mercado.
- El usuario debe poder ver el gráfico de precios de un mercado.
- El usuario debe poder ver el gráfico de profundidad de un mercado.
- El usuario debe poder ver el historial de sus órdenes.
- El usuario debe poder comprar o vender en el mercado instantáneo.
- El usuario debe poder ver el balance de todas sus criptomonedas y divisas.
- El usuario puede recibir alertas mediante notificaciones.
- La aplicación debe de soportar multilenguaje.

3.2.2 Requisitos de software

- La aplicación debe funcionar en base a cuentas de usuario, comunicándose con el servidor para el inicio de sesión.
- La aplicación debe poder comunicarse por *sockets* para recibir información en tiempo real y con ella poder desplegar tablas con diferente información y gráficos.
- La aplicación debe poder comunicarse por APIs con el servidor para peticiones autenticadas.

- La aplicación debe de soportar todos los tamaños de pantalla de los iPhone compatibles.
- La aplicación debe de usar llaves públicas y privadas para la comunicación con el servidor.
- La aplicación debe de tener la capacidad de cambiar de divisa base a nivel de toda la aplicación y guardar la elección del usuario.
- La aplicación, debe de cerrar sesión automáticamente cuando sea necesario renovar los accesos.
- La aplicación debe implementar notificaciones push para generar alertas para el usuario.
- La aplicación debe de ser segura.
- La aplicación debe de ser dinámica al momento de agregar un nuevo mercado, no debe de requerir actualización cuando hay un nuevo mercado disponible.

3.3 Arquitectura externa

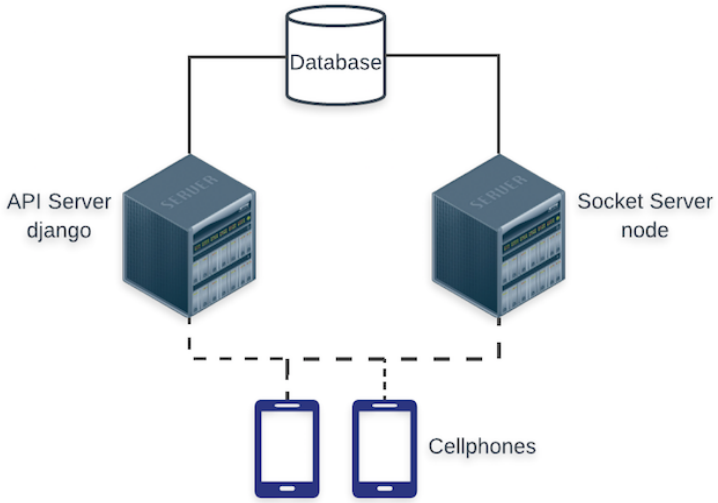


figura 6: arquitectura externa de CryptoMarket con la aplicación

En cuanto a arquitectura, los teléfonos se conectan (dependiendo de lo que necesitan) a dos diferentes servidores de CryptoMarket. El primer servidor *API Server* es el encargado de hacer todas las interacciones mediante *API* de la plataforma, este servidor esta hecho en *django* (*framework* web en *python*), mientras que el segundo servidor

Socket Server es el encargado de toda la comunicación por *sockets* de la plataforma, está hecho en *nodejs* (*framework* web de *javascript*).

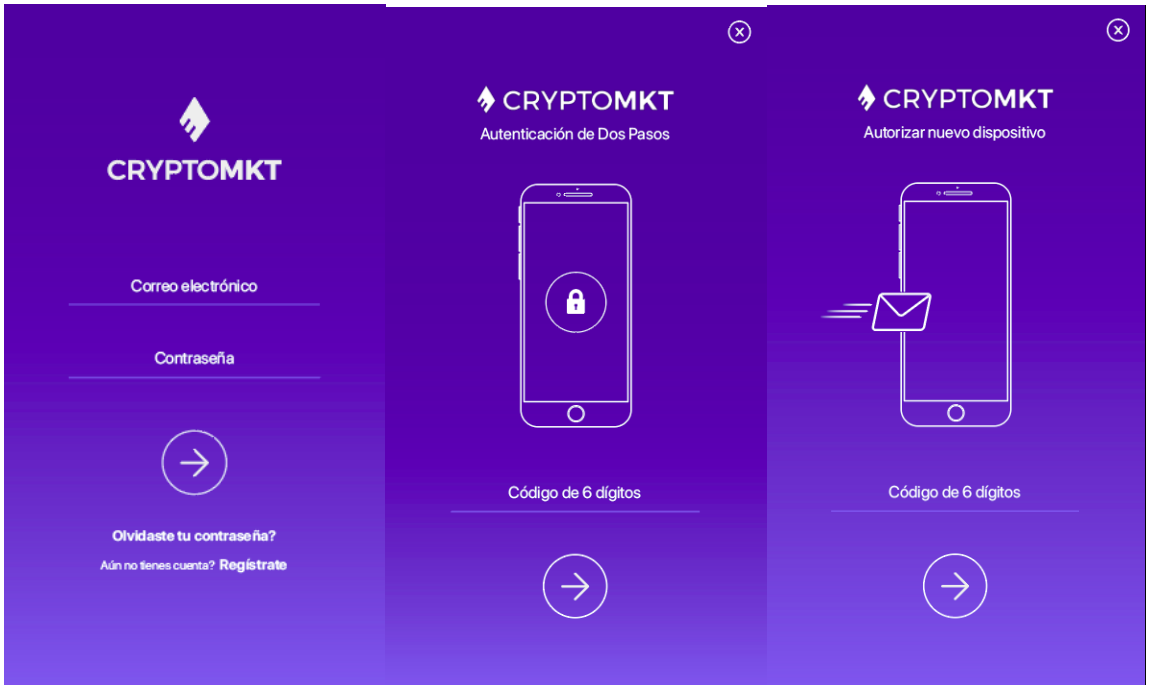
Estas funcionalidades están separadas para repartir la carga en varios servidores. Ambos servidores se conectan a la misma base de datos para tener consistencia en la entrega de información.

3.4 Diseño de la solución

Desde el inicio del diseño de la aplicación, se buscó que esta tuviera todas las funcionalidades necesarias para que los usuarios puedan realizar la mayoría de las acciones que hacen en la plataforma web. Es por esto que se dio mucho énfasis en el diseño y funcionalidades que tendrán todas las vistas de la aplicación. Además que en su conjunto fueran una versión robusta y muy útil para todos los usuarios, dándoles la capacidad de transar desde donde ellos estén sin la necesidad de un computador cerca y de manera cómoda, rápida y segura.

A continuación se detallarán las vistas con las funcionalidades esperadas en cada una y los mockups respectivos. Estos mockups fueron diseñados en conjunto con el diseñador de CryptoMarket, buscando que fuera amigable y familiar con el usuario. Cabe mencionar que para llegar al diseño final se fue iterando en las vistas y revisando diferentes propuestas. Los mockups mostrados en este capítulo no reflejan la funcionalidad final ni el diseño final. Los diseños y funcionalidad finales se mostrarán en el capítulo 4 al mostrar capturas de pantalla de la aplicación finalizada y corriendo en un dispositivo.

- Inicio de sesión, autenticación de dos pasos, autorización de dispositivo (figura 7)



(a) *(b)* *(c)*

figura 7: a) mockups inicio de sesión, b) 2FA, c) autorización de dispositivo.

En la vista de inicio de sesión (figura 7a) el usuario debe poder ingresar sus datos, si sus datos son erróneos la aplicación muestra una alerta del error correspondiente. Si sus datos están correctos pueden ocurrir tres cosas diferentes; iniciar sesión inmediatamente, ir a la vista de 2FA (si la cuenta posee 2FA) o ir a la vista de autorización de dispositivo (si el dispositivo aún no se ha autorizado).

Al ir a la vista de 2FA (figura 7b) el usuario debe de ingresar los 6 dígitos que le entrega la aplicación que utilice para 2FA. Al ingresarlo erróneamente se despliega una alerta, al ingresarlo correctamente pueden pasar dos cosas; iniciar sesión inmediatamente o llevarlo a la vista de autorización de dispositivo.

En la vista de autorización de dispositivo (figura 7c) el usuario recibe en su correo electrónico un código de 5 caracteres que debe de ingresar en la aplicación. Si es erróneo se despliega una alerta y si es correcto se inicia la sesión.

La vista de iniciar sesión debe desplegarse si un usuario que no ha iniciado sesión en la aplicación intenta acceder a las vistas que requieren la sesión iniciada.

- Market (figura 8)

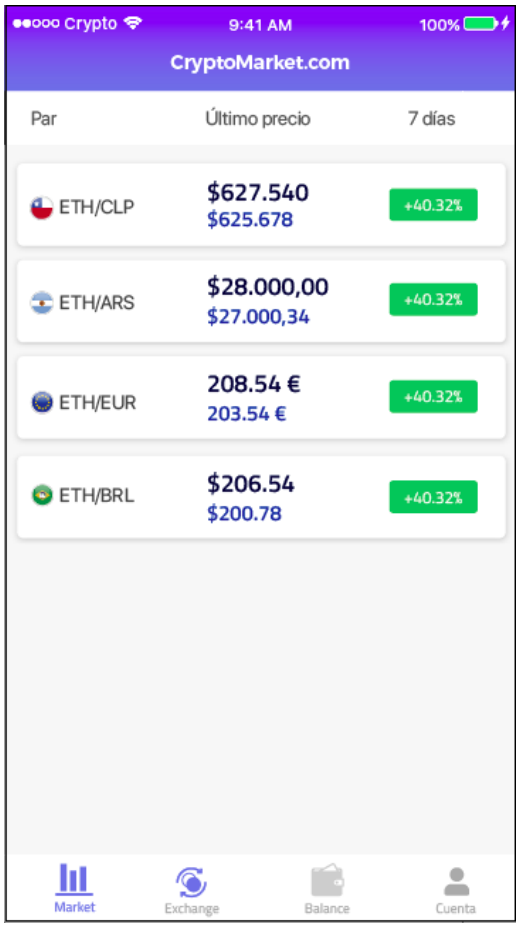


figura 8: mockup de la pestaña market

En esta vista se muestran los diferentes mercados disponibles en CryptoMarket. Se eligió el nombre *market* para mantener la terminología usada en la plataforma web (en vez de mercados). Estos mercados deben mostrar los precios de compra y venta de acuerdo al país del usuario, mostrar el par de mercado y la variación de precio en las últimas 24 horas. Por ejemplo, si el usuario elige como moneda base el Euro, se deben de mostrar todos los mercados cuya divisa base sea una criptomoneda y como contra divisa sea el Euro.

Todos estos datos son obtenidos desde el servidor en tiempo real. En el instante en que un usuario cualquiera abre una orden de compra o venta que modifique los precios de mercado, se ve el cambio reflejado inmediatamente en la aplicación.

Esta es la vista principal de la aplicación y primera opción de la tab bar inferior. En la tab bar se puede acceder a las otras partes de la aplicación, pero para acceder a ellas se debe estar autenticado.

- Exchange (figura 9)

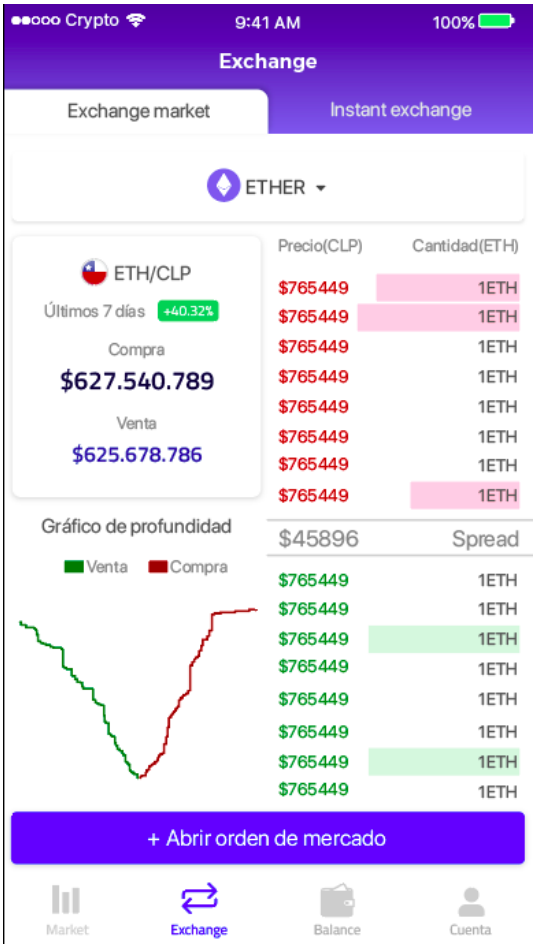


figura 9: mockup de la pestaña exchange

Esta es una de las vistas más complejas de la aplicación y es la segunda en la tab bar inferior. Se eligió el nombre *exchange* para mantener la terminología usada en la plataforma web. En esta vista se debe dar la capacidad al usuario de cambiar de criptomoneda. También debe de ser capaz de ver los precios y variación del mercado seleccionado, así como también el libro de órdenes y últimas transacciones, y un gráfico

de profundidad simple que representa el mercado. Por último, debe de poder acceder a abrir una orden de mercado y comprar o vender en el mercado instantáneo.
Todos los datos de esta vista deben ser actualizados en tiempo real desde el servidor. El gráfico de profundidad se construye a partir del libro de órdenes calculando la cantidad acumulada de órdenes y su respectivo precio.

- Cambio de mercado (figura 10)

figura 10: mockup de cambio de mercado

Esta vista despliega una tarjeta por cada mercado disponible para la divisa que el usuario tiene seleccionada (por ejemplo, si tiene la divisa peso chileno seleccionada, se mostrarán las criptomonedas que tienen como contra divisa el peso chileno). Luego de presionar una tarjeta de una criptomoneda se cambia a nivel de la aplicación el mercado seleccionado. El diseño de esta vista se utiliza para decidir otras opciones a nivel de la aplicación.

- Orden de mercado (figura 11)

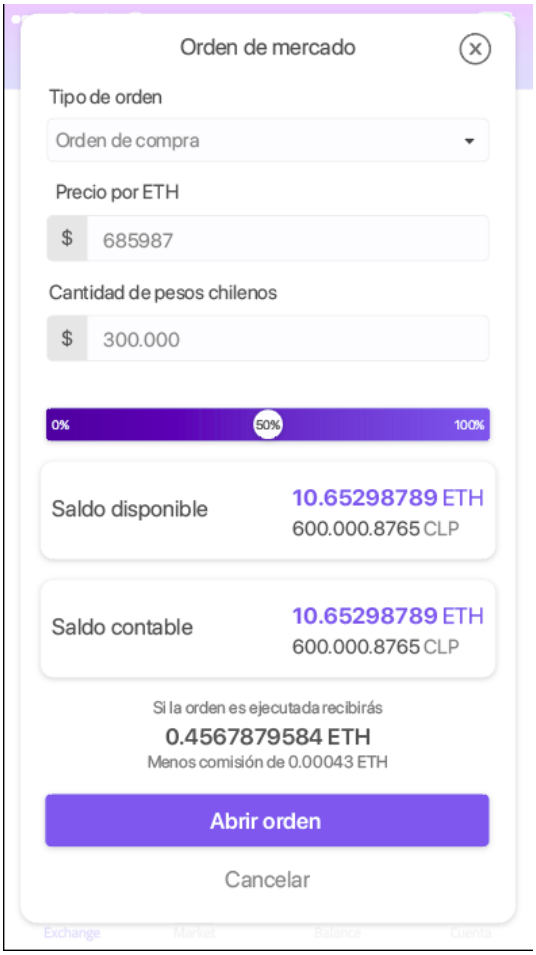


figura 11: mockup de orden de mercado

En esta vista, el usuario puede realizar una orden de mercado ya sea de compra o de venta. Debe de ingresar el precio que desea por la criptomoneda y luego la cantidad de dinero que quiere comprar de la criptomoneda, o la cantidad de criptomonedas que desea vender. También es posible usar un slider para elegir un porcentaje del balance del usuario para transar. El balance del usuario debe de ser visible. La vista debe

mostrar tanto la cantidad de dinero o criptomonedas a recibir, como la comisión que cobra CryptoMarket. Si hay algún problema en los datos ingresados por el usuario se despliega una alerta, mientras que si esta todo bien se envía la orden al servidor y luego de unos segundos se recibe una respuesta y se muestra al usuario.

- Instant exchange (figura 12)

figura 12: mockup de vista instant exchange

En esta vista el usuario puede comprar o vender instantáneamente a precio de mercado. Se eligió el nombre *instant exchange* para mantener la terminología usada en la plataforma web. El usuario también puede cambiar de mercado en todo momento y ver el balance de este. También puede usar el slider para decidir un porcentaje de su balance a transar. Si todo está bien se envia la consulta al servidor y luego de un tiempo se le señala al usuario el resultado. Debe de mostrarse un pequeño mensaje de que el monto a

recibir es una estimación y puede cambiar en cualquier segundo según las condiciones del mercado.

- Balance (figura 13)

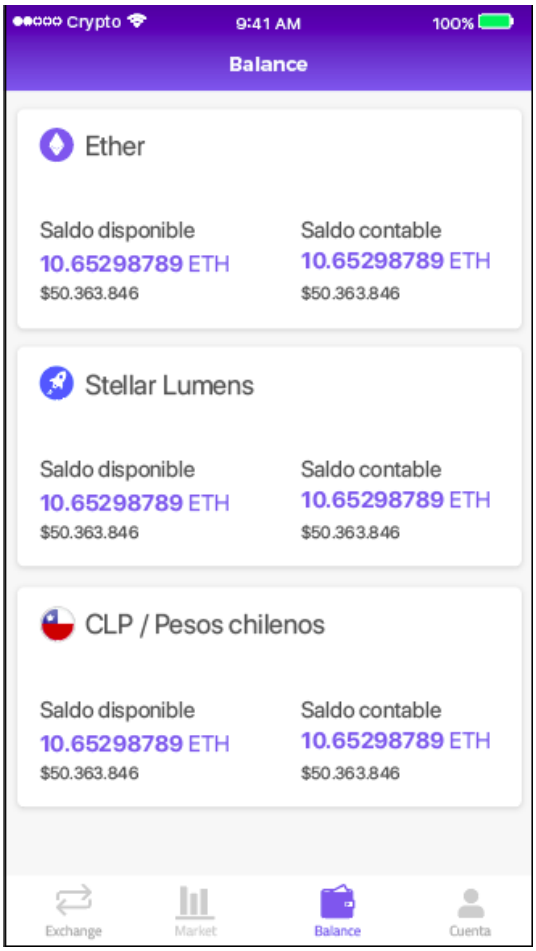


figura 13: mockup de pestaña de balance

Esta es la tercera vista del tab bar. En esta vista el usuario debe de ser capaz de ver todos los fondos que posee en su cuenta de CryptoMarket, independiente de la contra divisa seleccionada, debe de poder ver todos sus pesos chilenos, argentinos, euros, ether y criptomonedas en general que posee. Los saldos se actualizan en tiempo real.

- Configuración (figura 14)

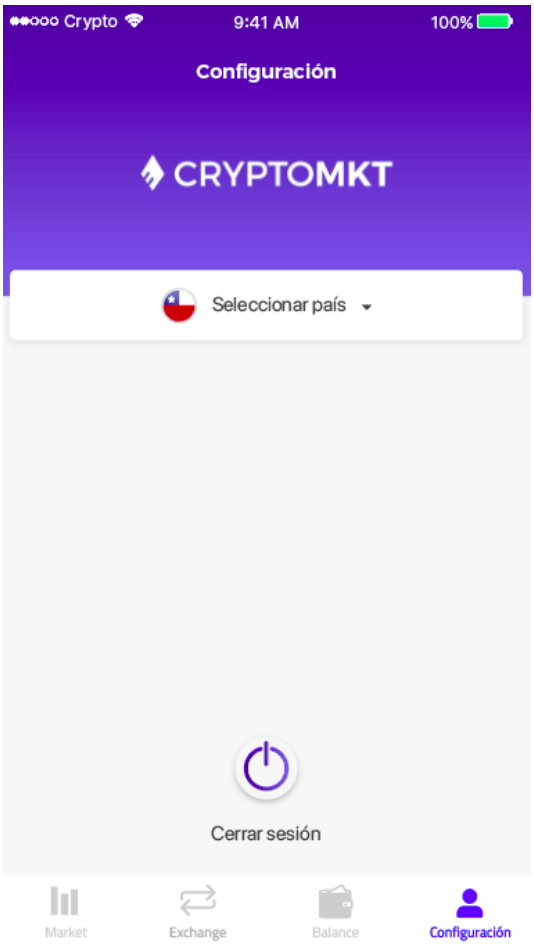


figura 14: mockup de pestaña de configuración

Esta es la cuarta vista del tab bar. En esta vista el usuario puede hacer diversas configuraciones de la aplicación, por ejemplo cambiar de contra divisa, cerrar sesión, cambiar de idioma y otras opciones a futuro. Al presionar cerrar sesión se despliega una alerta de confirmación que luego de que el usuario la acepta, se cierra la sesión de este y se le lleva a la primera vista de la aplicación que no requiere estar autenticado.

- Gráficos (figura 15)

figura 15: mockup de vista de gráficos

En esta vista, accediendo desde la pestaña de exchange, muestra un gráfico de precios en tiempo real del mercado seleccionado, así como una opción para cambiar los tiempos del gráfico. También al seleccionar el otro tipo de gráfico, gráfico de profundidad, se despliega el gráfico de profundidad del mercado seleccionado.

En esta vista se pueden ver las órdenes activas y cancelarlas luego de que el usuario lo confirme en una alerta que se despliega. Como también ver las órdenes históricas que ha realizado en ese mercado.

Capítulo 4

Implementación de la solución

4 IMPLEMENTACIÓN DE LA SOLUCIÓN

En este capítulo se describe en detalle la implementación de la aplicación desarrollada. Se comienza por describir la idea general de la aplicación, luego se explica la arquitectura del software, las vistas con su diseño final implementado en el dispositivo movil en conjunto con una descripción de la funcionalidad de esa vista.

4.1 Idea general de la aplicación

La aplicación está pensada como un acceso relativamente más cómodo y rápido a la plataforma de CryptoMarket en donde sus usuarios pueden ver los datos en tiempo real de un mercado, ver las órdenes de mercado e historial de ordenes, abrir y cerrar ordenes, comprar o vender instantáneamente, recibir notificaciones push cuando alguna de sus órdenes de mercado fue ejecutada y ver sus fondos. Esta aplicación se considera la primera versión de la plataforma móvil de CryptoMarket y será la base para futuras versiones con mayores funcionalidades.

4.2 Arquitectura del software

Para el desarrollo de la solución se utilizó principalmente el lenguaje *Swift* 4 en el IDE *XCode* 9 de Apple.

Adicionalmente se utilizó el paradigma *Model View Controller* el cual es ampliamente utilizado en el desarrollo de aplicaciones para iOS. El modelo es representado por clases que modelan objetos, código de llamadas a la red y persistencia (base de datos). El storyboard (figura 16) es el lugar en donde se diseñan las vistas y los flujos de la aplicación que son controladas (controlador) por las clases denominadas *ViewController*. Cada vista de la aplicación tiene una *ViewController* que se encarga de ver que datos despliega la vista y las diversas respuestas que puede tener una vista con la interacción de una persona en la pantalla.

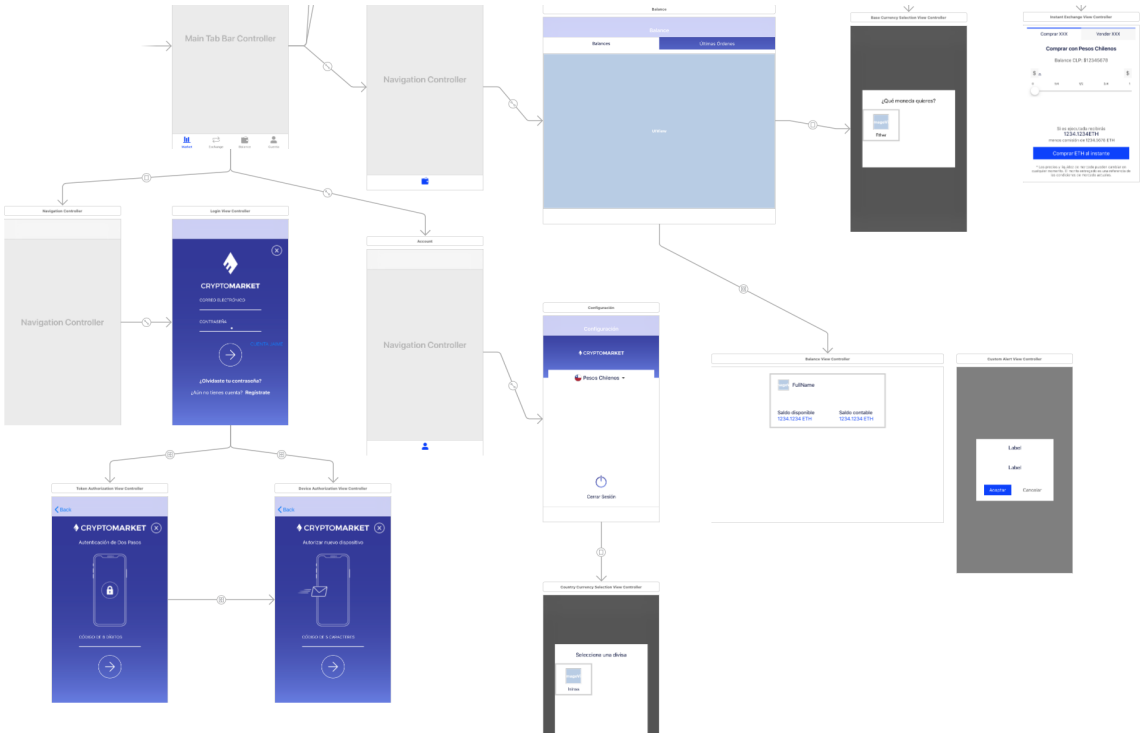


figura 16: captura del storyboard de XCode

Se utilizaron diversas librerías y frameworks, las principales son:

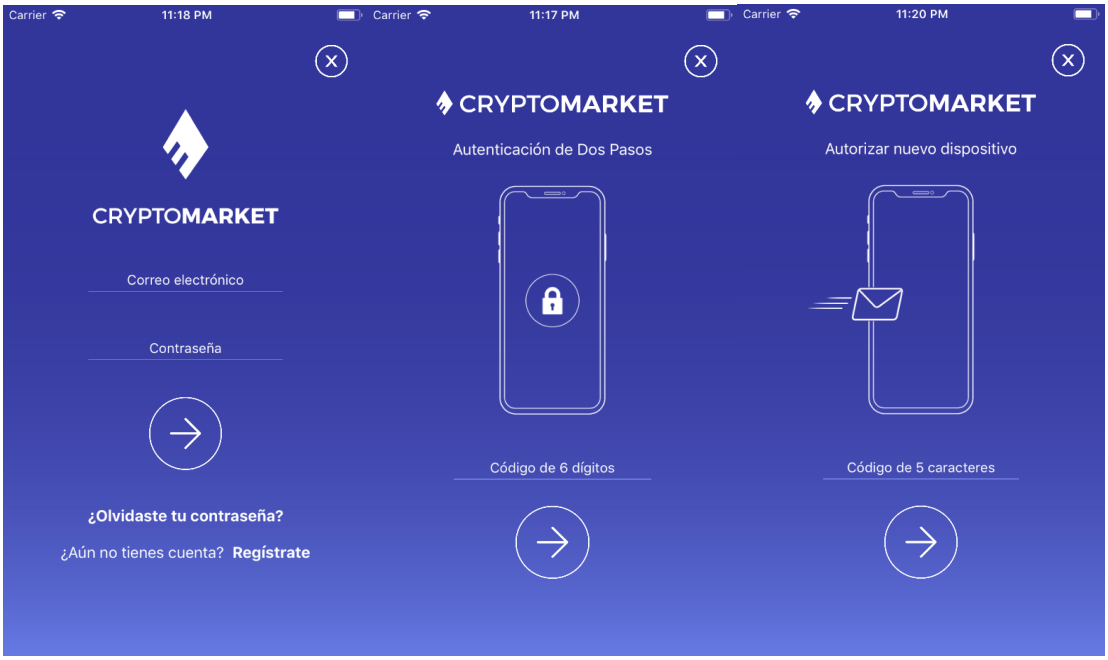
- Cocoapods: Es el gestor de librerías de *iOS* más conocido, con él se instalan nuevas librerías de manera muy sencilla [24].
- Alamofire: Librería utilizada para la comunicación por APIs con un servidor [18].
- Socket.IO: Librería utilizada para la comunicación por *WebSockets* con un servidor que implemente el protocolo Socket.IO [20].
- Charts: Librería utilizada para el manejo de gráficos de la aplicación [21].
- Firebase: Framework de Google con muchas herramientas para el desarrollo de aplicaciones, ya sea base de datos online, manejo de estadísticas, etc. En este caso fue utilizado para el manejo de notificaciones *push* enviadas por el servidor [22].
- Realm Swift: Librería para base de datos móvil [19].
- SwiftyJSON: Librería que permite un manejo más sencillo de objetos JSON en *Swift* [25].

El proyecto se mantuvo en un repositorio git en un servidor propio de CryptoMarket para el manejo de versiones de forma de asegurar el respaldo de los datos.

4.3 Vistas y códigos

En esta sección se mostrará el diseño final de las vistas que fueron previamente explicadas en el capítulo 3, implementadas ya en el dispositivo, en conjunto con una explicación de cómo funcionan internamente y de ser pertinente una muestra de código.

- Vistas inicio de sesión, autenticación de dos pasos, autorización de dispositivo



(a) *(b)* *(c)*

figura 17: a) vista de inicio de sesión, b) vista de 2FA, c) vista de autorización de dispositivo

Al ingresar las credenciales del usuario y presionar la flecha (figura 17a) se hace una *request* de tipo post a la *API* la cual puede retornar error (credenciales no existe, usuario incorrecto, etc) o retornar código 200 junto a la llave pública y privada del usuario . Estas llaves se guardan en *Keychain* y serán las que se utilicen para realizar todas las llamadas a la *API* que se necesite autenticación. Luego de una respuesta por parte del servidor, se redirecciona según corresponda (figura 18) a la vista de 2FA (figura 17b) o de autorización de dispositivo (figura 17c).

```
if json!["credential"]["authorized"].boolValue {

    UserDefaults.standard.setIsLoggedIn(value: true)
    self.dismiss(animated: true, completion: nil)

} else {

    self.email = email
    self.password = password
    self.performSegue(withIdentifier: "deviceAuthorizationSegue", sender: nil)

}
```

figura 18: trozo de código de success en llamada a API iniciar sesión

- Sockets

Los *sockets* son utilizados para la comunicación en tiempo real con el servidor. Se utilizan para actualizar los datos de los mercados, obtener los balances de los usuarios, órdenes de mercado, historial de ordenes, datos de gráficos, órdenes activas e históricas del usuario, etc. En este caso tanto el servidor como la aplicación implementan el protocolo de la librería de Socket.IO.

Para conectarse a un *socket* se necesitan tres identificadores necesarias para la autenticación con este, asi como tambien la URL host del *socket* el cual se nos otorgó para hacer la conección. Estos datos se piden a una *API* habiendo iniciado sesión previamente.

Para implementar *sockets* en una vista básicamente se deben hacer tres cosas: configurar el *socket* (inicializarlo y decirle la URL del *socket* host), agregar los handlers (código que se ejecuta al llegar un evento del *socket*) y hacer la conexión.

Luego de hacer la conexión, el servidor espera un máximo de 10 segundos para que se envíen los tres identificadores previamente mencionadas al *socket* para autenticarnos con él. Si no lo realizamos nos desconecta automáticamente. Una vez nos autenticamos, nos empieza a entregar todos los datos a los cuales nos suscribimos.

Los handlers son muy importantes ya que nos permiten hacer acciones para cada evento diferente que nos suscribimos. Por ejemplo en la siguiente imagen (figura 19) se puede ver como nos suscribimos a dos eventos. En este caso, cuando nos llega el evento .connect que significa que se realizó la conexión al *socket* correctamente, enviamos los tres identificadores previamente descritos al *socket* con la función *emit* utilizada para enviar datos al *socket*. En este caso *socket.emit("user-info", info)* que envía la información requerida.

Cabe decir que, por temas de seguridad, las credenciales de conexión al *socket* caducan cada 24 horas. Al haber una desconexión cualquiera (por la caducidad del acceso o algún otro error como por ejemplo problemas de internet) se debe volver a hacer el proceso de obtención de IDs y reconectarse al *socket*.

Para esto está el evento *.disconnect* que se gatilla al haber una desconección al *socket*. Este intenta volver a conectarse después de una espera de 5 segundos (se decidió dar un tiempo de espera, pues si no se tiene internet quedaría en un loop intentando conectar y fallando inmediatamente).

```
socket.on(clientEvent: .connect) { data, ack in

    print("-------- CONECTADO A SOCKET --------")

    let socketSocId = UserDefaults.standard.getSocketSocId()
    let socketUId = UserDefaults.standard.getSocketUId()
    let socketVerify = UserDefaults.standard.getSocketVerify()

    let info: NSDictionary = [

        "socid": socketSocId,
        "uid": socketUId,
        "verify": socketVerify

    ]

    self.socket.emit("user-info", info)
    self.socket.emit("subscribe","GLOBAL")

}

socket.on(clientEvent: .disconnect) { data, ack in

    print("-------- DESCONECTADO DE SOCKET --------")

    if self.socketConnectionWanted {

        DispatchQueue.main.asyncAfter(deadline: DispatchTime.now()
            + .seconds(5)) {

            print("ASYNC DESCONNECT")
            self.socketGetData()

        }
    }
}
```

figura 19: trozo de código de handlers de eventos de sockets

- Vista Market (figura 20)

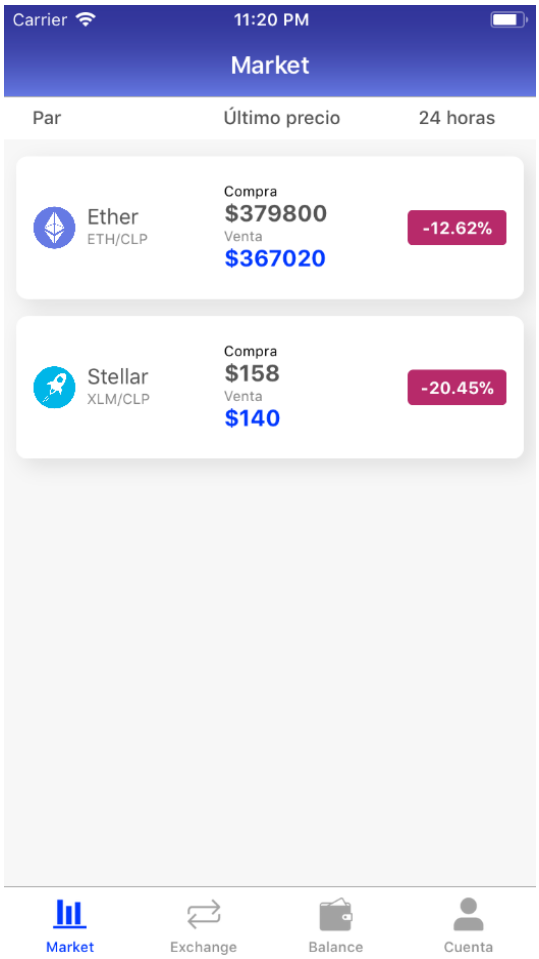


figura 20: pestaña market

Esta es la primera vista de la aplicación y la primera en utilizar *sockets*. Cuando aparece la vista en pantalla se inicia la conexión al *socket*, una vez realizada la conexión comienza a entregar datos con el evento *socket.on("board-alert")* (figura 21). Este evento es llamado cada vez que hay algún cambio en el mercado. Los *sockets* nos entregan *data* y esta data hay que transformarla a JSON para poder ser utilizada. Para esto se utiliza la librería SwiftyJSON que nos permite tomar la *data* y transformarla a un objeto JSON para mayor facilidad de uso. Luego de tener el objeto JSON se comienza a generar los objetos necesarios para posteriormente mostrar la información de los mercados.

```
socket.on("board-alert"){ data, ack in

    print("-------- BOARD-ALERT --------")

    let json = JSON(data[0])

    var index = 0
    for element in self.marketPairsNames {

        if json["board"][element].exists() {

            let tempBoardAlertData = BoardAlertData()

            tempBoardAlertData.ask = json["board"][element]["ASK"].doubleValue
            tempBoardAlertData.bid = json["board"][element]["BID"].doubleValue
            tempBoardAlertData.delta1d = json["board"][element]["delta1d"].doubleValue
            tempBoardAlertData.delta7d = json["board"][element]["delta7d"].doubleValue

            tempBoardAlertData.marketPair = self.marketPairs[index]
```

figura 21: trozo de código del handler de una alerta de mercado

- Vista exchange (figura 22)

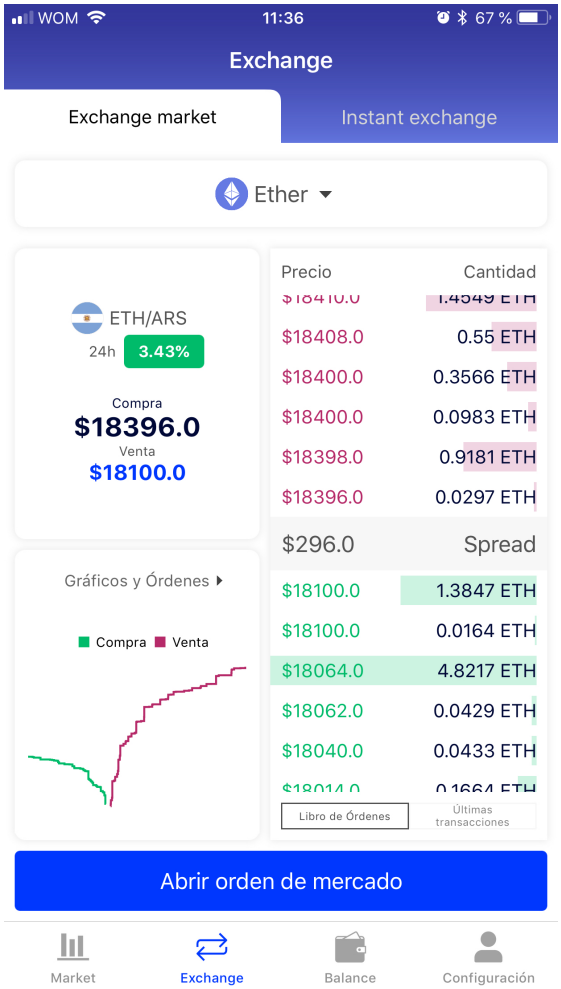


figura 22: pestaña exchange

En esta vista, se utiliza una conexión al socket para diferentes eventos; un evento para obtener los datos de mercado (tarjeta de precio de compra y venta), un evento para obtener el libro de órdenes (tabla con precios y cantidades), y un evento para obtener las últimas órdenes. Los rectángulos de colores bajo la cantidad representan qué tan relevante en cantidad es una orden con respecto a otra (mientras más largo es el rectángulo mayor es la cantidad). Al presionar la tarjeta que contiene al gráfico de profundidad se va a la vista de gráficos y órdenes.

- Vista libro de órdenes (figura 23)

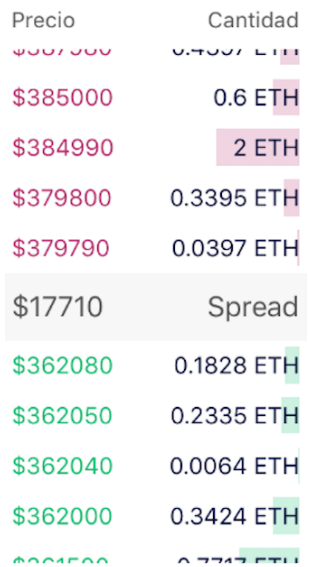


figura 23: vista libro de órdenes

El libro de órdenes es una vista contenida dentro de la pestaña exchange. Esto se hizo para poder separar los códigos de la vista padre (pestaña exchange) ya que el libro de órdenes tiene una lógica muy compleja. Para poder construir el libro de órdenes (figura 23) se realizan dos procesos. Primero se obtienen los arreglos de órdenes de venta (en color rojo) y órdenes de compra (en color verde) mediante el evento *socket.on("orders")*. Con estos dos arreglos se construye el libro de órdenes y se despliega en una tabla. Las barras de colores por debajo del texto representan qué tan relevante es cierta orden con respecto al resto, entregando una pequeña ayuda visual al usuario para saber qué orden es más difícil que se realice.

El libro de órdenes está cambiando constantemente ya que los usuarios de la plataforma están abriendo y cerrando órdenes de compra y venta en cada momento. Cada 1 segundo se crea un nuevo libro de órdenes. Si el servidor enviara cada 1 segundo los 1000 objetos

que tiene el libro de órdenes sería muy costoso ya que es una cantidad enorme de datos que se debe estar enviando cada 1 segundo a cada usuario. Es por esto que el servidor implementa una librería *javascript* de *diferencias* llamada jsondiffpatch para poder ir modificando el libro de órdenes sin la necesidad de estar enviándolo completo.

Funciona de la siguiente manera; inicialmente cuando un usuario se conecta a la plataforma, el cliente del usuario recibe todo el libro de órdenes. Luego cada 1 segundo el servidor le envía una serie de pasos (llamados *parches*) que debe aplicar a la vieja versión del libro de órdenes para poder llegar a la nueva versión del libro de órdenes. Si el cliente se quedó atrás por alguna razón (se demoró en aplicar un *parche*, intermitencia de internet, etc.) el servidor siempre está enviando varios parches que, dependiendo la versión del libro de órdenes en donde se encuentre el cliente, debe seguir en secuencia para poder llegar a la versión más actualizada del libro. Si por alguna razón ninguno de los parches nos sirve para actualizar el libro (pues nos quedamos muy atrás) se le debe decir al *socket* mediante la función *emit* una alerta de pánico para que nos entregue todo el libro de órdenes de nuevo y volver a comenzar este proceso.

Lamentablemente esta librería *javascript* no posee una contraparte para el lenguaje *Swift* de Apple. Es por esto que se tuvo que realizar un pequeño *parser* para esta librería para así poder entender en *Swift* que significa cada *parche* (figura 24) que nos entregan y poder aplicarlos.

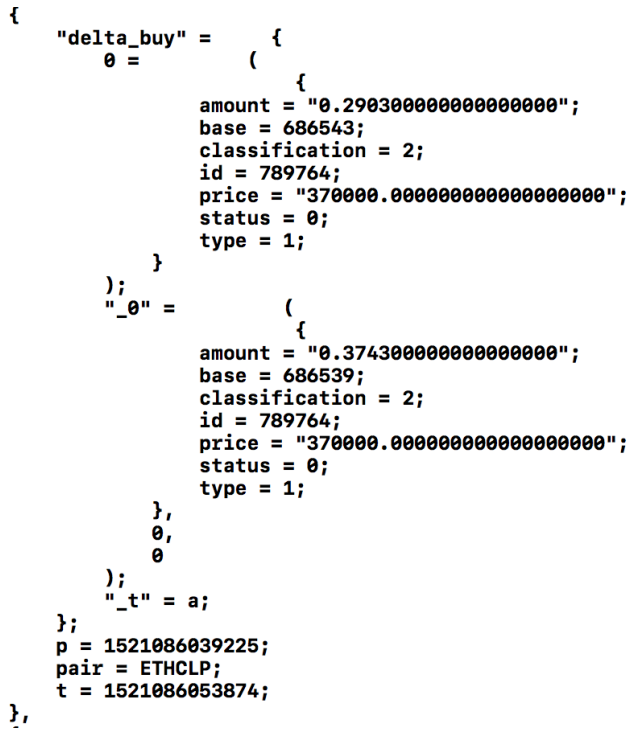


figura 24: ejemplo de un parche de jsondiffpatch

En la imagen (figura 24) se puede apreciar a qué corresponde un *parche*. Un *parche* señala a qué arreglo le afecta (en este caso *delta_buy*, al arreglo de compras) y también muestra para qué versión previa del libro se puede aplicar (*p = 1521086039225*) y a que versión del libro se llegará si se aplica (*t = 152108605384*). Cuando en el *parche*, viene una llave del objeto JSON como un número (en este caso 0) significa que se debe *insertar* el objeto que contiene en el arreglo en la posición 0. Mientras que una instrucción que comienza con _ seguido de un número (en este caso "_0") significa que hay que *eliminar* el objeto que está en esa posición del arreglo.

Como al usar la librería en *javascript* se llama a una función patch que recibe un arreglo y un parche y lo aplica, y además no existía ningún tipo de documentación en la manera u orden en que se aplicaban estos parches, se tuvo que descifrar cómo se debe aplicar cada uno.

Lo único que sí señala la librería es que si la llave es un solo número hay que insertar el objeto *JSON* en esa posición, mientras que si la llave parte con guión bajo seguido de un número hay que remover de esa posición el objeto.

Después de hacer muchas pruebas durante días, se llegó a la conclusión de que la forma en que se aplican los parches es la siguiente:

Luego de decidir que cierto parche hay que aplicarlo, se guardan los movimientos de eliminación y de inserción para poder aplicarlos posteriormente. Luego de guardar todos los movimientos que el parche dice que hay que aplicar, se ordenan decrecientemente (ordenando por la posición) los movimientos de eliminación y se aplican en ese orden. Luego se ordenan crecientemente (ordenando por la posición) los movimientos de inserción y se aplican en ese orden.

Si todo funciona correctamente, se tendrá un libro de órdenes que se va actualizando con tan solo aplicar los pequeños *parches*.

Cabe mencionar que las llamadas a los *handlers* (eventos) de los *sockets* son *asíncronas*. Es por esto que es sumamente importante que mientras se estén aplicando algunos *parches* por ningún motivo puede entrar otra llamada del mismo evento a aplicar otro *parche*, pues esto generaría una inconsistencia y podría provocar *dataraces* e incluso hacer que la aplicación se caiga. Es por esto que se utilizaron *semáforos* para impedir el acceso hasta que se haya completado la aplicación de cierto *parche*.

- APIs

Como se mencionó anteriormente, existen dos tipos de APIs que se pueden comunicar con el servidor; APIs públicas y APIs autenticadas. Las que son públicas simplemente se llaman sin ningún tipo de seguridad pues cualquiera puede llamarlas sin ser usuario y sin estar autenticado.

Sin embargo, por temas de seguridad, hay que seguir una serie de pasos para poder utilizar una *API* autenticada (figura 25).

Para llamar a este tipo de APIs hay que incluir información en el header de la llamada. En el campo *X-MKT-APIKEY* se debe incluir la llave pública del usuario. En el campo *X-MKT-TIMESTAMP* se debe incluir el timestamp en segundos del momento actual (si el *timestamp* difiere en más de 30 segundos del *timestamp* del servidor al recibir la petición, esta se descarta). En el campo *X-MOBILE* se debe incluir el *token* de *Firebase* para notificaciones *push* (se hablará de esto más adelante) en conjunto con el sistema operativo y versión del celular. Por último en el campo *X-MKT-SIGNATURE* debe de ir un mensaje generado creando un *HMAC SHA384* utilizando la llave privada del usuario para firmarlo. El contenido de este mensaje es la concatenación del *timestamp*, la ruta de la *API* (ejemplo /v1/create_order) y los *valores* de los parámetros concatenados lexicográficamente.

Si todo sale bien la llamada nos retorna un código de *success*.
Este tipo de seguridad es necesaria para que solo el usuario dueño de la cuenta pueda hacer la petición. Si un tercero está escuchando de alguna forma la petición del usuario no puede hacer nada ya que si intenta modificar los parámetros de la petición (por ejemplo algún precio), la firma (*el valor en el header X-MKT-SIGNATURE*) estaría incorrecta. Y si intenta enviar la misma petición, esta ya habrá caducado por la regla de los 30 segundos de tiempo para poder hacer la petición.

```
case .createOrder(let body, let timestamp, let parameters):
    let message = "\(timestamp)\(self.path)\(body)"
    //print(message)
    let signature = message.hmac(.sha384, key: UserDefaults.standard.getSecretKey())   //hmac
    urlRequest.addValue("\(UIDevice.current.modelName)#\(UIDevice.current.systemName) \
        (UIDevice.current.systemVersion)#\(UserDefaults.standard.getFirebaseToken())", forHTTPHeaderField: "X-
        MOBILE")
    urlRequest.addValue(UserDefaults.standard.getSessionToken(), forHTTPHeaderField: "X-MKT-APIKEY")
    urlRequest.addValue(signature, forHTTPHeaderField: "X-MKT-SIGNATURE")
    urlRequest.addValue(timestamp, forHTTPHeaderField: "X-MKT-TIMESTAMP")
    urlRequest.setValue("application/x-www-form-urlencoded", forHTTPHeaderField: "Content-Type")
    urlRequest = try URLEncoding.default.encode(urlRequest, with: parameters)  // POST URL ENCODING
```

figura 25: trozo de código que muestra cómo se forma una llamada a la API autenticada

- Vista orden de mercado (figura 26)

figura 26: vista de orden de mercado

En esta vista se permiten hacer órdenes de compra y venta. Para poder hacer una orden primero se debe poner un precio por la criptomoneda a comprar o vender, y luego la cantidad. Esta cantidad puede ser ingresada por un *slider* o manualmente pero debe de ser menor o igual al balance del usuario y mayor al mínimo de un mercado. Luego de ingresar estos valores la aplicación calcula cuánto dinero o cuantas criptomonedas obtendrá el usuario si se realiza la orden, como también la comisión que cobrará CryptoMarket al momento de que se ejecute la orden. Si todos los valores son correctos, al presionar el botón *abrir orden* se le desplegará una alerta al usuario para que confirme que está seguro de abrir la orden. Si acepta se utilizará una *API* autenticada para enviar la orden de mercado al servidor. Si el servidor acepta o rechaza la petición se le hará saber al usuario mediante una alerta.

- Vista alertas (figura 27)

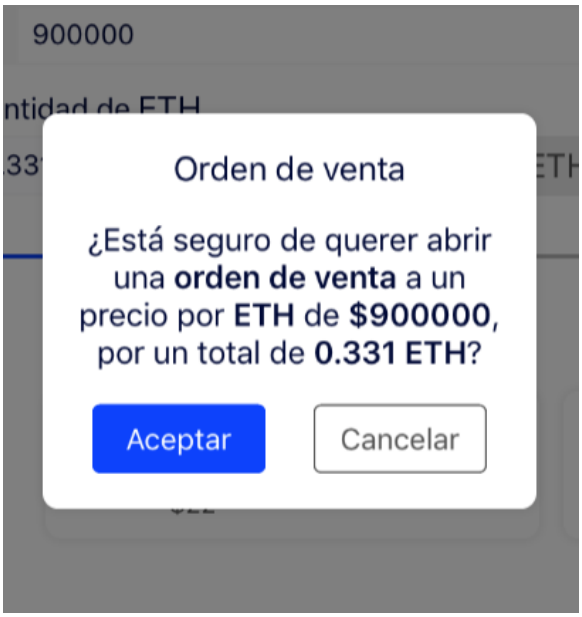


figura 27: vista de alertas

Se decidió hacer un sistema de alertas propio en vez de usar el sistema de alertas nativo pues se tendría más control sobre el diseño y las acciones al presionar los botones. El sistema de alertas puede ser llamado desde cualquier lugar de la aplicación llamando a una función estática que despliega la alerta. Para poder ejecutar código después de que el usuario presiona un botón se utilizaron las clausuras o *closures* para poder dar código como un argumento más a la función que despliega la alerta, como se puede apreciar en la siguiente imagen (figura 28).

```
static func showAlert(viewController: UIViewController, type:
    CustomAlertViewController.alertType, alertTitle: String,
    alertInformation: String, acceptClosure: @escaping () -> ()) {


    let normalAttributes = [NSAttributedStringKey.font:
        UIFont.systemFont(ofSize: 16)]
    let message = NSMutableAttributedString(string: alertInformation,
        attributes: normalAttributes)


    let customAlertViewController =
        viewController.storyboard?.instantiateViewController(withIdentifier:
        "CustomAlertViewController") as! CustomAlertViewController


    customAlertViewController.alertTitle = alertTitle
    customAlertViewController.alertInformation = message
    customAlertViewController.acceptClosure = acceptClosure
    customAlertViewController.selectedType = type

    customAlertViewController.modalPresentationStyle = .overFullScreen
    viewController.present(customAlertViewController, animated: false,
        completion: nil)
```

figura 28: trozo de código de alertas propias

Como argumentos de las alertas se deben dar la vista donde se desplegará la alerta, el título y mensaje, y la clausura a llamar una vez se presiona el botón confirmar.

- Vista instant exchange (figura 29)

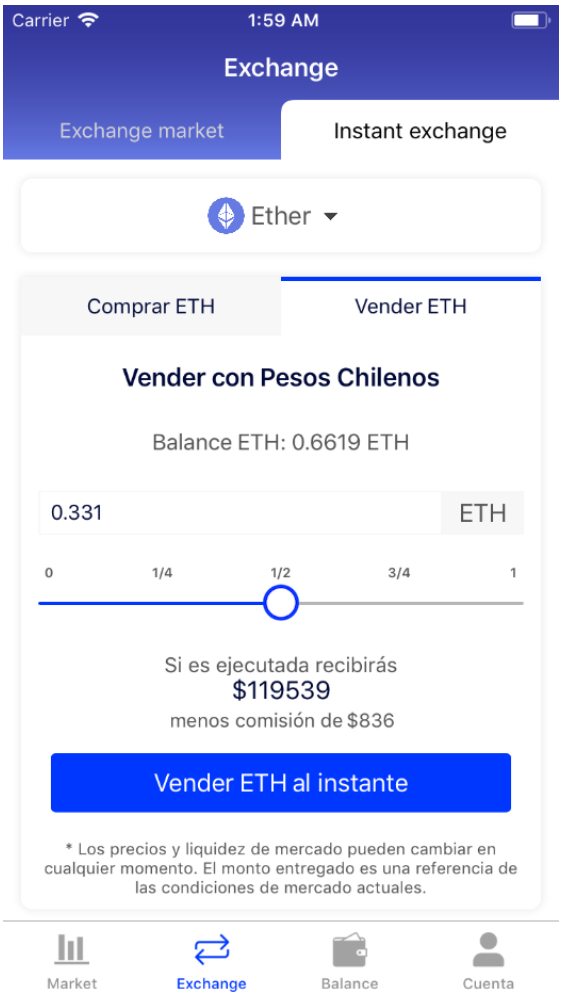


figura 29: vista instant exchange

Vista similar a la de orden de mercado, pero permite comprar o vender a precio de mercado (o sea, sin señalar un precio). Se debe elegir la cantidad que puede ser mediante un *slider* o manualmente. Si las cantidades son válidas, al presionar el botón de vender o comprar se despliega una alerta la que, luego de tener la confirmación del usuario, procede a hacer la petición a la *API* correspondiente para hacer la compra o venta.

- Vista selección de criptomoneda (figura 30)

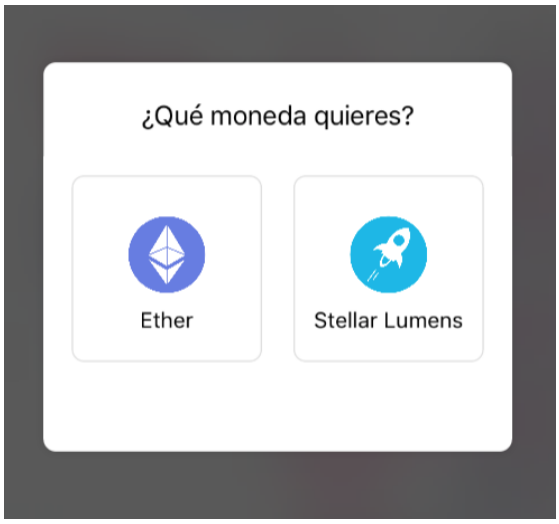


figura 30: vista para seleccionar una criptomoneda

En esta vista, se despliegan todas las criptomonedas que tiene el usuario para elegir siempre y cuando exista ese mercado en su divisa local. Para saber si existe ese mercado e n su divisa local se hace una consulta a *Realm* (figura 31) para saber todos los mercados cuya contra divisa sea igual a la divisa local del usuario en la aplicación (por ejemplo, si en la aplicación el usuario tiene escogido el peso chileno, aparecerán todos los mercados cuya contra divisa es CLP, como es el caso de ETH/CLP y XLM/CLP)

```
let realm = try! Realm()

selectedMarketPair = realm.object(ofType: MarketPair.self, forPrimaryKey:
    Int(UserDefaults.standard.getSelectedMarketId()))

markets = realm.objects(MarketPair.self).filter("counterCurrency.id == %@",
    Int(selectedMarketPair.counterCurrency!.id))
```

figura 31: trozo de código de una consulta a Realm

- Vista gráficos y órdenes del usuario (figura 32)

figura 32: vista gráficos y órdenes de usuario

En esta vista se despliega el gráfico de precios (el que se vé) o el de profundidad del mercado según decida el usuario. También se muestran las órdenes activas y órdenes históricas. Las órdenes activas del usuario se pueden cancelar al presionar el botón *x* y confirmándolo en la alerta que se despliega. La vista gráficos es una vista contenida dentro de la vista principal, al igual que la vista de órdenes.

- Vista gráficos (figura 33)

figura 33: vista de gráficos

En esta vista existen dos gráficos que se muestran dependiendo de la opción elegida por el usuario en la esquina superior izquierda. En la barra superior para el gráfico de precios (también llamado gráfico de velas) se puede elegir el intervalo de tiempo: M5 datos cada 5 minutos, M15 cada 15 minutos, H1 datos cada 1 hora, H4 cada 4 horas, D1 datos cada 1 día, W1 datos cada 1 semana. El cambio al presionar los botones es instantáneo ya que tan solo hay que redibujar el gráfico y no pedir nuevos datos ya que se tienen todos los datos del gráfico guardados en memoria. Para pedir los datos de los gráficos se requiere suscribirse al evento del *socket socket.on("pairs")* que entrega todos los datos de las velas del mercado seleccionado para dibujar el gráfico de velas y todos sus intervalos de tiempo.

- Notificaciones push

Las notificaciones *push* se realizan mediante el *framework* Firebase de Google. Al iniciar la aplicación Firebase le otorga a la aplicación un *token* que lo identifica en Firebase con el cual un servidor que implemente el *framework* puede enviarle notificaciones *push* al teléfono con ese *token*.

Estas notificaciones son para alertar al usuario de que una orden de mercado que él había enviado fue ejecutada, o que un nuevo dispositivo fue autorizado para entrar a su cuenta, etc.

El *token* de Firebase se envía al servidor al momento de iniciar sesión con lo que el servidor guarda el *token* y así sabe cuál dispositivo tiene cierto usuario para alertarle de sus notificaciones.

- Vista balance (figura 34)

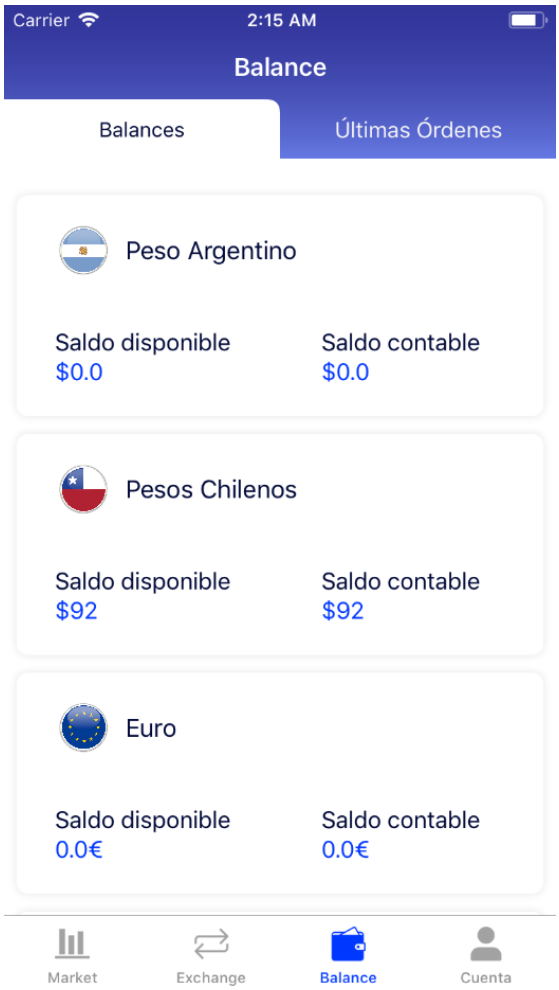


figura 34: pestaña de balance

En esta vista se obtienen los balances o fondos del usuario en tiempo real mediante conexión al *socket*. Estos datos se obtienen suscribiendose al evento *socket.on("balances")*

- Vista cuenta (figura 35)

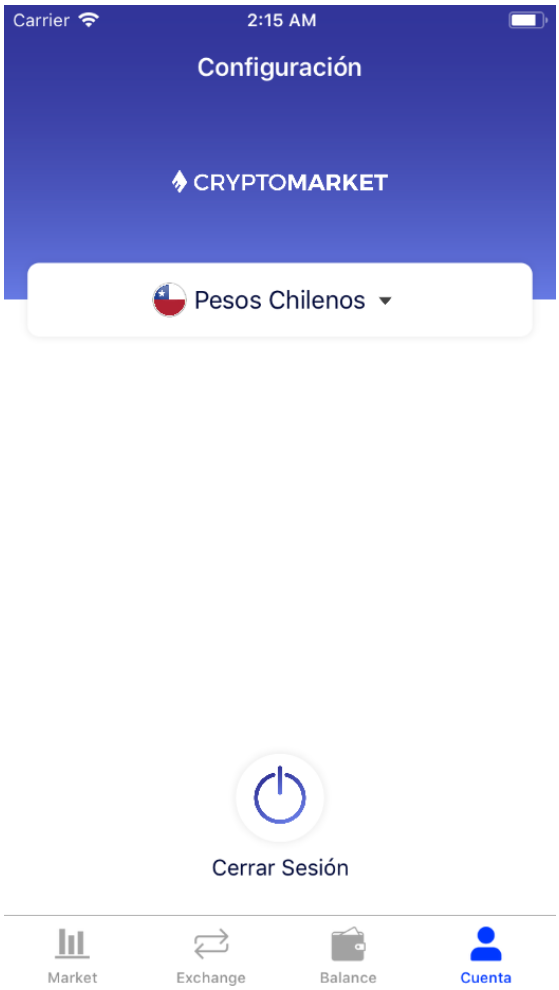


figura 35: pestaña de cuenta

En esta vista el usuario tiene acceso a una serie de opciones o preferencias. En esta vista puede cambiar de divisa local y cerrar sesión. Al cerrar sesión se borran todos los datos de la base de datos que permiten conectarse a los *sockets* y a las *APIs* autenticadas y luego se envía al usuario a la vista principal tal como se muestra en la imagen (figura 36).

```
static func logOut(currentViewController: UIViewController) {

    UserDefaults.standard.setIsLoggedIn(value: false)
    UserDefaults.standard.setSessionToken(value: "")
    UserDefaults.standard.setSecretKey(value: "")
    UserDefaults.standard.setSocketUId(value: "")
    UserDefaults.standard.setSocketSocId(value: "")
    UserDefaults.standard.setSocketVerify(value: "")
    UserDefaults.standard.setSocketHost(value: "")

    if let tabBar = currentViewController.tabBarController {

        tabBar.selectedIndex = 0

    }
}
```

figura 36: trozo de código de cerrar sesión

- Multilenguaje

En la primera versión de la aplicación se planea dar soporte a 2 lenguas diferentes; español y portugués de Brasil (y más a futuro).

Para hacer el soporte a multilenguaje hay que traducir dos grandes partes de la aplicación. La primera es el *Storyboard* (lugar donde se hace el diseño gráfico). Para hacer esto *XCode* facilita el trabajo ya que al activar el multilenguaje, se generan archivos de texto automáticamente para cada lenguaje en donde se puede hacer la traducción de cada frase que existe en el *Storyboard*.

Sin embargo, para traducir los *strings* que están en el código *Swift* de la aplicación, se deben generar estos archivos manualmente y uno a uno buscar todos los *strings* que se quieren traducir, y se debe generar un *id* para cada *string* y referenciando al archivo de traducciones.

Capítulo 5

Validación

5 VALIDACIÓN

En este capítulo se realizará un proceso de validación de la aplicación en donde se realizó un estudio de usabilidad sobre la aplicación desarrollada.

5.1 Estudio de usabilidad

Con el objeto de medir el nivel de usabilidad de la aplicación desarrollada, se llevará a cabo un estudio utilizando el método de *User Testing*. Así, será posible identificar las falencias y fortalezas de su diseño, con miras a resolver interrogantes relevantes, tales como ¿el usuario logra encontrar una determinada feature? o ¿es una tarea fácil de lograr?

Para esto, se encomendó la realización de una serie de tareas o actividades a usuarios experimentados, pues queremos medir usabilidad y no aprendizaje, en el sistema operativo iOS y en *trading* de criptomonedas.

El estudio se realizará a 5 personas, según el *paper Determining Usability Test Sample Size* [26] la mayor cantidad de problemas de usabilidad son detectados con las primeras 3 a 5 personas. Si se siguen realizando pruebas con más personas es improbable encontrar nuevos problemas de usabilidad.

5.1.1 Descripción del estudio

A continuación se describirán la serie de tareas a desarrollar, el tipo de usuario y qué aspectos de usabilidad se midieron y de cuál forma.

En cuanto a las tareas a realizar, son las siguientes:

1) Cree una orden de venta a un valor de $1.000 pesos de una cantidad de 1 XLM de la criptomoneda Stellar Lumens.
2) Cancelar una orden abierta cualquiera.
3) Encuentre el libro de órdenes de la criptomoneda Bitcoin.
4) Vea el gráfico de velas de un mercado.
5) Vea la cantidad de pesos chilenos y de la criptomoneda Ether que hay en la cuenta.
6) Encuentre el precio de venta de la criptomoneda Bitcoin.

Las tareas en general no mencionan las palabras exactas que hay en la aplicación para no facilitar la ejecución de las tareas.

Estas tareas se registraron mediante audio y video apuntando a la aplicación y a los usuarios usándola y haciendo que estos piensen en voz alta, teniendo consentimiento para hacer las grabaciones. También se contó con una cuenta en CryptoMarket con dinero cargado previamente para las pruebas, buena conexión a internet, dispositivo iOS con la aplicación instalada y un incentivo monetario en criptomonedas. También se irán tomando notas sobre la realización de las tareas.

Los aspectos de usabilidad a medir son las siguientes:

- ¿La información que se presenta es fácil de entender?
- ¿Es una de las tarea difícil de lograr?
- ¿Existe en algún momento una barrera para los usuarios?
- ¿Sienten frustración o incomodidad en algún momento?

Para esto se guardan los siguientes datos:

- Tasa de éxito de las tareas.
- Tiempo que toma realizar cada tarea.
- Valoraciones subjetivas de satisfacción, frustración, confianza, etc.
- Problemas de usabilidad.

La características de los usuarios son los siguientes:

- Usuario mayor de 18 años.
- Con experiencia en el sistema operativo iOS.
- Conocimientos en criptomonedas.
- No haya usado la aplicación móvil de CryptoMarket previamente.

Los usuarios fueron reclutados mediante la publicación de los requisitos en grupos de redes sociales sobre *blockchain* y criptomonedas en Chile, señalando además la existencia de una recompensa monetaria por la participación.

5.1.2 Ejecución del estudio

Se mostrarán los resultados de las 5 personas, y observaciones

Persona 1	Se completó	Duración (s)	Observaciones
Tarea 1	Si	80	
Tarea 2	Si	100	Señaló que tuvo dificultades para encontrar el lugar donde están las órdenes.
Tarea 3	Si	30	
Tarea 4	Si	25	
Tarea 5	Si	27	
Tarea 6	Si	20	

Persona 2	Se completó	Duración (s)	Observaciones
Tarea 1	Si	178	Intentó presionar las criptomonedas en la vista de market.
Tarea 2	Si	63	Señaló que tuvo dificultades para encontrar el lugar donde están las órdenes.
Tarea 3	Si	18	
Tarea 4	Si	11	
Tarea 5	Si	30	
Tarea 6	Si	12	

Persona 3	Se completó	Duración (s)	Observaciones
Tarea 1	Si	93	
Tarea 2	Si	24	Señaló que tuvo dificultades para encontrar el lugar donde están las órdenes.
Tarea 3	No	104	Intentó presionar las criptomonedas en la vista de market.
Tarea 4	Si	15	
Tarea 5	Si	36	
Tarea 6	Si	7	

Persona 4	Se completó	Duración (s)	Observaciones
Tarea 1	Si	88	
Tarea 2	Si	27	
Tarea 3	Si	18	
Tarea 4	Si	12	
Tarea 5	Si	26	
Tarea 6	Si	10	

Persona 5	Se completó	Duración (s)	Observaciones
Tarea 1	No	120	Ingresó bién los datos pero no seleccionó la criptomoneda correcta.
Tarea 2	No	62	
Tarea 3	No	58	Intentó presionar las criptomonedas en la vista de market.
Tarea 4	Si	24	
Tarea 5	No	83	
Tarea 6	Si	9	

	Cantidad de personas que realizaron la tarea	Duración promedio de tareas realizadas (s)
Tarea 1	4	109.75
Tarea 2	4	53.3
Tarea 3	3	22
Tarea 4	5	17.4
Tarea 5	4	29.75
Tarea 6	5	11.6

5.1.3 Análisis de usabilidad

- ¿La información que se presenta es fácil de entender?

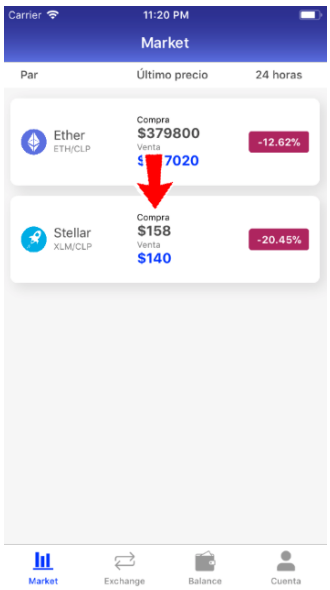
4 de 5 personas entendían la información que se mostraba, sin embargo la persona 5 tuvo dificultades con 4 de 6 de las tareas. Se cree que, a pesar que la persona señaló que tenía conocimientos con criptomonedas, tal vez no tenía los conocimientos en la profundidad requerida para usar una aplicación de *trading* como esta. A pesar de esto se incluyeron sus resultados pues también se destacaron problemas de usabilidad en su prueba.

- ¿Es una de las tarea difícil de lograr?

Si bien la tarea 1 demoró más tiempo en promedio que el resto, esta tarea es más lenta de realizar pues requiere de ingresar correctamente datos con el teclado.
Sin embargo, la tarea 2 es la tarea que más tasa de error tuvo e incomodidad por parte de los usuarios, por el hecho de no encontrar el lugar donde se pueden ver las órdenes abiertas y poder cancelarlas.

- ¿Existe en algún problema de usabilidad?

Se detectaron 2 problemas de usabilidad:



Algunos usuarios, al pedirles una tarea que usa una criptomoneda diferente a la seleccionada previamente, intentaban cambiar de criptomoneda presionando las tarjetas de información en la pestaña de market (figura 37).

figura 37: pestaña de market

4 de 5 usuarios tuvieron dificultades para encontrar las órdenes activas, o simplemente no pudieron realizar la tarea 2 por no poder encontrar el lugar correcto en la pestaña exchange (figura 38).

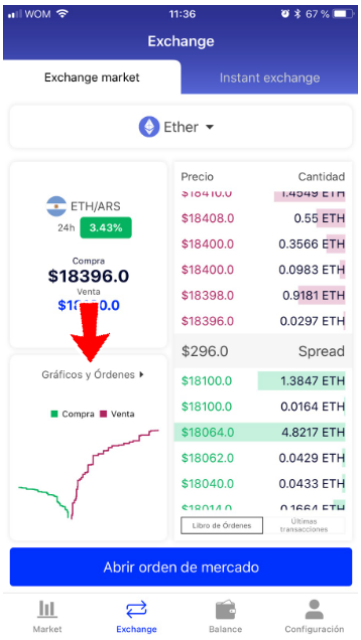


figura 38: pestaña de exchange

- ¿Sienten frustración o incomodidad en algún momento?

Al realizar la tarea 2, independiente de la tasa de éxito de esta tarea, 4 de 5 usuarios sintieron incomodidad al no encontrar fácilmente el lugar donde cancelar las órdenes.

5.2 Cantidad de usuarios

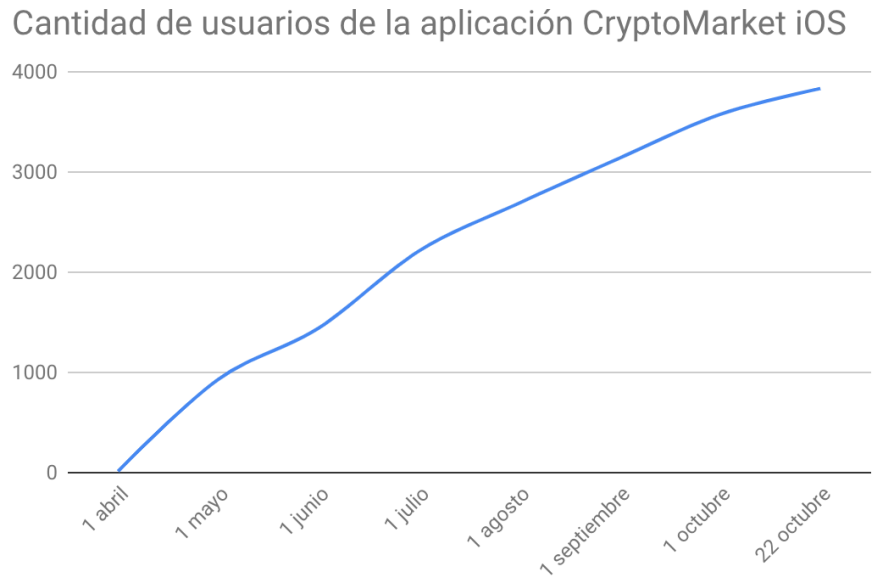


figura 39: cantidad de usuarios de la aplicación CryptoMarket iOS

Desde que se lanzó la aplicación a día de hoy cuenta con cerca de 4.000 usuarios. De estos usuarios más de 1.000 utilizan la aplicación al menos una vez al mes. No existen datos de cuántos usuarios tuvo la aplicación antigua de CryptoMarket iOS, pués no existía la integración con *Firebase* que entrega los datos estadísticos.
Si bien es una cantidad no despreciable de usuarios, sigue siendo un desafío atraer más personas pues en este momento CryptoMarket cuenta con cerca de 160.000 usuarios, de los cuales alrededor de 9.000 han descargado las aplicaciones móviles de CryptoMarket iOS y Android.

Capítulo 6

Conclusiones

6 CONCLUSIONES Y TRABAJO A FUTURO

En este trabajo de memoria se implementó una solución móvil iOS para la plataforma CryptoMarket, en la cual los usuarios pueden hacer diversas acciones de trading de criptomonedas, como comprar y vender criptomonedas, ver sus balances de criptomonedas y pesos, ver diferente información de los mercados como las órdenes abiertas e historial de transacciones, etc.

Respecto de la programación se realizó con el lenguaje de programación *Swift 4* de Apple, en conjunto con el IDE XCode 10 de Apple.

El objetivo general de la memoria se cumplió, la aplicación desarrollada ya está disponible en la App Store de iOS. A día de hoy se cuentan con cerca de 4.000 usuarios, y se han recibido decenas de comentarios y correos electrónicos positivos agradeciendo el desarrollo de esta solución, así como también *feedback* sobre más funcionalidades que desean los usuarios.

Finalmente se hizo un estudio de usabilidad en donde se encontraron ciertos problemas de usabilidad, que actualmente se está viendo cómo resolver en una futura actualización de la aplicación.

En cuanto al trabajo a futuro, se está desarrollando la capacidad de pagar por código QR a una *wallet* de otra persona, ver el valor del portafolio (lugar donde se muestran todas las criptomonedas que se poseen, y su valor aproximado en pesos), solicitar retiros y abonos a las cuentas bancarias de las personas. Y la capacidad de poder ver los datos de las *wallet* de criptomonedas de los usuarios.

7 BIBLIOGRAFíA

[1] Satoshi Nakamoto (2008) Bitcoin: A Peer-to-Peer Electronic Cash System
[2] uPort https://www.uport.me/
[3] Stellar Foundation https://www.stellar.org/lumens/
[4] Buda https://itunes.apple.com/cl/app/buda/id1321460860?mt=8
[5] Binance https://play.google.com/store/apps/details?id=com.binance.dev
[6] Nielsen, J., 1993. Usability engineering. New York: Academic Press Professional.
[7] Interest in Bitcoin https://news.bitcoin.com/interest-in-bitcoin-set-to-double-in-europe-new-survey-suggests/
[8] How Cryptocurrencies prices works https://cointelegraph.com/explained/how-cryptocurrency-prices-work-explained
[9] Native vs Hybrid Apps https://blog.techmagic.co/native-vs-hybrid-apps/
[10] Aplicación Buda https://play.google.com/store/apps/details?id=com.buda.crypto
[11] Orionx https://www.orionx.com
[12] KuCoin https://www.kucoin.com/
[14] Ionic https://ionicframework.com/
[15] Xamarin https://visualstudio.microsoft.com/es/xamarin/
[16] CryptoMarket Developers https://developers.cryptomkt.com/es
[17] XCode https://developer.apple.com/xcode/
[18] Alamofire https://github.com/Alamofire/Alamofire
[19] Realm Swift https://realm.io/docs/swift/latest/
[20] Socket.io https://socket.io/
[21] Charts https://github.com/danielgindi/Charts
[22] Firebase https://firebase.google.com/
[23] HMAC
https://www.wolfe.id.au/2012/10/20/what-is-hmac-authentication-and-why-is-it-useful/
[24] CocoaPods https://cocoapods.org/
[25] Swifty JSON https://github.com/SwiftyJSON/SwiftyJSON
[26] Carl W. Turner, James R. Lewis, Jakob Nielsen (2006) Determining Usability Test Sample Size