Universidad Católica de Chile
Escuela de Ingeniería – Departamento Ciencia de la Computación
IIC2113 – Diseño Detallado de Software
Segundo Semestre 2020

Bitfake Exchange

Entrega 1: Documentación

Fecha de entrega: Viernes 4 de septiembre de 2020

Grupo 2:
Kenzo Andrade
Matias Perez
Juan Pablo Contreras
Claudio Prieto
Rodrigo Heiremans
Maximiliano Schudeck

1. Introducción

En esta documentación presentamos a *Bitfake Exchange,* una aplicación web enfocada en la compra y venta de la criptomoneda *Bitfake* (BTF). En esta app cualquier cliente puede registrarse en nuestra plataforma, para así crear una cuenta propia, en donde se puede manejar un exchange de BTF y pesos chilenos (CLP). Además de permitir la compra y venta de BTF, la app guarda información sobre cada transacción personal, para ser compartida única y exclusivamente con el cliente dueño de la cuenta en *Bitfake Exchange*. El cliente en cualquier momento puede solicitar estados de su cuenta, lo que permite ver cuánto BTF y CLP tiene.

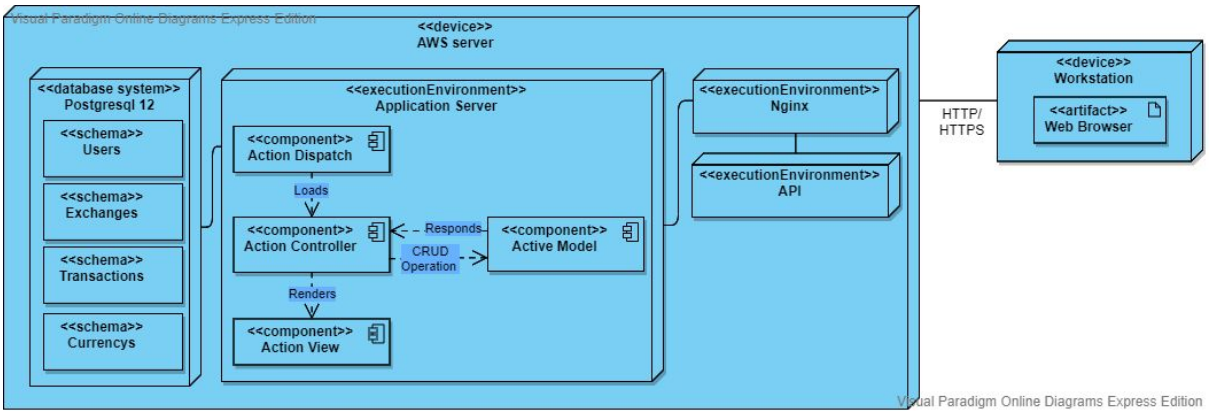
Como BTF es una moneda descentralizada los precios de ella fluctúan constantemente y *Bitfake Exchange* está al tanto de la situación y tiene actualizados sus precios en tiempo real, tanto para la compra y venta. Al completar cualquier transacción se le envía un correo electrónico al cliente, con el resumen de la compra/venta.

Además de la aplicación web *Bitfake Exchange*, expone una API, en donde cada cliente tiene un token para acceder a ella. A través de nuestra API se pueden hacer consultas del precio actual de compra y venta, además de consultar el balance actual de la cuenta. También se pueden hacer solicitudes para ejecutar órdenes de compra y venta de BTF en tiempo real.

2. Overview

2.1 Arquitectura

Bitfake Exchange es una aplicación construida en el framework Ruby on Rails, encima de ello, utilizamos Docker. Este último, permite a los desarrolladores empaquetar, enviar y ejecutar fácilmente cualquier aplicación como un contenedor ligero, portátil y autosuficiente, que puede ejecutarse prácticamente en cualquier lugar. Los contenedores hacen esto al permitir a los desarrolladores aislar el código en un solo lugar. Esto facilita la modificación y actualización del programa. En este proyecto tenemos cuatro contenedores distintos: *database, app, nginx y API*. Para la producción de la aplicación tenemos una instancia EC2 de Amazon Web Services (AWS), como servidor único que va a correr los cuatro contenedores distintos. Estos componentes se pueden apreciar en el diagrama UML de arquitectura:



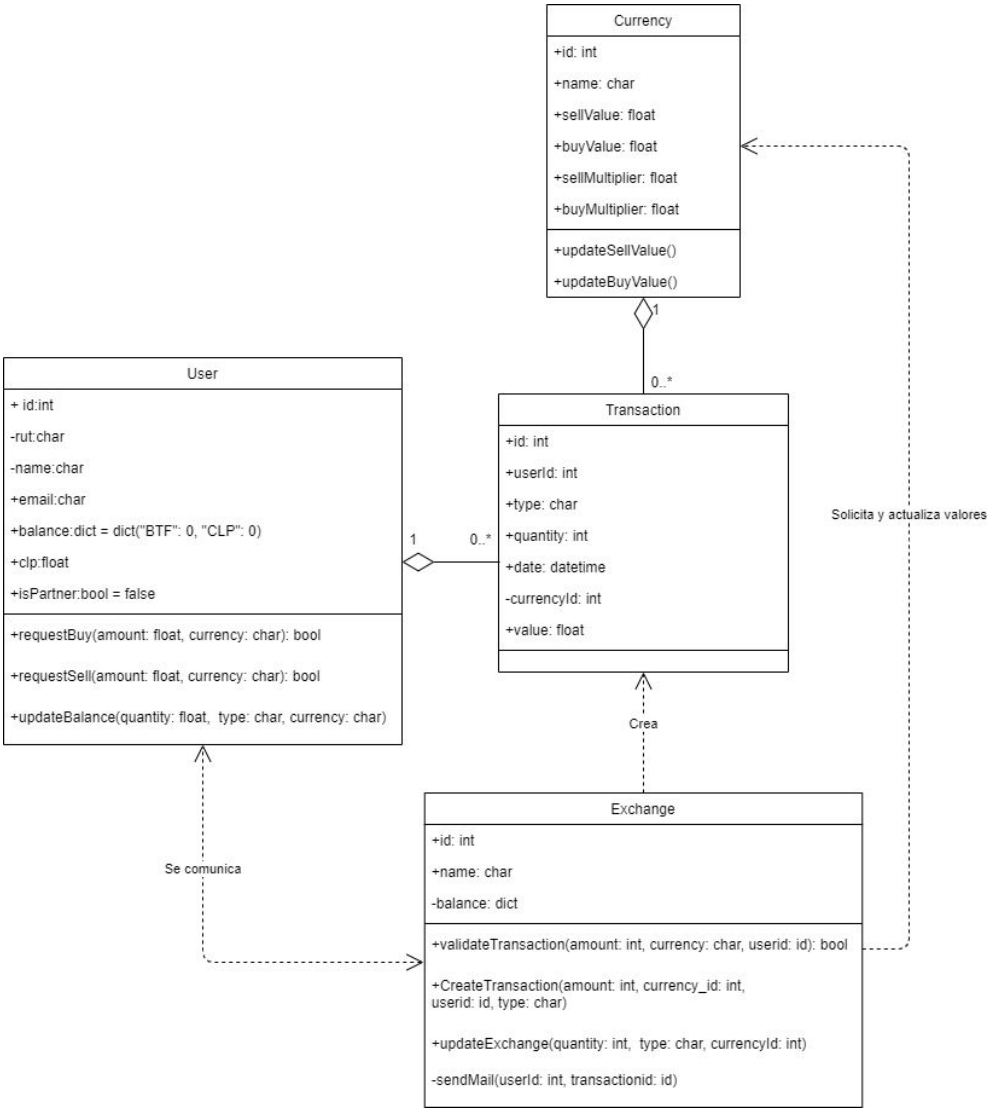
2.2 Diagrama de clases

Decidimos modelar nuestro sistema de exchange de Bitfake con 4 entidades. Una entidad *User* que se hará cargo de almacenar toda la información del usuario, cuando el usuario desee realizar una transacción *User* invocará su método *requestBuy()* o *requestSell(),* dependiendo del tipo de transacción que desee realizar. Estos métodos se comunicarán con nuestra segunda entidad, *Exchange*, la cual se hará cargo de llevar un registro de la cantidad de monedas que hay en el sistema.

Nuestra segunda entidad *Exchange* recibirá la solicitud de compra/venta realizada por *User*, y verificará que la transacción sea posible con el método *validateTransaction()*. Este método, con el objetivo de revisar que la transacción sea posible, se comunicará con nuestra tercera entidad *Currency*, la cual se hará cargo de llevar registro del valor de las monedas. El método *validateTransaction()* extraerá el valor de la moneda a transferir de *Currency* y revisará si la transacción es posible. En caso de ser posible la transacción *Exchange* llamará los métodos *CreateTransaction(), updateExchange(), sendMail()* y *updateSellValue()* o *updateBuyValue()*.

El método *CreateTransaction()* invocará a nuestra cuarta entidad *Transaction*, la cual se hará cargo de llevar registro de todas las transacciones realizadas exitosamente y añadirá la transacción exitosa al registro de transacciones llevado por esta entidad. Luego de esto, se llama al método *updateExchange()*, y se actualiza la cantidad de la moneda transferida en *Exchange.* Después se llama al método *updateSellValue()* o *updateBuyValue()* de *Currency,* dependiendo de si se realizó una compra o una venta, y se actualiza el valor de mercado de la moneda transferida. Finalmente, *Exchange* llama a *sendMail(),* este método le informa al usuario que su transacción ha sido exitosa a través de un mail, además llama la función *updateBalance()* de *User,* el cual actualizará las cantidades de monedas del usuario.

Los atributos y métodos de las cuatro clases mencionadas se desglosan en el diagrama de clases, para luego proceder a un desglose con una explicación breve punto por punto.



- **User:**
  - Atributos:
    - id: int
    - rut: char
    - name: char
    - email: char
    - balance: dict -> Diccionario que almacena los saldos de las distintas monedas del usuario, donde la *key* corresponde al nombre de la moneda y el *value* corresponde al saldo. La razón de esta modelación es para permitir futuras adiciones de nuevas monedas al sistema.
    - isPartner: bool = false

- Métodos:
  - requestBuy(amount: int, currency: char): bool: Solicitud del usuario de comprar una cantidad *amount* de monedas del tipo *currency*.
  - requestSell(amount: int, currency: char): bool: Solicitud del usuario de comprar una cantidad *amount* de monedas del tipo *currency*.
  - updateBalance(quantity: int, type:char, currency: char): Método para cambiar los saldos del usuario de la moneda *currency* en *quantity* unidades, aumentando o disminuyendo dependiendo del *type*.

- **Transaction:**
  - Atributos:
    - id: int
    - userId: int -> Id del usuario que realizó la compra.
    - type: char -> Tipo de transacción, es decir si se compró o se vendió.
    - quantity: float -> Cantidad de la moneda *currency* que se compró/vendió,
    - date: datetime
    - currencyId: int -> Id de la moneda *currency* involucrada en la transacción.
    - value: float -> Valor al cual se realizó la transacción de la moneda *currency.*

- **Currency**
  - Atributos:
    - id: int
    - name: char
    - sellValue: float -> valor actual del mercado para vender la moneda
    - buyValue: float-> valor actual del mercado para comprar la moneda
    - sellMultiplier: float -> Factor de cambio en el precio de la moneda *currency* por cada compra. La razón de esta modelación es para permitir eventuales cambios en el multiplicador de BTF (actualmente 0.03) y para poder registrar un factor de cambio en alguna nueva moneda creada en el futuro.
    - buyMultiplier: float -> Factor de cambio en el precio de la moneda *currency* por cada venta. La razón de esta modelación es para permitir eventuales cambios en el multiplicador de BTF (actualmente 0.03) y para poder registrar un factor de cambio en alguna nueva moneda creada en el futuro.

- Métodos:
  - updateSellValue(): Método para cambiar el valor de venta de la currency en base al multiplicador
  - updateBuyValue(): Método para cambiar el valor de compra de la currency en base al multiplicador

- **Exchange:**
  - Atributos:
    - id: int
    - name: char
    - balance: dict: Diccionario que almacena las cantidades de las distintas monedas que hay en el sistema, donde la *key* corresponde al nombre de la moneda y el *value* corresponde al saldo. La razón de esta modelación es para permitir futuras adiciones de nuevas monedas al sistema.

2.3 Vista y flujo de uso

El flujo de uso de la aplicación web parte de la página de inicio en donde se muestran gráficos con los precios históricos de las criptomonedas que se transan en el exchange, esto se puede ver en la figura 1. Gran parte del flujo de la aplicación se lleva a cabo mediante la barra superior en donde están implementadas la mayoría de features.

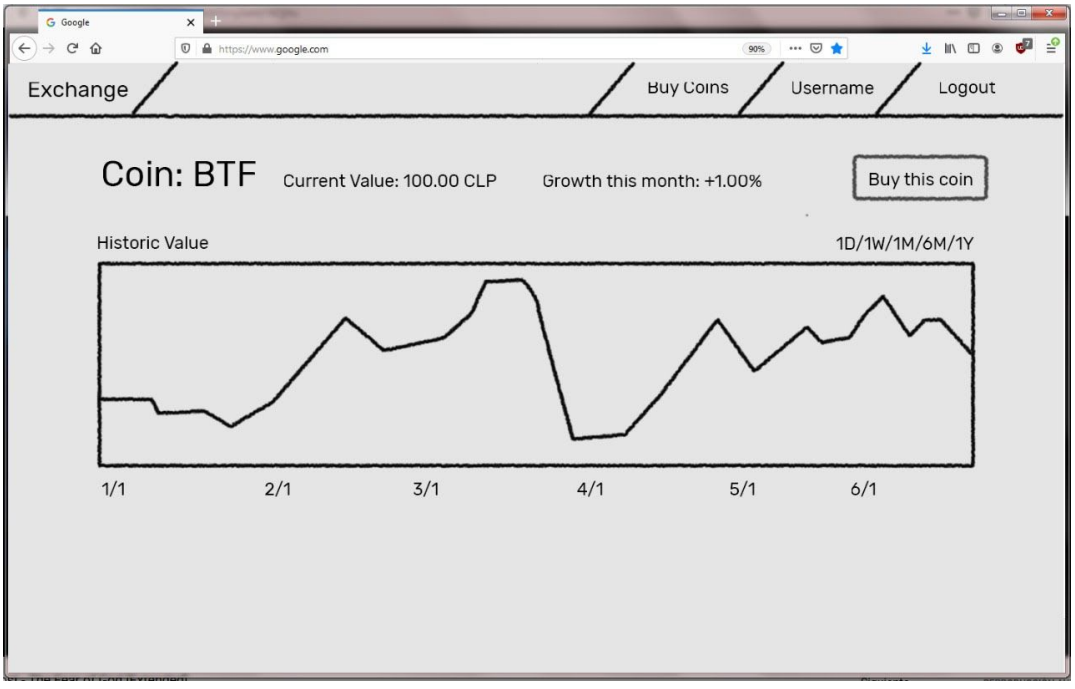


Figura 1: Vista de la página inicial con el gráfico del valor histórico de BTF.

En el caso de que el usuario no esté logueado en su cuenta, este no puede realizar ninguna acción. Para entrar a su cuenta se selecciona en la parte superior derecha Sign Up en el caso de no tener cuenta, o Login para ingresar a una cuenta ya existente.

Al seleccionar Sign Up envía al usuario a un formulario en donde debe ingresar todos sus datos para crear su cuenta. Si selecciona Login, se abre el formulario que se puede ver en la figura 2, donde el usuario ingresa su Username y Password de su cuenta previamente creada.

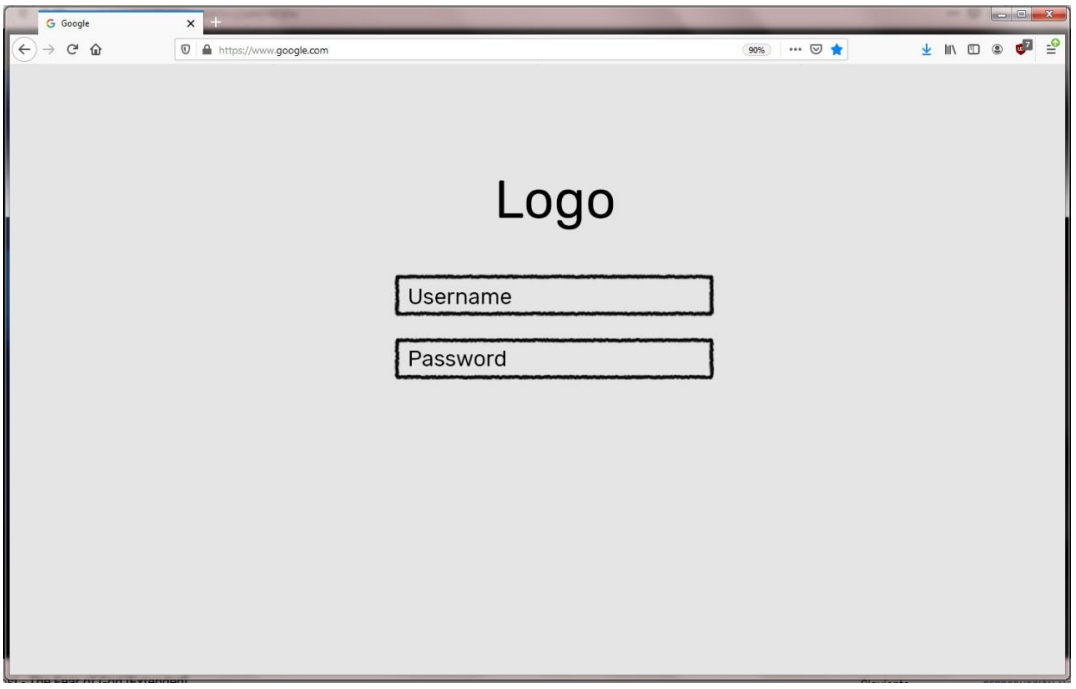


Figura 2: Vista del Login de usuario.

Una vez que el usuario ya ingresó a su cuenta puede ingresar a su perfil haciendo click en su Username en la barra superior. Una vez dentro del perfil se pueden ver el historial de transacciones, su portafolio y la cantidad actual de dinero ingresado en la cuenta, tal como se ve en la figura 3.

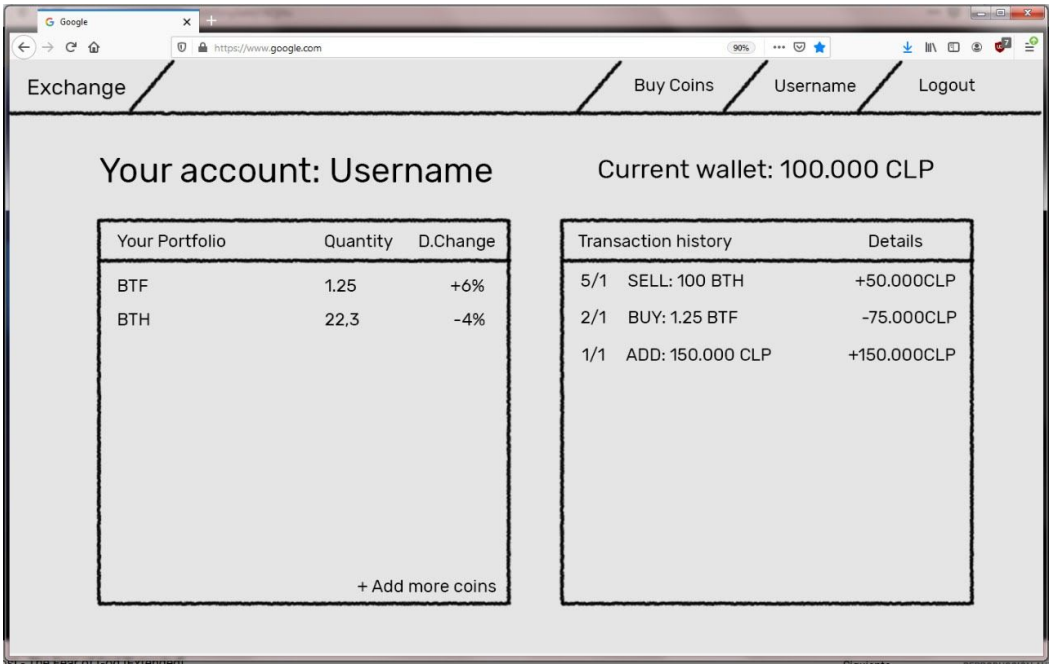


Figura 3: Vista del perfil del usuario con el historial de transacciones.

Ya con el usuario ingresado este puede comprar o vender cualquiera de las criptomonedas que ofrece el exchange tal como se ve en la figura 4. Se puede acceder a esto mediante el botón de la barra superior Buy Coins.

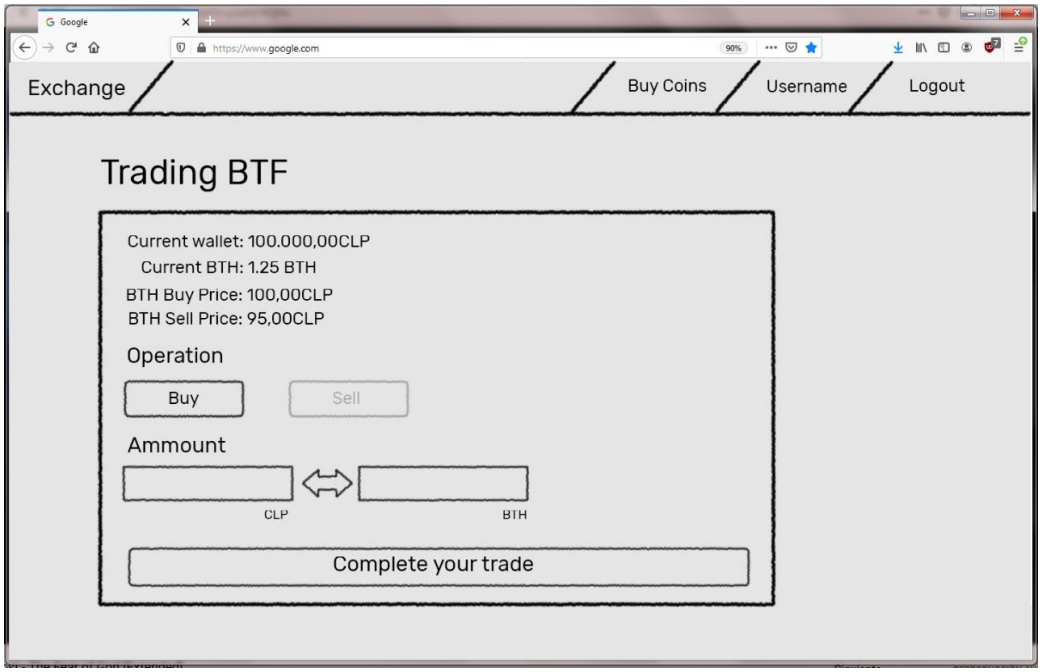


Figura 4: Vista de Compra y Venta de las diferentes criptomonedas

Para salir de la cuenta se debe seleccionar la opción Logout de la barra superior que solo está disponible si ya se encuentra logueado en una cuenta, luego se pide una confirmación, la cual de ser confirmada el usuario sale de su cuenta.