INSTITUTO TECNOLÓGICO DE BUENOS AIRES – ITBA

ESCUELA DE INGENIERÍA Y GESTIÓN

CRYPTOBOT

Creación de base de datos y estudio de series temporales de criptomonedas

AUTOR/ES: Oviedo Candelaresi, Fernán Darío (Leg. N° 56651)

Fratoni, Axel (Leg. N° 55749)

DOCENTE/S TITULAR/ES O TUTOR/ES: Parisi, Daniel

TRABAJO FINAL PRESENTADO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN INFORMÁTICA

Lugar: Buenos Aires

Fecha: 18 de diciembre de 2019

1. Resumen

Existe una plataforma de intercambio de criptomonedas llamada HitBTC que provee una API para interactuar con los mercados. El objetivo de este trabajo fue desarrollar un sistema de software que permitiera recabar información del mercado de forma automática a lo largo del tiempo, analizar esta información para caracterizar el comportamiento del mercado e implementar estrategias de decisión automática para ejecutar operaciones de compraventa.

2. Introducción

2.1. Criptomonedas y *exchanges*

El 18 de agosto de 2008 se registró el dominio bitcoin.org; el 31 de octubre de 2008 se publicó el *whitepaper* de Bitcoin [1]; el 3 de enero de 2009 se minó el primer bloque [2]. Estos sucesos dieron origen a la primera criptomoneda. Diez años después, existen en el mundo miles de criptomonedas [3]. Las criptomonedas son activos digitales que funcionan como medios de intercambio y utilizan criptografía para controlar su creación y gestión. Cada criptomoneda se implementa de una forma distinta. El funcionamiento de las criptomonedas con sus respectivos detalles técnicos está por fuera del alcance de este documento y no es necesario su entendimiento para comprender el presente trabajo.

Con el surgimiento de diversas criptomonedas aparecieron plataformas llamadas *exchange*. Estas plataformas son mercados digitales que permiten intercambiar criptomonedas por otras o por dinero fiduciario. Existen hoy en día cientos de *exchanges* [4]. Estas plataformas difieren entre sí en varios aspectos como el nivel de anonimato que ofrecen, las comisiones, las criptomonedas disponibles, los medios de pago soportados, la seguridad, el volumen de operaciones, la disponibilidad de una API y otras cuestiones. Los *exchange* ofrecen una serie de ventajas para el usuario: centralizan la operación de criptomonedas de distinta naturaleza (de otra forma, haría falta una herramienta distinta para cada una), permiten intercambiar una criptomoneda por otra en una sola operación atómica, ofrecen la posibilidad de intercambiar moneda digital de forma mucho más rápida de lo que quizás soporta la infraestructura subyacente de la criptomoneda en cuestión (por ejemplo, en Bitcoin se validan y agregan bloques de transacciones cada 10 minutos aproximadamente) y permiten al usuario abstraerse del funcionamiento de las diversas criptomonedas y pensarlas como cualquier otro activo financiero.

Uno de los *exchange* de criptomonedas más populares es HitBTC [5], que se encuentra operando desde 2013. El principal atractivo de esta plataforma para este trabajo es el API que ofrece [6]. Gracias a esta API es posible automatizar la interacción con los diversos mercados que ofrece la plataforma. Si bien HitBTC no ofrece la posibilidad de cambiar criptomonedas por dinero fiduciario o viceversa, sí es posible hacer operaciones con el Tether, una criptomoneda estable que siempre vale un dólar.

2.2. Funcionamiento de un mercado

Un mercado es el lugar donde se encuentran la oferta y la demanda de bienes o servicios. HitBTC permite el acceso a diferentes mercados de criptomonedas. Cada par de criptomonedas que se puede intercambiar en HitBTC es un mercado (por ejemplo, el mercado ETH/BTC es el mercado donde se intercambian Ethers por Bitcoins). Explicamos a continuación algunos conceptos sobre el funcionamiento del mercado que no son exclusivos de HitBTC, sino que son conceptos financieros presentes en diversos mercados de diferente naturaleza.

2.2.1. Oferta y demanda

En un mercado donde se intercambia algún activo por dinero fiduciario se suele identificar como la oferta (llamada *ask*) a la intención de dar el activo y recibir dinero (venta), mientras que se suele identificar como demanda (llamada *bid*) a la intención de dar dinero y recibir el activo (compra). Cada vez que alguien quiere intercambiar una criptomoneda por otra lo hace dentro del mercado correspondiente. Sin embargo, mercados de criptomonedas no está tan claro a priori quién es la oferta y quién la demanda. Tomamos como ejemplo el mercado BTC/USDT, donde se intercambian Bitcoins por Tethers. Toda oferta de Bitcoins puede ser vista como una demanda de Tether y viceversa. Esto dificultaría entender a qué se refiere uno cuando habla de compra o venta dentro del mercado. Es por esto que para cada mercado HitBTC le da un orden a las criptomonedas que determina cuál es el activo que se compra o vende y cuál es la moneda de cambio. En el caso del mercado BTC/USDT, el Bitcoin es el activo y el Tether la moneda de cambio. A cada mercado se lo identifica con un símbolo (al mercado de Bitcoin y Tether se lo identifica con “BTCUSD”).

2.2.2. Órdenes

Cada vez que alguien quiere operar en el mercado, debe emitir una orden de compra o de venta. Existen dos tipos principales de órdenes, tanto de compra como de venta. Por un lado están las *limit orders*. Estas órdenes manifiestan la intención de compra o venta de un determinado volumen a un precio que no sea superior o inferior (respectivamente) a un determinado valor. Por ejemplo, una orden de este estilo podría expresar “comprar 10 Bitcoins a un precio que no sea superior a 9.000 Tether” o “vender 5 Bitcoins a un precio que no sea inferior a 9.500 Tether”. Estas órdenes son almacenadas en una estructura contable llamada *order book* (libro de órdenes) y tienen vigencia hasta que se concreten o se cancelen. Por otro lado están las *market orders*. Estas órdenes manifiestan la intención de compra o venta de un determinado volumen al mejor precio disponible. Por ejemplo, “comprar 2 Bitcoins al precio más bajo posible” o “vender 3 Bitcoins al precio más alto posible”. Estas órdenes se ejecutan contra el libro de órdenes del mercado y son de ejecución inmediata.

Sum BTC	Amount	Bid	Ask	Amount	Sum BTC
0.12000	0.12000	9240.89	9243.16	0.00156	0.00156
0.52000	0.40000	9240.42	9243.20	2.00000	2.00156
1.22000	0.70000	9239.02	9243.34	0.00810	2.00966
3.22000	2.00000	9239.01	9246.28	0.15000	2.15966
3.51000	0.29000	9238.91	9247.20	0.20000	2.35966
3.62523	0.11523	9237.89	9247.21	0.15100	2.51066
3.81440	0.18917	9235.98	9248.10	0.07500	2.58566
3.88940	0.07500	9234.62	9248.11	0.34024	2.92590
5.03940	1.15000	9234.61	9248.15	0.24900	3.17490
5.05940	0.02000	9233.84	9248.30	0.42136	3.59626

Figura 1: parte central del libro de órdenes del mercado de Bitcoins y Tethers en HitBTC

2.2.3. Candlestick

Si para un mercado se realiza un gráfico de precio de operación (compra o venta) en función del tiempo, se obtiene un gráfico que representa una serie temporal de la evolución del precio del activo. Cada operación que se concrete agregaría un punto en este gráfico. Sin embargo, un gráfico en el que hay una operación por punto en un mercado de muchas operaciones puede dificultar su visualización para períodos largos de tiempo. Existe entonces el *candlestick* o gráfico de velas. Éste es una versión agregada por tiempo de la serie temporal de precios. Para construirlo se fracciona el eje temporal (abscisas) en intervalos homogéneos. Para cada intervalo, se grafica una vela. Una vela es un rectángulo verde o rojo acompañado de una línea vertical que lo atraviesa por el medio. Esta vela funciona como un resumen del movimiento del precio del activo durante el intervalo de tiempo al que corresponde. De ella se puede extraer el precio al inicio y al final del intervalo así como los precios máximo y mínimo a los que se concretó una operación en el intervalo. Los precios de apertura y cierre son el borde superior e inferior del rectángulo; si la vela es color verde, el precio de apertura es el de abajo y el de cierre es el de arriba, y al revés si es roja. El precio máximo y mínimo son el punto más alto y más bajo respectivamente de la línea vertical que atraviesa el rectángulo. HitBTC permite visualizar el gráfico de velas de los diferentes mercados con distintos intervalos de agregación temporal.

Figura 2: gráfico de velas del mercado de Bitcoin y Tether en intervalos de 15 minutos en HitBTC

2.2.4. Market depth chart

Para entender el estado actual del mercado hay que mirar el libro de órdenes. Sin embargo, la forma de visualizar el libro de órdenes no es graficándolo directamente, sino graficando una versión agregada del mismo. Este gráfico se llama *market depth chart*. En el libro de órdenes el máximo precio al que alguien está dispuesto a comprar un determinado volumen es siempre inferior al mínimo precio al que alguien está dispuesto a vender otro volumen (si dos órdenes se solaparan se liquidarían de forma inmediata). Con esto en mente, el gráfico consiste en tomar cada precio posible y analizar en qué lado del libro de órdenes cae o si cae en el medio. Si cae en el medio (entre el máximo precio de demanda y el mínimo precio de oferta) no se grafica nada para ese punto; si cae del lado de la demanda (*bid*) se pone como ordenada el volumen total de demanda que hay por encima de ese precio; si cae del lado de la oferta (*ask*) se pone como ordenada el volumen total de oferta que hay por debajo de ese precio. Para facilitar la visualización, el lado de la demanda se grafica en verde y el lado de la oferta en rojo. Además, puede que no se grafique para todos los precios posibles ni para todos los precios a los que haya alguna orden, sino que se tome un intervalo de precios fijo. Este nivel de agregación es configurable en HitBTC. A la diferencia de precio entre el mayor precio de demanda y el menor precio de oferta se lo conoce como *market spread*, y se lo ve gráficamente como la mínima distancia entre las dos curvas.

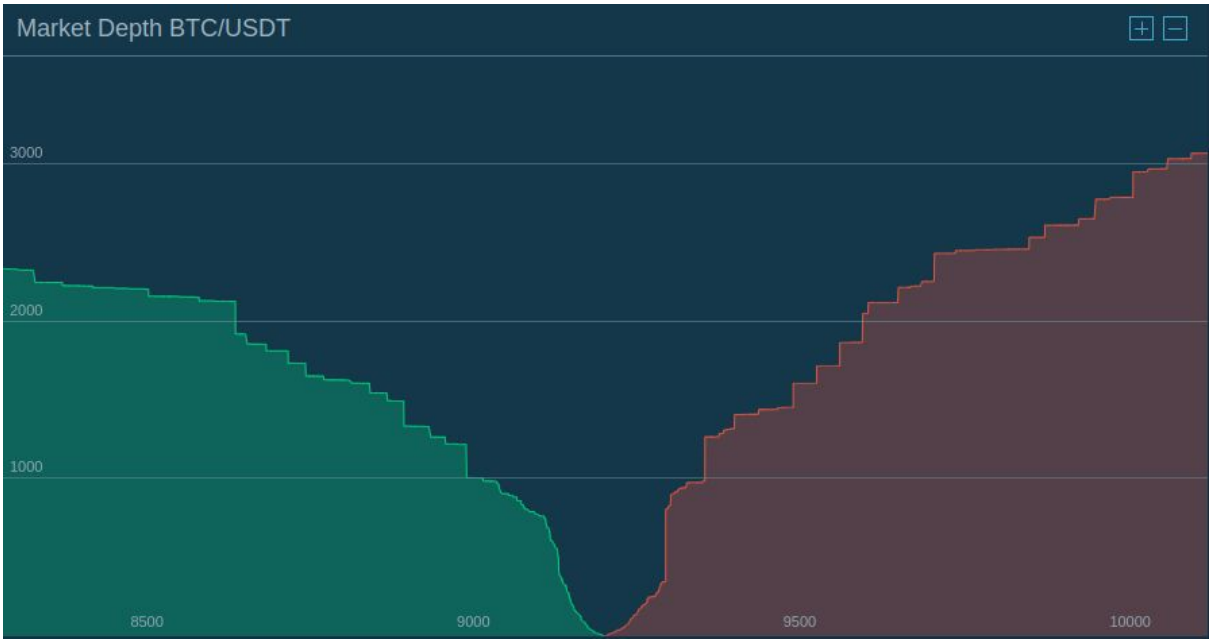

Figura 3: *market depth chart* del mercado de Bitcoin y Tether en HitBTC

3. Cryptobot

Este proyecto consiste en la creación de un sistema informático que permita interactuar con HitBTC para poder almacenar la información del mercado, analizar esta información y poder ejecutar estrategias de compraventa contra el mercado.

La información de series temporales históricas disponible sobre la evolución de criptomonedas es incompleta, errónea o paga. Alguna información ni siquiera está disponible de forma histórica (por ejemplo, en HitBTC sólo puede obtenerse el libro de órdenes actual). Ergo, una de las partes del trabajo consiste en crear un software que registre la información en vivo del mercado y la vaya almacenando en una base de datos. En particular, interesa guardar el historial de transacciones para cada mercado a estudiar y el estado del libro de órdenes a lo largo del tiempo. Con la información recolectada se puede tratar de caracterizar al mercado para intentar predecir la evolución del mismo y posiblemente implementar alguna estrategia de inversión. Otra parte del trabajo entonces está dedicada a construir un sistema que permita implementar estrategias que reciban la información tanto en vivo como histórica del mercado, tomen decisiones de compraventa y ejecuten órdenes contra el mercado.

Visto que este sistema corre estrategias programadas sin intervención directa del usuario, es fundamental que el sistema pueda ser controlado de manera remota. También resulta útil que el sistema dé notificaciones al usuario sin que éste deba estar activamente fijándose qué operaciones está realizando o si hubo algún problema técnico. Por último, es fundamental que el sistema deje registro de toda la información que pueda ser útil para encontrar potenciales errores o diagnosticar situaciones de falla.

4. Arquitectura

Para construir el sistema de software que alcance el objetivo propuesto se planteó la siguiente arquitectura.

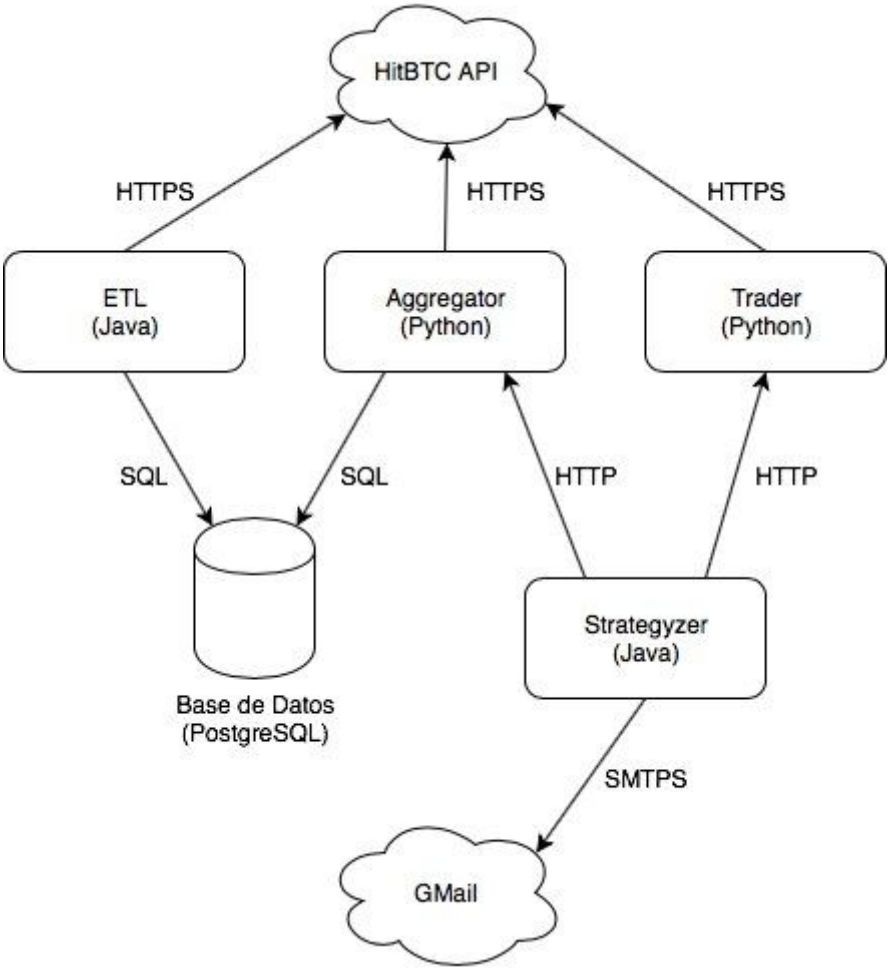

Figura 4: arquitectura de alto nivel del sistema de software construido

Llevar adelante esta arquitectura implicó construir cuatro módulos de software: ETL, *Aggregator*, *Strategyzer* y *Trader*. Cada uno de estos módulos tiene un rol y una responsabilidad particulares dentro del sistema de software.

4.1. ETL

Este módulo toma su nombre de *Extract, Transform, Load*. Es el nombre del procedimiento más estándar en el mundo de la informática para tomar datos de una o diversas fuentes, transformarlos y almacenarlos en una base de datos. Este módulo se ocupa de seguir la información en vivo de HitBTC e ir almacenando en una base de datos relacional PostgreSQL la información de todas las operaciones que se van ejecutando así como el libro de órdenes completo de forma periódica. De qué mercado se toma la información es configurable y el mismo módulo puede tomar información de más de un mercado a la vez.

4.1.1. Funcionamiento

Para obtener la información de las operaciones, el ETL pide a HitBTC cada 1 minuto las operaciones realizadas en el último minuto (esta información siempre está disponible). Para obtener el libro de órdenes, es necesario configurarle una periodicidad al ETL. Esta periodicidad puede ser diferente para cada mercado. Por ejemplo, una configuración podría expresar “almacenar el libro de órdenes del mercado BTC/USDT cada 1 minuto y el del mercado ETH/USDT cada 30 segundos”, y el mismo módulo se ocuparía de llevar adelante ambas tareas junto con el almacenamiento de todas las operaciones concretadas para ambos mercados.

A medida que se producen errores o eventos de particular interés, el ETL va dejando registro de éstos. A estos registros, como en cualquier software, los llamamos *logs*. Existen en el ETL dos tipos de *logs*: los de negocio y los técnicos. Los *logs* de negocio expresan los eventos sucedidos en un lenguaje que le permita entender a alguien que no pertenece al mundo técnico qué sucedió y qué implicancias tiene sobre el proceso de ETL. En cambio, los *logs* técnicos contienen información técnica detallada sobre lo sucedido que permiten a un informático resolver potenciales problemas. Cada evento de interés produce un *log* de negocio y un *log* técnico, pero además este último deja un archivo aparte con el detalle de la excepción ocurrida (en caso de que se produzca algún error). Por ejemplo, un *log* de negocio podría expresar “no se pudo almacenar el libro de órdenes del mercado BTC/USDT del 9 de septiembre de 2019 a las 12:35:00”, mientras que el correspondiente *log* técnico expresaría “falló la actualización del *prepared statement* de inserción libro de órdenes a la base de datos porque se cayó la conexión al puerto 5432 y no pudo ser restablecida (ver archivo de excepción *exception.txt*)”, y el archivo de excepción mencionado contendría todo el detalle técnico de cómo la ejecución del código del ETL llevó a esa situación.

4.1.2. Implementación

El ETL se encuentra implementado en Java 9. El siguiente diagrama da un paneo del funcionamiento de este módulo.

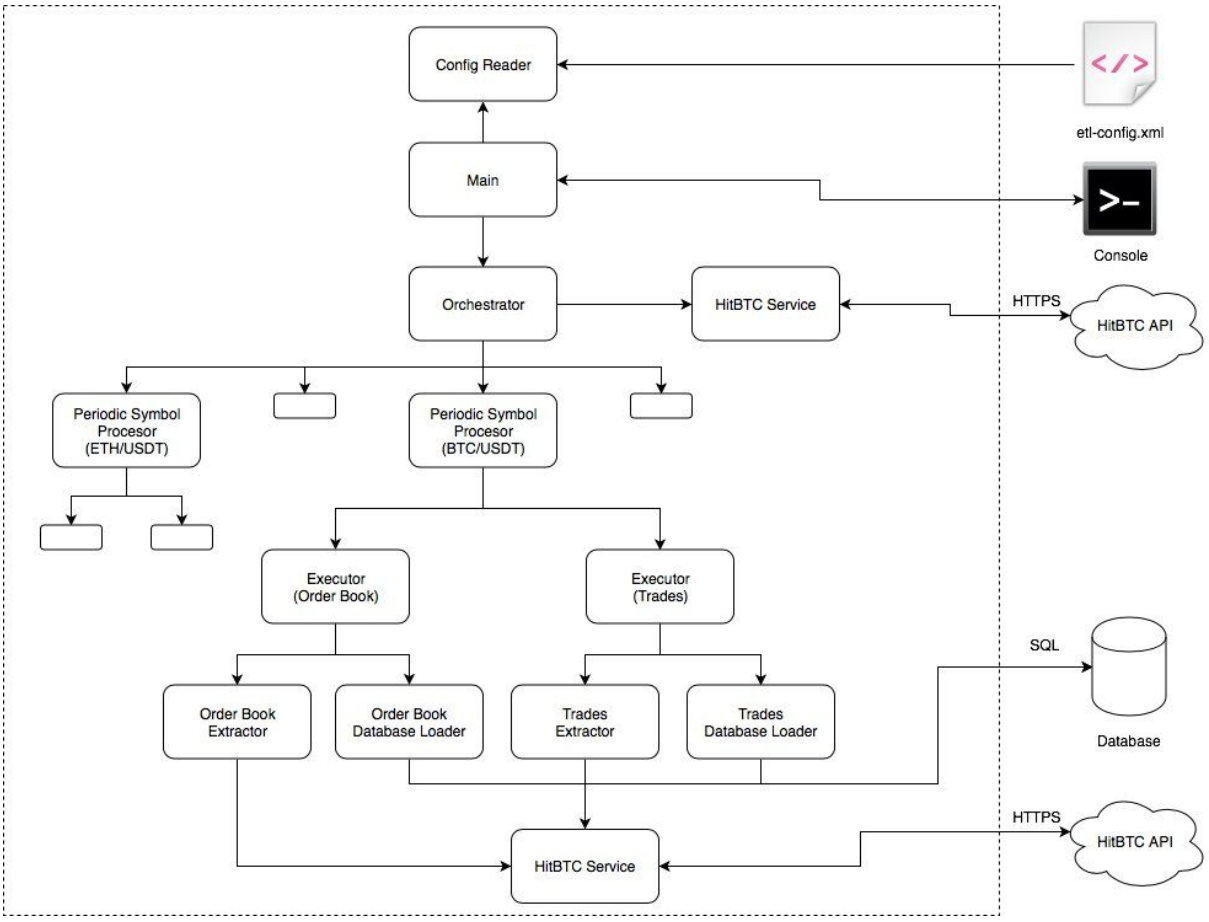

Figura 5: arquitectura del módulo ETL

Se detalla a continuación el rol de las partes relevantes del ETL.

- *Main*: es el punto de acceso al programa. Se encarga de instanciar al *ConfigReader* y al *Orchestrator*, pasar la configuración del primero al segundo, poner a correr al *Orchestrator* y quedarse escuchando en la consola comandos. El único comando que entiende el ETL es “STOP” que le indica frenar toda la operatoria de forma armoniosa y terminar el programa.
- *ConfigReader*: es el encargado de transformar el archivo de configuración escrito en XML en un objeto *ETLConfig* de Java y validar algunos aspectos. En el archivo de configuración se especifica los mercados a seguir, la frecuencia de extracción del libro de órdenes y la información necesaria para conectarse a la base de datos.
- *Orchestrator*: es quien maneja qué símbolos se están siguiendo o se dejan de seguir. A partir de la configuración, para cada mercado a seguir verifica la existencia del mismo en HitBTC y pone a correr un *PeriodicSymbolProcessor* para cada uno.
- *PeriodicSymbolProcessor*: cada instancia de esto se encarga del procesamiento periódico de toda la información de un mercado. Para esto pone a correr dos *Executors*, uno para las operaciones concretadas y otro para el libro de órdenes. Cada *Executor* es lanzado en un hilo de ejecución (*thread*) separado, de forma que todos puedan correr concurrentemente.
- *Executor*: es el responsable de orquestar la extracción del API y la carga en base de datos de un dato de interés de un mercado particular. Se ocupa de invocar periódicamente a un *Extractor* (que puede ser de operaciones o del libro de órdenes)

y pasarle la información extraída al correspondiente *Loader* para que la cargue en la base de datos.

- *TradesExtractor* y *OrderBookExtractor*: cada uno es responsable por extraer los datos de interés de HitBTC. Para ello utilizan un *HitBTCService* que les provee acceso al API.
- *HitBTCService*: permite a quien lo use obtener la información del API en objetos de Java. Se ocupa de conectarse al API por internet, hacer los pedidos sobre HTTPS y transformar los resultados en clases de Java (como *TradesData*).
- *TradesDatabaseLoader* y *OrderBookDatabaseLoader*: cada uno es responsable de almacenar los datos ya extraídos del API en la base de datos, comunicándose con ésta por medio de SQL.
- *BusinessLogger* y *TechnicalLogger*: son los responsables respectivamente de dejar *logs* de negocio y técnicos (no se muestran en el diagrama porque todos los módulos tienen acceso a ellos y su agregado entorpecería la interpretación del mismo). El *BusinessLogger* deja los *logs* en un archivo *etl.log*. El *TechicalLogger* deja los *logs* en la consola y cada excepción en un archivo separado dentro del directorio *log*.

4.2. Base de datos

La base de datos utilizada es una base de datos relacional PostgreSQL. A continuación se detalla un diagrama del modelo relacional que expresa el diseño lógico de los esquemas.

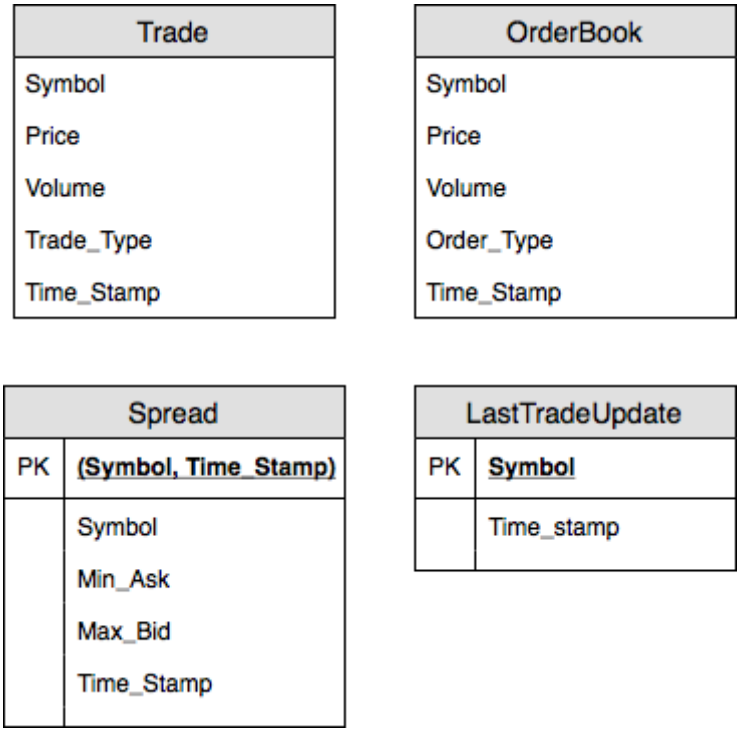

Figura 6: diagrama del diseño lógico en modelo relacional de la base de datos

Las tablas o relaciones involucradas son:

- *trade*: en esta tabla se guardan las operaciones para cada mercado que se está siguiendo, dejando asentado el símbolo del mercado, el precio de operación, el

volumen de operación, el tipo de operación (es decir, si la *market order* que concretó la operación fue de compra o de venta) y la fecha y hora de la operación.

- *orderbook*: en esta tabla se almacena toda la información de todos los libros de órdenes que se guardan. Cada registro representa una *limit order* de algún mercado en un determinado momento.
- *spread*: esta tabla contiene un registro por cada libro de órdenes que hay almacenado. Su finalidad es proveer un mecanismo mucho más eficiente de obtener cierta información que de otra forma habría que buscar en la tabla *orderbook*. Así, por cada libro de órdenes, esta tabla contiene el máximo precio de demanda y el mínimo precio de oferta. Esto permite, por ejemplo, obtener el *market spread* mucho más rápido.
- *lasttradeupdate*: esta tabla indica la fecha y hora de la última vez que se actualizó la tabla *trade* para cada símbolo. Esto permite a quienes usan la información en la base de datos distinguir un silencio (momentos en que no hay ninguna operación) de una falta de datos almacenados. Esta tabla la actualiza el ETL cada vez que almacena operaciones y la leen otros módulos.

Junto con las tablas hay tres índices que resultan fundamentales para resolver las consultas a la base de datos en tiempos razonables. Éstos son:

- *trade_index*: es un índice árbol B compuesto en la tabla *trade* sobre las columnas *symbol* y *time_stamp*; permite acelerar consultas como “las operaciones del mercado BTC/USDT de la última hora”
- *orderbook_index*: es un índice árbol B compuesto en la tabla *orderbook* sobre las columnas *symbol* y *time_stamp*; permite acelerar consultas como “el libro de órdenes más reciente del mercado BTC/USDT”
- *spread_pkey*: es un índice árbol B compuesto en la tabla *spread* sobre las columnas *symbol* y *time_stamp*; permite acelerar consultas como “la evolución del *spread* del mercado BTC/USDT en el último día”

Además de almacenar y dar acceso a la información, la base de datos se ocupa de borrar periódicamente la información vieja. Esto lo hace corriendo procesos una vez al día que borran las operaciones y los libros de órdenes que tienen más de 30 días. Estos procesos están implementados con *pg_cron* [7] y son fundamentales para evitar que la base de datos llene el disco del servidor y el sistema colapse.

4.3. Aggregator

Este módulo es el encargado de agregar la información almacenada en la base de datos mediante consultas SQL y obtener información en tiempo real de *HitBTC* para satisfacer las demandas del *Strategyzer*. Es un módulo pasivo y sólo responde a consultas. También consta de dos submódulos llamados *Candlestick* y *Orderbook*, encargados de generar series temporales de *candlestick* en tiempo real y extraer libros de órdenes respectivamente.

4.3.1. Funcionamiento

El *Aggregator* es una aplicación web que expone una *API REST* mediante la cual es posible realizarle consultas específicas utilizando los diferentes *endpoints* expuestos. Cada uno de éstos está diseñado con el objetivo de satisfacer las necesidades de las distintas estrategias implementadas en el *Strategyzer*. Estas consultas son realizadas mediante HTTP. Por cada consulta que realiza el *Strategyzer* (o quien lo use), el *Aggregator* realiza pedidos sobre HTTPS al API de HitBTC y/o consultas SQL a la base de datos. Esto permite al *Strategyzer* abstraerse del origen de los datos y simplemente utilizarlos para su toma de decisión.

La configuración necesaria para su funcionamiento, como las credenciales de la base de datos y el puerto en cual escuchar las consultas, se encuentran en un archivo *JSON* llamado *config.json*.

Este módulo utiliza la consola para dejar registro de todas las llamadas realizadas a su *API*, indicando el *endpoint* utilizado, la fecha del llamado y el código *HTTP* de la respuesta.

A su vez, el *Aggregator* consta de dos scripts independientes de su modo de ejecución normal que utilizan sus métodos de agregación. El primero, llamado *Candlestick*, crea una serie temporal en tiempo real de candlesticks según los parámetros deseados, como símbolo, período de agregación y opcionalmente fechas de inicio y fin de la serie temporal. El segundo, llamado, *Orderbook*, permite extraer libros de órdenes para un símbolo determinado en un intervalo de tiempo determinado. Estos libros se encontrarán separados en archivos, agrupando aquellas órdenes correspondientes al mismo instante de tiempo.

4.3.2. Implementación

El *Aggregator* está implementado en *Python* 3 utilizando *Flask*, un framework de aplicaciones web liviano.

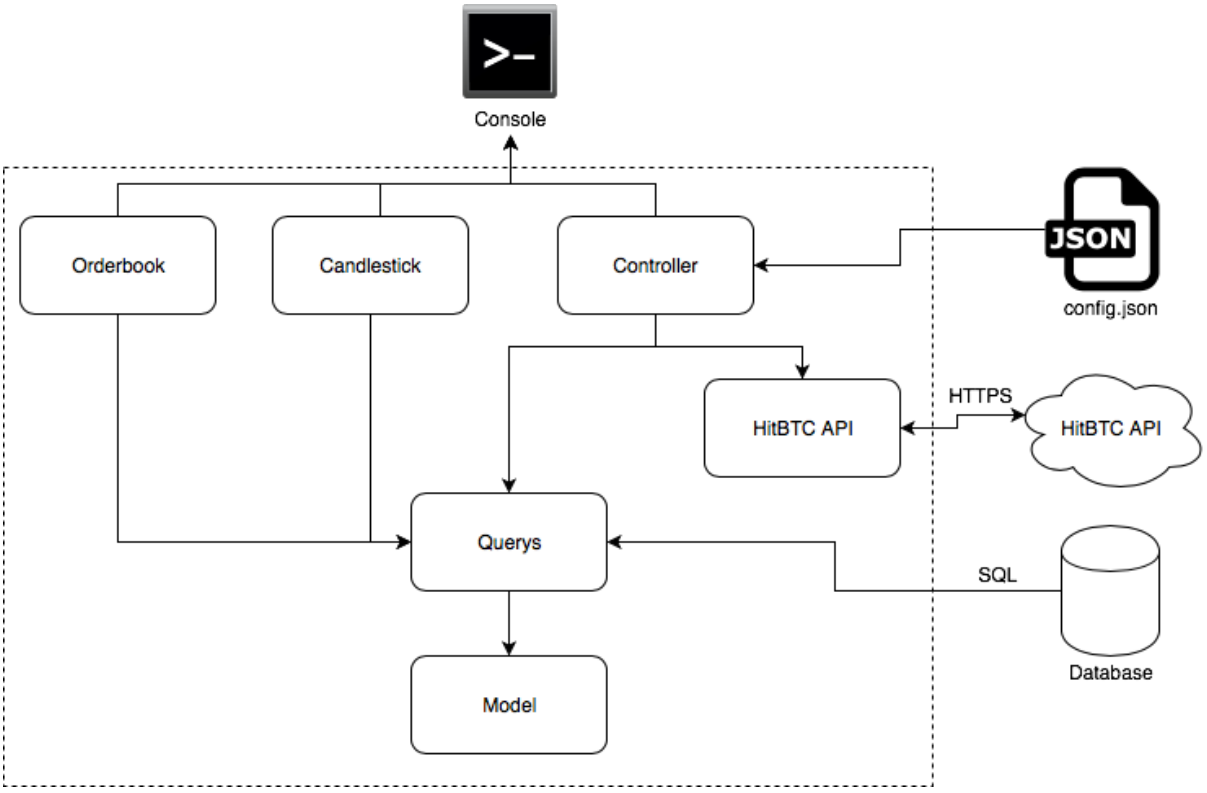

Figura 7: arquitectura del módulo *Aggregator*

El código está dividido en tres capas según la arquitectura *MVC*: controlador, servicio y modelo. La capa de controlador contiene la declaración de los *endpoints* expuestos y es el punto de acceso al módulo. La capa de servicio contiene las consultas *SQL* que se le realizan a la base de datos y los llamados a la *API* de *HitBTC*. La capa de modelo contiene una abstracción de la base de datos utilizando la librería *SQLAlchemy*, la cual es un *ORM* de *Python*.

Los submódulos *Candlestick* y *Orderbook* constan de scripts en *Python* que se ejecutan de manera independiente y utilizan la arquitectura descrita llamando a los métodos de la capa de servicio directamente, salteando la capa de controlador y la necesidad de comunicarse mediante HTTP.

4.4. Trader

Este módulo es el encargado de llevar a cabo las señales de compraventa indicadas por el *Strategyzer* creando órdenes en *HitBTC* y consultando el saldo de la cuenta en el *exchange*. Es un módulo pasivo y sólo responde a consultas.

4.4.1. Funcionamiento

El *Trader*, al igual que el *Aggregator*, es una aplicación web que expone una *API REST* mediante la cual es posible enviarle señales de compraventa y consultas de saldo usando los diferentes *endpoints* expuestos. Cada uno de estos, realiza una o más consultas a la *API* de *HitBTC,* las cuales requieren del uso de una clave privada a modo de autenticación, la cual debió haber sido habilitada previamente en la plataforma del *exchange.* Las comunicación entre el *Trader* y el *Strategyzer* se realiza mediante *HTTP*.

Este módulo permite al *Strategyzer* abstraerse del uso de la plataforma de *HitBTC* y de los *tokens* de autenticación necesarios para llevar a cabo las operaciones.

La configuración necesaria para su funcionamiento, como las credenciales de *HitBTC* y el puerto en cual escuchar las consultas, se encuentran en un archivo *JSON* llamado *config.json*.

Este módulo utiliza la consola para dejar registro de todas las llamadas realizadas a su *API*, indicando el *endpoint* utilizado, la fecha del llamado y el código *HTTP* de la respuesta.

4.4.2. Implementación

El *Trader* está implementado en *Python 3* utilizando el framework *Flask*.

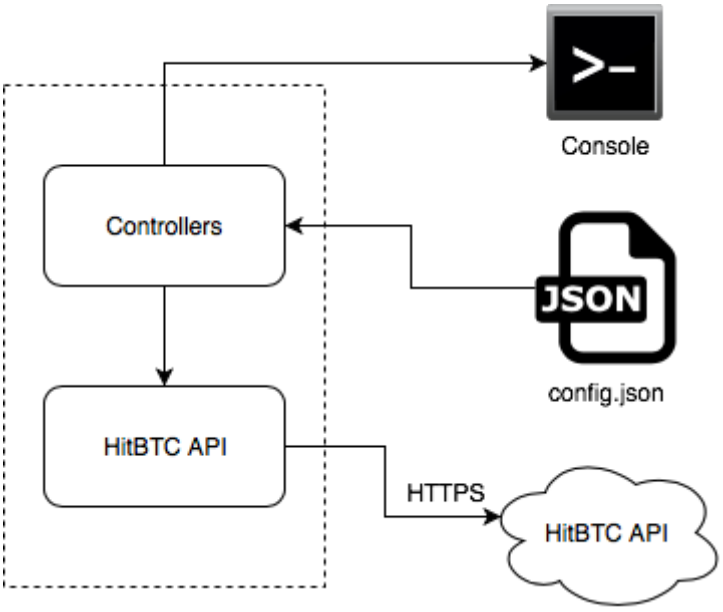

Figura 8: arquitectura del módulo *Trader*

Al igual que el *Aggregator*, este módulo tiene una arquitectura MVC formada por una capa de controlador y otra de servicio. En este caso, la capa de servicio contiene las consultas necesarias a la *API* de *HitBTC*, cuyas respuestas en algunos casos están transformadas para un uso más apropiado por parte del *Strategyzer*. La capa de controlador contiene la definición de los *endpoints*.

4.5. Strategyzer

Este módulo es el encargado de llevar adelante las estrategias de inversión. Para esto obtiene información del mercado del *Aggregator*, toma decisiones de compraventa y solicita al *Trader* que ponga las órdenes en el mercado.

4.5.1. Funcionamiento

El *Strategyzer* está diseñado de forma que la estrategia a correr sea configurable. Esto permite implementar nuevas estrategias sin tener que empezar de cero. Además de elegir una estrategia y un mercado, debe elegirse un modo de ejecución. Los modos de

ejecución son *run*, *mock-run* y *backtesting*. El modo *run* es la ejecución normal de la estrategia contra el mercado elegido. El modo *mock-run* es igual al *run* pero a la hora de poner una orden en lugar de informárselo al *Trader* la deja en el *log*; este modo sirve para poner a correr la estrategia en tiempo real sin llevar a cabo las operaciones. Por último, el modo *backtesting* simula la ejecución de la estrategia contra información histórica del mercado; para esto hay que configurarle un balance inicial, un instante de inicio y uno de finalización. Tanto la obtención de información como la puesta de órdenes la hace a través del *Aggregator* y del *Trader* respectivamente, con los cuales se comunica a través del protocolo HTTP. El *Strategyzer* es agnóstico tanto de la existencia de la base de datos como de HitBTC.

Este módulo es responsable por el balance de la cuenta. Para evitar que una mala estrategia de inversión resulte en la pérdida total de fondos, se implementa una política de *stop loss*. Cuando se inicia la ejecución de la estrategia, el *Strategyzer* pide al *Trader* el balance si está en modo *run*, o simplemente lo toma de la configuración si está en modo *backtesting*. Este balance lo pasa totalmente a la moneda de cambio según el estado del libro de órdenes. A medida que realiza operaciones, revisa este valor del balance para asegurarse de que no haya disminuido significativamente. Si este valor cae un determinado porcentaje configurable o más, se aplica la política de *stop loss* y se finaliza la ejecución de la estrategia. Por ejemplo, supongamos que se está ejecutando una estrategia en el mercado BTC/USDT con un valor de *stop loss* del 15%. Se inicia con un balance de 0,00005 BTC y 0,45 USDT y el mejor precio para vender es 8.000 USDT/BTC. Pasando todo el balance a USDT, queda un total de 0,85 USDT (0,45 USDT más el valor en USDT de los 0,00005 BTC). Pasado un tiempo y algunas operaciones, el balance es 0,00006 BTC y 0,35 USDT, con un precio de mercado de 6.000 USDT/BTC. Esto deja un balance total de 0.71 USDT, por lo que el valor del balance cayó un 16%. Al haber caído más que el valor de *stop loss*, se finaliza la ejecución de la estrategia.

El sistema de *logs* es similar al del *ETL*. Están el *log* de negocio y el *log* técnico con archivos de excepción aparte. Pero además, el *Strategyzer* agrega dos componentes más. Por un lado tiene la capacidad de enviar mails. Para esto tiene creada una cuenta de mail (cryptopf2019@gmail.com) desde la cual envía por protocolo SMTPS un mail a una lista configurable de remitentes cada vez que se concreta una operación de compra o venta, cuando se termina la estrategia por *stop loss* o cuando se produce un error. De esta forma se evita tener que estar constante o periódicamente monitoreando al *Strategyzer*. Por otro lado, a medida que realiza operaciones las va dejando reportadas en un archivo en formato CSV que hace las veces de libro contable.

4.5.2. Implementación

El *Strategyzer* se encuentra implementado en Java 9. El siguiente diagrama da un paneo del funcionamiento de este módulo.

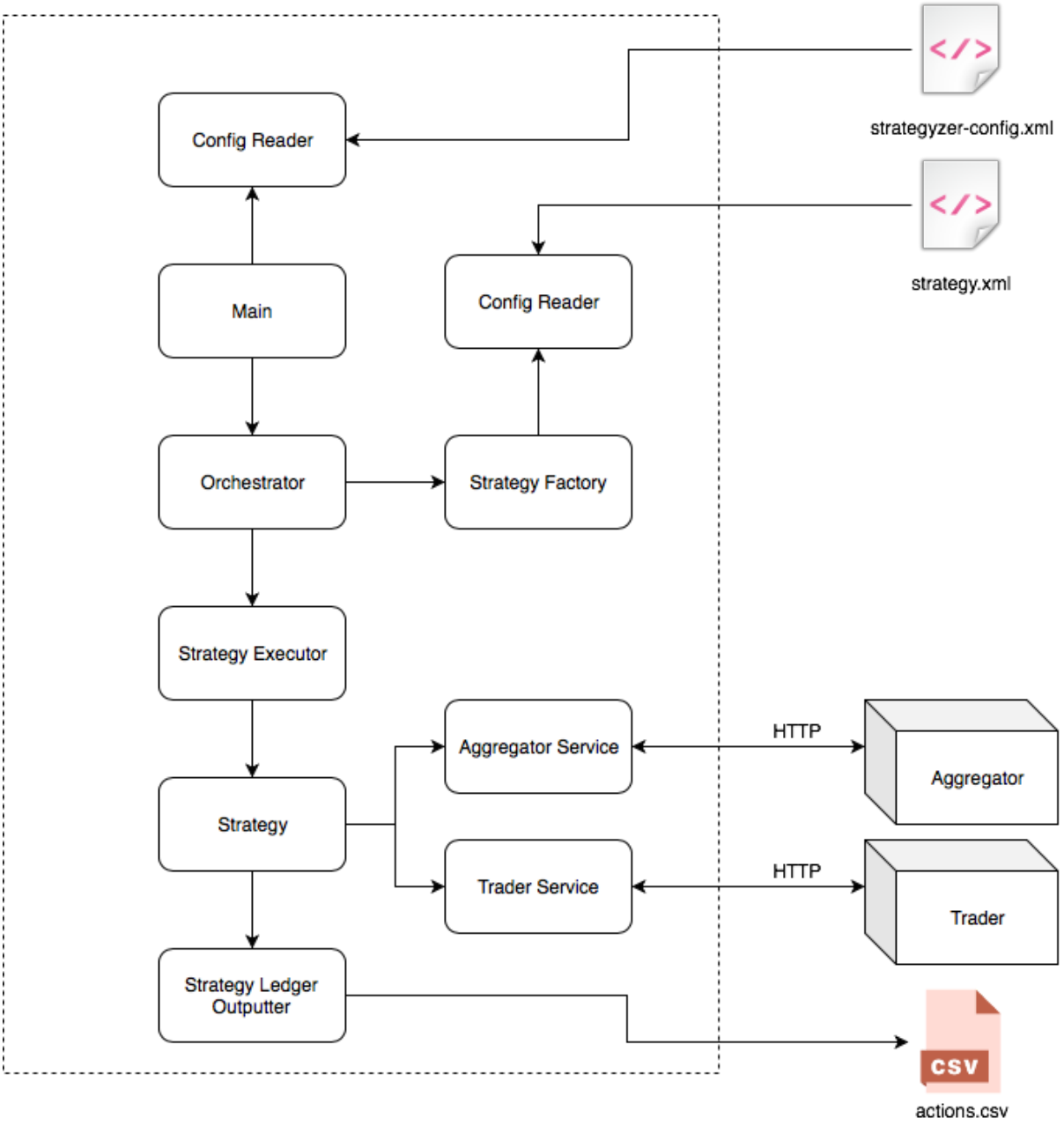

Figura 9: arquitectura del módulo *Strategyzer*

Se detalla a continuación el rol de las partes más relevantes del *Strategyzer*.

- *Main*: es el punto de acceso al programa. Se encarga de instanciar al *ConfigReader* y al *Orchestrator*, pasar la configuración del primero al segundo y poner a correr al *Orchestrator*
- *ConfigReader*: es el encargado de transformar los archivos de configuración escritos en XML en objetos de Java y validar algunos aspectos. Hay un archivo de configuración global que se transforma en un objeto *StrategyzerConfig* que indica qué estrategia correr, contra qué mercado, en qué modo, qué porcentaje de *stop loss*, cómo conectarse al *Aggregator*, al *Trader* y a GMail, qué periodo usar para *backtesting* y otras cuestiones generales. Cada estrategia puede además tener su propio archivo de configuración aparte que se transforma en otro objeto Java.

- *Orchestrator*: se encarga de instanciar un *StrategyFactory* con la configuración, tomar la estrategia y pasársela a un *StrategyExecutor* para que la ejecute en un hilo de ejecución (*thread*) aparte.
- *StrategyFactory*: es el encargado de crear la estrategia especificada por la configuración global; para ello toma esta configuración, en caso de ser necesario instancia un *ConfigReader* para leer la configuración específica de la estrategia, instancia la estrategia y la devuelve al *Orchestrator*.
- *StrategyExecutor*: toma la estrategia y la pone a correr en el modo y con la configuración especificadas
- *Strategy*: es una interfaz que representa la forma general que tienen las estrategias; las estrategias implementan esta interfaz (como lo hace *ExpMovAvgStrategy*)
- *AggregatorService*: permite a quien lo use obtener la información del *Aggregator* en objetos de Java. Se ocupa de conectarse al *Aggregator*, hacer los pedidos sobre HTTP y transformar los resultados en clases de Java (como *ExpMovAvgData*)
- *TraderService*: permite a quien lo use interactuar con el *Trader* a través de métodos y objetos de Java. Se ocupa de conectarse al *Trader*, hacer los pedidos sobre HTTP y transformar los resultados en clases de Java
- *StrategyLedgerOutputter*: a medida que se realizan operaciones, la estrategia las comunica a este objeto para que las vaya dejando asentadas en un archivo CSV
- *BusinessLogger* y *TechnicalLogger*: son los responsables respectivamente de dejar *logs* de negocio y técnicos (no se muestran en el diagrama porque todos los módulos tienen acceso a ellos y su agregado entorpecería la interpretación del mismo). El *BusinessLogger* deja los *logs* en la consola y envía algunos *logs* por mail. El *TechicalLogger* deja los *logs* en un archivo *strategyzer.log* y cada excepción en un archivo separado dentro del directorio *log*.

5. Infraestructura

Para correr todo el sistema de software se nos dio acceso a tres servidores en la red interna del ITBA, de los cuales sólo tuvimos el control de dos. En estos servidores instalamos Ubuntu Server 18.04 junto con todo el software adicional necesario (la máquina virtual de Java, el motor de base de datos PostgreSQL, etc). También le dimos nombre a estos servidores en el DNS interno del ITBA, llamándolos Curly, Larry y Moe (siendo Moe el que quedó fuera de nuestro control). A estos servidores se puede acceder por SSH desde la red interna del ITBA con el usuario *crypto* (desde afuera se puede acceder a Pampero y de ahí a los servidores). Por ejemplo, se podría ejecutar desde una computadora del ITBA:

```
ssh crypto@curly.it.itba.edu.ar
```

De esta forma se accedería a *Curly* por SSH. Para poder lanzar los procesos del sistema de software y poder dejarlos corriendo luego de cerrar la conexión SSH utilizamos *tmux* [8], un multiplexor de terminales que permite desde una misma consola lanzar varios procesos y mantenerlos vivos aun habiendo cerrado la conexión SSH.

6. Estrategias

El *Strategyzer* está diseñado de forma modular para poder programar y agregar estrategias sin mayores dificultades. A continuación se detallan las dos estrategias que ya fueron implementadas.

6.1. User Control

Esta estrategia sólo sirve de prueba y sólo está disponible en el modo *run*. Al iniciarla habilita la consola para que el usuario ingrese los comandos *BUY* o *SELL*. Esta estrategia fue creada para probar el correcto funcionamiento de la interacción del *Strategyzer* con el *Trader* y de este último con el mercado.

6.2. *Exponential Moving Average*

Esta estrategia está basada en la media móvil exponencial. Es una estrategia conocida cuya explicación detallada puede encontrarse en Investopedia [9].

6.2.1. Cálculo

La media móvil exponencial (EMA) es una media móvil ponderada sobre los precios de las operaciones a lo largo del tiempo que pone más peso a las operaciones más recientes. Como toda media móvil, tiene un parámetro que es la cantidad de puntos o periodos a tomar. Esta media puede calcularse teniendo en cuenta todas las operaciones o dividiendo la serie temporal de precios en intervalos homogéneos de tiempo (como hace el *candlestick*) y tomando el precio de cierre de cada período. Así, una media móvil exponencial de 20 períodos de 15 minutos tomará los precios de cierre de los últimos 20 períodos (como mínimo) de 15 minutos, por lo que necesita mínimo 5 horas de información para calcularse.

El procedimiento para calcular el EMA es recursivo y está expresado por la siguiente fórmula:

$$EMA_{p,n} = CP_n \times s(p) \ + \ EMA_{p,n-1} \times (1 - s(p)) \qquad (1)$$

siendo:
$EMA_{p,n}$: el EMA de *p* periodos en el periodo *n*
CP_n: el precio de cierre del periodo *n*
s(p): la función de suavizado evaluada en *p*

La función de suavizado tiene la siguiente ecuación:

$$s(p) = \frac{2}{1+p}$$

Para entender cómo esta media pondera más a los precios más recientes, podemos expandir la ecuación 1 y analizar sus términos. Primero, llamamos *α* a *s(p)*:

$$EMA_{p,n} = CP_n \times \alpha \ + \ EMA_{p,n-1} \times (1 - \alpha) \qquad (2)$$

Reordenando términos:

$$EMA_{p,n} = EMA_{p,n-1} \ + \ \alpha(CP_n - EMA_{p,n-1}) \qquad (3)$$

Expandiendo una cantidad arbitraria de términos $EMA_{p,n}$ y reordenando:

$$EMA_{p,n} = \alpha[CP_n + (1-\alpha)CP_{n-1} + (1-\alpha)^2 CP_{n-2} + (1-\alpha)^3 CP_{n-3} + ...] \qquad (4)$$

En la ecuación 4 vemos que cada precio de cierre está multiplicado por un factor que decrece exponencialmente (*α* es menor a 1 siempre que se tomen más de dos periodos), lo que le da el nombre a la media y muestra cómo los precios más recientes pesan más en el cálculo del *EMA*. También es importante notar que el valor de *α* es menor si la cantidad de periodos *p* es mayor. Esto hace que cuanto menor sea *p* más peso esté puesto en los precios más recientes. Por esto se dice que cuantos menos periodos se toman más rápida es la respuesta del *EMA* al cambio en el precio.

A partir de la ecuación 1 se puede calcular el *EMA* de cualquier periodo sabiendo el *EMA* del periodo anterior. Este *EMA* del periodo anterior se obtiene a su vez con la misma ecuación 1 usando el *EMA* dos periodos anterior. Esta recursión necesita un caso base para estar bien definida, de otra forma se estaría calculando *EMAs* anteriores infinitamente. El caso base de esta recursión es el *SMA* (*simple moving average*). Cuántos periodos se toman hasta el caso base (es decir, cuántas veces se aplica la recursión antes de usar el *SMA*) es arbitrario; alcanza con que queden suficientes periodos *p* hacia atrás para calcular el *SMA*. Hay que tener en cuenta que a medida que los precios son más viejos, éstos pesan menos en el cálculo del *EMA*, por lo que llega un punto en que su influencia es despreciable.

6.2.2. Estrategia

La estrategia está basada en las medias móviles exponenciales de 5, 20 y 50 periodos, con periodos del tiempo que se desee (es configurable) pero típicamente de 15 minutos. La estrategia indica que si $EMA_{5,n}$ cruza de abajo hacia arriba al $EMA_{20,n}$ estando ambos por encima del $EMA_{50,n}$ y el precio de cierre también por encima del $EMA_{50,n}$, es un indicador de que hay que comprar. Por el contrario, si el $EMA_{5,n}$ cruza de arriba hacia abajo al $EMA_{20,n}$ estando ambos por debajo del $EMA_{50,n}$ y el precio de cierre también por debajo del $EMA_{50,n}$, es un indicador de que hay que vender. Cuánto se compra o vende en cada caso es algo que no es parte de la estrategia en sí, sino una decisión de cuán agresiva o conservadoramente se quiere invertir. Si se usan o no las comparaciones con el $EMA_{50,n}$ es opcional y se puede apagar en la configuración de la estrategia.

7. Resultado

El 22 de octubre de 2019 a las 11:38:59 pusimos a correr en el servidor Larry el *Strategyzer* con la estrategia *Exponential Moving Average* con periodos de 15 minutos y con

la comparación con el $EMA_{50,n}$ activada contra el mercado BTC/USDT. El balance en ese momento era de $5x10^{-5}$ BTC y 0.408869508 USDT. Con el estado del libro de órdenes en ese momento, el valor estimado de todo el balance en USDT era de 0,819766008 USDT. Antes de iniciar el *Strategyzer*, el ETL había estado corriendo durante varios días para poder recabar información histórica y hacer algo de *backtesting* antes de poner a correr la estrategia contra el mercado. El *Strategyzer* corrió durante 6 días, durante los cuales llevó a cabo las siguientes operaciones:

Time (-03)	Price (USDT)	Amount (BTC)	Total (USDT)
Oct 27, 21:45:01	9640.06	0.00001	0.0964006
Oct 27, 19:15:01	9656.38	0.00001	0.0965638
Oct 25, 01:30:01	7425.60	0.00001	0.0742560
Oct 24, 20:00:01	7453.30	0.00001	0.0745330
Oct 24, 16:00:01	7447.36	0.00001	0.0744736
Oct 24, 14:00:02	7456.83	0.00001	0.0745683
Oct 24, 09:15:01	7449.37	0.00001	0.0744937
Oct 23, 22:30:01	7451.67	0.00001	0.0745167
Oct 23, 21:45:01	7452.15	0.00001	0.0745215
Oct 23, 08:45:01	7953.79	0.00001	0.0795379

Figura 10: operaciones realizadas por el *Strategyzer* con la estrategia EMA (sacadas de HitBTC)

Estas operaciones fueron registrándose en el archivo CSV que hace las veces de libro contable.

TIME STAMP	OPERATION	PRICE	VOLUME	FEE
2019-10-23T08:45:01Z	SELL	7953.79	1.0E-5	1.590758E-4
2019-10-23T21:45:01Z	SELL	7452.15	1.0E-5	1.49043E-4
2019-10-23T22:30:01Z	SELL	7451.67	1.0E-5	1.490334E-4
2019-10-24T09:15:01Z	BUY	7449.37	1.0E-5	1.489874E-4
2019-10-24T14:00:02Z	BUY	7456.83	1.0E-5	1.491366E-4
2019-10-24T16:00:01Z	BUY	7447.36	1.0E-5	1.489472E-4
2019-10-24T20:00:01Z	BUY	7453.3	1.0E-5	1.49066E-4
2019-10-25T01:30:01Z	SELL	7425.6	1.0E-5	1.48512E-4
2019-10-27T19:15:01Z	BUY	9656.38	1.0E-5	1.931276E-4
2019-10-27T21:45:01Z	BUY	9640.06	1.0E-5	1.928012E-4

Figura 11: operaciones realizadas por el *Strategyzer* con la estrategia EMA (sacadas del CSV)

Por cada una de estas operaciones, el *Strategyzer* manda un mail como el siguiente:

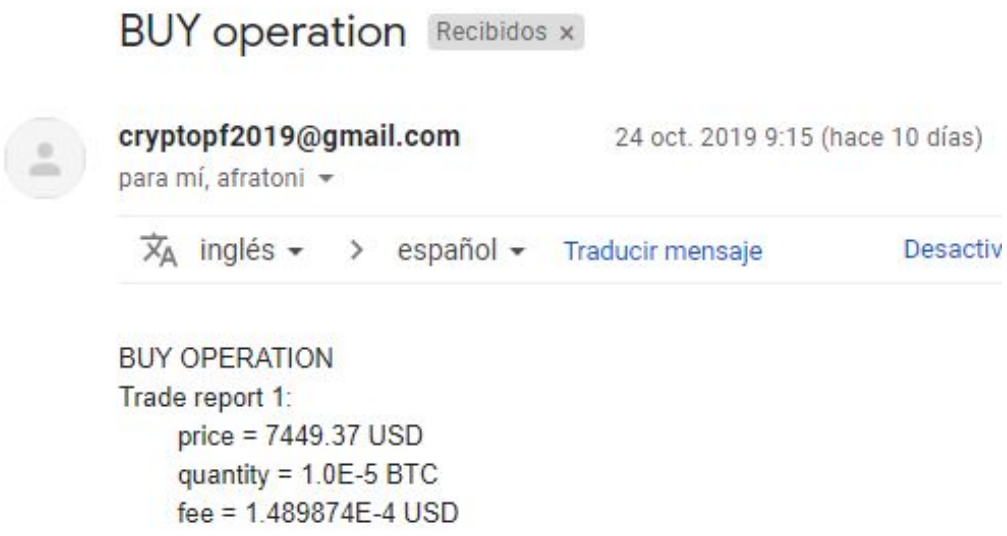

Figura 12: mail enviado por el *Strategyzer* informando una operación de compra

Pasados los 6 días, el balance era de $7x10^{-5}$ BTC y 0,219085308 USDT. Con el estado del libro de órdenes de ese entonces, el balance total en USDT era de 0,872223108 USDT. Esto quiere decir que hubo una ganancia del 6,4%. Este valor se presenta sólo a modo indicativo, ya que no es parte del objetivo del trabajo diseñar un sistema que saque provecho del mercado.

8. Conclusiones

Se diseñó e implementó un sistema de software que permite recabar información del mercado, analizar la información y ejecutar estrategias de compraventa automáticas.

La arquitectura que propusimos y llevamos a código permite gracias al ETL almacenar la información de interés en una base de datos, mientras que el *Aggregator*, el *Trader* y el *Strategyzer* proveen los mecanismos para analizar esta información y poner a correr estrategias de compraventa que operen contra el mercado o contra información histórica.

El sistema alcanzó una versión estable hasta el momento de la redacción del presente informe.

9. Referencias

[1] https://bitcoin.org/bitcoin.pdf
[2] https://en.bitcoin.it/wiki/Category:History
[3] https://www.coinlore.com/all_coins
[4] https://coin.market/exchanges
[5] https://hitbtc.com/
[6] https://api.hitbtc.com/
[7] https://github.com/citusdata/pg_cron
[8] https://github.com/tmux/tmux/wiki
[9] https://www.investopedia.com/terms/e/ema.asp