Cryptick - A Digital Asset Portfolio Management Platform
— Final Report —

Charith Amarasinghe, Simon Spurrier, Julian Vossen, Dimitris Nikolaou, James Griffiths, Konstantin Hemker.
{ca508, ss19616, jv1914, dn1217, jrg17, knh116}@doc.ic.ac.uk

Supervisor: Prof. William Knottenbelt
Course: CO530, Imperial College London

16th May, 2018

Contents

1 Introduction 1
  1.1 Background .............................................................. 1
  1.2 Motivation .............................................................. 1

2 Specification 2
  2.1 Description .............................................................. 2
  2.2 Use Case ................................................................. 2
  2.3 Product Requirements .................................................. 2
  2.4 Risks ...................................................................... 3
  2.5 Product Deliverables .................................................... 3

3 Design 3
  3.1 Architecture Overview ............................................... 3
  3.2 Strategy Execution (Cryptick) ....................................... 4
  3.3 Front-end (Cryptock) .................................................. 6
  3.4 Exchange Interaction (xWrap) ....................................... 8

4 Methodology 9
  4.1 Development strategy .................................................. 9
    4.1.1 Project Management .............................................. 9
    4.1.2 Software Tools ................................................ 11
    4.1.3 Development Workflow ........................................ 11
  4.2 Subproblems .......................................................... 14
    4.2.1 Cryptick .......................................................... 14
    4.2.2 Front-end ....................................................... 17
    4.2.3 xWrap Methods ................................................ 18
    4.2.4 Deployment ..................................................... 20
  4.3 Testing ................................................................. 20
    4.3.1 Testing Overview ................................................. 20
    4.3.2 Approach ......................................................... 21
    4.3.3 Testing Modules ................................................. 21
    4.3.4 Testing Risks & Notable Omissions ............................. 23
    4.3.5 Current Results ................................................ 23
  4.4 Challenges ............................................................. 24
4.4.1 Architecture .................................................. 24
4.4.2 Wallet Management ......................................... 24
4.4.3 Poorly Documented Exchange APIs ..................... 24
4.4.4 Testing Trading Strategies ................................. 24

5 Group Work ......................................................... 25
5.1 Project Milestones .............................................. 25
5.2 Group Organisation ............................................... 26
5.3 Meeting Log ....................................................... 26
5.4 Work Log .......................................................... 26

6 Final Product .......................................................... 26
6.1 Project Summary .................................................. 26
6.2 Front-end Overview ............................................. 26
6.3 Strategy Overview ............................................... 28
6.4 Exchange Wrapper Overview ............................... 32
6.5 Next steps for Cryptick ........................................ 33

Appendices ................................................................. 40
A Installation Instructions ......................................... 40
A.1 System Requirements .......................................... 40
A.2 Installing Dependencies ....................................... 40
B xWrap API Documentation ......................................... 41
C Strategy API .......................................................... 43
D Test coverage ......................................................... 45
D.1 Strategy coverage ............................................... 45
D.2 Front-end coverage .............................................. 47
D.3 xWrap coverage ................................................ 48
E Meeting Log .......................................................... 48
E.1 Commit Log ........................................................ 50
E.1.1 Strategy ......................................................... 50
E.1.2 Front-end ....................................................... 50
E.1.3 xWrap .......................................................... 51
E.2 Facts and Figures ................................................. 51
1 Introduction

1.1 Background

The Digital Asset Boom\(^1\) Since Bitcoin was introduced in 2008, with a white-paper describing the world’s first decentralised currency\(^{[12]}\), its market cap has grown to over $100 billion and spawned an asset class worth almost $1 trillion by the end of 2017\(^{[3]}\).

Today, there are thousands of digital assets, attracting media attention across the world as they challenge traditional ideas of money and exchange. However, the market for digital assets remains highly volatile and relatively untouched by government regulation.

Institutional and Retail Adoption With the unprecedented increase in value of the digital asset market during 2017, retail investment grew at an astonishing rate. Institutions, keen to meet this demand and gain exposure to a volatile and lucrative asset class in an era of dormancy in traditional markets, are starting to take notice.

1.2 Motivation

A Challenging Market The market for digital assets remains highly fragmented - a large number of online only exchanges, operating in various jurisdictions, each offering a different subset of trading pairs. Very rarely do two exchanges list the same price for a given trading pair because of differences in fee structure and trading restrictions.

In most cases, despite the billions of dollars of daily trading volume, these markets are not subject to traditional financial regulation. As a result exchanges are run with varying levels of competence and are often subject to price manipulation.

Given these conditions, a need for a platform providing controlled and uniform access to these markets has arisen. This makes the exchange of digital assets more straightforward and less risky, optimising for price and speed.

Investment Industry Investment funds not only need to trade effectively but must maximise returns. This may be on a long term basis, where funds should be allocated efficiently to minimise unsystematic risk and gain exposure to outperforming assets, or a short term basis, using optimal high-frequency trading algorithms to capture immediate profits.

Institutions require a reliable tool for portfolio allocation and trading strategy testing in the market for digital assets.

Miners and Infrastructure Cryptocurrency miners drive the ecosystem and generate and hold large amounts of crypto-currency. To stabilise their portfolio and grow their digital assets they need automated portfolio balancing and a robust trading system that can deal with large volumes of trades without impacting the market (trade slicing).

Other infrastructure providers, such as cryptocurrency payment services, need to manage their digital asset holdings in a similar way, and be able to quickly find liquidity to serve customer’s needs whilst minimising risk.

The Consumer For digital assets to experience true mass adoption there must be a consumer level trading application for managing and investment a cryptocurrency portfolio, analogous to online banking services today. Consumers must be able to optimise and control their own portfolio through a user-friendly interface.

\(^1\)In this section, and throughout this final report, some paraphrased and verbatim content from the first\(^{[21]}\) and second\(^{[22]}\) reports is included without further citation.
2 Specification

2.1 Description

The Cryptick platform is a portfolio management tool for cryptocurrencies. It incorporates an algorithm to auto-diversify a portfolio in real time and respond to market events and market microstructure. The platform provides a unified interface to different exchange to overcome the fragmented market. Users are able to monitor the performance of their portfolio through a web dashboard and tweak parameters to reflect their desired portfolio composition. More advanced users are also able to write their own trading strategy and test its functionality using a backtest environment [21].

2.2 Use Case

There are two major use cases for the Cryptick platform:

Cryptocurrency Portfolio Diversification The first type of user is one that typically has a constant stream of income in a cryptocurrency (e.g. a mining company) and wants to diversify their holdings into a variety of other currencies with minimal effort to reduce cluster risk. Cryptick leverages a capital allocation theory named Markowitz Portfolio Theory that has been heavily used in the financial markets over the past half century. The theory argues that investors can optimise the risk-return ratio of their portfolios through diversifying across different assets (i.e. cryptocurrencies), benefiting from the imperfect correlation between those assets. Based on this theory, the user provides Cryptick with their current portfolio and the platform suggests multiple optimal portfolio mixes which the user can select.

Cryptocurrency Trading Strategy Backtesting The second type of user is typically a more sophisticated cryptocurrency investor who wants to have an environment to backtest and execute their trading strategies. This customer will purchase the software package as a virtual appliance and install it on a host with local network access. An administrator will then configure the software with API keys for cryptocurrency exchanges, wallet addresses and the currency types allowed for trade. The customer (or their authorised employees) will then be able to view their portfolio balances, choose a target portfolio from choices presented by the software, and start a trading process that would trade towards the selected target.

2.3 Product Requirements

All of the minimum requirements set at the onset of this project were met. Additionally, the majority of stretch goals were also met. What follows is a list of the requirements met, the last two bullets of which state the stretch goals that were met [21].

- A strategy for portfolio diversification, grounded in Markowitz Portfolio Theory, parametrised by a client-specified risk level, and tested on historical data.
- A trading system which interfaces with live exchanges to receive market data and issues appropriate orders determined by the strategy to an exchange mock which allows paper-trading.
- An interactive browser-based dashboard for visualisation of the current state of the portfolio and strategy, and for adjusting parameters of the strategy through UI controls.
- Support for data from cryptocurrency exchanges Bitfinex, GDAX and Poloniex.
- Support cryptocurrencies Bitcoin, Bitcoin Cash, Litecoin, Ethereum, ZCash and Monero.
- Implement a trading architecture which supports other strategy types - for example, arbitrage strategies through the unified exchange interface.
- In-application strategy backtesting facilities.
- Central management and logical partitioning of exchange wallets.
2.4 Risks

A high level view of the risks faced in this project are listed below:

- Markowitz Portfolio Theory was developed based on traditional financial markets which are significantly less volatile than cryptocurrencies. It is possible that digital asset market behaviour does not conform to traditional financial modelling posing a moderate risk to the project - deeper research is required [7].

- The trading and simulation features depend on external data. Hence, there is a moderate risk of external data sources becoming unavailable.

2.5 Product Deliverables

- An application that is supposed to be long-running in a headless environment, which fetches real-time market data, evaluates this data using a trading strategy, and sends orders to virtual exchanges accordingly.

- A browser-based interface for the above described application which provides inputs for user defined strategy parameters, a visualisation of the portfolio’s state, and trade execution monitoring [21].

3 Design

3.1 Architecture Overview

The broad function of the Cryptick platform makes system architecture critical in building an extensible and maintainable product. The Separation of Concerns principle was applied to determine the three primary modules of the platform: exchange interaction (xWrap), strategy execution (Cryptick) and front-end (Cryptock).

For overall application structure, a micro-service architecture was selected, as the exchange interaction and strategy execution components best benefited from two concurrency patterns (coroutines and threading) that would not integrate together in a single monolithic structure. Though the front-end could be hosted by one of the other components, it was decided to make it a dedicated micro-service in order to simplify development. Further reasoning behind this choice is discussed in Section 4.4. An initial high-level diagram for the system architecture is shown in Figure 1.

Strategy Execution (Cryptick) Cryptick is a framework for writing and testing Python-based cryptocurrency trading strategies using a simple, easy-to-use API. Individual strategies run inside dedicated processes and have their lifecycle managed by a main Cryptick process. Strategies are able to create and update UI components visible from the front-end through an API layer made available by Cryptick (Table 6 - Appendix B), as well as call the xWrap API to connect the strategy execution to the available cryptocurrency exchanges (Table 5 - Appendix B). The user of the platform can choose either to use provided strategies (e.g. the efficient frontier strategy) or write their own strategy using the APIs.

Front-end User Interface (Cryptock) Cryptock is a React-based web interface that displays the status and operation of running Cryptick strategies and enables wallet-management. It connects to both back-end components (Cryptick/xWrap) to post strategy information, wallet balances and order streams. Information is available to the user at different levels:

- Strategy-level: information per run
- User-level: aggregated information about all strategies and wallets
- Market-level: general cryptocurrency market information
Exchange Interaction (xWrap)  xWrap builds an abstraction layer to multiple cryptocurrency exchanges and manages the lifecycle of buy/sell orders and the collection of pricing data. It additionally provides a simulated trading mode (termed ‘paper’ trading) atop supported exchanges in order to test strategies in a sandbox environment without risking real money.

3.2 Strategy Execution (Cryptick)

Requirements  The following requirements for the strategy execution engine are derived from the project requirements:

- Support execution of different trading strategies
- Communicate with the exchange interface (xWrap) to receive exchange data and manage orders
- Simulate the execution of trading strategies using historical data (backtesting)
- Expose strategy management API to the front-end

In addition to the above requirements, the ease of operation and usability without the front-end for strategy developers was prioritised.

Design Overview  Figure 2 provides a high-level overview of the strategy runner architecture. A server process provides an HTTP API for the front-end and creates processes for executing strategies. Each of these instantiations has a central communication node - the message hub - as well as communication interfaces and the strategy logic itself.

Internal Dataflow  The message hub serves as distributor for all data in the trading system and is handled by each strategy block. The exchange wrapper can connect to the message hub through an adapter and send or receive messages through a fixed message format. The chosen format of messages exchanged over the hub is similar to JSON-RPC 2.0, which specifies two message formats.


```json
{
    "mhubrpc": "0.0.1",
```
"method": "exchange.order",
"params": ["bitfinex", "eth_usd", 10000],
"response_method": "1231709"
}

- Result/Error: specifies ‘response_method’ from initial call in ‘method’ field. Example:

  {
    "mhubrpc": "0.0.1",
    "method": "1231709",
    "result": "success"
  }

As shown in the above example of a trade order being sent, hierarchical method names of the format 'provider.method' were used. The provider of a message is usually the name of a data provider (e.g. 'twitter', 'exchange'). The method would be the action executed on the provider, such as sending off an order.

Using a central message hub enables auditing the information flow. For this purpose, the strategy runner provides a web dashboard, which shows all messages that went through the message hub for each strategy run. The dashboard is a small single-page application based on AngularJS, which is hosted by the same server which provides the HTTP API as described later in this section. A screenshot of the dashboard is shown in figure 14 in the results section.

Strategy Logic All trading logic for a strategy is encapsulated in a single class. On instantiation, the strategy class registers callback functions for messages of interest with the message hub. Within these callbacks, the strategy can react to messages such as exchange events, user actions or periodic self-scheduled triggers. Such reactions can range from simple internal state updates to placing orders by sending appropriate messages to the message hub.

This strict encapsulation of the trading logic makes it easy to implement new strategies and support the execution of arbitrary trading strategies. Moreover, encapsulating the trading logic is also required for backtesting, where the exact same trading logic needs to be simulated using historical exchange data.
UI Components UI components establish a bidirectional communication between a user and a strategy run. These components consist of a format specification and a data array and can be rendered by the front-end according to a defined set of supported component types. A strategy can create and update components by posting messages to the message hub. The UI component manager has registered callbacks for these messages and writes the component data to the local filesystem. For each component, a JSON file holds the format specification, and a CSV file stores the data itself. We chose simple files over database management systems according to the design priority of low operational complexity to enable strategy developers to run Cryptick easily. When users log to components, they can directly inspect the data in CSV files. Advantages of DBMS diminish in this setting where only a single writer instance (UIComponentManager) writes to the data for each run, the total data volume is low and the data structure is simple.

For the communication from the user to the strategy, the front-end can post data to individual components. These posts are then converted into valid message hub messages and forwarded via the message hub socket interface.

CLI As users should be able to develop new trading strategies without running the front-end server, the strategy runner comes with a command-line interface (CLI) itself. Using this CLI, the user can run single strategy instantiations by specifying the mode and initial conditions. For example, the following command will start a backtest run of a strategy called HodlStrategy, for a specified time, with an initial balance of 10000 USD:

```bash
python -m cryptick.run HodlStrategy -m backtest --gdax-usd 10000 \
-s '01-01-2018' -e '16-05-2018'
```

Using the CLI, a strategy will run synchronously in the console.

API Server Given the requirement of exposing the strategy execution management to the front-end, the strategy runner provides an HTTP API server to manage strategy runs. With this API, the front-end can request available strategies, instantiate and terminate runs, and communicate with the running strategies. The full API documentation can be found in the appendix.

As opposed to the CLI, the API server will start strategies asynchronously, by creating separate processes for each strategy run.

Backtesting For backtesting, a run process is instantiated with an exchange simulator instead of the XWrapInterface. Hence, the exchange simulator provides callbacks for all the messages which would normally be send to xWrap. The simulator requests historical exchange data from xWrap to simulate order matching and determine the corresponding prices.

3.3 Front-end (Cryptock)

Design Overview The main design of the front-end is shown in Figure 3 and shows both internal dataflows through the front-end as well as the building blocks required for communication with the strategy and the exchange wrapper. The control and data flow of the front-end is discussed in the following section and a more detailed discussion about implementation considerations is included in section 4.

Reducer All state management happens through the reducer, which keeps track of the state of the application and specifies how the application’s state changes in response to actions. This is just a composition of pure functions from state-to-state. The initial state of the application is stored in the reducer as an immutable object whose members are aligned with the format expected by the xWrap and front-end API.
**Actions** Fundamentally, all communication throughout the application occurs through actions. State updates may only occur in response to actions, whose types are matched on to produce the desired effect. Actions can be triggered by any part of the application upstream of the reducer, such as upon loading a UI component, on receipt of websocket communication, or by user interaction (e.g. `onClick`, `onHover`, etc.). Actions consist of types and payloads of information, used for updating the application’s state, or intercepted by middleware (in our case, `sagas`) to perform intermediate transformations or effects (most notably, AJAX requests). An example of an action would be adjusting the portfolio weights on the efficient frontier strategy using the sliders component. As soon as this action is dispatched, the reducer, which is listening for all actions, matches on this action type and updates the state of the application corresponding to its payload. Actions within the Redux architecture are the only source of information for the reducer, leading to consistent state management with a single source of truth.

**Container Components and Selectors** Container components manage the data used by the UI. Using container components separated from the rendering (UI) components separates the process of data-fetching and rendering for each component. Any state updates broadcast by the reducer are heard by container components, which map the state of the application (and in our case, router state) to the required properties of the UI component. The main upside of separating rendering and container components is that components can be re-used more easily and the rendering component is not concerned with state management. Containers themselves use selectors to efficiently compute derived data from application state. This efficiency is achieved through memoisation and functional composition, allowing the functions to be called on every state update, but only being fully evaluated if the inputs have changed. The `reselect` library was chosen for selectors.

**UI Components** The container components map the state of the application to the properties of the rendering components that were built using React in conjunction with Google’s Material-UI. All chart representations were rendered using `react-chartjs-2`. Rendering components also provide an interface to the actions available to the user (see above).

**Sagas** Sagas are a relatively old concept from the database world to manage state-updating transactions, but novel in the front-end development context. For our application, sagas act as ‘database’ middleware, sitting just in front of the reducer. These listen for particular action types which are used
to communicate requests for API-calls and fork threads to perform these calls asynchronously, or sit waiting for socket communication, before dispatching actions denoting API success or failure to the reducer. This isolates all network side-effects to this portion of the app. Redux-saga is employed as an implementation of the saga concept in our application.

**API Interaction** API interaction is done with the help of two front-end servers. Each front-end server serves the application itself and associated assets, and talks to the back-end services. One front-end server serves assets for the live-trading version of the application, and communicates with the live-trading xWrap service. The other does the same but for paper and backtest trading. Both of these server instances speak to the same instance of the Cryptick strategy runner micro-service, as that instance manages all trading modes of strategies. These front-end server instances act as amalgamation of proxy servers and static-asset servers, built in a fairly simple manner with express.

### 3.4 Exchange Interaction (xWrap)

The project specification calls for a trading system which is capable of fetching live data from cryptocurrency exchanges and virtually trading according to real-time market conditions in paper trading mode. Exchange API formats can differ significantly and offer different levels of functionality to users, and a unified abstraction would make extending the system in the future much easier.

The xWrap service acts as a local cryptocurrency exchange, transparently handing orders to the GDAX, BitFinex and Poloniex APIs while providing high level management and error recovery capability to client applications. It is built atop a single high performance asynchronous event loop that services network connections, handles events and carries out scheduled tasks. What follows is a discussion of key design concepts; full documentation is bundled with this report and made available in the appendix.

![Figure 4: Simplified Diagram of xWrap](image)

**Design** The component diagram for the xWrap service is shown in figure 4. The service is packaged as a Python module and requires CPython version 3.6 or later on MacOS or Linux. A brief description of the various components follows.

**API Layer** The API layer is a set of functions which expose the core functionality of xWrap to client services. Consult the xWrap documentation in the appendix for a full list of supported API methods. These functions are implemented in the `service.ExchangeWrapper` class.
HTTP REST API The HTTP API module translates incoming HTTP requests into API calls on the API layer. The HTTP server is provided by the `sanic` library for Python and the API stubs are specified in the `http_server` module.

WebSocket RPC API The RPC API module translates incoming RPC requests into API calls on the API layer. This is implemented by the `rpc.RpcInterface` and `wsock_server.WebsocketServer` classes.

Config The Config module provides a global configuration to all other modules. Variables can be set either through a JSON configuration file, or in certain cases, overridden by OS environment variables. Example configuration variables include ports for serving APIs, security credentials for exchange API access and tradeable currencies. The Config module allows for flexible configuration of the xWrap service when deployed.

Exchange Adapters The exchange adapters form the core of the exchange interaction tasks. These are implemented as Python modules in the `exchange` package which abstracts the Web APIs for different providers into a unified set of Python methods. The modules also spawn and maintain recurring tasks that poll exchanges for certain types of data. For exchanges delivering Level 2 market data, exchange adapters additionally subscribe to push APIs to maintain real-time exchange order books.

Logical Wallets Logical wallets allow the logical partitioning of exchange wallets into wallet sets. This allows users to sandbox portions of funds held on exchanges to a single strategy and ensure that strategies are limited to seeing and using only those allocated funds. This functionality is implemented in the `models.VirtualWallet` and `logical_wallet.LogicalWalletManager` classes.

4 Methodology

4.1 Development strategy

4.1.1 Project Management

To ensure the timely completion of the project, Agile software development was used. Agile specifies an iterative and flexible approach to the development of lean software that requires the regular input of members of all teams.

In particular, two key Agile frameworks called Scrum and Kanban were employed. Scrum focuses on breaking down the development work into blocks and completing them in an iterative fashion within a set time limit. Each iteration is called a Sprint and the time of each Sprint is set by the Scrummaster (in our case the team leader)[16]. We chose small iterations of two weeks as this size of time frame allowed us to be more adaptive. Scrum also entails regular quick meetings to track progress. Therefore, we met on average twice a week for a total of four times per sprint. The Kanban framework places its focus on the visualisation of the blocks that are created on every sprint and this was very valuable in tracking progress within each sprint.

Agile was chosen mainly due to its adaptive and iterative nature. Moreover, some team members previously worked on projects using these techniques. The implementation of Agile was made on Asana, a software that allows us to easily use Scrum and Kanban. Figure 5 is a snippet of the `Cryptick` Kanban board on the 22/02/2018.
Figure 5: Kanban Board 22/02/2018.
4.1.2 Software Tools

**Programming Language**  The choice of programming language, unless constrained by application specific requirements, is largely determined by the skill-base of the organisation\(^2\). For this project, the ability to easily interface with third-party web services and perform fixed-precision computation on large datasets is critical. Python was chosen as the main back-end development language due to the team’s past experience and for its strong library support in the above areas.

**Computer Resources**  As the end-product is a web application, the only external resources required are servers for hosting the application and any supporting services. No special compute or storage requirements were needed and the Department of Computing’s Apache CloudStack virtualisation platform was used to create VMs. The Cryptick platform is hosted on 3 VMs, one for production, one for development and one for the time-series database. A fourth VM provides a GitLab CI Runner for Continuous Integration.

**Software Packages**  The final product (section 6) depends on a number of third-party libraries and services. Associated license requirements can pose significant limitations on commercial use. Open-source dependencies should thus be licensed under a BSD-style or similarly permissive license that permits free commercial use without any restrictions on the end-product. Web services to be used, such as cryptocurrency exchanges, will be priced on a per-transaction commission or subject to volume-pricing agreements. These costs will be unavoidable for any participant in the ecosystem.

**Version Control and Code Review**  Git has been chosen as the version control system for this project. This is largely due to the wide familiarity with Git within the team. Alternative tools such as Mercurial and SVN pose no specific advantage within the scope of this project. GitLab and Imperial’s GitHub Enterprise were evaluated for hosting a co-ordinating repository, code review functionality, a bug tracker and support for continuous integration. GitLab was selected due to simpler CI integration with GitLab Runner.

4.1.3 Development Workflow

**Workflow**  A structured workflow forms an integral part of managing day-to-day collaborative software development. It primarily guides the effective use of version control software, issue trackers and code review processes to ensure the creation of high quality, readily deployable code. Given the size of the team, a branch-review-merge strategy is proposed\(^2\) as it maps smoothly with the built-in code-review functionality on GitLab[20]. In this model, a new feature is developed in its own branch by one or more developers. When a feature is completed, the primary developer will raise a merge request to the upstream branch and resolve any merge conflicts that have developed. The subsequent diff will be brought up for approval by the maintainer of the upstream branch. A merge request that passes this review process will be merged upstream. Figure 6 shows an example review of a merge request using GitLab.

**Continuous Delivery and Continuous Integration**  CI/CD practices enforce rapid releases of code to production environments and incremental changes to software. Such a system is adopted in-line with modern best practice in order to achieve highly visible iterative progress. Figure 7 shows an extract of the GitLab CI system for the xWrap process. Branches failing tests or missing coverage need to be fixed before they can be merged upstream.

**Development Practices**  Various project-specific practices such as Git workflow, code style, commit strategy, documentation requirements and test practices are documented and maintained within a central docs repository on GitLab.

\(^2\)Certain assertions in this section are based on the teams experiences or observations from industry.
Figure 6: GitLab Merge Request

crypto-portfolio-mgmt / cryptick

```python
+ PID_FILENAME = '.pid'
+ PORT_FILENAME = '.port'
+ app = Flask(__name__)
+ def new_run_id(name):
+     """Generates a new unique id for a run by appending a number to its name"
+     for i in range(100000000):
+         run_id = '{}_{}'.format(name.lower(), i)
+         if not os.path.exists(os.path.join(STORE_DIR, run_id)):

ca508 @ca508 commented a month ago
This can cause a race condition?

Something like:

```with lock:
    p = os.path.join(STORE_DIR, run_id)
    if not os.path.exists(p):
        os.makedirs(p)
    return run_id
```

Julian Vossen @jv1914 commented a month ago
Yeah, well spotted. Will fix.

Julian Vossen @jv1914 changed this line in version 5 of the diff a month ago
**Figure 7: GitLab CI Pipelines**

<table>
<thead>
<tr>
<th>Status</th>
<th>Pipeline</th>
<th>Commit</th>
<th>Stages</th>
<th>Description</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>failed</td>
<td>#49878 by</td>
<td>feat_live_ord.. → 4a8b51d</td>
<td></td>
<td>very theoretically poloniex should work</td>
<td>01:00:04</td>
</tr>
<tr>
<td></td>
<td>latest</td>
<td></td>
<td></td>
<td></td>
<td>about 5 hours ago</td>
</tr>
<tr>
<td>passed</td>
<td>#49862 by</td>
<td>v0.5 → e7831655</td>
<td></td>
<td>Dirty fixes</td>
<td>02:02:50</td>
</tr>
<tr>
<td></td>
<td>latest</td>
<td></td>
<td></td>
<td></td>
<td>about 12 hours ago</td>
</tr>
<tr>
<td>passed</td>
<td>#49833 by</td>
<td>v0.5 → c6c47daf</td>
<td></td>
<td>Fix nasty notifications - bad blinker, b...</td>
<td>02:02:52</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>about 13 hours ago</td>
</tr>
<tr>
<td>failed</td>
<td>#49825 by</td>
<td>feat_live_ord.. → 95f13ac3</td>
<td></td>
<td>bitfinex needs debugging</td>
<td>01:00:05</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>about 13 hours ago</td>
</tr>
<tr>
<td>failed</td>
<td>#49810 by</td>
<td>feat_live_ord.. → c8f7f4d7</td>
<td></td>
<td>bitfinex update formatting and subsc...</td>
<td>01:00:05</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>about 14 hours ago</td>
</tr>
<tr>
<td>failed</td>
<td>#49789 by</td>
<td>feat_live_ord.. → ccbe8d9b</td>
<td></td>
<td>start bitfinex livebook version</td>
<td>01:00:04</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>about 17 hours ago</td>
</tr>
</tbody>
</table>
Internal Communications  Team communications were largely carried out through the Slack platform. In addition to providing convenient web and mobile applications, Slack allowed the creation of ‘Channels’ for each workstream. This enabled interested parties to passively monitor design discussions on parallel streams as well as to keep a log of all discussions. The support for chat bots also enabled integration with third-party services such as GitLab and Sentry.IO.

Metrics and Error Reporting  As large parts of the Crypick platform are long running processes, bugs can cause errors or unexpected behaviour at any time, and developers might not always notice them. As such, error reporting and proactive monitoring are key to identifying issues and fixing them. Two major strategies were adopted to address this requirement: The third-party Sentry.IO service[5] is used to log and alert on runtime exceptions and the Prometheus monitoring system[14] is used to instrument key sections of code to monitor for critical variables such as response times, error rates and message counts[15]. Figure 8 shows the dashboard of the Sentry.IO web application and Figure 9 shows graphs of some of the metrics monitored on the xWrap service.

Documentation  APIs for the Cryptick platform are documented using Python Doc-Strings and text files in Markdown format. The Python sphinx library is used to generate HTML and PDF documentation from these sources. The reference documentation attached in the appendix to this report was generated in this manner.

4.2 Subproblems

As mentioned in section 3, the main workstreams of the project were broken down according to its subproblems - the strategy, the exchange wrapper and the front-end. On top of these general subproblems, the main overarching subproblems are discussed in the following, namely integration of the three workstreams and deployment of the Cryptick platform.

4.2.1 Cryptick

Portfolio Diversification Strategy  The Markowitz Portfolio Theory was used as the portfolio diversification method. This theory is very commonly applied to traditional financial portfolios (e.g. bonds, stocks) and as such it was trusted by the Cryptick team.

Input  The strategy receives two inputs. The first one is the Open-High-Low-Close data of the supported cryptocurrency on a 5 minute interval over the past 180 days. The strategy does not need to communicate with any exchanges as the xWrap component of Cryptick collects and adjusts the data. Therefore, the strategy communicates with xWrap and collects the data needed. The second input is from the users of the platform. When diversifying their portfolio the user is asked to specify a risk/return trade-off.

Data Processing  The strategy then processes the data and generates 10,000 portfolios with random weights over the cryptocurrencies selected. For these 10,000 portfolios an expected return and a variance is calculated based on the historic prices. The return of a portfolio is calculated as the weighted average of the historic returns. The portfolio variance is calculated by taking into account portfolio weights and the correlation coefficient between the portfolio assets [10]. The formula for portfolio variance can be found in the equation below.

\[
\text{Variance} = \sum_{i=1}^{N} \sum_{j=1}^{N} (w_j \ast \rho_{ij}) \ast (w_i)
\]

\[ w_i : \text{weight of asset } i \]
\[ N : \text{number of assets in portfolio} \]
\[ \rho_{ij} : \text{correlation coef. between asset } i \text{ and } j \]
### Unresolved Issues (13)

<table>
<thead>
<tr>
<th>Issue</th>
<th>Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>xwrap-18</td>
<td>Traceback (most recent call last):</td>
<td>unresolved</td>
</tr>
<tr>
<td>xwrap-17</td>
<td>ConnectionError requests.adapters in send</td>
<td>unresolved</td>
</tr>
<tr>
<td>xwrap-16</td>
<td>HTTPConnectionPool(host=localhost, port=8086): Max retries exceeded with url:</td>
<td>unresolved</td>
</tr>
<tr>
<td>xwrap-15</td>
<td>xwrap-14</td>
<td>JSONDecodeError</td>
</tr>
<tr>
<td>xwrap-13</td>
<td>Error occurred while executing <code>make_order</code> from client 163eb70c-a18d</td>
<td>unresolved</td>
</tr>
</tbody>
</table>

Figure 8: Sentry.IO Dashboard
Figure 9: Prometheus Stats for xWrap Rendered in Grafana.
For each random portfolio, a Sharpe ratio is calculated which reflects the return earned in *excess* of the risk free rate. As a risk free rate, a semi-annual return of 3% was used as that appears to be the rate of US 10Y bonds [1]. It follows that the portfolio with the highest Sharpe ratio should be the one selected assuming that the investor is willing to take any level of risks. The formula for the Sharpe ratio is given below [11].

\[
\text{Sharpe ratio} = \frac{(\bar{r}_p - r_f)}{\sigma_p}
\]

\( r_f \) : risk free asset  
\( \bar{r}_p \) : mean portfolio return  
\( \sigma_p \) : portfolio standard deviation

**Output**  
Out of the 10,000 portfolios, the ones with the highest return for every level of risk are determined. These portfolios are said to create the efficient frontier as it would be inefficient for investors to choose any portfolios other than these. An illustration of the random portfolios and the Efficient Frontier can be viewed in Figure 10. The strategy writes all these portfolios that make up the Efficient Frontier together with their expected return, variance and Sharpe ratio into a UI component which is then displayed on the front-end. It also adds the covariance matrix between all selected cryptocurrencies together with all single-asset portfolios for recomputing metrics for user-specified portfolios.

### 4.2.2 Front-end

The methods used to meet the requirements of the front-end application are described below. While these are to an extent touched upon by the content of 3.3, we here go into more detail as to the methods used to overcome both intellectual and technical problems, and alternatives that were considered.
Virtualising the UI  One of the significant challenges faced by the front-end has been maintaining responsiveness in the face of large and unpredictable volumes of data from back-end sources, both in stream- and batch-form. These two forms of data-source have different requirements for maintaining responsiveness: streams require being able to handle unpredictable frequency of messages effectively, while batch data requires being able to process large volumes of data and update the UI without resulting in noticeable slowdown. Both of these requirements are satisfied through a combination of virtualising the UI and batching updates to the actual DOM based on a ‘virtual’ DOM. The displaying of large quantities of batch data is mitigated through the use of virtual tables, which render only as much data as is visible, calculated according to scroll position.

Unified Theming  While CSS lays the foundation for web-UI styling, operating on a team-project requiring consistent theming while also having granular control over how the theme is expressed in particular components exposes its inadequacy as a way of managing styling in itself. To remedy this, we used the high-level Material UI library for React to enable automatic theme- and styling-injection, as well as access to a consistent set of pre-styled components. This provides a basic set of building-block components all determined by a single theme configuration. If properties of the theme are changed, these changes propagate down through the entire application. Moreover, when needed, theme properties can be overridden by particular components. This enables a uniformly styled UI, where components designed by team members automatically adopt the theme of the application.

Unidirectional data-flow & Immutability  To facilitate an easy division-of-labour between members of the front-end team and reduce mental overhead, a strict unidirectional data-flow and pub-sub architecture was adopted for data management. This is implemented through the Redux library. All communication is done through a single channel (via actions), and actions may only be dispatched from outside of the data-store. While this leads to greater initial overhead in setting the project up, it has enabled us to easily reason about the project as time has gone on. Utilising seamless-immutable to enforce immutability throughout the project has also ensured the absence of bugs caused by state-mutation due to side-effecting functions in other parts of the project.

Development server  One of the challenges of developing the front-end in tandem with the back-end has been managing data-dependencies for the development process. Thus rather than initially developing against the changing back-end, we have primarily developed against sample data in our production server, and transitioned to live servers once the project was bootstrapped.

4.2.3 xWrap Methods

Some of the key design decisions were already outlined in section 3.4. On top of these design choices, the following methods were used in the implementation of the exchange wrapper.

Paper Trading  As xWrap must support both paper and live trading on an exchange, it was desirable to reduce code duplication between these two modes of operation. Lightweight subclasses were considered but this would require additional work from a developer adding support for a new exchange, thus the two modes of operation were implemented via dynamic inheritance at start-up, whereby the exchange adapter selection can be overwritten with simulation routines when an exchange is loaded in ‘paper’ trading mode. Trades are matched against real-time order books maintained via subscriptions to level 2 trading data, and as such have a high degree of accuracy.

Easing Client Development  Developing for an API only web service is sometimes difficult during the early stages of a project due to the absence of a user friendly interface to test API calls. From the API developers perspective, it is also difficult to spot the nuances of API usage and address any design issues pro-actively. To overcome this obstacle, xWrap provides its own web-based dashboard to allow users to view data and perform most core operations. This dashboard as shown in section 6.4 uses the same APIs as client micro-services and serves as a reference example for API usage.
Event Loop  The xWrap service is largely I/O driven, acting as both a client and server, and carrying out tasks when certain events are triggered (a new level opens on a cryptocurrency order book, an order is filled by an exchange, etc.). An Event Loop design pattern lends itself well to this type of application, allowing programmers to segment behaviour into tasks triggered by events (incoming/outgoing messages, timer ticks, etc.). Python provides a number of libraries to implement such a loop ranging from the DIY solutions around the select() system call to fully fledged frameworks such as Tornado. For the design of xWrap, three main approaches were considered for implementing an event loop:

Worker Threads  Threads are a popular concurrency primitive. Any parallelism advantages however are nullified thanks to the CPython Global Interpreter Lock (GIL) which prevents parallel execution of Python bytecode. Moreover, threaded applications can be notoriously difficult to debug when data races occur, and are in this respect inferior to the following solutions.

Custom Asynchronous Loop  The select() system call and newer alternatives such as epoll and libuv provide a versatile mechanism to implement non-blocking I/O. This combined with a simple task manager can be used to implement a basic event loop in Python. The downside is that any I/O participant must declare and manage callbacks whenever it waits on I/O and the management of these callbacks can become tedious and difficult to debug.

AsyncIO  The asyncio library was added in Python 3.4 to provide a more programmer friendly approach to asynchronous programming in Python. The library adds a new programming paradigm to the language and thus presents a steep learning curve to even experienced Python programmers. However, it presents the most elegant solution to the problem of an event loop, as Python functions can be transformed into asynchronous tasks through addition of a single keyword. The downside however is that co-routines (as created by asyncio) do not support pre-emption and thus tasks must voluntarily yield execution for other tasks to run. Any blocking operations inside a co-routine will block the entire process. The lack of pre-emption however has an advantage in simplifying the process of mutual exclusion, as context switches are solely under the control of the programmer.

After prototyping each implementation option, the asyncio option was selected as it presented the most modern and concise solution to designing an event loop. Some tasks such as rate limited HTTP operations are simpler to express with blocking I/O, and the Python built-in concurrent.futures library was used to synchronise threads and co-routines. The third-party janus library further provides a queue implementation that provides both synchronous and async friendly access.

Storing Time-Series Data  Strategies require historic market data in order to compute metrics such as asset risk. These volumes of data can span multiple years and cannot be queried from exchanges on-demand. Thus the xWrap service must collect and store them such that they can be later queried by strategies. This data is indexed by time and can potentially reach volumes of a few hundred datapoints per minute.

A number of specialised Time-Series Databases (TSDB) and relational databases were considered for storing this data and PostgreSQL and InfluxDB were shortlisted for consideration. InfluxDB was selected for use mainly due to its simplicity and ease-of-use compared to PostgreSQL which required additional tuning for the expected volumes of data.

Simulated Trading  It is impossible to model true market impact - the reaction the market would have to your trading activity. However, assuming any market impact is negligible - that an individual’s trading activity is insignificant compared to traded volume over the period of their order - a paper or back-testing trading environment has value for testing trading strategies, particularly those focussing mostly on high frequency, short term trades.

At a level as granular as a a level 2 order book, it is necessary to maintain control over what live liquidity ”should be” available to prevent trading against the same liquidity twice. A record of
past virtual trades was implemented that matches off with incoming orders as they update the locally
maintained book.

Matching Logic  The virtual exchange implements a matching engine according to a typical central
limit order book[17]. Orders are matched according to price time priority - virtual orders they are
always traded at the best available price. Any order not immediately executable sits ”on the book”
until it matches against an incoming order. We model this by maintaining a record of open order
which may match against new updates to the order book.

Four order types are available: limit order, market order, fill or kill (FOK) and immediate or cancel
(IOC). These are the most commonly used order types on live exchanges. A limit order will execute
the order up to a specified price limit and will remain ”on the book” until it is matched or cancelled.
A market order executes the full size of the order at the best currently available price. An IOC order
trades all available size up to a specified limit and cancels the residual. A FOK order will trade the
entire size of the order up to the limit or cancel the order.

This matching logic is typical of major trading venues across asset classes.

Maintaining Real-Time Order Books  Maintaining live local order books allows high frequency
trading strategies to be thoroughly tested and to competitively trade live. It also provides better
pricing information for strategy analysis and ensures that trade execution achieves an optimal price
in live trading and a realistic price during testing.

A live order book type was created with a universal message format for book updates and trades,
implementing double trade protection. At least 100 price levels are maintained for every traded
currency on every supported exchange, with size aggregated at each level. A snapshot provides the
order book baseline and update messages are pushed from the exchange over a websocket interface as
soon as the exchange’s order book changes.

4.2.4 Deployment

Motivations for Docker  Deploying micro-services , dependencies and configuration can introduce
significant systems administration overhead. Docker containerisation was adopted to minimise this
overhead and speed up deployment. Each micro-service component is packaged with its own Dock-
erfile which defines how to build a container for that service. These individual containers are then
orchestrated with the docker-compose tool in order to bring up the entire Cryptick platform with
a single command-line invocation. This process makes deployment far more convenient than with
manual installation and management of individual components.

4.3 Testing

4.3.1 Testing Overview

As mentioned in section 3, the project was broadly split into three main components - the strategy,
an exchange wrapper and a front-end web interface. For testing, the three components are unit
tested individually and integration tested against the predefined API contracts, resulting in a high
test coverage and confidence in each component [2]. The final end-to-end testing will then attempt to
capture any contractual disagreements between the three micro-services by running in paper trading
mode. This simulates a real trading environment and captures the full range of functionality and
behaviour by trading virtual assets according to real exchange data. The three test types used for the
Cryptick platform are summarised below:

Unit Tests These are white-box tests that are performed on individual functions and methods in
isolation. The completeness of the unit tests was measured by looking at total code coverage
(see section 4.3.5). Unit tests were kept relatively simple and rely on mock functions to mimic
the behaviour of external dependencies.
Integration Tests These tests check the links between unit tested objects and/or external services (databases, web APIs) to ensure that the individual components of a micro-service work together correctly. They are most commonly grey-box tests and have a lower reliance on mocks.

System Testing These black-box tests verify the functionality of the entire application. As the prior stages would verify the correctness of individual blocks to a significant extent, end-to-end tests seek to verify that the application functions at a high level. It is achieved by running the application against typical use cases and evaluating the resulting behaviour against expected outcomes.

4.3.2 Approach

- **Designing test cases:** Design of test cases depended heavily on the component of the project that was being tested. At a high level, the design of every test was composed of the input data, the order of execution of the test cases and the output of the test.

- **Preparing test data:** Test data was prepared in order to ensure high coverage with the use of credible, reliable and most of all representative data. Additionally, keeping the size of the data to a minimum level was also an objective.

- **Running tests with mock data:** The tests were first run with no data to ensure correct test behaviour. That was followed by testing with invalid data to ensure that correct errors would be raised. The last step was running the test with a valid dataset and ensuring no errors were raised.

- **Interpreting results:** To interpret the results, the design of the test had already specified an expected outcome with which the test result would be compared. An error was raised if there was a mismatch between the two.

4.3.3 Testing Modules

**Exchange Wrapper (xWrap)**

**Testing Suite** xWrap tests were written using the Python standard library unittest module which provided a powerful testing framework as well as a mocking facility (via unittest.mock in Python3). The nose2 package was used as a test-runner to run the test-suite and generate branch and line coverage with coverage.py. A GitLab CI [6] script runs unit and integration tests on every commit and alerts users via Slack and Email of any failures.

**Unit & Integration Testing** Unit tests were written with in-depth knowledge of the underlying code (white box) and individual tests were designed to cover method-level scope. These tests were designed primarily to test correct business logic as well as to ensure robust error-handling on unexpected input/output. In order to test methods independently of their dependencies, Mock objects which synthesised return values were used to replace them. In Python, mocking was easily achieved by way of monkey patching code within test fixtures.

Integration tests take a grey-box or black-box approach to testing and rely on minimal knowledge of internal structure. The aim is to test combinations of unit-tested components for correct functionality. Integration tests for xWrap range from testing the integration of independent test units to the testing of xWrap as a stand-alone micro-service.

**Challenges** xWrap interacts with a number of external web services. In testing, this posed a significant challenge as live services could not be relied upon in tests. Several options were evaluated, but ultimately mocking was used to fake web requests and respond with pre-recorded API responses.

The use of asyncio for cooperative multitasking with co-routines added additional complexity to the testing process as unittest does not natively support testing co-routines. Following a common
workaround for the unittest framework, a lightweight async wrapper was implemented to simplify the test implementation [8]. For integration tests, a server thread was spawned with its own event loop for code under test while the test fixture ran in the main thread. Interactions between the two were performed over local network connections and threadsafe queues.

**Current Status & Improvements** The exchange wrapper currently achieves 90% code coverage and 85% branch coverage through unit and integration tests. Most of the untested lines are in error handling code for system level functions such as thread and socket handling. These facilities will need to be mocked in order to achieve coverage or extensively code-reviewed as the underlying events that trigger them are difficult to simulate inside test fixtures.

**Strategy (Cryptick)**

**Testing Suite** As the trading strategy framework, cryptick, builds on an identical technology stack as the exchange wrapper, it was decided to adapt the same testing suite. A similar GitLab Continuous Integration (CI) chain was also configured using Python’s unittest, nose2 and coverage.py modules.

**Unit testing** White-box unit tests are used on a single class or function scope to ensure each individual part’s correctness. This is particularly crucial for the logic of the trading strategy itself. Without appropriate unit tests, potentially severe errors in the trading logic are hard to identify. For strategy testing, outputs generated from historical data were compared against manually computed values to test the correctness of the implementation.

**Challenges** Large changes in code structure can occur regularly in early stages of the Agile development process. As a result, a very fine-grained unit test suite could become obsolete in subsequent iterations. Therefore, an adaptive approach to unit testing was selected such that tests increase in granularity as the codebase matures.

**Current Status & Improvements** The strategy component currently achieves 65% code coverage through unit and integration tests. Some lines of code are currently untested as they have been added to adhere to functionality that will be implemented in later stages of production. It is projected that once this functionality is added, the strategy component will achieve 80% code coverage.

**Front-end (Cryptock)**

**Testing Suite** The primary testing technology used on the front-end is Facebook’s Jest. This was chosen for its powerful mocking capabilities and simple integration with React (the front-end UI library). Enzyme, a library for shallow rendering, is used to aid UI-testing.

**Unit Testing** The most crucial aspects of the application (state management and inter-component communication) are given the most focus in unit-testing, though all units are thoroughly tested through white- and black-box tests (for those components using external libraries which were not mocked or stubbed out). API units were tested with the help of a mock adapter for our Ajax library (axios-mock-adapter).

**GUI Testing** GUI testing was performed iteratively with each new feature addition. As the front-end was designed in accordance with Google’s Material-UI guidelines, it can be considered a composition of tried-and-tested components, and thus we needed only test this composition of GUI elements, rather than needlessly starting from the ground up.
Challenges   The main challenge for testing the frequently-changing front-end has been the same as that mentioned in 4.3.3. Namely, the challenge has been striking a balance between the test suite being sufficiently coarse to describe application functionality across rapid iterations and sufficiently fine so as not to pass trivially. Accordingly, smoke-tests were employed during the early stages of development until a more concrete structure was settled on, at which point more comprehensive tests were written.

End to End Testing   End-to-end testing is currently performed by running the system in ‘paper’ mode. Due to the different languages and environments (docker containers) that the individual micro-services run in, coverage cannot be measured in this scenario. It is proposed to implement a custom automated end-to-end test suite once interface APIs stabilise. However, as mentioned in Section 4.3.1, the high coverage of individual services minimises the importance of these tests.

4.3.4 Testing Risks & Notable Omissions

The testing methods used so far do not cover comprehensive end-to-end functionality, but are unit-specific to each of the three work streams. This is due to the development workflow which is optimised for time. End-to-end functionality will be black-box tested at a later stage with a fully integrated product.

Some code was ‘smoke-tested’ to ensure stability and basic functionality at this stage of development. This will be further verified in future testing to ensure code correctness.

4.3.5 Current Results

A summary of code coverage for the project is seen below. Full output from the coverage tools are included in Appendix D.

<table>
<thead>
<tr>
<th>Work Unit</th>
<th>Line Count</th>
<th>Line Coverage</th>
<th>Branch Count</th>
<th>Branch Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exchange wrapper (xWrap)</td>
<td>2405</td>
<td>82%</td>
<td>690</td>
<td>85%</td>
</tr>
<tr>
<td>Strategy (Cryptick)</td>
<td>1401</td>
<td>70%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Front-end (Cryptock)</td>
<td>(66%)</td>
<td>67%</td>
<td></td>
<td>50%</td>
</tr>
</tbody>
</table>
4.4 Challenges

Many challenges were encountered and overcome during the development process of the Cryptick platform. A selection of these challenges and the means by which they were addressed are discussed below.

4.4.1 Architecture

The Cryptick platform is composed of multiple components which have widely differing functionality (e.g. strategy development, exchange management, user interface, etc...). Developing these functionalities in parallel inside a monolithic architecture leads to multiple organisational bottlenecks and requires a high degree of coordination between work streams. In addition this approach complicates testing as code units are often tightly coupled with each other and require heavy use of mocking for testing.

This challenge was overcome by organising the project in a micro-service architecture. This design approach allowed the full functionality of the project to be divided into three smaller standalone sub problems that could be independently solved. Pre-defining application programming interfaces (APIs) between these units during early development and regular integration testing mitigated the risk of incompatibility between sub-units at final integration.

4.4.2 Wallet Management

Cryptocurrency can be stored on hardware cold wallets, hot/cold software wallets or on cryptocurrency exchanges. The Cryptick platform currently supports exchange wallets only, and a challenge arises in partitioning funds held on these wallets between strategies that may be executing in parallel. The supported exchanges do not provide any wallet management capability within a single customer account. In addition, transferring funds between cryptocurrency wallets can incur significant transfer fees and introduce processing delays.

This issue was addressed by implementing a logical wallet feature in the xWrap service. A strategy can create a set of logical wallets and allocate funds to them from any unallocated exchange wallet balances. xWrap ensures that any trades generated by the strategy are accepted only if the logical wallets hold sufficient balance. Trades leading to movements or holds of funds on exchange wallets are reflected on logical wallets by the xWrap service ensuring consistency with exchange behaviour. On termination, the strategy can deactivate a logical wallet set and allow users to reallocate those funds to another strategy run.

4.4.3 Poorly Documented Exchange APIs

Certain cryptocurrency exchanges were found to have inaccurate API documentation and, in some cases, non functional APIs. This posed a risk to the implementation of key features of the Cryptick platform. The unpredictable availability of the Bitfinex API and the undocumented nature of the Poloniex websocket API are examples of this challenge.

In all cases, these challenges were overcome by workarounds at the cost of reduced capability or accuracy. Though non-ideal, the effects of these mitigations are systematic to any cryptocurrency trading software that utilises the affected exchanges and cannot be resolved until the underlying bugs are fixed.

4.4.4 Testing Trading Strategies

Unit-testing strategies posed a challenge as, in most cases, extensive mocking was required for the supply of user input (as provided by the front-end) and trade execution (as provided by xWrap). Mocking extensively in this way was not feasible within the time constraints of the project.
This challenge was overcome through a combination of two approaches. Firstly, the backtest feature allowed for the verification of the correctness of a strategy across large timescales. Secondly, integration tests were built to incorporate a running instance of the xWrap service and fake user input from the front-end. The latter tests covered the minutiae of trade execution and response, while the former covered algorithmic correctness. This two-pronged approach allowed for a high test confidence, but also complicated the test coverage reports.

5 Group Work

5.1 Project Milestones

At the start of the project, an initial milestone table was set to track internal deadlines. The status at the conclusion of the project is shown in Table 2.

<table>
<thead>
<tr>
<th>Milestone</th>
<th>Due Date</th>
<th>Completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finalise high-level architecture</td>
<td>31/01/2018</td>
<td>30/01/2018</td>
</tr>
<tr>
<td>Specify interfaces between components</td>
<td>09/02/2018</td>
<td>06/02/2018</td>
</tr>
<tr>
<td>Exchange: Receive exchange data streams</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Front-end: Basic dashboard</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exchange: Make historical OHLC data available</td>
<td>16/02/2018</td>
<td>16/02/2018</td>
</tr>
<tr>
<td>Strategy: Send visualisations to front-end</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strategy: Efficient Frontier strategy making orders</td>
<td>23/02/2018</td>
<td>28/02/2018</td>
</tr>
<tr>
<td>Exchange: Simulate orders on a virtual exchange</td>
<td>23/02/2018</td>
<td>28/02/2018</td>
</tr>
<tr>
<td>Front-end: Display strategy visualisations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exchange: Send logs to front-end</td>
<td>02/03/2018</td>
<td>03/03/2018</td>
</tr>
<tr>
<td>Front-end: Display exchange logs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strategy: Backtest Efficient Frontier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Second Report</td>
<td>05/03/2018</td>
<td>04/03/2018</td>
</tr>
<tr>
<td>Final integration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Product meets minimum requirements</td>
<td>10/03/2018</td>
<td>15/03/2018</td>
</tr>
<tr>
<td>Front-end: render live data from server</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exchange: message hub communications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strategy: general backtesting environment</td>
<td>30/03/2018</td>
<td>14/04/2018</td>
</tr>
<tr>
<td>Exchange: live trading</td>
<td>14/04/2018</td>
<td>15/04/2018</td>
</tr>
<tr>
<td>Front-end: front-end finalised</td>
<td>28/04/2018</td>
<td>15/05/2018</td>
</tr>
<tr>
<td>Final product</td>
<td>30/04/2018</td>
<td>15/05/2018</td>
</tr>
<tr>
<td>Finalise documentation</td>
<td>30/04/2018</td>
<td>11/05/2018</td>
</tr>
<tr>
<td>Final report</td>
<td>16/05/2018</td>
<td>15/05/2018</td>
</tr>
</tbody>
</table>

After the high-level architecture was finalised and the interface between components specified, the focus moved to implementation. OHLC (open-high-low-close) data was made available to the trading strategy, a key dependency for the strategy work stream. This was followed by basic integration of the platform, whereby the efficient frontier portfolio strategy was able to place paper trading orders on a virtual exchange. This integration milestone lagged behind its due date but, due to an extensible and robust code base, the accelerated forward timeline meant that the critical schedule remained mostly unchanged [22].

Completing the final integration took longer than planned. The minimum requirements mentioned in the specification section were all met by mid-March, relatively close to deadline that was set from the initial planning. The majority of the remaining stretch goals were also met, however, that only took place with a two week delay relative to initial deadline. The only goal that was not met by the
time of the report submission was the live trading feature. However, it is planned to implement this
functionality by the date of the software demonstration.

5.2 Group Organisation

Workstreams  As shown by table 5.2, this project consists of three major workstreams, each with
two team members responsible for primary implementation and scheduling. This approach allows for
pair programming and development collaboration between workstreams in the Scrum Sprints. Sub-
team members have been allocated based on previous experience and personal preference. Regular
code reviews with the full team were used in order to coordinate between workstreams in addition to
the Scrum and Kanban practices.

<table>
<thead>
<tr>
<th>Workstream</th>
<th>Team Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trading System</td>
<td>Charith Amarasinghe, Simon Spurrier</td>
</tr>
<tr>
<td>Front-end &amp; UI</td>
<td>Konstantin Hemker, James Griffiths</td>
</tr>
<tr>
<td>Portfolio strategy</td>
<td>Julian Vossen, Dimitris Nikolaou</td>
</tr>
</tbody>
</table>

5.3 Meeting Log

Through the course of the project, a number of well-documented meetings were held which are sum-
marised in Appendix D. After each meeting, notes were circulated via email to each member and To
Dos were assigned on Asana. As the group was relatively close-knit outside of university, there were
numerous undocumented discussions and meetings in less formal settings.

5.4 Work Log

On top of the group meetings, the most representative proxy of the time expenditure of each work-
stream spent on each topic is the commit history. Note that on top of the git history, there were some
code review sessions and pair programming sessions that contributed to the total time expenditure in
table 5.4. A commit history over time with the relevant milestones is available in Appendix D.

6 Final Product

6.1 Project Summary

The Cryptick platform implements a fully featured cryptocurrency trading suite, incorporating a
default portfolio diversification strategy, support for custom strategies, a web-based front-end, and a
unified exchange interface (xWrap).

Adherence to spec  All aspects of the core specification were met. This in addition to key stretch
goals which complete the back-testable strategy architecture.

6.2 Front-end Overview

Run configuration  The final front-end has a tab-based routing system connected by a navigation
bar at the top of each page (see Figure 11). The landing page is the run configuration page (Fig.
11) and allows the user to view its recent runs, add wallets and start a new strategy run. Using the
drop-down menus and sliders, the user can allocate its wallet balances from the different exchanges
to that run. Strategies can be started in three modes, as discussed in the previous sections - paper
trading, backtesting and live trading.
Table 4: Work Log by Topic

<table>
<thead>
<tr>
<th>Workstream/Topics</th>
<th>Time expenditure (in man hrs)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General</strong></td>
<td></td>
</tr>
<tr>
<td>Idea generation &amp; spec</td>
<td>37</td>
</tr>
<tr>
<td>High level design</td>
<td>14</td>
</tr>
<tr>
<td>Project admin</td>
<td>23</td>
</tr>
<tr>
<td>Report writing</td>
<td>163</td>
</tr>
<tr>
<td>Group Meetings</td>
<td>280</td>
</tr>
<tr>
<td><strong>Subtotal</strong></td>
<td>517</td>
</tr>
<tr>
<td><strong>Strategy</strong></td>
<td></td>
</tr>
<tr>
<td>Vis. to front-end</td>
<td>30</td>
</tr>
<tr>
<td>EF making orders</td>
<td>42</td>
</tr>
<tr>
<td>EF backtest</td>
<td>10</td>
</tr>
<tr>
<td>Final API spec</td>
<td>30</td>
</tr>
<tr>
<td>General backtesting env</td>
<td>50</td>
</tr>
<tr>
<td>MessageHub integration</td>
<td>20</td>
</tr>
<tr>
<td>Full integration</td>
<td>40</td>
</tr>
<tr>
<td>Tests</td>
<td>25</td>
</tr>
<tr>
<td><strong>Subtotal</strong></td>
<td>247</td>
</tr>
<tr>
<td><strong>Front-end</strong></td>
<td></td>
</tr>
<tr>
<td>Basic dashboard</td>
<td>38</td>
</tr>
<tr>
<td>Strategy visualisations</td>
<td>31</td>
</tr>
<tr>
<td>Render Orderbook</td>
<td>5</td>
</tr>
<tr>
<td>Render live data</td>
<td>20</td>
</tr>
<tr>
<td>Strategy API calls</td>
<td>15</td>
</tr>
<tr>
<td>Wallet management</td>
<td>28</td>
</tr>
<tr>
<td>Full integration</td>
<td>50</td>
</tr>
<tr>
<td>Tests</td>
<td>32</td>
</tr>
<tr>
<td><strong>Subtotal</strong></td>
<td>302</td>
</tr>
<tr>
<td><strong>xWrap</strong></td>
<td></td>
</tr>
<tr>
<td>Overall architecture</td>
<td>19</td>
</tr>
<tr>
<td>Recv. data streams</td>
<td>35</td>
</tr>
<tr>
<td>OHLC data available</td>
<td>27</td>
</tr>
<tr>
<td>Virtual exchange</td>
<td>51</td>
</tr>
<tr>
<td>Logical wallets</td>
<td>32</td>
</tr>
<tr>
<td>Dashboard</td>
<td>16</td>
</tr>
<tr>
<td>Strategy API</td>
<td>29</td>
</tr>
<tr>
<td>Live book</td>
<td>36</td>
</tr>
<tr>
<td>Full integration</td>
<td>38</td>
</tr>
<tr>
<td>Tests</td>
<td>15</td>
</tr>
<tr>
<td><strong>Subtotal</strong></td>
<td>298</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>1364</td>
</tr>
</tbody>
</table>

**Strategy Tracker**  Once a run has started, a new strategy overview tab appears displaying details about that run. Figure 12 shows the strategy overview page of a running efficient frontier strategy. The blue lines on the graph show the weight history of the portfolio, i.e. the “path” it took to reach the efficient frontier. The efficient frontier is made up of the red points marked in the graph, where each point can be selected to start the diversification process towards those portfolio weights. To provide full customizability of the portfolio, the slider components in Figure 12 allow for the full customisation of the target portfolio. Moreover, actions can be dispatched that pause or update the strategy based on the current selection. Any update that is selected must be confirmed in an extra pop-up window.
before trading starts based on the selection. The strategy page also has an overview of the orderbook which displays trades that were executed based on that strategy (Fig. 18).

**Overview page** The overview page carries any information relevant to the user across all strategies. These include things such as trades executed, wallet distribution across strategies and performance metrics of their strategies. At the time of writing, this page only displayed the trading data (orderbook), but it is expected to be completed by the date of the software demonstration.

**Further features** To further improve the front-end’s functionality one could implement an in-browser editor e.g. through codemirror [13] which allows the user to utilise the API supplied by the front-end to implement custom trading strategies. The functionality on the back-end already exists as a command-line tool, where it is possible write a `strategy.py` file that leads to the execution of this strategy through the `MessageHub` and the `exchange wrapper`. Therefore, providing a clean interface of this functionality to the user would be of great value.

6.3 Strategy Overview

The strategy component supports the execution of different trading strategies, by communicating with xWrap to obtain data and execute trades. The strategy can simulate the execution of trading strategies using historical data in a backtest environment. Strategies may either be pre-defined (e.g. efficient frontier diversification) or user-defined.

**Event-Driven Backtesting** Cryptick implements event-driven backtesting which grants significant advantages over conventional approaches. This means high frequency trading strategies, triggered by particular events, are truly backtestable, assuming no market impact. This is in contrast to popular open source alternatives to Cryptick, where trading logic is periodically executed in a loop, independent of any events or market conditions. Cryptick also support periodic execution by self-scheduling periodic trigger events. As all communication during trading and trading simulations is routed through a central message hub, the system is easily auditable. A screenshot of the message hub dashboard displaying some messages is shown in Figure 14.
Figure 11: Configuring a run on the front-end
<table>
<thead>
<tr>
<th>Status</th>
<th>Side</th>
<th>Pair</th>
<th>Order Size</th>
<th>Limit</th>
<th>Filled Size</th>
<th>Avg Filled Price</th>
<th>Exchange</th>
<th>Run</th>
<th>At</th>
</tr>
</thead>
<tbody>
<tr>
<td>open</td>
<td>sell</td>
<td>BTC_USD</td>
<td>0.5882</td>
<td>7649.57</td>
<td>0.5882</td>
<td>8499.530000000</td>
<td>gdax</td>
<td>ef01</td>
<td>2018-05-15T18:37:32...</td>
</tr>
<tr>
<td>done</td>
<td>sell</td>
<td>BTC_USD</td>
<td>0.5882</td>
<td>7649.57</td>
<td>0.5882</td>
<td>8499.530000000</td>
<td>gdax</td>
<td>ef01</td>
<td>2018-05-15T18:37:32...</td>
</tr>
<tr>
<td>open</td>
<td>sell</td>
<td>BTC_USD</td>
<td>0.5882</td>
<td>7649.57</td>
<td>0.5882</td>
<td>8499.530000000</td>
<td>gdax</td>
<td>ef01</td>
<td>2018-05-15T18:37:32...</td>
</tr>
<tr>
<td>done</td>
<td>sell</td>
<td>BTC_USD</td>
<td>0.5882</td>
<td>7649.57</td>
<td>0.5882</td>
<td>8499.530000000</td>
<td>gdax</td>
<td>ef01</td>
<td>2018-05-15T18:37:32...</td>
</tr>
<tr>
<td>open</td>
<td>buy</td>
<td>ETH_USD</td>
<td>5.6153</td>
<td>783.47</td>
<td>5.6153</td>
<td>712.260000000</td>
<td>gdax</td>
<td>ef01</td>
<td>2018-05-15T18:37:32...</td>
</tr>
<tr>
<td>done</td>
<td>buy</td>
<td>ETH_USD</td>
<td>5.6153</td>
<td>783.47</td>
<td>5.6153</td>
<td>712.260000000</td>
<td>gdax</td>
<td>ef01</td>
<td>2018-05-15T18:37:32...</td>
</tr>
</tbody>
</table>

Figure 13: Front-end Overview Page
User-Defined Strategies  Cryptick supports development of custom strategies by the user which may utilize the back-testable architecture and interface with other services of the platform.

Efficient Frontier  The strategy module includes a default diversification strategy based on Markowitz portfolio theory. The existing portfolio is automatically diversified towards a risk-return optimal point, based on user preferences.

6.4 Exchange Wrapper Overview

xWrap provides a uniform interface to GDAX, Bitfinex, and Poloniex for virtual trading in all supported currency pairs. It implements a live order book and corresponding double trade protection for exchanges that reliably support a live level 2 websocket data feed. An independent dashboard provides additional order monitoring and execution capabilities.

Data gathered from exchanges is available to other platform components and stored for future analysis and use in back-testing.

Virtual Exchange  xWrap implements a virtual exchange with central limit order book matching logic, that matches trades against live market data from supported exchanges. It supports FOK, IOC,
limit and market orders in all trade-able currencies. This includes management of virtual wallets for each exchange and currency.

**Dashboard**  An interactive dashboard allows the user to manually manage orders and monitor trading activity. It is also useful for audit purposes and platform debugging. Additionally, the user may view and manage virtual wallets.

**Live Order Books**  It was not possible to provide a live order book for Poloniex or Bitfinex due to poor quality or non-existent documentation. For example, Poloniex do not officially support or document their push API at all. In these cases, a polled order book is used and immediate double trading is prevented with a trading time delay.

**Live trading**  Comprehensive live trading on-exchange was not implemented due to time and resource constraints. However, it is demonstrable as a proof of concept.

**Data management**  The exchange wrapper stores exchange data and live book update data that it uses in normal operation. This data is accessible to other platform components for analysis and testing.

6.5  Next steps for Cryptick

Assuming future implementation of live trading, whilst reliably operational, the project was constrained by time and resource, so the platform cannot be guaranteed for operation in a live business environment in its current state. However, with further development, the platform could be developed into a stable, viable product for financial applications.

Several options exist for the future of Cryptick:

- **Open source** - Relinquish ownership and allow the open source community to continue development.
- **Private use** - Use the platform for trading personal funds or as a basis for a trading business.
- **Develop as product** - Develop the product for use as a supported financial application for institutional or retail users.
- **Sell codebase** - Sell the codebase as-is or with some further development.
### Exchanges

<table>
<thead>
<tr>
<th>Exchange</th>
<th>Status</th>
<th>Mode</th>
<th>Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdax</td>
<td>🔄 ready</td>
<td>paper</td>
<td>BTC_USD, LTC_BTC, LTC_USD, ETH_USD, BCH_BTC, ETH_BTC, BCH_USD</td>
</tr>
<tr>
<td>poloniex</td>
<td>🔄 ready</td>
<td>paper</td>
<td>XMR_USDT, ZEC_ETH, ETH_USDT, BTC_USDT, ZEC_USDT, LTC_USDT, ETH_BTC, BCH_USDT, BCH_ETH, XMR_BTC, LTC_USD, XMR_BTC, ZEC_BTC, BCH_BTC, LTC_BTC</td>
</tr>
<tr>
<td>cryptocompare</td>
<td>🔄 ready</td>
<td>live</td>
<td>BCH_USD, LTC_BTC, LTC_USD, XMR_USDT, ETH_USD, XMR_BTC, ZEC_USDT, BCH_BTC, ETH_BTC, ZEC_BTC, BTC_USD</td>
</tr>
<tr>
<td>bitfinex</td>
<td>🔄 ready</td>
<td>paper</td>
<td>LTC_BTC, LTC_USD, BCH_BTC, ETH_BTC, BTC_USDT, XMR_USD, BCH_ETH, XMR_BTC, BCH_USDT, ZEC_BTC, ZEC_USDT, ETH_USD</td>
</tr>
</tbody>
</table>

### Wallets

**Wallet Set: 1 (paperex)**

**Paper Exchange Simulated**

<table>
<thead>
<tr>
<th>Id</th>
<th>Exchange</th>
<th>Currency</th>
<th>Balance</th>
<th>Available</th>
<th>Hold</th>
<th>Balance USD</th>
<th>Available USD</th>
<th>Hold USD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>bitfinex</td>
<td>ETH ¥</td>
<td>5402.0</td>
<td>5402.0</td>
<td>0.0</td>
<td>US$3,725,381.26</td>
<td>US$3,725,381.26</td>
<td>US$0.00</td>
</tr>
<tr>
<td>2</td>
<td>bitfinex</td>
<td>BTC ¥</td>
<td>35521.0</td>
<td>35521.0</td>
<td>0.0</td>
<td>US$292,382,586.46</td>
<td>US$292,382,586.46</td>
<td>US$0.00</td>
</tr>
<tr>
<td>3</td>
<td>bitfinex</td>
<td>USD $</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>US$0.00</td>
<td>US$0.00</td>
<td>US$0.00</td>
</tr>
<tr>
<td>4</td>
<td>bitfinex</td>
<td>LTC ¥</td>
<td>161176.0</td>
<td>161176.0</td>
<td>0.0</td>
<td>US$22,029,535.68</td>
<td>US$22,029,535.68</td>
<td>US$0.00</td>
</tr>
</tbody>
</table>

Figure 15: xWrap Exchange Status
### xWrap Order List

**Orders**

**Open Orders**

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>COID</th>
<th>Exchange</th>
<th>Run ID</th>
<th>Type</th>
<th>Side</th>
<th>Status</th>
<th>Pair</th>
<th>Size</th>
<th>Limit</th>
<th>Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Recent Orders**

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>COID</th>
<th>Exchange</th>
<th>Run ID</th>
<th>Type</th>
<th>Side</th>
<th>Status</th>
<th>Pair</th>
<th>Size</th>
<th>Limit</th>
<th>Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>16/05</td>
<td>07:48</td>
<td>0c3760</td>
<td>gdax</td>
<td>ef01</td>
<td>limit</td>
<td>buy</td>
<td>done</td>
<td>ETH_USD</td>
<td>5.6285</td>
<td>781.72</td>
<td>100</td>
</tr>
<tr>
<td>16/05</td>
<td>07:48</td>
<td>8ec66a</td>
<td>gdax</td>
<td>ef01</td>
<td>limit</td>
<td>sell</td>
<td>done</td>
<td>BTC_USD</td>
<td>0.5874</td>
<td>7659.75</td>
<td>100</td>
</tr>
<tr>
<td>16/05</td>
<td>07:47</td>
<td>75c0fc</td>
<td>gdax</td>
<td>ef01</td>
<td>limit</td>
<td>buy</td>
<td>done</td>
<td>ETH_USD</td>
<td>5.6285</td>
<td>781.72</td>
<td>100</td>
</tr>
<tr>
<td>16/05</td>
<td>07:47</td>
<td>fbd807</td>
<td>gdax</td>
<td>ef01</td>
<td>limit</td>
<td>sell</td>
<td>done</td>
<td>BTC_USD</td>
<td>0.5874</td>
<td>7659.75</td>
<td>100</td>
</tr>
<tr>
<td>16/05</td>
<td>07:46</td>
<td>f29ba0</td>
<td>gdax</td>
<td>ef01</td>
<td>limit</td>
<td>buy</td>
<td>done</td>
<td>ETH_USD</td>
<td>5.6285</td>
<td>781.72</td>
<td>100</td>
</tr>
<tr>
<td>16/05</td>
<td>07:46</td>
<td>ad0f8c</td>
<td>gdax</td>
<td>ef01</td>
<td>limit</td>
<td>sell</td>
<td>done</td>
<td>BTC_USD</td>
<td>0.5874</td>
<td>7659.75</td>
<td>100</td>
</tr>
<tr>
<td>16/05</td>
<td>07:45</td>
<td>7f458</td>
<td>gdax</td>
<td>ef01</td>
<td>limit</td>
<td>buy</td>
<td>done</td>
<td>ETH_USD</td>
<td>5.6285</td>
<td>781.72</td>
<td>100</td>
</tr>
<tr>
<td>16/05</td>
<td>07:44</td>
<td>139f10</td>
<td>gdax</td>
<td>ef01</td>
<td>limit</td>
<td>sell</td>
<td>done</td>
<td>BTC_USD</td>
<td>0.5874</td>
<td>7659.75</td>
<td>100</td>
</tr>
<tr>
<td>16/05</td>
<td>07:44</td>
<td>42b3ba</td>
<td>gdax</td>
<td>ef01</td>
<td>limit</td>
<td>buy</td>
<td>done</td>
<td>ETH_USD</td>
<td>5.6285</td>
<td>781.72</td>
<td>100</td>
</tr>
<tr>
<td>16/05</td>
<td>07:43</td>
<td>03e28a</td>
<td>gdax</td>
<td>ef01</td>
<td>limit</td>
<td>sell</td>
<td>done</td>
<td>BTC_USD</td>
<td>0.5874</td>
<td>7659.75</td>
<td>100</td>
</tr>
<tr>
<td>16/05</td>
<td>07:43</td>
<td>d64747</td>
<td>gdax</td>
<td>ef01</td>
<td>limit</td>
<td>buy</td>
<td>done</td>
<td>ETH_USD</td>
<td>5.6285</td>
<td>781.72</td>
<td>100</td>
</tr>
<tr>
<td>16/05</td>
<td>07:42</td>
<td>686cd8</td>
<td>gdax</td>
<td>ef01</td>
<td>limit</td>
<td>sell</td>
<td>done</td>
<td>BTC_USD</td>
<td>0.5874</td>
<td>7659.75</td>
<td>100</td>
</tr>
<tr>
<td>16/05</td>
<td>07:41</td>
<td>07b4c7</td>
<td>gdax</td>
<td>ef01</td>
<td>limit</td>
<td>buy</td>
<td>done</td>
<td>ETH_USD</td>
<td>5.6285</td>
<td>781.72</td>
<td>100</td>
</tr>
<tr>
<td>16/05</td>
<td>07:41</td>
<td>232c5e</td>
<td>gdax</td>
<td>ef01</td>
<td>limit</td>
<td>sell</td>
<td>done</td>
<td>BTC_USD</td>
<td>0.5874</td>
<td>7659.75</td>
<td>100</td>
</tr>
</tbody>
</table>

Figure 16: xWrap Order List
### xWrap Wallet List

**Wallet Set: 1 (paperex)**

Paper Exchange Simulated

<table>
<thead>
<tr>
<th>Id</th>
<th>Exchange</th>
<th>Currency</th>
<th>Balance</th>
<th>Available</th>
<th>Hold</th>
<th>Balance USD</th>
<th>Available USD</th>
<th>Hold USD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>bitfinex</td>
<td>ETH</td>
<td>5402.0</td>
<td>5402.0</td>
<td>0.0</td>
<td>US$3,725,327.24</td>
<td>US$3,725,327.24</td>
<td>US$0.00</td>
</tr>
<tr>
<td>2</td>
<td>bitfinex</td>
<td>BTC</td>
<td>35521.0</td>
<td>35521.0</td>
<td>0.0</td>
<td>US$292,310,123.62</td>
<td>US$292,310,123.62</td>
<td>US$0.00</td>
</tr>
<tr>
<td>3</td>
<td>bitfinex</td>
<td>USD $</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>US$0.00</td>
<td>US$0.00</td>
<td>US$0.00</td>
</tr>
<tr>
<td>4</td>
<td>bitfinex</td>
<td>LTC</td>
<td>161176.0</td>
<td>161176.0</td>
<td>0.0</td>
<td>US$22,013,418.08</td>
<td>US$22,013,418.08</td>
<td>US$0.00</td>
</tr>
<tr>
<td>5</td>
<td>bitfinex</td>
<td>XMR</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>US$0.00</td>
<td>US$0.00</td>
<td>US$0.00</td>
</tr>
<tr>
<td>6</td>
<td>bitfinex</td>
<td>ZEC</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>US$0.00</td>
<td>US$0.00</td>
<td>US$0.00</td>
</tr>
<tr>
<td>7</td>
<td>bitfinex</td>
<td>BCH</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>US$0.00</td>
<td>US$0.00</td>
<td>US$0.00</td>
</tr>
</tbody>
</table>

**Make Transaction**

**Movement:**

0

A positive or negative number.

*Figure 17: xWrap Wallet List*
Logs

xWrap Version: 0.5.5
Currencies: ["ZEC", "BTC", "ETH", "USD", "XMR", "LTC", "USDT", "BCH"]
Exchanges: ["gdax", "poloniex", "bitfinex"]

Figure 18: xWrap Logs
References


[18] S. Spurrier and C. Amarasinghe. xwrap documentation 0.4.3. https://www.doc.ic.ac.uk/project/2017/530/g1753006/xwrap_docs/v0.4/, April 2018.


Glossary

backtesting  Backtesting is the process of mocking a strategy run in paper trading mode over past time, usually in order to test its performance before live trading. 26

CLI  Command-line interface. 6

DBMS  Database Management System. 6

efficient frontier  The efficient frontier is the most efficient portfolio mix of any combination of assets that optimises for risk (std. dev.) and return. 17

FOK  Fill or kill order. The order is immediately executed in its full size at its limit or is cancelled. 20

high-frequency trading  A style of trading whereby a large number of orders are placed in a short period of time that may respond to market conditions at very fast speeds. 1

IOC  Immediate or cancel order. The order is executed against all liquidity priced better than its limit and any residual is cancelled. 20

limit order  An order executed against any liquidity better than the limit. Any amount not traded will remain on the order book at the limit price. 20

live trading  Trading real money on real exchanges. 20

logical wallet  Logical partition of funds within an exchange wallet. 9

market cap  The number of shares of a stock multiplied by the quantity issued. Taking inspiration from this, the market capitalisation of a cryptocurrency is the value of one unit multiplied by the number of units in circulation. 1

market order  An order executed in its full size at the best available price at time of execution. 20

order book  A record of buy and sell orders with price and size, ordered by price and time of order submission. 9

paper trading  Paper trading is a trading feature that attempts to reproduce all features of a live trading market so that a user may practice or test trade behaviour without financial risk. 20

portfolio  A portfolio is a grouping of financial assets, usually comprising of many different trade-able assets on financial markets, such as a stocks, bonds, and cryptocurrencies. 1

run  A strategy run is an instantiation of a runnable. Note that there can be multiple runs of a single runnable. 44

runnable  Runnable strategies are the set of all strategies available to the user that he can start. 44

trading pair  Commonly referred to as currency pair is the quotation and pricing structure of the currencies traded in the foreign exchange market. An example of a trading pair would be USD/EUR or BTC/ETH in this context. 1

UI  User interface. 2

virtual wallet  A simulated exchange wallet. 33
Appendices

A Installation Instructions

A.1 System Requirements

It is recommended that the host environment meets the following requirements:

- A dual-core CPU running at 2GHz or higher
- Minimum 4GB of RAM (2GB if InfluxDB hosted separately)
- Ubuntu 16.04 or later (Ubuntu 16.04 LTS recommended)
- CPython 3.6 or later
- Docker CE 1.8 or later
- Docker Compose 1.21.2 or later
- InfluxDB (OSS) 1.5.0 or later
- NodeJS 9.0 or later

A.2 Installing Dependencies

Each host machine should have the required dependencies installed. The easiest way to accomplish this is by using the scripts in the devops package bundled with this project.

InfluxDB and Cryptick can be co-located on the same VM, but due to InfluxDB’s greedy memory allocation, it is best if it is hosted on a dedicated VM when used in production.

Deployment

1. To run Cryptick, InfluxDB and xWrap, the easiest solution is to use the bundled docker-compose.yml file.

    docker-compose build

2. To bring up the services:

    docker-compose up

3. This should bring up the services on the ports 4200 (xWrap RPC), 4242 (xWrap HTTP API), 5726 (Cryptick API) and 8086 (InfluxDB). These can be tested by:

    curl localhost:5726/runnables
    curl localhost:4242/xwrap/v1/status

4. To run the cryptick front-end, npm must be installed and a reverse proxy must should be used:

    cd cryptick
    npm install
    npm build-paper
    npm serve-paper

5. The front-end server will now be up on port 8200. Proxy this to port 80 (or for testing, use reverse SSH tunneling).

    ssh -L 8200:localhost:8200 remote_server_address
### B  xWrap API Documentation

**Exchange Interaction API**  The Exchange Interaction API is supplied by the xWrap microservice and is consumed by both strategy instances and the Front-end UI. To support both these use-cases, a RPC based and HTTP based API is provided. The main xWrap API methods are outlined in Table 5, detailed documentation can be found in the xWrap Reference Documentation[18].

**RPC API**  RPC API allows connected clients to execute methods on the xWrap service as if they were local function calls. This functionality requires a message format that encodes a function name, arguments list and return value into a network packet, and stub functions that marshal and unmarshal Python function calls into these messages.

Multiple RPC implementations were evaluated including gRPC, XML-RPC and JSON-RPC. The JSON-RPC format transported over WebSockets was eventually selected due to its simplicity and wide platform support. The tinyrpc library for Python is used for standards compliant message serialisation and de-serialisation.

JSON-RPC allows the transport of method arguments either as a list (positional arguments in Python) or a dictionary (keyword arguments in Python), but not both. It was decided to utilise keyword arguments across all RPC APIs in order to reduce the risk of mistakenly swapping arguments in a function call. An example Python client API call is shown below:

```python
wallet = xwrapapi.get_wallet(exchange="gdax", base="BTC", wallet_set_id=2)
```

When executed, the stub function for `get_wallet` will encode the specified parameters into a JSON-RPC message and send it to the server via the RPC websocket. The serialised wire-format of the above call is as follows:

```json
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "get_wallet",
  "args": {"exchange": "gdax", "base": "BTC"}
}
```

The server decodes this message and maps it to an internal Python function call. This server-side method is declared with the signature `get_wallet(self, *, exchange, base)`. This definition uses Python keyword only argument syntax and will result in the remote call failing if there is an argument mismatch[19]. An example server response is shown below.

```json
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {"balance": "100.0", "id": 9}
}
```
After de-serialisation on the client-side, the `wallet` variable will now be set to the python `dict` 
{"balance": "100.0", "id": 9}. The JSON-RPC standards document can be consulted for a more detailed description of the protocol[9].

**HTTP API** The HTTP API provides easy access to most data held by the xWrap service such as wallet balances and open orders. Each exposed API method maps to a URI on a xWrap HTTP server and request parameters can be passed as part of the URI structure. For example, the `get_wallet` API could be invoked via the HTTP API as a simple HTTP GET call as follows:

```
GET /xwrap/v1/wallets/gdax/BTC HTTP/1.1
Host: cloud-vm-45-137.doc.ic.ac.uk
Connection: keep-alive
```

With the matching response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Connection: Closed

{"balance": "100.0", "id": 9}
```

API calls that fetch data from the server are implemented by HTTP GET methods while calls that change or add data to the server are implemented with HTTP POST. APIs were designed with REST (Representational State Transfer) concepts in mind, but adherence to these concepts was not strict[4].

The API version in the URL string allows backward incompatible changes to API formats to be deployed without breaking support for existing clients.

**Notifications** The xWrap service generates notification messages whenever it becomes aware of an update from one or more upstream services such as cryptocurrency exchanges or data providers. JSON-RPC provides a convenient mechanism to deliver these messages to clients (JSON-RPC notifications), however the HTTP 1.1 protocol does not provide a mechanism to push data from the server to clients in this way. The `SocketIO` library is thus used to serve notifications to HTTP clients by way of WebSocket connections initiated through the HTTP server.

**Alternatives** The choice of building JSON-RPC over Websocket and HTTP APIs as discussed above was mainly due to their simplicity and cross-platform capability in a micro-service environment. Several other RPC formats and transports were considered such as:

**Message Queue (Transport)** Message Queues are a common method of inter-process communication in a microservice environment. Options range from brokered queues such as RabbitMQ to lightweight decentralised alternatives such as NSQ or ZMQ. However, as the microservices described above are not to be used in a clustered environment, message queues would add unnecessary overhead for little benefit.

**TCP Network Sockets (Transport)** Plain TCP sockets could be used for transporting API call payloads, however this introduces new requirements such as message framing (if streaming API calls) and the implementation of an additional compatibility layer for web browser clients. The WebSocket and HTTP protocols do not have these disadvantages.

**MsgPack or Protobuf RPC (Format)** MsgPack and Protobuf are alternative means to serialise RPC calls over the wire. Both would result in much shorter message lengths and, in the latter case, means of defining a strict schema. The disadvantage of both however is the lack of human readability. As all the above microservices operate on a single machine, the additional data overhead has negligible impact.
gRPC (Transport+Format) gRPC is a powerful framework for developing cross-platform, cross-language RPC services using Protobuf and HTTP/2. However, its design requires the automated generation of stub code at both client and server and adds additional dependencies. As the APIs being built are mainly for internal use in a non-cluster environment, it was thought that employing gRPC would add too much development overhead.

C Strategy API

The strategy API is implemented via HTTP GET and PUT calls.

<table>
<thead>
<tr>
<th>API call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET/runnables</td>
<td>Gets list of runnable strategies</td>
</tr>
<tr>
<td>GET/runs</td>
<td>Get list of all runs and their status</td>
</tr>
<tr>
<td>GET/runs/run-id</td>
<td>Get details on a run</td>
</tr>
<tr>
<td>GET/runs/run-id/components</td>
<td>Get list of all UI components created by a run with given ID</td>
</tr>
<tr>
<td>GET/runs/run-id/messages</td>
<td>Returns all messages that went through the message hub during a run</td>
</tr>
<tr>
<td>PUT/runnables/runnable-name/start</td>
<td>Runs a strategy with given name and returns run-ID</td>
</tr>
<tr>
<td>PUT/run/run-id/stop</td>
<td>Stops the run with given ID</td>
</tr>
<tr>
<td>PUT/runs/run-id/components/component-id</td>
<td>Send json-encoded data to specific component</td>
</tr>
</tbody>
</table>

**HTTP GET API calls** GET methods allow the front-end to query data on running and runnable strategies. GET methods are called by front-end components when rendering to query relevant data. A full list of GET methods can be found in Table 1 as well as the full documentation (citation needed). An example response from the back-end for querying the actively running strategies would be the following:

```
GET /runs HTTP/1.1
Host: localhost
Connection: close

HTTP/1.1 200 OK
Content-Type: application/json
Connection: Closed

{
    "demostrategy_1": {
        "mode": "backtesting",
        "run_id": "demostrategy_1",
        "running": false,
        "strategy": "DemoStrategy"
    },
    "echostrategy_0": {
        "mode": "paper",
        "run_id": "echostrategy_0",
        "running": false,
        "strategy": "EchoStrategy"
    },
    "efficientfrontier_0": {
```
"mode": "live",
"run_id": "efficientfrontier_0",
"running": false,
"strategy": "EfficientFrontier"
}
}

Note that by differentiating between runnables and running strategies, the API allows for creating multiple runs (instantiations) of one type of runnable. For example, if a user wanted to run multiple efficient frontier strategies at the same time, this would be possible by creating multiple runs of a runnable strategy type "EfficientFrontier".

**HTTP PUT API calls**  PUT methods are usually used to start and stop runnables/running strategies or to adjust the portfolio weights of a running strategy. The full list of PUT methods can be found in Table 1 and a sample request to start a backtest run can be found below:

```http
PUT /runnables/HodlStrategy/start HTTP/1.1
Host: localhost
Content-Type: application/json
Connection: close

{
    "mode": "backtest",
    "start": 1523003864,
    "end": 1523823864,
    "balances": [
        {
            "exchange": "gdax",
            "base": "USD",
            "balance": 1000
        },
        {
            "exchange": "gdax",
            "base": "BTC",
            "balance": 10
        }
    ]
}
```
D  Test coverage

D.1  Strategy coverage

<table>
<thead>
<tr>
<th>Module</th>
<th>statements</th>
<th>missing</th>
<th>excluded</th>
<th>coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>cryptick/<strong>init</strong>.py</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>cryptick/api/<strong>init</strong>.py</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>cryptick/api/filewriter.py</td>
<td>87</td>
<td>36</td>
<td>0</td>
<td>59%</td>
</tr>
<tr>
<td>cryptick/api/socket.py</td>
<td>34</td>
<td>20</td>
<td>0</td>
<td>41%</td>
</tr>
<tr>
<td>cryptick/exchange/<strong>init</strong>.py</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>cryptick/exchange/xsim/<strong>init</strong>.py</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>cryptick/exchange/xsim/xsim.py</td>
<td>150</td>
<td>66</td>
<td>0</td>
<td>56%</td>
</tr>
<tr>
<td>cryptick/exchange/xsim/xwrapprovider.py</td>
<td>77</td>
<td>67</td>
<td>0</td>
<td>13%</td>
</tr>
<tr>
<td>cryptick/exchange/xwrap/<strong>init</strong>.py</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>cryptick/exchange/xwrap/api.py</td>
<td>118</td>
<td>47</td>
<td>0</td>
<td>60%</td>
</tr>
<tr>
<td>cryptick/exchange/xwrap/interface.py</td>
<td>92</td>
<td>58</td>
<td>0</td>
<td>37%</td>
</tr>
<tr>
<td>cryptick/messagehub.py</td>
<td>102</td>
<td>9</td>
<td>0</td>
<td>91%</td>
</tr>
<tr>
<td>cryptick/run.py</td>
<td>60</td>
<td>24</td>
<td>0</td>
<td>60%</td>
</tr>
<tr>
<td>cryptick/server.py</td>
<td>222</td>
<td>106</td>
<td>0</td>
<td>52%</td>
</tr>
<tr>
<td>cryptick/strategy/<strong>init</strong>.py</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>cryptick/strategy/component_demo.py</td>
<td>28</td>
<td>8</td>
<td>0</td>
<td>71%</td>
</tr>
<tr>
<td>cryptick/strategy/demo.py</td>
<td>30</td>
<td>2</td>
<td>0</td>
<td>93%</td>
</tr>
<tr>
<td>cryptick/strategy/echo.py</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>cryptick/strategy/efficient_frontier/<strong>init</strong>.py</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>cryptick/strategy/efficient_frontier/ef_strategy.py</td>
<td>159</td>
<td>114</td>
<td>0</td>
<td>28%</td>
</tr>
<tr>
<td>cryptick/strategy/efficient_frontier/ef_util.py</td>
<td>159</td>
<td>10</td>
<td>0</td>
<td>94%</td>
</tr>
<tr>
<td>cryptick/strategy/hodl.py</td>
<td>31</td>
<td>15</td>
<td>0</td>
<td>52%</td>
</tr>
<tr>
<td>cryptick/strategy/strategy.py</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>cryptick/utils/<strong>init</strong>.py</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>cryptick/utils/sync.py</td>
<td>25</td>
<td>1</td>
<td>0</td>
<td>96%</td>
</tr>
</tbody>
</table>

**Total**  

<table>
<thead>
<tr>
<th>statements</th>
<th>missing</th>
<th>excluded</th>
<th>coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1401</td>
<td>583</td>
<td>0</td>
<td>58%</td>
</tr>
</tbody>
</table>

Figure 19: Strategy - Unit Test Coverage Report
## Coverage report: 70%

<table>
<thead>
<tr>
<th>Module</th>
<th>statements</th>
<th>missing</th>
<th>excluded</th>
<th>coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>cryptick/_init_py</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>cryptick/api/_init_py</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>cryptick/api/filewriter.py</td>
<td>87</td>
<td>18</td>
<td>0</td>
<td>79%</td>
</tr>
<tr>
<td>cryptick/api/socket.py</td>
<td>34</td>
<td>20</td>
<td>0</td>
<td>41%</td>
</tr>
<tr>
<td>cryptick/exchange/_init_py</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>cryptick/exchange/xsim/_init_py</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>cryptick/exchange/xsim/xsim.py</td>
<td>150</td>
<td>12</td>
<td>0</td>
<td>92%</td>
</tr>
<tr>
<td>cryptick/exchange/xsim/xw rapp ro_provider.py</td>
<td>77</td>
<td>15</td>
<td>0</td>
<td>81%</td>
</tr>
<tr>
<td>cryptick/exchange/xw_rap/_init_py</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>cryptick/exchange/x_rap/api.py</td>
<td>118</td>
<td>13</td>
<td>0</td>
<td>89%</td>
</tr>
<tr>
<td>cryptick/exchange/x_rap/_interface.py</td>
<td>92</td>
<td>10</td>
<td>0</td>
<td>89%</td>
</tr>
<tr>
<td>cryptick/messagehub.py</td>
<td>102</td>
<td>19</td>
<td>0</td>
<td>81%</td>
</tr>
<tr>
<td>cryptick/run.py</td>
<td>60</td>
<td>24</td>
<td>0</td>
<td>60%</td>
</tr>
<tr>
<td>cryptick/server.py</td>
<td>222</td>
<td>222</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>cryptick/strategy/_init_py</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>cryptick/strategy/component_demo.py</td>
<td>28</td>
<td>18</td>
<td>0</td>
<td>36%</td>
</tr>
<tr>
<td>cryptick/strategy/demo.py</td>
<td>30</td>
<td>15</td>
<td>0</td>
<td>50%</td>
</tr>
<tr>
<td>cryptick/strategy/_echo.py</td>
<td>10</td>
<td>6</td>
<td>0</td>
<td>40%</td>
</tr>
<tr>
<td>cryptick/strategy/efficient_ frontier/_init_py</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>cryptick/strategy/efficient_ frontier/_of_strategy.py</td>
<td>159</td>
<td>15</td>
<td>0</td>
<td>91%</td>
</tr>
<tr>
<td>cryptick/strategy/efficient_ frontier/_of_util.py</td>
<td>159</td>
<td>10</td>
<td>0</td>
<td>94%</td>
</tr>
<tr>
<td>cryptick/strategy/hedl.py</td>
<td>31</td>
<td>6</td>
<td>0</td>
<td>81%</td>
</tr>
<tr>
<td>cryptick/strategy/strategy.py</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>cryptick/utils/_init_py</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>cryptick/utils_sync.py</td>
<td>25</td>
<td>1</td>
<td>0</td>
<td>96%</td>
</tr>
</tbody>
</table>

**Total**: 1401 424 0 70%

Figure 20: Strategy - Integration Test Coverage Report
### D.2 Front-end coverage

![Front-end coverage report](image)

**Figure 21: Front-end (Cryptock) - Test Coverage Report**

<table>
<thead>
<tr>
<th>File</th>
<th>%Stmts</th>
<th>%Branch</th>
<th>%Funcs</th>
<th>%Lines</th>
<th>Uncovered Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>47.5</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
D.3 xWrap coverage

Coverage report: 82%

<table>
<thead>
<tr>
<th>Module</th>
<th>statements</th>
<th>missing</th>
<th>excluded</th>
<th>branches</th>
<th>partial</th>
<th>coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>exchange_wrapper/<strong>init</strong>.py</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>exchange_wrapper/client.py</td>
<td>104</td>
<td>29</td>
<td>0</td>
<td>18</td>
<td>1</td>
<td>70%</td>
</tr>
<tr>
<td>exchange_wrapper/config.py</td>
<td>47</td>
<td>5</td>
<td>0</td>
<td>22</td>
<td>2</td>
<td>87%</td>
</tr>
<tr>
<td>exchange_wrapper/connection.py</td>
<td>98</td>
<td>13</td>
<td>0</td>
<td>20</td>
<td>7</td>
<td>83%</td>
</tr>
<tr>
<td>exchange_wrapper/data_sinks/<strong>init</strong>.py</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>exchange_wrapper/data_sinks/influx.py</td>
<td>76</td>
<td>13</td>
<td>0</td>
<td>18</td>
<td>2</td>
<td>76%</td>
</tr>
<tr>
<td>exchange_wrapper/data_sinks/sink.py</td>
<td>22</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>95%</td>
</tr>
<tr>
<td>exchange_wrapper/db.py</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>exchange_wrapper/exchange/<strong>init</strong>.py</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>exchange_wrapper/exchange/base.py</td>
<td>200</td>
<td>28</td>
<td>0</td>
<td>56</td>
<td>6</td>
<td>84%</td>
</tr>
<tr>
<td>exchange_wrapper/exchange/bitfinex.py</td>
<td>57</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>exchange_wrapper/exchange/cryptocompare.py</td>
<td>84</td>
<td>9</td>
<td>0</td>
<td>20</td>
<td>3</td>
<td>88%</td>
</tr>
<tr>
<td>exchange_wrapper/exchange/gdax.py</td>
<td>132</td>
<td>8</td>
<td>0</td>
<td>42</td>
<td>1</td>
<td>95%</td>
</tr>
<tr>
<td>exchange_wrapper/exchange/poloniex.py</td>
<td>62</td>
<td>1</td>
<td>0</td>
<td>14</td>
<td>1</td>
<td>97%</td>
</tr>
<tr>
<td>exchange_wrapper/exchange/virtual.py</td>
<td>146</td>
<td>21</td>
<td>0</td>
<td>60</td>
<td>13</td>
<td>80%</td>
</tr>
<tr>
<td>exchange_wrapper/http_server.py</td>
<td>238</td>
<td>72</td>
<td>0</td>
<td>46</td>
<td>17</td>
<td>66%</td>
</tr>
<tr>
<td>exchange_wrapper/live_book/<strong>init</strong>.py</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>exchange_wrapper/live_book/live_book.py</td>
<td>57</td>
<td>23</td>
<td>0</td>
<td>36</td>
<td>2</td>
<td>52%</td>
</tr>
<tr>
<td>exchange_wrapper/logical_wallet.py</td>
<td>74</td>
<td>9</td>
<td>0</td>
<td>36</td>
<td>4</td>
<td>85%</td>
</tr>
<tr>
<td>exchange_wrapper/matching.py</td>
<td>37</td>
<td>1</td>
<td>0</td>
<td>22</td>
<td>1</td>
<td>97%</td>
</tr>
<tr>
<td>exchange_wrapper/models.py</td>
<td>166</td>
<td>8</td>
<td>0</td>
<td>42</td>
<td>4</td>
<td>94%</td>
</tr>
<tr>
<td>exchange_wrapper/pair.py</td>
<td>22</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>96%</td>
</tr>
<tr>
<td>exchange_wrapper/pc.py</td>
<td>92</td>
<td>3</td>
<td>7</td>
<td>12</td>
<td>0</td>
<td>97%</td>
</tr>
<tr>
<td>exchange_wrapper/service.py</td>
<td>414</td>
<td>93</td>
<td>55</td>
<td>144</td>
<td>30</td>
<td>73%</td>
</tr>
<tr>
<td>exchange_wrapper/util/<strong>init</strong>.py</td>
<td>30</td>
<td>2</td>
<td>0</td>
<td>14</td>
<td>0</td>
<td>95%</td>
</tr>
<tr>
<td>exchange_wrapper/util/token_bucket.py</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>exchange_wrapper/wallet.py</td>
<td>27</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>93%</td>
</tr>
<tr>
<td>exchange_wrapper/wsock_client.py</td>
<td>50</td>
<td>7</td>
<td>0</td>
<td>12</td>
<td>2</td>
<td>82%</td>
</tr>
<tr>
<td>exchange_wrapper/wsock_server.py</td>
<td>109</td>
<td>9</td>
<td>0</td>
<td>36</td>
<td>5</td>
<td>88%</td>
</tr>
<tr>
<td>exchange_wrapper/wsocklog/<strong>init</strong>.py</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>exchange_wrapper/wsocklog/formatter.py</td>
<td>14</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>62%</td>
</tr>
<tr>
<td>exchange_wrapper/wsocklog/handler.py</td>
<td>22</td>
<td>6</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>67%</td>
</tr>
</tbody>
</table>

Total 2405 368 62 690 101 82%

Figure 22: xWrap - Test Coverage Report

E Meeting Log

Table 7: Meeting log

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
<th>Main issues</th>
<th>Ext. present</th>
</tr>
</thead>
<tbody>
<tr>
<td>30 min 20/04/2018</td>
<td>Next Steps for Cryptick</td>
<td>Final Integration, post-project use</td>
<td>W. Knottenbelt, S. Werner</td>
</tr>
<tr>
<td>2h 11/04/2018</td>
<td>Integration</td>
<td>Report 3 to dos, final deadline to dos</td>
<td>None</td>
</tr>
<tr>
<td>2h 30/03/2018</td>
<td>Integration</td>
<td>API issues, L3 orderbook, wallet management</td>
<td>None</td>
</tr>
</tbody>
</table>

48
<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
<th>Main issues</th>
<th>Ext. present</th>
</tr>
</thead>
<tbody>
<tr>
<td>3h 24/03/2018</td>
<td>Workstreams</td>
<td>Currency conversion issues, L3 orderbook, Strategy backtesting</td>
<td>None</td>
</tr>
<tr>
<td>2h 21/03/2018</td>
<td>Integration</td>
<td>Wallet Management, detailed API discussion, etc.</td>
<td>None</td>
</tr>
<tr>
<td>30 min 14/03/2018</td>
<td>Progress update</td>
<td>Individual workstreams on track, final implementation</td>
<td>W. Knottenbelt</td>
</tr>
<tr>
<td>2h 28/02/2018</td>
<td>Report 2</td>
<td>Finalising report 2</td>
<td>None</td>
</tr>
<tr>
<td>4h 21/02/2018</td>
<td>Integration, Testing, Backtesting</td>
<td>Integrating xWrap with Strategy, Testing strategies, Backtester</td>
<td>None</td>
</tr>
<tr>
<td>4h 18/02/2018</td>
<td>Interaction front-end with strategy</td>
<td>xWrap update, MessageHub status, Web Socket clients</td>
<td>None</td>
</tr>
<tr>
<td>3h 13/02/2018</td>
<td>Workstream communication</td>
<td>MessageHub implementation</td>
<td>None</td>
</tr>
<tr>
<td>3h 09/02/2018</td>
<td>Workstream communication</td>
<td>Data exchange strategy to front-end, paper trading GDAX, web socket clients</td>
<td>None</td>
</tr>
<tr>
<td>4h 05/02/2018</td>
<td>Detailed functionality per workstream</td>
<td>MessageHub format, min API spec, strategy backtesting, front-end data requirements</td>
<td>None</td>
</tr>
<tr>
<td>30 min 25/01/2018</td>
<td>Project Outline</td>
<td>Expand on main project objectives</td>
<td>W. Knottenbelt</td>
</tr>
<tr>
<td>2h 21/01/2018</td>
<td>Report</td>
<td>Report 1, further architecture decisions</td>
<td>None</td>
</tr>
<tr>
<td>3h 17/01/2018</td>
<td>Further implementation</td>
<td>Further implementation, front-end wireframes, xWrap optimisation</td>
<td>None</td>
</tr>
<tr>
<td>3h 14/01/2018</td>
<td>Further implementation</td>
<td>Efficient frontier discussion, data wish lists, communication of workstreams (message-Hub)</td>
<td>None</td>
</tr>
<tr>
<td>4h 11/01/2018</td>
<td>Starting implementation</td>
<td>Implementing the efficient frontier, High-level architecture, Report One To Dos</td>
<td>None</td>
</tr>
<tr>
<td>2h 09/01/2018</td>
<td>Term outline</td>
<td>Main project schedule and milestones, election of responsibilities</td>
<td>None</td>
</tr>
<tr>
<td>2h 18/12/2017</td>
<td>Internal Project Kick-off</td>
<td>Problem breakdown, overview of trading processes, value chain of the product</td>
<td>None</td>
</tr>
</tbody>
</table>
E.1 Commit Log

E.1.1 Strategy

Figure 23: Strategy - Commit History January 17th 2018 - May 15th 2018

E.1.2 Front-end

Figure 24: Front-end - Commit History January 17th 2018 - May 15th 2018
E.1.3 xWrap

Figure 25: xWrap - Commit History January 17th 2018 - May 15th 2018

E.2 Facts and Figures

<table>
<thead>
<tr>
<th>CRYPTICK –FACTS AND FIGURES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total No. of Lines</td>
</tr>
<tr>
<td>Total No. of Files</td>
</tr>
<tr>
<td>Total No. of Commits</td>
</tr>
<tr>
<td>No. of Issues</td>
</tr>
<tr>
<td>No. of merges</td>
</tr>
<tr>
<td>No. of sentry errors</td>
</tr>
<tr>
<td>Total Hours Worked</td>
</tr>
<tr>
<td>Litres of coffee consumed</td>
</tr>
<tr>
<td>No. of poor jokes made</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Files</th>
<th>Comment</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaScript</td>
<td>105</td>
<td>725</td>
<td>9,670</td>
</tr>
<tr>
<td>Python</td>
<td>85</td>
<td>1,564</td>
<td>6,946</td>
</tr>
<tr>
<td>HTML</td>
<td>3</td>
<td>21</td>
<td>188</td>
</tr>
<tr>
<td>Bourne Shell</td>
<td>6</td>
<td>0</td>
<td>64</td>
</tr>
<tr>
<td>YAML</td>
<td>5</td>
<td>0</td>
<td>61</td>
</tr>
<tr>
<td>CSS</td>
<td>2</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>Make</td>
<td>1</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>SUM</td>
<td>2,017</td>
<td>2,316</td>
<td>16,956</td>
</tr>
</tbody>
</table>

Figure 26: Cryptick - Facts and Figures
1 Contents
  1.1 Getting Started ................................. 3
  1.2 Strategy API ..................................... 5
  1.3 Frontend API .................................... 6
  1.4 Exposed Functions .............................. 10
  1.5 Notifications ................................... 17
  1.6 exchange_wrappper package ....................... 20

HTTP Routing Table .............................. 41

Python Module Index .............................. 43

Index .............................................. 45
The xWrap module provides a web-service that abstracts different trading and wallet management APIs from cryptocurrency exchanges to a common RPC friendly API. The wrapper seeks to bridge the different capabilities provided by upstream exchanges such that client code can largely be exchange agnostic.

xWrap also provides basic paper trading functionality against supported cryptocurrency exchanges.
1.1 Getting Started

To run xWrap in development make sure a valid virtual environment is activated with at least Python 3.6.

```bash
# build the dashboard
$ cd dashboard/
$ npm install
$ npm run build

$ virtualenv --python=/usr/bin/python3 env
$ . env/bin/activate
$ pip install -r requirements.txt
$ export INFLUXDB_DSN="influxdb://influx.cryptick.fun/cryptick"
$ export SQLITE_DB="xwrap.db"
$ python -m exchange_wrapper.service init_db $SQLITE_DB
$ python -m exchange_wrapper.service run --config /path/to/config.json
```

For production environments, use the Docker image.

1.1.1 Configuration

The xWrap service can be configured via environment variables and a configuration file.

Configuration JSON File

Use the `settings-sample.json` file in the root directory as a starting point. The configuration file sets the available currencies and exchange classes to load.

```json
{
    "currencies": [
        "ZEC",
        "BTC",
        "ETH",
        "USD",
        "XMR",
        "LTC",
        "USDT",
        "BCH"
    ],
```

(continues on next page)
"exchanges": 
  "gdax": { 
    "mode": "paper",
    "class": "exchange_wrapper.exchange.gdax.GDAXExchange"
  },
  "poloniex": { 
    "mode": "paper",
    "class": "exchange_wrapper.exchange.poloniex.PoloniexExchange"
  },
  "bitfinex": { 
    "mode": "paper",
    "class": "exchange_wrapper.exchange.bitfinex.BitfinexExchange"
  }
}

Required Environment Variables

SQLite

SQLite is used to store order state information and wallet transactions. The SQLITE_DB environment variable must be set to a path for the database file.

To setup the initial database:

```bash
$ export SQLITE_DB="/path/to/xwrap.db"
$ python -m exchange_wrapper.service init_db $SQLITE_DB
Create Table for WalletSet
Create Table for Order
Create Table for WalletLine
Create Table for VirtualWallet
Create Table for Trade
```

Influx DB (Optional)

InfluxDB is used for logging various time-series datapoints such as historical tick data and logs of wallet balances. To enable InfluxDB support, a DSN must be supplied via the INFLUXDB_DSN environment variable.

```bash
export INFLUXDB_DSN="influxdb://influx.local:8086/cryptick"
```

If InfluxDB is not configured the following features will be unavailable:

- get_ohlc API calls on exchange
- get_wallet_history API calls

API Endpoint Listen IPs and Ports

xWrap currently exposes a RPC interface and a REST/SocketIO interface. The listen IP and port can be configured via environment variables. By default the services will listen for global traffic (binding to 0.0.0.0) and the RPC service will be on port 4200 and the REST/SocketIO service on port 4242.
1.1.2 Running Tests

Use nose2 to run tests.

```
. env/bin/activate
nose2 --with-coverage
```

1.1.3 Building a Docker Image

```
$ docker build -t xwrap .
```

1.2 Strategy API

The strategy connects to xWrap via a Websocket Stream that supports the JSON-RPC protocol.
Messages on the stream interface should be assumed to be asynchronous. I.e. responses may be out of order. See the spec.

JSON-RPC supports two main types of message:
- A remote procedure call where the message will elicit a response
- A notification where the notification will not be responded to

Methods accessible via this RPC are described here

1.2.1 Request Format

Each message expecting a response should have a unique id generated by the client. This id will be used to match requests to responses and must be unique to the connection.

```
{
    "jsonrpc": "2.0",
    "method": "make_order",
    "params": {...},
    "id": 240
}
```

The params object contains the call specific parameters. If id is omitted, the message is considered to be a notification and will not elicit a response.
1.2.2 Response Format

On success:

```json
{
    "jsonrpc": "2.0",
    "result": "accepted",
    "id": 240
}
```

On failure:

```json
{
    "jsonrpc": "2.0",
    "error": {
        "code": -32000,
        "message": "Insufficient balance in wallet",
    },
    "id": 240
}
```

1.3 Frontend API

The frontend can utilise a read-only API to observe the status of xWrap.

1.3.1 REST Endpoints

The following REST endpoints are served by a `sanic` web server.

Provides a HTTP API to xWrap

`exchange_wrapper.http_server.create_new_order(request)`

**POST /xwrap/v1/orders/new**

Maps to `make_order()`

`exchange_wrapper.http_server.deactivate_wallet_set(request)`

**POST /xwrap/v1/wallets/new**

Maps to `deactivate_wallet_set()`

`exchange_wrapper.http_server.get_allocated_wallet_sets(request)`

**GET /xwrap/v1/wallets/allocated**

Maps to `get_allocated_wallet_set()`

`exchange_wrapper.http_server.get_book(request, exchange, pair)`

**GET /xwrap/v1/(str: exchange)/book/**

**str: pair** Maps to `get_book()`

`exchange_wrapper.http_server.get_exchange_pricing(request, exchange)`
GET /xwrap/v1/(str: exchange)/pricing
The best bid offer from the exchange (exchange).

Example request:

```
GET /xwrap/v1/gdax/pricing HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "BCH_BTC":{
    "bid":0.1145,
    "ask":0.1204
  },
  "BCH_USD":{
    "bid":1259.33,
    "ask":1273.73
  }
}
```

Status Codes
- 200 OK – no error
- 404 Not Found – there’s no such exchange

exchange_wrapper.http_server.get_ohlc(request, exchange, pair)

GET /xwrap/v1/ohlc/(str: exchange)/
str: pair OHLC data for the specified exchange and pair.

Returned datapoints have the following format: [time, low, high, open, close, volume, received_at]

Example request:

```
GET /xwrap/v1/ohlc/gdax/BTC_USD HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "exchange": "gdax",
  "pair": "BTC_USD",
  "generated_at": 1520551670
  "points": [
    [1520525700, 9879.59, 9879.59, 9857.2, 9857.21, 19.04536155, 1520551661]
  ]
}
```
Query Parameters

- **start** – an integer unix timestamp in seconds
- **end** – an integer unix timestamp in seconds
- **received_before** – an integer unix timestamp to filter by `received_at`
- **freq** – frequency of datapoints - valid are *highest* and *daily*

Status Codes

- **200 OK** – no error
- **400 Bad Request** – `start` or `end` set to a non-integer value
- **503 Service Unavailable** – underlying backend database not reachable/configured

```python
exchange_wrapper.http_server.get_total_wallet_sets(request)
```

**GET /xwrap/v1/wallets/total**
Maps to `get_total_wallet_set()`

```python
exchange_wrapper.http_server.get_unallocated_exchange(request)
```

**GET /xwrap/v1/wallets/unallocated_exchange**
Maps to `~exchange_wrapper.service.ExchangeWrapper.get_unallocated_per_exchange`

```python
exchange_wrapper.http_server.get_unallocated_wallet_sets(request)
```

**GET /xwrap/v1/wallets/unallocated**
Maps to `get_unallocated_wallet_set()`

```python
exchange_wrapper.http_server.get_wallet_set(request, set_id)
```

**GET /xwrap/v1/wallets/(int: set_id)**
Maps to `get_wallet_set()`

```python
exchange_wrapper.http_server.get_wallet_set_history(request, set_id)
```

**GET /xwrap/v1/wallets/history/(int: set_id)**
Get history of wallet set state.

**Example request:**

```
GET /xwrap/v1/wallets/history/4 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

**Example response:**

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "timestamp":1523146996.0901162624,
  "run_id":"test02",
  "id":4,
}
```

(continues on next page)
"current":{
    "timestamp":1523146996.0630502701,
    "description":"Logical Wallet for Run: test02",
    "id":4,
    "run_id":"test02",
    "wallets":[
        {
           "currency":"BTC",
           "exchange":"gdax",
           "id":30,
           "balance":"883",
           "available":"883",
           "hold":"0",
           "present_value":{
              "available_usd":"6167023",
              "balance_usd":"6167023",
              "hold_usd":"0"
            }
        },
        ...
    ],
    "type":"logical"
},
"data":[
    
    1523146633,
    [
    
        {
           "currency":"BTC",
           "id":"30",
           "balance":882.93316402,
           "available":882.93316402,
           "hold":0,
           "present_value":{
              "balance_usd":6125560.7293481147,
              "available_usd":6125560.7293481147,
              "hold_usd":0
            }
        },
        ...
    ]
]}

Status Codes

- 200 OK – no error
- 404 Not Found – Cannot find wallet set with id
- 503 Service Unavailable – underlying backend database not reachable/configured

exchange_wrapper.http_server.get_wallet_sets(request)

GET /xwrap/v1/wallets/sets
Maps to get_all_wallet_sets()
exchange_wrapper.http_server.list_exchanges(request)

1.3. Frontend API
GET /xwrap/v1/exchanges
Maps to \texttt{get\_exchanges()} 
exchange\_wrapper.http\_server.list\_open\_orders(request)

GET /xwrap/v1/orders/open
Maps to \texttt{list\_open\_orders()}
exchange\_wrapper.http\_server.list\_recent\_orders(request)

GET /xwrap/v1/orders/recent
Maps to \texttt{list\_recent\_orders()}
exchange\_wrapper.http\_server.main(request)

GET /xwrap/v1/status
Maps to \texttt{get\_status()}
exchange\_wrapper.http\_server.make\_logical\_transaction(request)

POST /xwrap/v1/wallets/virtual/transaction
Maps to \texttt{add\_virtual\_wallet\_tx()}
exchange\_wrapper.http\_server.make\_transaction(request)

POST /xwrap/v1/wallets/new
Maps to \texttt{create\_wallet\_set()}
exchange\_wrapper.http\_server.metrics(request)

Metrics endpoint for prometheus.
exchange\_wrapper.http\_server.notify(sender, *, notification, payload)
Callback for notifications to be sent to socketio clients.
exchange\_wrapper.http\_server.uri\_map(request)

GET /uri\_map
Returns a list of URIs and the docstring of the handler.

### 1.3.2 SocketIO Channels

Notifications described here are broadcast on SocketIO channels. The channel name will be the notification name, eg: \texttt{order} or \texttt{trade}.

### 1.4 Exposed Functions

\texttt{xWrap} exposes a number of functions to client code via a JSON-RPC or REST interface. Both API mechanisms expose the same underlying python functions as described here. These function calls only accept keyword arguments and will use Keyword-Only Arguments as specified in PEP 3102.

c\texttt{onnection\_id} is a special keyword argument inserted by \texttt{xWrap} to all method calls. It is used in situations where the function call can result in later asynchronous notifications that need to be sent to originating connection.
1.4.1 Common Conventions

- A pair is represented in the format BASE_QUOTE where BASE and QUOTE will be valid currency codes.
- An exchange value must correspond to the name of a loaded exchange.
- If an API call that specifies an exchange parameter is made before that exchange is in the ready state, the API call will raise an error.

1.4.2 Utility Functions

**Get Status (get_status)**

`ExchangeWrapper.get_status(**kwargs)`

Get the status of xWrap and configuration information.

**Returns:** A dict with status information.

```
{
    "status": "online",
    "version": "0.4.1",
    "config": {
        "currencies": ["BTC", "LTC", "USD"],
        "exchanges": ["gdax"]
    }
}
```

**Echo (echo)**

Will echo any parameters that this method is called with to the sender as a response. Use to test RPC interfaces.

Eg: calling `echo(foo="bar", baz=9)` will return a dict `{"foo": "bar", "baz": 9}`.

`ExchangeWrapper.echo(*, connection_id, **kwargs)`

Echos sent keyword arguments back to the sender.

**Echo Notify (echo_notify)**

Will echo any parameters that this method is called with to the sender as a notification. Use to test notifications.

Eg: calling `echo_notify(foo="bar", baz=9)` will return a dict `{"status": "ok", "receivers": 2}` if there were 2 receivers listening for notifications. Each receiver would receive a `echo_ret` notification with the payload `{"foo": "bar", "baz": 9}`.

`ExchangeWrapper.echo_notify(*, connection_id, **kwargs)`

Creates an `echo_notify` notification with the sent keywords.

**Returns:** A dictionary with “status”: “ok” and the number of notification receivers that would receive this notification.

1.4.3 Exchange Functions
Get Exchanges (get_exchanges)

ExchangeWrapper.get_exchanges(*, mode=None, **kwargs)

Return a list of dictionaries describing the registered exchanges on this server.

Args:

  param mode  Filters the exchanges by their mode. Valid modes are paper or live.

Returns: A list of dict’s corresponding to exchanges supported by the system.

- exchange: The exchange id.
- class: The python class handling the exchange.
- status: The status of the exchange.
- pairs: A list of pairs supported by the exchange.
- currencies: List of currencies supported by the exchange.
- mode: The mode the exchange is loaded in.

```
[
    {
        "exchange": "gdax",
        "class": "xwrap.exchange.gdax.GDAXExchange",
        "status": "starting",
        "pairs": ["BTC_USD", "LTC_USD"],
        "currencies": ["BTC", "LTC", "USD"],
        "mode": "paper"
    },
    ...
]
```

A few notes:

- See get_status() in the Exchange class for valid state values.
- If the exchange status is not ready then some data might be out of date, and api calls can fail.
- type can be paper or live. A paper exchange will simulate trades against real-time orderbooks.

1.4.4 Wallet Functions

Get Wallet Set (get_wallet_set)

ExchangeWrapper.get_wallet_set(set_id=None, aggregate_only=False, **kwargs)

Return the wallet set with the specific ID.

Get All Wallet Sets (get_all_wallet_sets)

ExchangeWrapper.get_all_wallet_sets(aggregate_only=False, **kwargs)

Get all the active wallet sets on the exchange.

Returns: A dict of WalletSet objects keyed by id.

Example:
The total, unallocated and allocated wallet sets are described in the corresponding sibling function documentation.

A WalletSet object is represented as:

```
{
  "timestamp": 153998293,
  "description": "This is a set of logical wallets",
  "id": 1,
  "wallets": [
    [Wallet Object],
    [Wallet Object],
    [Wallet Object]
  ],
  "type": "logical"
}
```

A Wallet object is represented as:

```
{
  "currency": "IOTA",
  "exchange": "bitfinex",
  "id": 24,
  "balance": "1000000.00",
  "available": "1000000.00",
  "hold": "0.00",
  "present_value": {
    "balance_usd": "10.00",
    "available_usd": "10.00",
    "hold_usd": "0.00"
  }
}
```

See `_wallet_to_dict()` for field descriptions.

For more information on field values see documentation for `WalletSet` and `VirtualWallet`.

**Get Allocated Wallets (get_allocated_wallet_set)**

`ExchangeWrapper.get_allocated_wallet_set(**kwargs)`

Get a aggregate of all active wallet sets aggregated by currency.

**Returns:** A WalletSet object as described in `get_all_wallet_sets()`.

**Get Unallocated Wallets (get_unallocated_wallet_set)**

`ExchangeWrapper.get_unallocated_wallet_set(**kwargs)`

Get a wallet set of unallocated exchange funds aggregated by currency.
Get All Wallets (get_total_wallet_set)

ExchangeWrapper.get_total_wallet_set(**kwargs)
Get the total funds (allocated+unallocated). This will equal the aggregate of exchange wallet balances.

Create Wallet Set (create_wallet_set)

ExchangeWrapper.create_wallet_set(*, run_id, wallets, **kwargs)
Create a new wallet set.

Args:

- **run_id** The strategy_run_id to associate this set with.
- **wallets** A list of Wallet objects.

Each Wallet object in the wallets list must contain the following fields: exchange, currency and allocation.

Example:

```json
{
    "run_id": "my_great_run_01",
    "wallets": [
        {
            "exchange": "gdax",
            "currency": "USD",
            "allocation": "500.0"
        },
        {
            "exchange": "gdax",
            "currency": "LTC",
            "allocation": "5.0"
        },
        {
            "exchange": "gdax",
            "currency": "ETH",
            "allocation": "10.0"
        }
    ]
}
```

Returns: A dict with the id of the new wallet set.

```json
{ "id": 5 }
```

If the wallet cannot be allocated, then an error response will be returned:

```json
{"error": "error message"}
```

Possible errors are:

- insufficient funds: Insufficient unallocated funds.
- bad wallet config: One or more bad wallet/exchange pairs

1.4.5 Order Functions
Make Order (**make_order**)

**ExchangeWrapper.make_order**(*, connection_id, run_id, exchange, client_oid, type, pair, size, side, wallet_set_id=None, limit=0, cancel_after=0, **kwargs)

Make an order on the specified exchange.

**Args:**

- **param exchange** Identifier for the exchange.
- **param connection_id** Identifier for connection sending this request. Use to deliver Trade and Order notifications.
- **param client_oid** Unique identifier for this order from the client.
- **param run_id** identifier for a strategy instance.
- **param wallet_set_id** Wallet Set to run this transaction on.
- **param side** *buy* or *sell*
- **param type** A valid order type string.
- **param pair** A pair name supported by this exchange.
- **param size** Order size.
- **param limit** Order limit value for limit orders.
- **param cancel_after** Seconds after which this order should cancel.
- **param **kwargs** Other keyword arguments which are ignored.

**Returns:** A tuple `(status, message)` is returned. If `status` returns as invalid or rejected then `message` will return a descriptive reason.

Several different order types are supported:

- **limit:** Standard limit order that is good till cancelled
- **limit-immediate-cancel:** Cancels the remaining size on an order instead of putting on the book
- **limit-fill-kill:** Rejects the order if the entire size cannot be matched
- **market:** Executed at the best price available

Of the above, only the plain **limit** order will be left on the order book.

List Open Orders (**list_open_orders**)

**ExchangeWrapper.list_open_orders**(**kwargs)

Get a list of open orders.

**Returns:** A list of Order objects, eg:

```
[   
    {   
        "id":30, 
        "created_at":"2018-04-07T02:29:58.933222", 
        "wallet_set_id":2, 
        "exchange":"gdax", 
        "client_order_id":"8eed73b3-309c-4153-b6e5-0735b484aad3", 
        "run_id":"asda", 
        "type":"limit",   
    }   
] (continues on next page)```
The field descriptions are described in *Order*.

### List Recent Orders(list_recent_orders)

`ExchangeWrapper.list_recent_orders(*, include_open=False)`

Get a list of open orders.

**Returns:** A list of Order objects. See `list_open_orders` for example.

### 1.4.6 Pricing Functions

#### Convert to USD (convert_to_usd)

`ExchangeWrapper.convert_to_usd(*, currency, value, side='bid')`

Convert the specified currency value into USD.

**Args:**

- `param currency` A currency code to convert into USD.
- `param value` The value to convert.

**Returns:** The converted currency value. If the currency cannot be converted `None` will be returned.

#### Get Book (get_book)

`ExchangeWrapper.get_book(*, exchange, pair, **kwargs)`

Get the recent order book snap.

**Args:**

- `param exchange` A valid exchange.
- `param pair` A exchange on the pair.

**Returns:** A `dict` with `asks` and `bids` keys that reference lists of `[level, size]`.

Example:
Get Pricing (get_pricing)

ExchangeWrapper.get_pricing(*, exchange, **kwargs)
Returns pricing from an exchange.

See exchange documentation for how this is computed.

Args:

   param exchange  The exchange name.

Returns: A dict with pair ids as keys and {"bids": value, "asks": value} as values:

   {"ETH_USD": {"bids": 105, "asks": 102}, ...}

1.4.7 Virtual Wallet Functions

Add Virtual Wallet Transaction (add_virtual_wallet_tx)

ExchangeWrapper.add_virtual_wallet_tx(*, exchange, base, movement, wallet_id=None, **kwargs)
Add a transaction to a paper exchange virtual wallet.

Args:

   param exchange  Identifier for the exchange.
   param base      Currency for the selected wallet.
   param movement  The credit (positive) or debit (negative) amount.
   param wallet_id Virtual wallet Id. Optional.
   param **kwargs  Other keyword arguments which are ignored.

Returns: Returns the id for the wallet trasaction. This id can be used with the modify_vwallet_line() call.

1.5 Notifications

xWrap generates notifications to certain events such as executed trades or completed orders that client code may wish to subscribe to.

In the sections titles below, the notification name is shown in monospace.
1.5.1 Order Status

Order (order)

Order notifications are issued on the creation or update of an order placed by a `make_order` command. They contain the following fields which are generated by the `to_dict()`.

Order. `to_dict()`

Encode an order as a `dict`.

Keys:

- **id**: xWrap Order ID
- **created_at**: Timestamp for order creation (ISO8601 format)
- **wallet_set_id**: Wallet Set ID (Optional)
- **exchange**: Exchange name
- **client_order_id**: Client Order ID
- **run_id**: Run ID associated with this order
- **type**: Order type (Options: `limit`, `immediate-or-cancel limit-fill-kill`, `market`)
- **pair**: A pair in BASE_QUOTE format (BTC_USD)
- **size**: Order size
- **side**: Order size (buy or sell)
- **limit**: Limit price (if market order, then 0)
- **cancel_after**: Seconds after which orders automatically cancel
- **status**: Order status
- **done_reason**: Descriptive reason for status
- **done_at**: Timestamp at which order closed (ISO8601 format)
- **exchange_at**: Timestamp at which order arrived at exchange
- **last_updated_at**: Timestamp for last order update
- **filled_size**: Size of order which has been filled
- **avg_price**: Average price for fill

Trade (trade)

Trade notifications are issued at every fill from an exchange.

Trade. `to_dict()`

Encode trade as a `dict`.

Keys:

- **client_order_id**: Client Order ID
- **run_id**: Run ID for the order associated with this trade
- **type**: Type of Order associated with this Trade
- **side**: Side for associated Order
• pair: Pair for associated Order
• exec_amount: Size of this trade
• exec_price: Amount this trade executed at
• fee: Exchange fee for this order
• maker: True if this order was a liquidity maker
• ext_id: Exchange ID for this trade (if any)

1.5.2 General

Log Messages (logmessages)

Log Messages are issued from the python logging library. Any messages from the xwrap logger greater or equal to WARNING or any messages from the user logger are emitted by default.

Example:

```
{
   "created":1523120885.1184688,
   "filename":"virtual.py",
   "funcName":"trade_executor",
   "levelname":"INFO",
   "levelno":20,
   "lineno":188,
   "module":"virtual",
   "msecs":118.4687614440918,
   "name":"user.orders.test01",
   "pathname":"./exchange_wrapper/exchange/virtual.py",
   "process":3067,
   "processName":"MainProcess",
   "relativeCreated":15570915.82942009,
   "thread":139932965160256,
   "threadName":"MainThread",
   "message":"Order 1faaae49-64bf-4dc5-847b-b4a92fd791fc is done"
}
```

Exchange Status (exchstatus)

Exchange notifications are issued when an Exchange changes state.

Example:

```
{
   "exchange": "gdax",
   "from": "starting",
   "to": "ready"
}
```
1.6 exchange_wrapper package

1.6.1 Subpackages

exchange_wrapper.util package

Submodules

exchange_wrapper.util.token_bucket module

Token Bucket —
Implement a token bucket algorithm to rate limit connections. This implementation will be local to a thread and will provide a method that can block until a token is available.

Does not support bursts.

class exchange_wrapper.util.token_bucket.TokenBucket (tokens_per_second)
    Bases: object
        request_token()
            Decrements token and returns if tokens are available. Otherwise blocks until a token can be awarded.

Module contents

exchange_wrapper.util.dec2str (decimal_value, dp=8)
exchange_wrapper.util.dt_to_msts (dt)
exchange_wrapper.util.dt_to_str (dt)
exchange_wrapper.util.dt_to_ts (dt)
exchange_wrapper.util.fnmatch_dicts (param, ref)

1.6.2 Submodules

1.6.3 exchange_wrapper.client module

class exchange_wrapper.client.Client (name='generic')
    Bases: object
        add_virtual_wallet_tx (exchange, base, amount)
        cancel_order (exchange, client_order_id)
        get_wallets ()
        limit_buy (exchange, pair, size, limit, oid=None, type='limit')
        limit_sell (exchange, pair, size, limit, oid=None, type='limit')
        parse_recv (json_str)
exchange_wrapper.client.collate_responses (resp, ws)
exchange_wrapper.client.execute_via_rpc (ws_url, client, rpc_msg, timeout=0.5, filter_notifications=None)
exchange_wrapper.client.websocket_query(url, message, count=1)
exchange_wrapper.client.websocket_query_timeout(url, message, timeout)

1.6.4 exchange_wrapper.config module

class exchange_wrapper.config.Config(envvar_config_key='env')
  Bases: object
    get(key, default=None, cast=None)
    getenv(key, default=None, cast=None)
    load(config_file_path)
exchange_wrapper.config.cast_bool(value)
exchange_wrapper.config.recursive_update(old_dict, new_dict)

1.6.5 exchange_wrapper.connection module

Models a HTTP connection to an exchange. Accepts requests in a queue and pushes data to the exchange based on a rate-limit.

class exchange_wrapper.connection.Connection(name, config, server_address=", max_queue=0, rate_limit=None)
  Bases: object
    config is a dict containing secrets for HMAC auth
    init()
      Init the session and test API?
    run()
      Start the worker thread
    send_request(method, endpoint, priority=99, **kwargs)
      Enqueue a HTTP request for dispatch to the server.
        Returns: Returns a (concurrent.futures.Future) object for the request if sufficient space was available in the queue, otherwise returns None. The Future’s value will be set to requests. Response object on completion of the request.
    send_request_async(*args, loop=None, **kwargs)
    stop()
exception exchange_wrapper.connection.ConnectionError(e)
  Bases: Exception
exchange_wrapper.connection.connection_worker(connection)
  Thread to process requests on a connection
1.6.6 exchange_wrapper.db module

class exchange_wrapper.db.BaseModel(*args, **kwargs)
    Bases: peewee.Model

    DoesNotExist
        alias of BaseModelDoesNotExist

    id = <peewee.AutoField object>

1.6.7 exchange_wrapper.http_server module

Provides a HTTP API to xWrap

class exchange_wrapper.http_server.HttpServer(xwrap, host, port)
    Bases: object

    notify_connect(sid)

    start()

exchange_wrapper.http_server.create_new_order(request)

    POST /xwrap/v1/orders/new
    Maps to make_order()

exchange_wrapper.http_server.deactivate_wallet_set(request)

    POST /xwrap/v1/wallets/new
    Maps to deactivate_wallet_set()

exchange_wrapper.http_server.get_allocated_wallet_sets(request)

    GET /xwrap/v1/wallets/allocated
    Maps to get_allocated_wallet_set()

exchange_wrapper.http_server.get_book(request, exchange, pair)

    GET /xwrap/v1/str: exchange)/book/
        str: pair Maps to get_book()

exchange_wrapper.http_server.get_exchange_pricing(request, exchange)

    GET /xwrap/v1/str: exchange)/pricing
        The best bid offer from the exchange (exchange).

    Example request:

        GET /xwrap/v1/gdax/pricing HTTP/1.1
        Host: example.com
        Accept: application/json, text/javascript

    Example response:
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "BCH_BTC": {
    "bid": 0.1145,
    "ask": 0.1204
  },
  "BCH_USD": {
    "bid": 1259.33,
    "ask": 1273.73
  }
}

Status Codes
• 200 OK – no error
• 404 Not Found – there’s no such exchange

exchange_wrapper.http_server.get_exchange_unallocated(request)
exchange_wrapper.http_server.get_ohlc(request, exchange, pair)

GET /xwrap/v1/ohlc/(str: exchange) /
str: pair OHLC data for the specified exchange and pair.
Returned datapoints have the following format: [time, low, high, open, close, volume, received_at]
Example request:

GET /xwrap/v1/ohlc/gdax/BTC_USD HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

Example response:

HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "exchange": "gdax",
  "pair": "BTC_USD",
  "generated_at": 1520551670
  "points": [
    [1520525700, 9879.59, 9879.59, 9857.2, 9857.21, 19.04536155, 1520551661]
  ]
}

Query Parameters
• start – an integer unix timestamp in seconds
• end – an integer unix timestamp in seconds
• received_before – an integer unix timestamp to filter by received_at
• freq – frequency of datapoints - valid are highest and daily

Status Codes
• 200 OK – no error
• 400 Bad Request – start or end set to a non-integer value
• 503 Service Unavailable – underlying backend database not reachable/configured

exchange_wrapper.http_server.get_total_wallet_sets(request)

GET /xwrap/v1/wallets/total
Maps to get_total_wallet_set()

exchange_wrapper.http_server.get_unallocated_exchange(request)

GET /xwrap/v1/wallets/unallocated_exchange
Maps to :py:meth:`exchange_wrapper.service.ExchangeWrapper.get_unallocated_per_exchange`

exchange_wrapper.http_server.get_unallocated_wallet_sets(request)

GET /xwrap/v1/wallets/unallocated
Maps to get_unallocated_wallet_set()

exchange_wrapper.http_server.get_wallet_set(request, set_id)

GET /xwrap/v1/wallets/ (int: set_id)
Maps to get_wallet_set()

exchange_wrapper.http_server.get_wallet_set_history(request, set_id)

GET /xwrap/v1/wallets/history/ (int: set_id)
Get history of wallet set state.

Example request:

```
GET /xwrap/v1/wallets/history/4 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
    "timestamp":1523146996.0901162624,
    "run_id":"test02",
    "id":4,
    "current":{
    "timestamp":1523146996.0630502701,
    "description":"Logical Wallet for Run: test02",
    "id":4,
    "run_id":"test02",
    "wallets":[]
    }
}
```

(continues on next page)
{  
  "currency":"BTC",
  "exchange":"gdax",
  "id":30,
  "balance":"883",
  "available":"883",
  "hold":"0",
  "present_value":{
    "available_usd":"6167023",
    "balance_usd":"6167023",
    "hold_usd":"0"
  }
},
...
,"type":"logical"
},
"data":[
    
    1523146633,
    
    {
      "currency":"BTC",
      "id":"30",
      "balance":882.93316402,
      "available":882.93316402,
      "hold":0,
      "present_value":{
        "balance_usd":6125560.7293481147,
        "available_usd":6125560.7293481147,
        "hold_usd":0
      }
    }
  
  ...  
]
}

**Status Codes**

- **200 OK** – no error
- **404 Not Found** – Cannot find wallet set with *id*
- **503 Service Unavailable** – underlying backend database not reachable/configured

**exchange_wrapper.http_server.get_wallet_sets** *(request)*

**GET /xwrap/v1/wallets/sets**
Maps to **get_all_wallet_sets()**

**exchange_wrapper.http_server.list_exchanges** *(request)*

**GET /xwrap/v1/exchanges**
Maps to **get_exchanges()**

**exchange_wrapper.http_server.list_open_orders** *(request)*
GET /xwrap/v1/orders/open
Maps to list_open_orders()
exchange_wrapper.http_server.list_recent_orders(request)

GET /xwrap/v1/orders/recent
Maps to list_recent_orders()
exchange_wrapper.http_server.log_404(request, exception)
exchange_wrapper.http_server.main(request)

GET /xwrap/v1/status
Maps to get_status()
exchange_wrapper.http_server.make_logical_transaction(request)

POST /xwrap/v1/wallets/virtual/transaction
Maps to add_virtual_wallet_tx()
exchange_wrapper.http_server.make_transaction(request)

POST /xwrap/v1/wallets/new
Maps to create_wallet_set()
exchange_wrapper.http_server.metrics(request)
Metrics endpoint for prometheus.
exchange_wrapper.http_server.notify(sender, *, notification, payload)
Callback for notifications to be sent to socketio clients.
exchange_wrapper.http_server.on_connect(sid, environ)
exchange_wrapper.http_server.uri_map(request)

GET /uri_map
Returns a list of URIs and the docstring of the handler.

1.6.8 exchange_wrapper.logical_wallet module

class exchange_wrapper.logical_wallet.LogicalWalletManager(xwrap)
Bases: object

get_allocated()
get_exchange_total()
get_unallocated()
get_unallocated_per_exchange()
get_wallet_set(id)
get_wallet_sets(type)
1.6.9 exchange_wrapper.matching module

exchange_wrapper.matching.match_limit_order(book, side, size, limit)
Execute limit order in paper trading environment.

exchange_wrapper.matching.match_market_order(book, side, size)
Execute market order in paper trading environment.

1.6.10 exchange_wrapper.models module

class exchange_wrapper.models.Order(*args, **kwargs)
Bases: exchange_wrapper.db.BaseModel
Represent a cryptocurrency order

    DoesNotExist
    alias of OrderDoesNotExist

    avg_price = <peewee.DecimalField object>
    cancel_after = <peewee.IntegerField object>
    client_order_id = <peewee.CharField object>
    created_at = <peewee.DateTimeField object>
    done_at = <peewee.DateTimeField object>
    done_reason = <peewee.CharField object>
    exchange = <peewee.CharField object>
    exchange_at = <peewee.DateTimeField object>
    filled_size = <peewee.DecimalField object>
    get_quote_amounts()
    get_wallets(wallet_set=None)
    Returns the quote and base wallet from the wallet set.
    Args:
        param wallet_set A wallet set. If None given (default), will
        use Order.wallet_set.
    Returns: A tuple, (base_wallet, quote_wallet) which are VirtualWallet objects for the quote and
    base currency respectively.

    id = <peewee.AutoField object>
    last_updated_at = <peewee.DateTimeField object>
    limit = <peewee.DecimalField object>
    logger
    orders
    pair = <peewee.CharField object>
    remaining_size
    settled = <peewee.BooleanField object>
side = <peewee.CharField object>
size = <peewee.DecimalField object>
status = <peewee.CharField object>
strategy_run_id = <peewee.CharField object>
to_dict()
    Encode an order as a dict.
    Keys:
    • id: xWrap Order ID
    • created_at: Timestamp for order creation (ISO8601 format)
    • wallet_set_id: Wallet Set ID (Optional)
    • exchange: Exchange name
    • client_order_id: Client Order ID
    • run_id: Run ID associated with this order
    • type: Order type (Options: limit, immediate-or-cancel limit-fill-kill, market)
    • pair: A pair in BASE_QUOTE format (BTC_USD)
    • size: Order size
    • side: Order size (buy or sell)
    • limit: Limit price (if market order, then 0)
    • cancel_after: Seconds after which orders automatically cancel
    • status: Order status
    • done_reason: Descriptive reason for status
    • done_at: Timestamp at which order closed (ISO8601 format)
    • exchange_at: Timestamp at which order arrived at exchange
    • last_updated_at: Timestamp for last order update
    • filled_size: Size of order which has been filled
    • avg_price: Average price for fill

type = <peewee.CharField object>
unfilled_size
update_wallets(wallet_set=None)
    Update the wallet_set associated with an order.
    Args:
        param wallet_set  The wallet set to update, if this is set to None
        (default) then the wallet_set from the Order object will be used.
class exchange_wrapper.models.Trade(*args, **kwargs)
    Bases: exchange_wrapper.db.BaseModel

    Represents a single trade that makes up an order

    DoesNotExist
        alias of TradeDoesNotExist

exec_amount = <peewee.DecimalField object>
eexec_price = <peewee.DecimalField object>
ext_id = <peewee.CharField object>
fee = <peewee.DecimalField object>
id = <peewee.AutoField object>
maker = <peewee.BooleanField object>
order = <ForeignKeyField: "trade"."order">
order_id = <ForeignKeyField: "trade"."order">
order_type = <peewee.CharField object>
to_dict()
    Encode trade as a dict.

    Keys:
    • client_order_id: Client Order ID
    • run_id: Run ID for the order associated with this trade
    • type: Type of Order associated with this Trade
    • side: Side for associated Order
    • pair: Pair for associated Order
    • exec_amount: Size of this trade
    • exec_price: Amount this trade executed at
    • fee: Exchange fee for this order
    • maker: True if this order was a liquidity maker
    • ext_id: Exchange ID for this trade (if any)

class exchange_wrapper.models.VirtualWallet(*args, **kwargs)
    Bases: exchange_wrapper.db.BaseModel

    Funds put on hold will not be available to the user. So if a debit is held then that affects the ‘available’ balance
    but not the ‘ledger’ balance itself. For a credit on hold, it does not affect the ‘available’ balance but does affect the
    ‘ledger’ balance.

    DoesNotExist
        alias of VirtualWalletDoesNotExist

address = <peewee.CharField object>
allocation = <peewee.DecimalField object>
available

balance

1.6. exchange_wrapper package
base = <peewee.CharField object>  
exchange = <peewee.CharField object>  
get_held_for(order)  
hold  
id = <peewee.AutoField object>  
make_transaction(movement, order=None, held=False)  

wallet_lines  

wallet_set = <ForeignKeyField: "virtualwallet"."wallet_set">  
wallet_set_id = <ForeignKeyField: "virtualwallet"."wallet_set">  

class exchange_wrapper.models.WalletLine(*args, **kwargs)  
Bases: exchange_wrapper.db.BaseModel  

Represent a transaction in a wallet.

Variables

- **date** – Date column for when this transaction was created
- **wallet_id** – Id for this wallet
- **movement** – Transaction amount
- **on_hold** – If this transaction is being held (waiting for settlement)
- **order** – Order that this transaction relates to (if any)

DoesNotExist  
alias of WalletLineDoesNotExist

date = <peewee.DateTimeField object>  
id = <peewee.AutoField object>  
movement = <peewee.DecimalField object>  
on_hold = <peewee.BooleanField object>  
order = <ForeignKeyField: "walletline"."order">  
order_id = <ForeignKeyField: "walletline"."order">  

wallet = <ForeignKeyField: "walletline"."wallet">  
wallet_id = <ForeignKeyField: "walletline"."wallet">  

class exchange_wrapper.models.WalletSet(*args, **kwargs)  
Bases: exchange_wrapper.db.BaseModel  

A set of logical wallet used by a strategy.

DoesNotExist  
alias of WalletSetDoesNotExist

active = <peewee.BooleanField object>  
aggregate  
Returns a dict of wallets aggregated by currency.

created = <peewee.DateTimeField object>  
description = <peewee.CharField object>
get_wallet (exchange, currency)
    Return the wallet for an exchange and currency from this set.

    Args:
        param exchange  An exchange name.
        param currency  A currency code.

    Returns: A Wallet if a matching wallet exists, otherwise None.

id = <peewee.AutoField object>
orders
run_id = <peewee.CharField object>
type = <peewee.CharField object>
wallets

1.6.11 exchange_wrapper.pair module

class exchange_wrapper.pair.Pair (x_id, base, quote, price_prec, min_order, max_order)
    Bases: object

    Models a cryptocurrency pair available on an exchange

    Creates a Pair object. A pair string is defined as \(\text{base}_{\text{order}}\).

    For a sell: base is the currency you have, quote what you want. For a buy: base is the currency you want, quote what you have.

    For uniformity, this class maintains all currencies codes and pair codes in uppercase.

    Args:  x_id: An identifier for this pair from the exchange base: base currency for this pair quote: quote currency for this pair price_prec: price precision for the quote in decimal places min_order: minimum size for any order max_order: maximum size for any order

    check_price (price)
    Check if the given price does not exceed the precision provided by the exchange. To be used to check that any limit prices sent to the exchange are valid.

    Args:  price: any numeric type or convertible string.

    Returns: boolean: True if the price is within the precision requirements.

    check_size (order_size)
    Check if a given order_size is within the min and max order sizes specified.

    Internally converts order_size to a fixed precision number. Should check that the order_size is a multiple of the max_order.

    Args:  order_size: any numeric type or convertible string.

    Returns: boolean: True if the order size is valid.

    name ()
    Returns a currency pair in the format \(\text{base}_{\text{quote}}\). All characters will be uppercase.
1.6.12 exchange_wrapper.rpc module

exception exchange_wrapper.rpc.ApiUsageError
    Bases: Exception
    data = None
    jsonrpc_error_code = -32001

exception exchange_wrapper.rpc.MethodNotFoundError
    Bases: Exception
    data = None
    jsonrpc_error_code = -32601

exception exchange_wrapper.rpc.ProcessingError
    Bases: Exception
    data = None
    jsonrpc_error_code = -32002

class exchange_wrapper.rpc.RpcInterface(app_klass, config)
    Bases: object

    Creates a server object that will run an asyncio loop and host the one or more service classes.

    audit_loop(scheduled_at)
        Audit the event loop and log the latency (if any).
        Args:
            param scheduled_at  The time at which this function was scheduled
                on the loop via callLater

    get_connection()

    handle_message(connection_id, message_str)
        Handle messages from a connection and hand them over an installed service. Pass back any response over
        the originating connection.
        Params:
            connection_id: Id which can be used to address the connection  this message arrived on.
            message_str: The received message in string form.

    on_connect_hook(callback)

    on_disconnect_hook(callback)

    run_forever()

    send_to(conn_id, method, args, kwargs, one_way=False)

    send_to_all(method, args, kwargs)

    shutdown()
        Shuts down the event loop.
1.6.13 exchange_wrapper.service module

class exchange_wrapper.service.ExchangeWrapper(wrapper)
    Bases: object

    Implements a service that gets data from cryptocurrency exchanges and provides RPC and PubSub interfaces
    via an abstracted API format to interact with the upstream exchanges.

    add_virtual_wallet_tx(*, exchange, base, movement, wallet_id=None, **kwargs)
        Add a transaction to a paper exchange virtual wallet.
        Args:
            param exchange Identifier for the exchange.
            param base Currency for the selected wallet.
            param movement The credit (positive) or debit (negative) amount.
            param wallet_id Virtual wallet Id. Optional.
            param **kwargs Other keyword arguments which are ignored.
        Returns: Returns the id for the wallet trasaction. This id can be used with the
            modify_vwallet_line() call.

    allocate_to_wallet_set(*, wallet_set_id, exchange, base, movement)

    cancel_order(*, exchange, client_order_id, **kwargs)
        Request a cancellation of the specified order.

    convert_to_usd(*, currency, value, side='bid')
        Convert the specified currency value into USD.
        Args:
            param currency A currency code to convert into USD.
            param value The value to convert.
        Returns: The converted currency value. If the currency cannot be converted None will be returned.

    create_wallet_set(*, run_id, wallets, **kwargs)
        Create a new wallet set.
        Args:
            param run_id The strategy_run_id to associate this set with.
            param wallets A list of Wallet objects.
        Each Wallet object in the wallets list must contain the following fields: exchange, currency and
        allocation.
        Example:

        ```python
        {
            "run_id": "my_great_run_01",
            "wallets": [
                {
                    "exchange": "gdax",
                    "currency": "USD",
                    "allocation": "500.0"
                },
                {
                    "exchange": "bfx",
                    "currency": "BTC",
                    "allocation": "100.0"
                }
            ]
        }
        ```

(continues on next page)
{ "exchange": "gdax", 
"currency": "LTC", 
"allocation": "5.0"
},
{
"exchange": "gdax", 
"currency": "ETH", 
"allocation": "10.0"
}
]

Returns: A dict with the id of the new wallet set.

{ "id": 5 }

If the wallet cannot be allocated, then an error response will be returned:

{"error": "error message"}

Possible errors are:

• insufficient funds: Insufficient unallocated funds.
• bad wallet config: One or more bad wallet/exchange pairs

**deactivate_wallet_set** (*, wallet_set_id)
Deactivate a wallet set.

**echo** (*, connection_id, **kwars)
Echos sent keyword arguments back to the sender.

**echo_notify** (*, connection_id, **kwars)
Creates an echo_ret notification with the sent keywords.

Returns: A dictionary with “status”: “ok” and the number of notification receivers that would receive this notification.

**get_all_wallet_sets** (aggregate_only=False, **kwars)
Get all the active wallet sets on the exchange.

Returns: A dict of WalletSet objects keyed by id.

Example:

```
{
  "1": [WalletSet Object],
  "2": [WalletSet Object],
  "3": [WalletSet Object],
  "total": [WalletSet Object],
  "unallocated": [WalletSet Object],
  "allocated": [WalletSet Object],
}
```

The total, unallocated and allocated wallet sets are described in the corresponding sibling function documentation.

A WalletSet object is represented as:
A Wallet object is represented as:

```
{
    "currency": "IOTA",
    "exchange": "bitfinex",
    "id": 24,
    "balance": "1000000.00",
    "available": "1000000.00",
    "hold": "0.00",
    "present_value": {
        "balance_usd": "10.00",
        "available_usd": "10.00",
        "hold_usd": "0.00"
    }
}
```

See `_wallet_to_dict()` for field descriptions.

`get_allocated_wallet_set(**kwargs)`

Get a aggregate of all active wallet sets aggregated by currency.

**Returns:** A WalletSet object as described in `get_all_wallet_sets()`.

`get_book(*, exchange, pair, **kwargs)`

Get the recent order book snap.

**Args:**

- **param exchange** A valid `exchange`.
- **param pair** A exchange on the `pair`.

**Returns:** A `dict` with `asks` and `bids` keys that reference lists of `[level, size]`.

Example:

```
{
    "bids": [
        [290, 0.01],
        [291, 0.01],
        ...
    ],
    "asks": [
        [300, 0.01],
        [301, 0.01],
        ...
    ]
}
```
get_exchanges(*, mode=None, **kwargs)
Return a list of dictionaries describing the registered exchanges on this server.

Args:

param mode  Filters the exchanges by their mode. Valid modes are paper or live.

Returns: A list of dict’s corresponding to exchanges supported by the system.

• exchange: The exchange id.
• class: The python class handling the exchange.
• status: The status of the exchange.
• pairs: A list of pairs supported by the exchange.
• currencies: List of currencies supported by the exchange.
• mode: The mode the exchange is loaded in.

```
[
    {
        "exchange": "gdax",
        "class": "xwrap.exchange.gdax.GDAXExchange",
        "status": "starting",
        "pairs": ["BTC_USD", "LTC_USD"],
        "currencies": ["BTC", "LTC", "USD"],
        "mode": "paper"
    },
    ...
]
```

get_pricing(*, exchange, **kwargs)
Returns pricing from an exchange.

See exchange documentation for how this is computed.

Args:

param exchange  The exchange name.

Returns: A dict with pair ids as keys and {"bids": value, "asks": value} as values:

```
{"ETH_USD": {"bids": 105, "asks": 102}, ...
```

get_status(**kwargs)
Get the status of xWrap and configuration information.

Returns: A dict with status information.

```
{
    "status": "online",
    "version": "0.4.1",
    "config": {
        "currencies": ["BTC", "LTC", "USD"],
        "exchanges": ["gdax"]
    }
}
```

get_total_wallet_set(**kwargs)
Get the total funds (allocated+unallocated). This will equal the aggregate of exchange wallet balances.
get_unallocated_per_exchange(**kwargs)
Get the unallocated funds deaggregated by exchange.

**Returns:** The return of this function is a dict of dict objects:

Example:

```{json}
{
    "gdax": {
        "BTC": WalletObject,
        "LTC": WalletObject,
        "USD": WalletObject
    }
}
```

get_unallocated_wallet_set(**kwargs)
Get a wallet set of unallocated exchange funds aggregated by currency.

get_wallet_set(set_id=None, aggregate_only=False, **kwargs)
Return the wallet set with the specific ID.

handle_request(method, args, kwargs, connection_id)
Entry point for RPC calls.

init(config)

list_open_orders(**kwargs)
Get a list of open orders.

**Returns:** A list of Order objects, eg:

```{json}
[
    {
        "id":30,
        "created_at":"2018-04-07T02:29:58.933222",
        "wallet_set_id":2,
        "exchange":"gdax",
        "client_order_id":"8eed73b3-309c-4153-b6e5-0735b484aad3",
        "run_id":"asda",
        "type":"limit",
        "pair":"LTC_USD",
        "size":"1",
        "side":"buy",
        "limit":"200",
        "cancel_after":0,
        "status":"done",
        "done_reason":"filled",
        "done_at":"2018-04-07T02:29:59.119495",
        "exchange_at":"2018-04-07T02:29:59.070921",
        "last_updated_at":"2018-04-07T02:29:32.387727",
        "filled_size":"1",
        "avg_price":"117.55"
    }
]
```

The field descriptions are described in `Order`.

list_recent_orders(*, include_open=False)
Get a list of open orders.
Returns: A list of Order objects. See list_open_orders for example.

log_wallet_set (sender, *, wallet_set, key='wallet_set_changes')
Log wallet balances

make_order (*, connection_id, run_id, exchange, client_oid, type, pair, size, side, wallet_set_id=None, limit=0, cancel_after=0, **kwargs)
Make an order on the specified exchange.

Args:

  param exchange Identifier for the exchange.
  param connection_id Identifier for connection sending this request. Use to deliver Trade and Order notifications.
  param client_oid Unique identifier for this order from the client.
  param run_id identifier for a strategy instance.
  param wallet_set_id Wallet Set to run this transaction on.
  param side buy or sell
  param type A valid order type string.
  param pair A pair name supported by this exchange.
  param size Order size.
  param limit Order limit value for limit orders.
  param cancel_after Seconds after which this order should cancel.
  param **kwargs Other keyword arguments which are ignored.

Returns: A tuple (status, message) is returned. If status returns as invalid or rejected then message will return a descriptive reason.

notify_wallet_set_update (sender, *, wallet_set)
Create a notification signal with the updated wallet set as JSON.

register_exchange (klass, name, config, mode, currencies)

shutdown ()

subscribe_notifications (conn, connection_id)

unsubscribe_notifications (conn, connection_id)

exchange_wrapper.service.tag_type_exception (tag, typ, value)

1.6.14 exchange_wrapper.wallet module

class exchange_wrapper.wallet.Wallet (exchange, base, available, balance, on_hold, allocation=0)

Bases: object
Dumb object to hold wallet balances

allocation
available
balance
hold
1.6.15 exchange_wrapper.wsock_server module

A websocket server that can track multiple client connections.

```python
class exchange_wrapper.wsock_server.WebsocketServer(message_handler):
    Bases: object
    add_on_connect_hook(callback)
    add_on_disconnect_hook(callback)
    handler(websocket, path)
    make_server(host, port, **kwargs)
    receive_handler(id, websocket, path)
    run_server(*args, **kwargs)
    send_handler(id, websocket, path)
    send_to(id, message)
    send_to_all(message)
    shutdown()

exchange_wrapper.wsock_server.address_to_str(address_tup)
```

1.6.16 Module contents
HTTP ROUTING TABLE

/uri_map
GET /uri_map, 10

/xwrap
GET /xwrap/v1/(str:exchange)/book/(str:pair), 6
GET /xwrap/v1/(str:exchange)/pricing, 6
GET /xwrap/v1/exchanges, 9
GET /xwrap/v1/ohlc/(str:exchange)/(str:pair), 7
GET /xwrap/v1/orders/open, 10
GET /xwrap/v1/orders/recent, 10
GET /xwrap/v1/status, 10
GET /xwrap/v1/wallets/(int:set_id), 8
GET /xwrap/v1/wallets/allocated, 6
GET /xwrap/v1/wallets/history/(int:set_id), 8
GET /xwrap/v1/wallets/sets, 9
GET /xwrap/v1/wallets/total, 8
GET /xwrap/v1/wallets/unallocated, 8
GET /xwrap/v1/wallets/unallocated_exchange, 8
POST /xwrap/v1/orders/new, 6
POST /xwrap/v1/wallets/new, 10
POST /xwrap/v1/wallets/virtual/transaction, 10
PYTHON MODULE INDEX

e
exchange_wrapper, 39
exchange_wrapper.client, 20
exchange_wrapper.config, 21
exchange_wrapper.connection, 21
exchange_wrapper.db, 22
exchange_wrapper.http_server, 22
exchange_wrapper.logical_wallet, 26
exchange_wrapper.matching, 27
exchange_wrapper.models, 27
exchange_wrapper.pair, 31
exchange_wrapper.rpc, 32
exchange_wrapper.service, 33
exchange_wrapper.util, 20
exchange_wrapper.util.token_bucket, 20
exchange_wrapper.wallet, 38
exchange_wrapper.wsock_server, 39
A

active (exchange_wrapper.models.WalletSet attribute), 30
add_on_connect_hook() (exchange_wrapper.wsock_server.WebsocketServer method), 39
add_on_disconnect_hook() (exchange_wrapper.wsock_server.WebsocketServer method), 39
add_virtual_wallet_tx() (exchange_wrapper.client.Client method), 20
add_virtual_wallet_tx() (exchange_wrapper.service.ExchangeWrapper method), 33
address (exchange_wrapper.models.VirtualWallet attribute), 29
address_to_str() (in module exchange_wrapper.wsock_server), 39
aggregate (exchange_wrapper.models.WalletSet attribute), 30
allocate_to_wallet_set() (exchange_wrapper.service.ExchangeWrapper method), 33
allocation (exchange_wrapper.models.VirtualWallet attribute), 29
balance (exchange_wrapper.models.VirtualWallet attribute), 29
balance (exchange_wrapper.wallet.Wallet attribute), 38
BaseModel (class in exchange_wrapper.db), 22
client_order_id (exchange_wrapper.models.Order attribute), 27
cancel_order() (exchange_wrapper.client.Client method), 20
cancel_order() (exchange_wrapper.service.ExchangeWrapper method), 33
cast_bool() (in module exchange_wrapper.config), 21
collate_responses() (in module exchange_wrapper.Order attribute), 20
check_price() (exchange_wrapper.pair.Pair method), 31
collate_responses() (in module exchange_wrapper.client), 20
check_size() (exchange_wrapper.pair.Pair method), 31
Client (class in exchange_wrapper.client), 20
Client (class in exchange_wrapper.config), 21
Connection (class in exchange_wrapper.connection), 21
connection_worker() (in module exchange_wrapper.connection), 21
ConnectionError, 21
cast_bool() (in module exchange_wrapper.config), 21
convert_to_usd() (exchange_wrapper.service.ExchangeWrapper method), 33
date (exchange_wrapper.models.WalletLine attribute), 30
deactivate_wallet_set() (exchange_wrapper.service.ExchangeWrapper method), 34
data (exchange_wrapper.rpc.ApiUsageError attribute), 32
data (exchange_wrapper.rpc.MethodNotFoundError attribute), 32
data (exchange_wrapper.rpc.ProcessingError attribute), 32
date (exchange_wrapper.models.ProcessingError attribute), 32
date (exchange_wrapper.models.WalletLine attribute), 30
deactivate_wallet_set() (exchange_wrapper.service.ExchangeWrapper method), 34
deactivate_wallet_set() (in module exchange_wrapper.http_server), 22

dec2str() (in module exchange_wrapper.util), 20
description (exchange_wrapper.models.WalletSet attribute), 30

DoesNotExist (exchange_wrapper.db.BaseModel attribute), 22

DoesNotExist (exchange_wrapper.models.Order attribute), 27

DoesNotExist (exchange_wrapper.models.Trade attribute), 29

DoesNotExist (exchange_wrapper.models.VirtualWallet attribute), 29

DoesNotExist (exchange_wrapper.models.WalletLine attribute), 30

DoesNotExist (exchange_wrapper.models.WalletSet attribute), 30

done_at (exchange_wrapper.models.Order attribute), 27
done_reason (exchange_wrapper.models.Order attribute), 27
dt_to_msts() (in module exchange_wrapper.util), 20
dt_to_str() (in module exchange_wrapper.util), 20
dt_to_ts() (in module exchange_wrapper.util), 20

E

echo() (exchange_wrapper.service.ExchangeWrapper method), 34

echo_notify() (exchange_wrapper.service.ExchangeWrapper method), 34

exchange (exchange_wrapper.models.Order attribute), 27

exchange (exchange_wrapper.models.VirtualWallet attribute), 30

exchange_at (exchange_wrapper.models.Order attribute), 27

exchange_wrapper (module), 39

exchange_wrapper.client (module), 20

exchange_wrapper.config (module), 21

exchange_wrapper.connection (module), 21

exchange_wrapper.db (module), 22

exchange_wrapper.http_server (module), 22

exchange_wrapper.logical_wallet (module), 26

exchange_wrapper.matching (module), 27

exchange_wrapper.models (module), 27

exchange_wrapper.pair (module), 31

exchange_wrapper.rpc (module), 32

exchange_wrapper.service (module), 33

exec_amount (exchange_wrapper.models.Trade attribute), 29

exec_price (exchange_wrapper.models.Trade attribute), 29

execute_via_rpc() (in module exchange_wrapper.client), 20

ext_id (exchange_wrapper.models.Trade attribute), 29

fnmatch_dicts() (in module exchange_wrapper.util), 20

F

fee (exchange_wrapper.models.Trade attribute), 29

filled_size (exchange_wrapper.models.Order attribute), 27

G

get() (exchange_wrapper.config.Config method), 21

get_all_wallet_sets() (exchange_wrapper.service.ExchangeWrapper method), 34

get_allocated() (exchange_wrapper.logical_wallet.LogicalWalletManager method), 26

get_allocated_wallet_set() (exchange_wrapper.service.ExchangeWrapper method), 35

get_allocated_wallet_sets() (in module exchange_wrapper.http_server), 22

get_book() (exchange_wrapper.service.ExchangeWrapper method), 35

get_book() (in module exchange_wrapper.http_server), 22

get_connection() (exchange_wrapper.rpc.RpcInterface method), 32

get_exchange_pricing() (in module exchange_wrapper.http_server), 22

get_exchange_total() (exchange_wrapper.logical_wallet.LogicalWalletManager method), 26

get_exchange_unallocated() (in module exchange_wrapper.http_server), 23

get_exchanges() (exchange_wrapper.service.ExchangeWrapper method), 35

get_held_for() (exchange_wrapper.models.VirtualWallet method), 30

get_ohlc() (in module exchange_wrapper.http_server), 23

get_pricing() (exchange_wrapper.service.ExchangeWrapper method), 36

get_quote_amounts() (exchange_wrapper.models.Order method), 27

get_status() (exchange_wrapper.models.Order method), 27

get_total_wallet_set() (exchange_wrapper.service.ExchangeWrapper method), 36

get_total_wallet_sets() (in module exchange_wrapper.http_server), 24
maker (exchange_wrapper.models.Trade attribute), 29
match_limit_order() (in module exchange_wrapper.matching), 27
match_market_order() (in module exchange_wrapper.matching), 27
MethodNotFoundError, 32
metrics() (in module exchange_wrapper.http_server), 26
move (exchange_wrapper.models.WalletLine attribute), 30

N
name() (exchange_wrapper.pair.Pair method), 31
notify() (in module exchange_wrapper.http_server), 26
notify_connect() (exchange_wrapper.http_server.HttpServer method), 22
notify_wallet_set_update() (exchange_wrapper.service.ExchangeWrapper method), 38

O
on_connect() (in module exchange_wrapper.http_server), 26
on_connect_hook() (exchange_wrapper.rpc.RpcInterface method), 32
on_disconnect_hook() (exchange_wrapper.rpc.RpcInterface method), 32
on_hold (exchange_wrapper.models.WalletLine attribute), 30
Order (class in exchange_wrapper.models), 27
order (exchange_wrapper.models.Trade attribute), 29
order_id (exchange_wrapper.models.WalletLine attribute), 30
order_id (exchange_wrapper.models.Trade attribute), 29
order_type (exchange_wrapper.models.Trade attribute), 30
orders (exchange_wrapper.models.Order attribute), 27
orders (exchange_wrapper.models.WalletSet attribute), 31

P
Pair (class in exchange_wrapper.pair), 31
pair (exchange_wrapper.models.Order attribute), 27
parse_recv() (exchange_wrapper.client.Client method), 20
ProcessingError, 32

R
receive_handler() (exchange_wrapper.wsock_server.WebsocketServer method), 39
recursive_update() (in module exchange_wrapper.config), 21
register_exchange() (exchange_wrapper.service.ExchangeWrapper method), 38
remaining_size (exchange_wrapper.models.Order attribute), 27
request_token() (exchange_wrapper.util.token_bucket.TokenBucket method), 20
RpcInterface (class in exchange_wrapper.rpc), 32
run() (exchange_wrapper.connection.Connection method), 21
run_forever() (exchange_wrapper.rpc.RpcInterface method), 32
run_id (exchange_wrapper.models.WalletSet attribute), 31
run_server() (exchange_wrapper.wsock_server.WebsocketServer method), 39

S
send_handler() (exchange_wrapper.wsock_server.WebsocketServer method), 39
send_request() (exchange_wrapper.connection.Connection method), 21
send_request_async() (exchange_wrapper.connection.Connection method), 21
send_to() (exchange_wrapper.rpc.RpcInterface method), 32
send_to() (exchange_wrapper.wsock_server.WebsocketServer method), 39
send_to_all() (exchange_wrapper.rpc.RpcInterface method), 32
send_to_all() (exchange_wrapper.wsock_server.WebsocketServer method), 39
settlement (exchange_wrapper.models.Order attribute), 27
shutdown() (exchange_wrapper.rpc.RpcInterface method), 32
shutdown() (exchange_wrapper.service.ExchangeWrapper method), 38
shutdown() (exchange_wrapper.wsock_server.WebsocketServer method), 39
side (exchange_wrapper.models.Order attribute), 27
size (exchange_wrapper.models.Order attribute), 28
start() (exchange_wrapper.http_server.HttpServer method), 22
status (exchange_wrapper.models.Order attribute), 28
stop() (exchange_wrapper.connection.Connection method), 21
strategy_run_id (exchange_wrapper.models.Order attribute), 28
subscribe_notifications() (exchange_wrapper.service.ExchangeWrapper method), 38
T
tag_type_exception() (in module exchange_wrapper.service), 38
to_dict() (exchange_wrapper.models.Order method), 28
to_dict() (exchange_wrapper.models.Trade method), 29
TokenBucket (class in exchange_wrapper.util.token_bucket), 20
Trade (class in exchange_wrapper.models), 28
type (exchange_wrapper.models.Order attribute), 28
type (exchange_wrapper.models.WalletSet attribute), 31

U
unfilled_size (exchange_wrapper.models.Order attribute), 28
unsubscribeNotifications() (exchange_wrapper.service.ExchangeWrapper method), 38
update_wallets() (exchange_wrapper.models.Order method), 28
uri_map() (in module exchange_wrapper.http_server), 26

V
VirtualWallet (class in exchange_wrapper.models), 29

W
Wallet (class in exchange_wrapper.wallet), 38
wallet (exchange_wrapper.models.WalletLine attribute), 30
wallet_id (exchange_wrapper.models.WalletLine attribute), 30
wallet_lines (exchange_wrapper.models.Order attribute), 28
wallet_lines (exchange_wrapper.models.VirtualWallet attribute), 30
wallet_set (exchange_wrapper.models.Order attribute), 28
wallet_set (exchange_wrapper.models.VirtualWallet attribute), 30
wallet_set_id (exchange_wrapper.models.Order attribute), 28
wallet_set_id (exchange_wrapper.models.VirtualWallet attribute), 30
WalletLine (class in exchange_wrapper.models), 30
wallets (exchange_wrapper.models.WalletSet attribute), 31
WalletSet (class in exchange_wrapper.models), 30
websocket_query() (in module exchange_wrapper.client), 20
websocket_query_timeout() (in module exchange_wrapper.client), 21
WebsocketServer (class in exchange_wrapper.wsock_server), 39