Imperial College London

COURSEWORK

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

QSync

Capturing and Visualising Cryptocurrency Market Data

Ayoob Ahmed
aa1319

Sanchit Ajmera
sa3119

Tyrell Duku
tld19

Mazen Hussein
mah319

Mustafa Ilyas
mi819

Luqman Liaquat
lml19

Abdur Sharif
ars319

Dr. Paul Bilokon
Supervisor

Executive Summary

The popularity of cryptocurrencies has surged in recent years, with the number of cryptocurrency holders doubling in 2021 from 100 million to 200 million [1]. Many traders have identified cryptocurrencies as a lucrative opportunity due to their volatility compared to traditionally regulated markets. Numerous trading strategies require simultaneous access to market data of multiple cryptocurrencies across different exchanges. However, this is not easily obtainable. Traders would need to connect to distinct exchange APIs or have multiple screens open to display the data.

Unlike well-established markets, which would be described by economists as 'efficient', cryptocurrency markets are still relatively immature and often present arbitrage opportunities. This generates the need for a tool to aggregate and display the required data across centralised exchanges coherently and in real-time. The tool should inform traders of the current state of the market, enabling them to easily carry out strategies such as *cross-exchange arbitrage*: Taking advantage of slight differences in the price of the same crypto (short for cryptocurrency) on different exchanges.

Introducing QSync, a web application that gives users access to various visualisations of real-time and historical *level-2* cryptocurrency data from across multiple exchanges. This involves the bids and asks for a given asset, aggregated by price. The scalability of our data collection infrastructure allows for new currencies and exchanges to be integrated into the application with ease. QSync is highly performant, enabling traders to be first-movers on identifying profitable opportunities.

Some of our visualisations include a table of the best cross-exchange arbitrage opportunities in real-time, detailing the crypto to trade, the exchanges involved and the potential profit that can be made. With the rise in new cryptocurrency users, our visualisations also cater to the less-experienced trader. We provide explanations for key terminology, descriptions for supported crypto assets and information on how visualisations may be used in a particular trading strategy.

QSync successfully meets the objectives our client set out. It provides high performance for metric calculations, with our benchmarks demonstrating 300,000 data points being queried in under 0.04 seconds. Our data collection infrastructure enables low latencies for sending real-time and historical data to the user. We have over 10 visualisations giving insight into popular cryptocurrencies' price data, arbitrage opportunities and the latest news on the state of the market. Our application supports the most traded cryptocurrencies such as Bitcoin, Ethereum and Solana with many more integrations planned for the future.

Our client, Dr. Paul Bilikon, the CEO and founder of Thalesians ltd [2], agreed that the objectives were met successfully and stated:

"*The team has implemented a sophisticated modular system that deals with level-2 data arriving in real-time. The application provides cryptocurrency traders with a flexible general-purpose tool for summarising the cryptocurrency markets and detecting arbitrage opportunities.*"

1 Introduction

1.1 Motivation

Real-time data capture and visualisation is an integral component of industry-standard financial tools used to make trading decisions. Based on the current state of the market, traders need to quickly determine what trades to execute for maximal profit, and when. The real-time aspect is even more crucial for cryptocurrency trading due to the high volatility of the markets when compared to traditional trading instruments, such as stocks.

Over the last five years, there has been an exponential increase in the number of cryptocurrency wallet users, rising from 9.26 million in October 2016 to 79.3 million in October 2021 [3]. This has led to a surge in the number of daily transactions, which can exceed one million per day on a single blockchain (e.g. Ethereum in 2020) [4]. Financial tools need to ensure their infrastructure is capable of supporting this ever-expanding pool of users which is challenging due to the real-time nature of the product.

The expanse of the cryptocurrency space is another challenging factor for financial tools. There are over 1500 currencies [5] and 300 exchanges [6] that allow fiat-for-crytocurrency (fiat is government-issued currency such as the Dollar) or crypto-to-crypto transactions [4]. Traders need to know prices on these exchanges to make the best judgment on transactions to execute. As an example, strategies such as cross-exchange arbitrage involve buying a cryptocurrency from one market and selling it on another for profit, taking advantage of slight price differences. This strategy requires the trader to constantly monitor multiple exchanges, which is challenging, as each one provides its own distinct APIs and applications.

1.2 Main Objectives

The primary objective of the project is to construct a scalable infrastructure for capturing high-frequency cryptocurrency market data for a number of cryptocurrencies across various centralised exchanges. Hence, the system design should have the capacity to support the integration of new exchanges and cryptocurrencies to collect data from, with minimal additional development. This directly feeds into the secondary goal of producing visualisations of metrics most important to our client. These should assist the client in understanding the state of the market to best inform their possible trading decisions.

The client requirements for our application are:

- Appropriate storage techniques for high-frequency data
- Real-time prices for Bitcoin and Ethereum
- Provide real-time market data feeds to clients

- Provide a simple method to add new currency/exchange data feeds
- Historical graph of prices and price changes
- Price comparisons between exchanges (arbitrage)

1.3 Achievements

QSync meets the goal of producing a complete and scalable system for capturing cryptocurrency market data across exchanges. We provide a number of visualisations for Bitcoin, Ethereum, Solana, Dogecoin and Cardano.

From an engineering standpoint, we were able to create a high-speed data capture system through the use of *kdb+*, a performance-oriented time-series database [7, 8]. This allows us to process millions of data points in a few seconds, enabling us to further expand the number of cryptocurrencies supported in the future.

We were able to connect the data capture system to a robust front-end built using *React* and *TypeScript*. *LightningChartJS* was used to display graphs elegantly and purposefully, and together this system provided many meaningful metrics for our client, including breakdowns for several top cryptos, and more in-depth arbitrages to aid with certain trading strategies.

2 Design and Implementation

2.1 Development Techniques

We aggregated our understandings of useful software engineering practices for development in this project. We took an Agile approach, using a mixture of Kanban and Scrum methodologies [9] which fit well with the pre-defined sprints of the project.

2.1.1 Software Engineering Practices

Our development process began by identifying the requirements of each iteration which our product manager converted into tickets on *Jira* and assigned to members. These tickets contained detailed and specific requirements which would be evaluated against in code reviews. Each ticket would be its own branch in our *GitLab* repository, making it easy to track the progress of any task. Additionally, we used a *GitLab-Jira* integration which enabled our branches and commits to link directly to a ticket in Jira, so all members were aware of what each commit was working towards. The board contained columns for *to-do*, *in-progress*, *blocked*, *code review*, *merge* and *done*. These columns enabled a clear and easy-to-track development process throughout the project and let all members of the team know the state of the project at any given time.

We had daily stand-ups which allowed members to update everyone on how their task was progressing. This also allowed us to identify any blockers and allocate support to members that required it. We had retrospectives which enabled us to discuss how smooth the development process was over the previous iteration and identify how we can improve the way tasks were being completed. We used *Confluence* to write documentation including tutorials and notes for other members to understand complicated technologies.

2.2 System Architecture

Figure 1 shows the final System Architecture of our project. The following sections will explain how we developed and rectified areas of our architecture to improve performance, as well as scalability, and arrive at this final design.

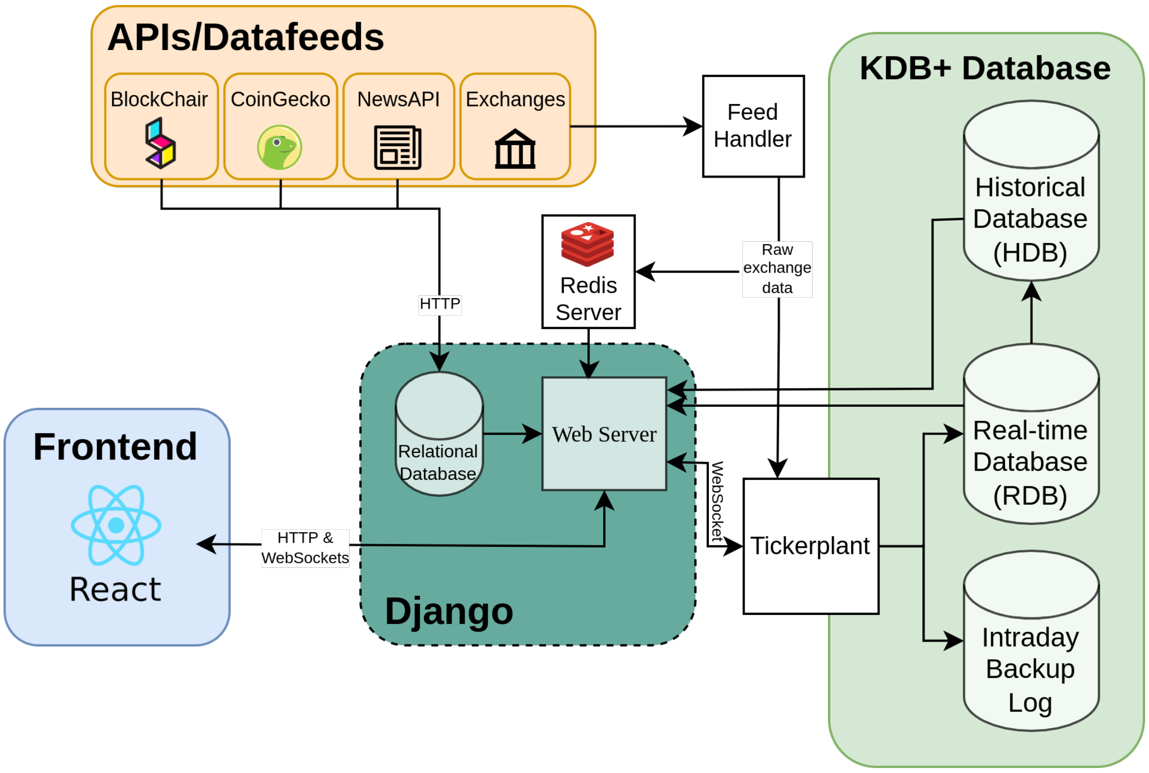


Figure 1: Our final system architecture diagram

2.3 Choosing a Database - kdb+

Since cryptocurrency usage is growing rapidly, storage and data aggregation will become ever more important as our client will look to apply trading strategies to new cryptocurrencies. Our choice of a database system will need to cater to the large volumes of real-time data we would collect.

The cryptocurrency market data being collected falls under the category of 'time-series data' which is data stored in order of time. The group took an interest in the kdb+ [8] database system due to its columnar structured tables, which allow for exceptionally fast analysis of large-scale data. Applications of kdb+ on a single server have yielded performance metrics such as "*aggregating over 40 billion sensor readings in under 2 minutes*" and searching "*in-memory tables at 4 billion records per second*" [8].

2.4 Our kdb+ Architecture

We use an architecture called kdb+tick for the main setup of the database, which is a three-process setup. It includes three parts, known as a **tick triplet**:

- Tickerplant
- RDB (real-time database)
- HDB (historical database)

The Tickerplant is the brain of the operation. It manages incoming real-time data and forwards it to subscribed processes, which includes the RDB. The Tickerplant allows using a publish-subscribe pattern enabling clients (i.e. our back-end) to subscribe to all updates to certain tables. It also makes use of an on-disk log file for recovery of today's data in case of an RDB failure.

The RDB is an in-memory database, storing data for only the current day (since midnight). This is done to allow significantly faster queries and insertions of data compared to on-disk storage.

The HDB is the final component of the triplet, storing all data before the current day. The HDB is on-disk storage but still allows for fast querying of data by utilising specialised storage techniques such as splaying [10].

Below is an architecture diagram for the database we created. The real-time subscriber in this case is simply our back-end.

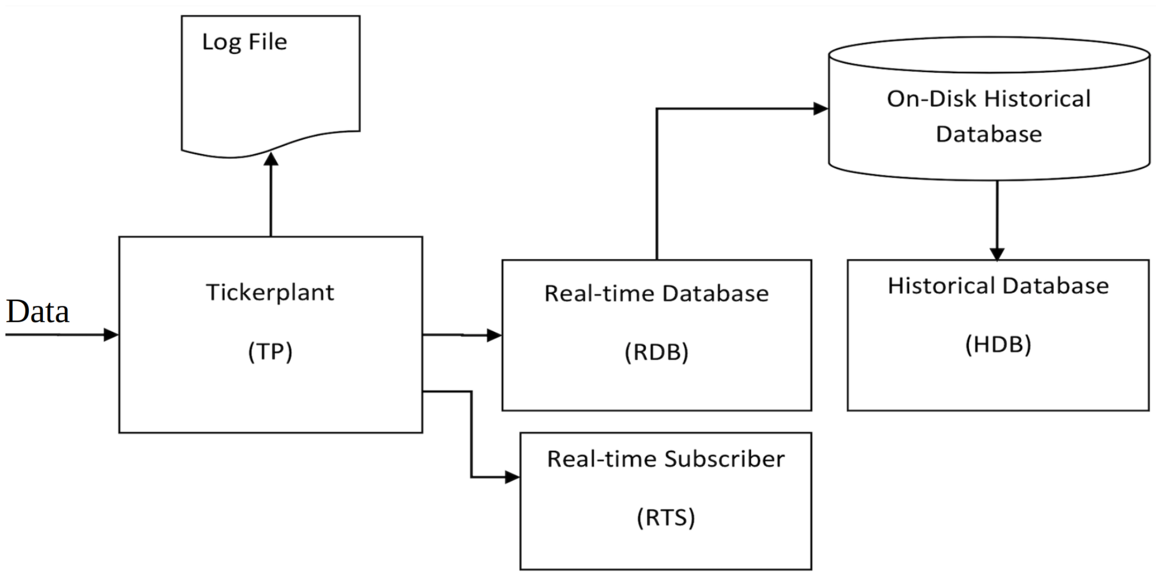

Log File
On-Disk Historical Database
Data
Tickerplant (TP)
Real-time Database (RDB)
Historical Database (HDB)
Real-time Subscriber (RTS)

Figure 2: A kdb+tick architecture diagram [11]

2.5 Database Challenges

This database system was new to all members of the team leading to the considerable effort expended to gain familiarity with the architecture, connecting the database to our Django web app, and learning the *q* language (this is the powerful query language for kdb+).

Sending tick-level data from the data feed to the clients would be unnecessary for many types of metrics. For example, price updates which are more fine-grained than once per second is unneeded for non-automated trading. To accommodate for this in q, we would use a **chained tickerplant** which is a second tickerplant connected to the first, allowing updates to be sent periodically. It also acts as a safety mechanism to prevent direct access should we decide to allow clients to connect directly to kdb+ in the future.

2.6 Front-end

We utilised *React* as our front-end framework due to the simplicity of its component-based structure. React's virtual DOM enabled us to efficiently render our planned components with live-data feeds, which is essential for the real-time performance of the application. We used React alongside the React-Bootstrap library to create reusable user-friendly UI elements. Angular was another framework we considered but the benefits were not justified compared to the learning overhead for our team as we already had experience with React. We decided to implement the front-end in *TypeScript* over *JavaScript* due to the additional type-safety, making code more reliable.

2.6.1 Charting Library

To render the appropriate graphs and figures, we used *LightningChartJS* due to its extensive API, real-time time-series support and great performance. We considered other graphing libraries such as *amcharts*, *Plotly.js* and *SciChart* but found *LightningChartJS* provided more support for easily visualising large quantities of real-time time-series data. A performance comparison report, comparing the performance of different graphing libraries including those mentioned above, showed *LightningChartJS* to have the best performance characteristics for high-frequency data [12]. We also found that *Plotly.js* amassed numerous reports of performance issues when rendering a large number of data points, and *SciChart* did not have a free licence to use which further supported our choice.

2.6.2 WebSockets and HTTP Communication

We decided on the use of WebSockets to facilitate all live data feeds. This choice was clear due to the full-duplex communication required for the user to send a request and the server to respond with a stream of real-time data. WebSockets also offer less overhead and latency compared to protocols such as HTTP [13]. We provided a REST API for historical data where latency is less important.

2.7 Back-end Framework and Languages

We chose Python as our back-end programming language as it provided a library, qPython, to facilitate interprocess communication with the q language used by kdb+ [14]. This warranted its use over other languages where we would have to create the integration ourselves. Coupled with Python's natural focus for data-driven projects, this led us to use the Django web framework. We chose Django over the micro-framework Flask due to its larger array of features that sped up development time. We were also familiar with Django which reduced the time we spent on additional learning.

2.7.1 Choosing an Exchange Data Feed Handler

In order to collect raw market data from various centralised cryptocurrency exchanges, we were presented with the problem of creating distinct adapters for the varying exchange APIs. To overcome this, we used the *Cryptofeed* library which creates WebSocket (or REST endpoint when not available) connections to the specified exchanges and provides a uniform way of collecting the required data [15]. We chose to focus on centralised exchanges as they provided trading instruments, such as futures, which were most important to our client.

2.7.2 Considering Market-Wide API Options

Some of the metrics required by the client were market-wide, i.e. aggregation of data across all centralised and decentralised exchanges, such as Market Capitalisation [16]. Therefore, we decided to use the API services CoinGecko [17] and Blockchair [18], which provide a variety of market-wide metrics, as it was not possible to calculate these metrics by simply using the exchange data we collected via Cryptofeed.

2.7.3 Django Channels

To support WebSockets through Django, we utilised a python module called *Django Channels*. This provided support to implement *WebSocket* servers through classes called *Consumers*. Furthermore, this module allowed us to incorporate WebSockets with asynchronous message passing via a *Redis* instance. This meant that we could receive data directly from *Cryptofeed* and distribute it to all connected clients as soon as we received it in *Django*.

2.8 Data Flow Summary

From the above sections, we can now summarise the data flow in our architecture diagram (Figure 1). A Feed-handler process collects data through WebSockets from various exchanges and sends data to both Django and the ticker plant process. The ticker plant sends the received data to the real-time in-memory database, which will write all of its contents into the historical database at the end of the day. *Django* utilises *Channels* to send the received data to all WebSocket servers subscribed to that type of data. Each server may perform some processing on the data before sending it to all of its clients. When a client wants real-time data, a WebSocket connection is established with the appropriate WebSocket server in the Django backend. The client specifies, in a request, the type of data they would like to receive. When the request is received by the server, the server will subscribe to receive all updates for the data type specified by the client.

2.9 Continuous Integration/Deployment

As another major practice used in software engineering, we used CI/CD throughout the project. Each feature branch had to pass the pipeline (and a code review from other members of the team) to allow for merging into the master branch.

2.9.1 Pipeline

We created a GitLab CI/CD pipeline which enabled us to set requirements for pushed code. The stages were *compile*, *style*, *test* and *deploy*. Compile simply checked the compilation of the code for the front-end and back-end. The style checks were a combination of format checks using *prettier* and *autopep*, along with linting checks using *eslint* and *pylint*. This allowed us to maintain a uniform codebase in style and format across our files. We had a test stage that carried out any unit and integration tests for our project. Finally, we had our deployment stage which is detailed further below.

2.9.2 Deployment

Our client requested that the data capture occurred on their own server, allowing us to utilise large amounts of disk space which would have otherwise been expensive. As we used a range of complex technologies, the deployment process had to amalgamate these in a simple method that could work regardless of the environment. This led us to containerize our services for front-end, back-end and the database system using Docker. These were paired with custom *run* scripts, enabling developers to spin up environments quickly to test changes. We further simplified the process of automated deployment to the server using Docker-Compose which allowed us to spin up all our services, create our application network and assign bind mounts to the correct disk drives.

2.9.3 Challenges

Creation of the environments proved to be challenging as the kdb+ database system did not have any official Docker base images. This required us to make our own using a licence from *Kx Systems*, the developers of kdb+. As it was our first time using many of the technologies such as docker-compose and kdb+, connecting the containers was another challenge. This was solved through careful analysis of the ports that were being used, and which services were attempting to communicate with each other through web sockets or API requests. We also had issues on the server with data collection which were resolved but had meant we missed out on collecting data for a few days. A useful method to prevent this in the future would be to build a notification system, e.g. using Graphana [19], to monitor the data collection and know when an issue (that could not be automatically resolved) has occurred.

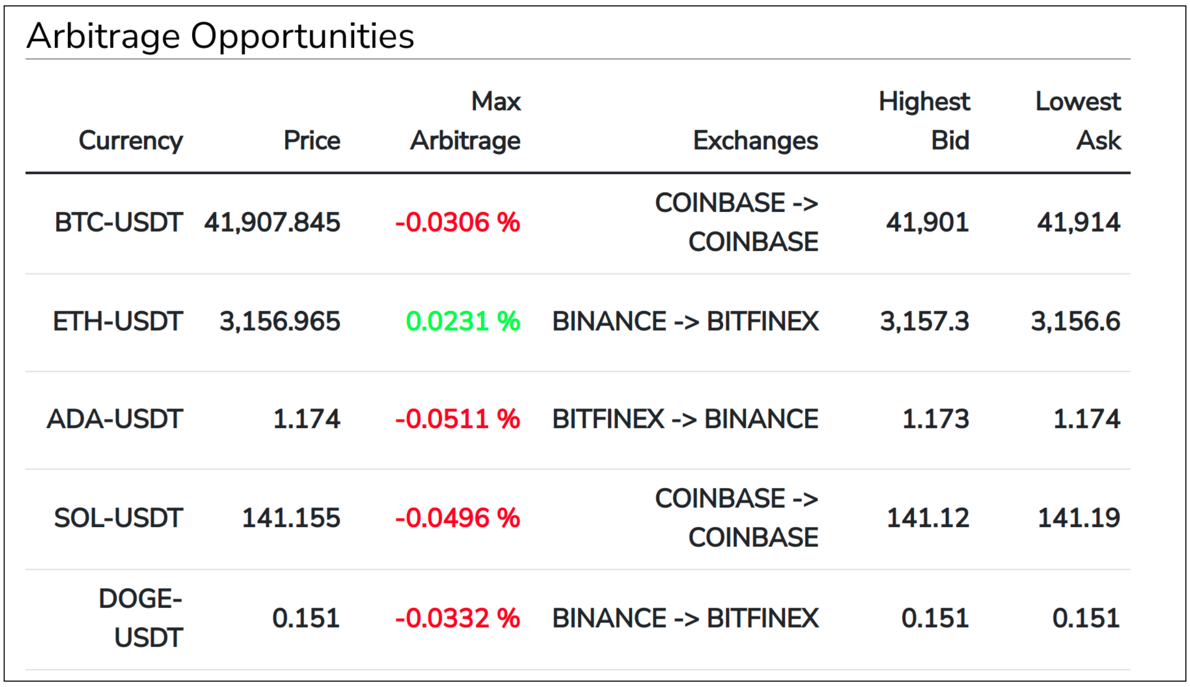
3 Visualisations

Our client requested specific visualisations that they would find useful. As a result, we first created mock-ups to get their opinion on how to best present the data and once approved, we would develop the components. The following are examples of the types of visualisations we created.

3.1 Arbitrage

A major use case of our application providing data on cryptocurrencies across exchanges was to identify arbitrage opportunities. These are trading opportunities where a user could take advantage of slight price differences of an asset across different markets. For example, if an apple cost £1 in Market One and £1.02 in Market Two, then a trader could purchase the apple from Market One and sell it in Market Two, yielding a profit of 2%.

An arbitrage opportunity component we created identifies the largest differences in prices across different exchanges for the same asset (figure 3).



Arbitrage Opportunities

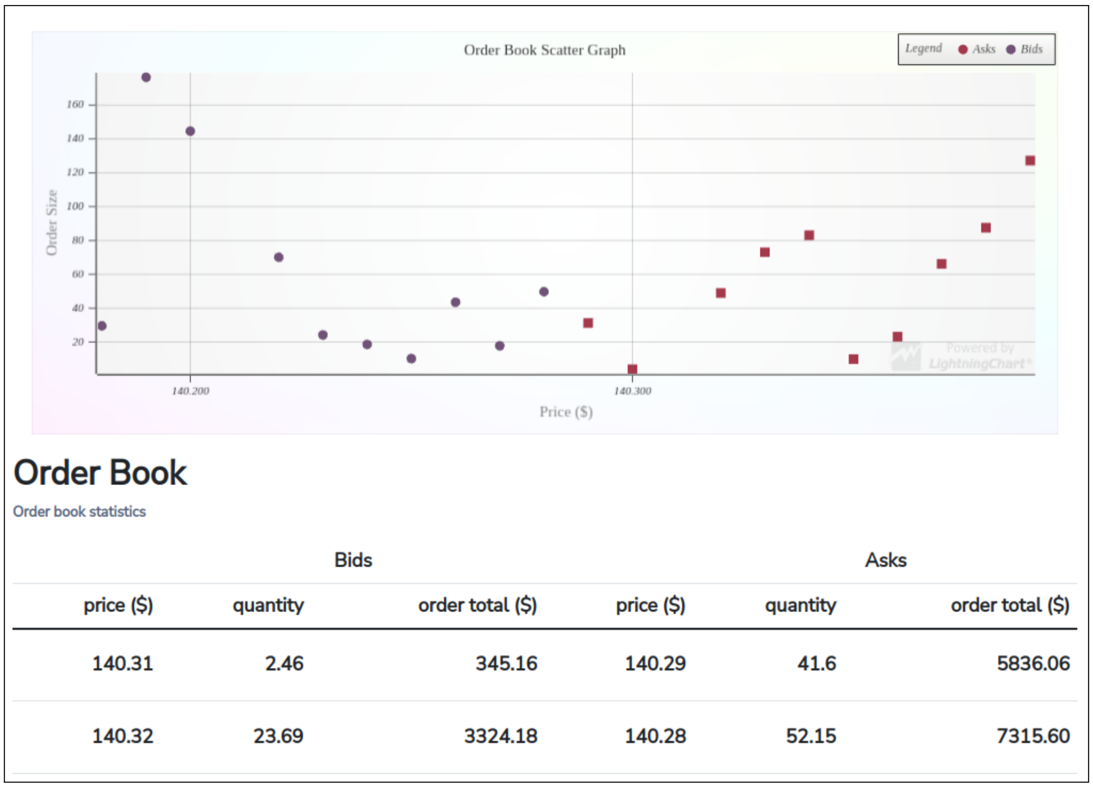
Currency	Price	Max Arbitrage	Exchanges	Highest Bid	Lowest Ask
BTC-USDT	41,907.845	-0.0306 %	COINBASE -> COINBASE	41,901	41,914
ETH-USDT	3,156.965	0.0231 %	BINANCE -> BITFINEX	3,157.3	3,156.6
ADA-USDT	1.174	-0.0511 %	BITFINEX -> BINANCE	1.173	1.174
SOL-USDT	141.155	-0.0496 %	COINBASE -> COINBASE	141.12	141.19
DOGE-USDT	0.151	-0.0332 %	BINANCE -> BITFINEX	0.151	0.151

Figure 3: A table providing possible cross-exchange arbitrage opportunities

A more complex arbitrage strategy uses a 'basis table', which displays differences in 'spot' (standard market) prices and 'future' (a bet on the future price) prices of an asset. A trader would require this information and a history of these differences, which we provide via an interactive table and corresponding graph (Appendix A).

3.2 Order Book and Price History

The most basic data to provide for the client was real-time and historical data on L2 order books (bids and asks made for a currency pair, grouped by price). We created an order book table component, which we paired with a scatter graph to present this data. We also created a component for the price history of a chosen cryptocurrency (Appendix B).



Order Book

Order book statistics

Bids			Asks		
price ($)	quantity	order total ($)	price ($)	quantity	order total ($)
140.31	2.46	345.16	140.29	41.6	5836.06
140.32	23.69	3324.18	140.28	52.15	7315.60

Figure 4: An order book table and scatter graph visualisation for Bitcoin

3.3 Additional Visualisations

Additionally, we proposed our own visualisations which impressed our client, leading to approval for development. An example of this is a news feed showing a list of the latest news articles on cryptocurrencies (Appendix C). This gives further insight into why some cryptocurrencies' prices may have changed. We also constructed a set of graphs and tables which show key metrics on the top currencies we support, including Bitcoin and Ethereum (Appendix D).

4 Evaluation

We believe the application that we built is a success overall. Primarily, we met our client's needs of developing a scalable infrastructure for high-frequency data collection. We also developed visualisations, specified by our client, which has been shown in an informative manner. An example of this is a spot-futures arbitrage basis table which solves a major problem the client had in using this particular arbitrage strategy. Previously, they would have to go to each exchange independently and did not have access to the basis history, but our visualisations now provide these features for them to use effectively.

4.1 kdb+ Performance

As part of evaluating our usage of kdb+, we wanted to test how our architecture as well as queries that we programmed scale to more intensive and data-heavy calculations.

4.1.1 RDB Benchmark

The RDB is important due to its faster in-memory calculations, and regular polling. We wanted to ensure that our architecture was performant for future scalability. To do this, we studied how a set of our queries were performing on up to 500,000 data points ($\approx$ half a day's data). These are executed on our deployment server (12-core CPU @ 3.20GHz with 64GB DDR4 RAM).

In particular, we decided to look at how our query, which calculated the *historical change in price*, grew the total query time when using an increasing volume of data points.

Figure 5 indicates kdb+ is scaling linearly with an increasing volume of data. This shows that the architecture can withstand a higher volume, meaning that any bottlenecks will likely be a result of hardware rather than implementation. This also shows that the query times themselves are fast and assuming linear scalability, it would take 120ms to aggregate over a day's worth of 1 million data points on a single table.

We also looked at how some of our other queries were performing to find weak points in our database. A query for a single data point by a given timestamp took 20 milliseconds on average over 100 iterations. This highlighted that the database wasn't well optimised for returning single data points, but this is reasonable as q is built for data aggregation over thousands of data points. If a requirement for this use case arises, then we can implement caching in Django to increase query speeds.

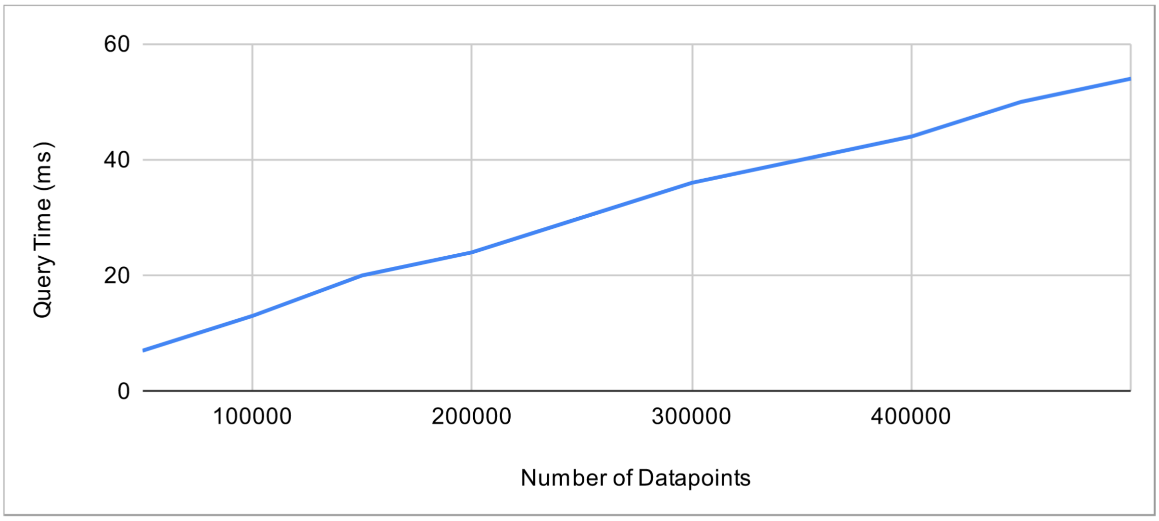


Figure 5: Time for percentage change history query to complete for an increasing volume of data points.

4.2 Code Evaluation

As a software engineering project, evaluating our code thoroughly was a core practice we employed through code reviews and automated tests.

4.2.1 Code Reviews

From the start of our project we set up systems to enforce approval of a merge request before allowing it to be merged into production. This approval had to come from a member of the group who was not working on that branch. This allowed us to identify and solve problems that might have been missed as a result of tunnel vision from developers. We used GitLab's merge request *comments* feature along with scheduling calls to resolve any issues. We also used the agile practice of pair programming on more complex tickets that required careful development, such as writing queries for large volumes of data.

4.2.2 Automated Tests

We implemented automated functional tests to ensure that updates to our code did not introduce unexpected breaking changes. Our front-end used the lightweight React *testing library* to test our different components rendered as expected. The *Django testing framework* allowed us to extensively test our data consumers and views along with integration tests to verify the addresses of our other services were resolved correctly. Finally, we wrote unit tests for our kdb+ database system to ensure the q functions we wrote returned the correct data. This was done using a mock database we created to directly calculate expected function outputs. We composed these automated tests and ran them in our pipeline so every code push could verify that an unexpected error did not occur.

4.3 Qualitative Evaluation

4.3.1 User Testing

Our supervisor was our user. We ensured that throughout every stage of the project, their ideas, requirements and feedback were taken into consideration to build the application. We had two meetings every week of the project which enabled us to understand the features that needed to be developed. The meetings also allowed us to receive feedback on the value and usefulness of previously developed features. With this high frequency of meetings with our user, we were able to constantly evaluate how well our project met the needs of the user. As a result, we were able to quickly add, remove, or update features to match the expectations of the client successfully, and solve any issues they identified early. This was evident in our iteration marks, where we were given a 3/5 or above in every iteration. This use of continuous user testing and evaluation led to a more successful project and is a methodology we will employ in future projects.

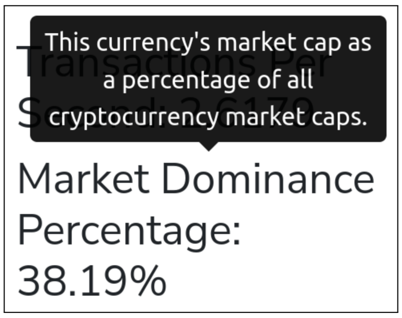
4.3.2 External Feedback

We believed getting qualitative feedback from sources outside of our client, particularly from a professional, would bring an alternative perspective that could give us further insight into the tools we were building. Therefore, we conducted an interview and demo with a systematic quantitative trader at BlackRock, where we asked for feedback on the mockups and implementation of our application. He praised the use of our *kdb+* tick triplet setup, stating that they were commonly used in industry for similar purposes to ours. He stated the importance of particular metrics he, and other traders, would find useful, namely the order book imbalance of given cryptocurrencies. Following the feedback, we made changes to the front-end and back-end to implement the display of order book imbalances, which our client approved of.

4.3.3 Internal Feedback

We also obtained feedback from other members of the Department of Computing. Professor William Knottenbelt [20] praised our data-capture architecture while stating one aspect to be improved upon was our front-end. He felt that some of the visualisations were presented in a confusing manner, leading to a loss of usefulness. Therefore, we made sure all metrics and visualisations were presented and explained clearly. For example, we labelled graphs more clearly and added tool-tips to headings throughout the application (figure 6).

We believe the feedback from both internal and external parties helped to improve the application and gave us a chance to evaluate different aspects of the project in more depth. We would like to thank Professor William Knottenbelt for his feedback which helped us significantly in improving our project.


This currency's market cap as a percentage of all cryptocurrency market caps.

Market Dominance Percentage: 38.19%

Figure 6: An example of a tool-tip to explain the Market Dominance Metric

5 Ethical Considerations

5.1 The Platform as a Financial Tool

QSync is a platform for visualising cryptocurrency market data to assist with trading decisions. The platform only displays information and does not facilitate any trading of financial assets. The legal considerations we make are based on UK law.

5.1.1 Financial Risk

As a highly volatile and largely unregulated market, cryptocurrencies can be dangerous to trade for beginners. We have therefore decided to place a financial disclaimer on first entering the page to clarify the risk of trading.

5.1.2 Derivatives

Used in finance and investing, a derivative is a type of contract. An example of a derivative is a futures contract, an agreement between two parties for the purchase and delivery of an asset at an agreed-upon price at a future date [21]. Cryptocurrency derivatives are banned by the FCA for retail (non-professional) investors in the UK, and exchanges have recently been working to implement these laws [22, 23].

Our platform provides data on futures, through a spot-futures arbitrage, which aims to provide data on how profits can be made through investing in futures contracts. A consideration we need to make is to perhaps verify that clients have approval under UK law to trade these products and hence restrict access to less knowledgeable users.

5.2 Cookies and Personal Data

QSync does not process any personal data. QSync utilises a cookie to determine whether the user has visited the site before, in order to to display a financial warning as described in 5.1.1. This cookie is necessary as it informs the user of an important message, and hence there is no need to obtain consent under UK Data Protection Laws [24]. We inform the user of this cookie.

5.3 Accessibility

For the future, our application would need to ensure it is accessible to all. For example, we would need to add alt-text and update colour choices to cater to different types of visual impairments.

5.4 External Libraries

A legal consideration to make is concerned with the *LightningChartJS TypeScript* library used for rendering real-time time-series graphs. *QSync* uses the Community edition of *LightningChartJS*, which watermarks all graphs rendered and forbids the use of any graphs for commercial use or websites under its licence. To resolve this, in the future we should purchase a licence for commercial use from *Arction* or use other libraries. Similarly, kdb+ requires a commercial licence.

All other libraries we use fall under open-source licences, therefore we are permitted to use them for commercial purposes. We would need to take care, to check whether any libraries require credit to be given before publishing our application.

6 Improvements & Future Development

A general task in future development is to continue to add new exchanges and cryptocurrency pairs to collect data on. Another improvement is to create more metrics and add support for more complicated arbitrage strategies. In the future, it would also be useful to construct a feed handler to collect data from *decentralised* exchanges, such as Uniswap, to give our users a more complete view of the market and present new opportunities.

Adding the ability to have an account system would also open the application to a whole new range of users, and allow us to create more personalised visualisations. For example, a machine learning system could identify what the best chart would be to display on the home screen for a particular user.

7 Conclusion

In conclusion, *QSync* satisfies the objectives outlined by our client in *section 1.2*. The system we developed is robust and functions correctly end-to-end. The data-capture architecture, in particular, allows for simple access of information on multiple currency pairs across exchanges. The user-friendly front-end presents a range of information, such as arbitrage opportunities and Level 2 order book data in an organised and easy-to-read fashion.

Our client praised the “sophisticated system” that we built to help him “summarise the cryptocurrency markets and detect arbitrage opportunities”. With QSync, traders will now be able to access a holistic view of the cryptocurrency landscape.

8 References

[1] Wang K. Measuring Global Crypto Users, A Study to Measure Market Size Using On-Chain Metrics. crypto.com; 2021.

[2] People – Thalesians [Internet]. Thalesians.com. 2022. Available from: https://thalesians.com/about/people/

[3] Best R. Blockchain wallets 2011-2021 — Statista [Internet]. Statista. 2021 . Available from: https://www.statista.com/statistics/647374/worldwide-blockchain-wallet-users/

[4] Best R. Daily cryptocurrency transactions 2017-2021 — Statista [Internet]. Statista. 2021 . Available from: https://www.statista.com/statistics/730838/ number-of-daily-cryptocurrency-transactions-by-type

[5] Best R. Number of crypto coins 2013-2021 — Statista [Internet]. Statista. 2021 . Available from: https://www.statista.com/statistics/863917/number-crypto-coins-tokens

[6] Top Cryptocurrency Exchanges Ranked By Volume — CoinMarketCap [Internet]. CoinMarketCap. 2022 . Available from: https://coinmarketcap.com/rankings/exchanges/

[7] Taylor S. Developing with kdb+ and the q language - Kdb+ and q documentation [Internet]. Code.kx.com. 2022 . Available from: https://code.kx.com/q/

[8] Tomczak P. What Makes Time-Series Database kdb+ So Fast? [Internet]. KX. 2022 . Available from: https://kx.com/blog/what-makes-time-series-database-kdb-so-fast

[9] REHKOPF M. Kanban vs Scrum — Atlassian [Internet]. Atlassian. 2022 . Available from: https://www.atlassian.com/agile/kanban/kanban-vs-scrum

[10] Taylor S. Splayed tables — Knowledge Base — kdb+ and q documentation - Kdb+ and q documentation [Internet]. Code.kx.com. 2022 . Available from:

https://code.kx.com/q/kb/splayed-tables/

[11] Perrem N. Building real-time tick subscribers — White Papers — kdb+ and q documentation - Kdb+ and q documentation [Internet]. Code.kx.com. 2022 . Available from: https://code.kx.com/q/wp/rt-tick/

[12] GitHub - Arction/javascript-charts-performance-comparison: Public comparison of JavaScript chart libraries performance in visualizing a real-time multichannel ECG chart. [Internet]. GitHub. 2022 . Available from: https://github.com/Arction/javascript-charts-performance-comparison

[13] Nickerson B, Pimentel V. Communicating and Displaying Real-Time Data with WebSocket. IEEE; 2022.

[14] GitHub - exxeleron/qPython: interprocess communication between Python and kdb+ [Internet]. GitHub. 2019 . Available from: https://github.com/exxeleron/qPython

[15] Moscon B. GitHub - bmoscon/cryptofeed: Cryptocurrency Exchange Websocket Data Feed Handler [Internet]. GitHub. 2022 . Available from: https://github.com/bmoscon/cryptofeed

[16] What is market cap? [Internet]. Coinbase. 2022 [cited 10 January 2022]. Available from: https://www.coinbase.com/learn/crypto-basics/what-is-market-cap

[17] Cryptocurrency Prices by Market Cap [Internet]. CoinGecko. 2022 [cited 10 January 2022]. Available from: https://www.coingecko.com/en?__cf_chl_jschl_tk__= _2xbkEs47rA6o0iaIEEpChQXYD66z6kTepRRFn4vQFI-1641796860-0-gaNycGzNCJE

[18] Blockchair — Universal blockchain explorer and search engine [Internet]. Blockchair.com. 2022. Available from: https://blockchair.com/

[19] Grafana [Internet]. Grafana.com. 2022 . Available from: https://grafana.com/

[20] Knottenbelt W. Will's home page [Internet]. Doc.ic.ac.uk. 2022 . Available from: https://www.doc.ic.ac.uk/ wjk/

[21] FERNANDO J. What Is a Derivative? [Internet]. Investopedia. 2021 . Available from: https://www.investopedia.com/terms/d/derivative.asp

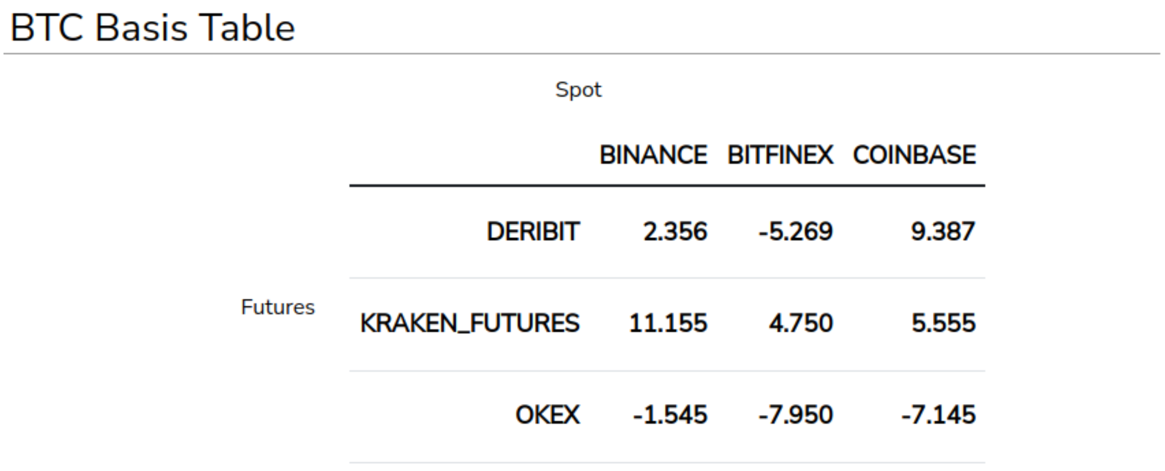
[22] FCA bans the sale of crypto-derivatives to retail consumers [Internet]. FCA. 2020. Available from: https://www.fca.org.uk/news/press-releases/fca-bans-sale-crypto-derivatives-retail-consumers

[23] Oliver J. Binance curbs UK trader access to risky crypto derivatives [Internet]. Ft.com. 2021 . Available from: https://www.ft.com/content/bbbd8ca5-1f13-45fd-9784-ace058dec37e

[24] Consent [Internet]. Ico.org.uk. 2021 . Available from: https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/lawful-basis-for-processing/consent

A Basis Table and Graph

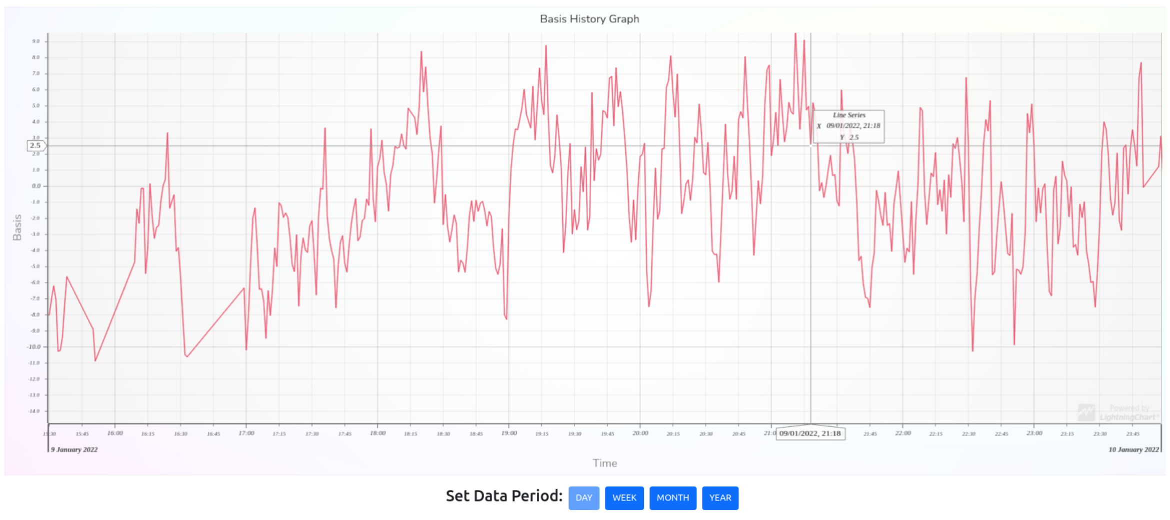
A.1 Bitcoin Basis Table for Spot-Futures Arbitrage



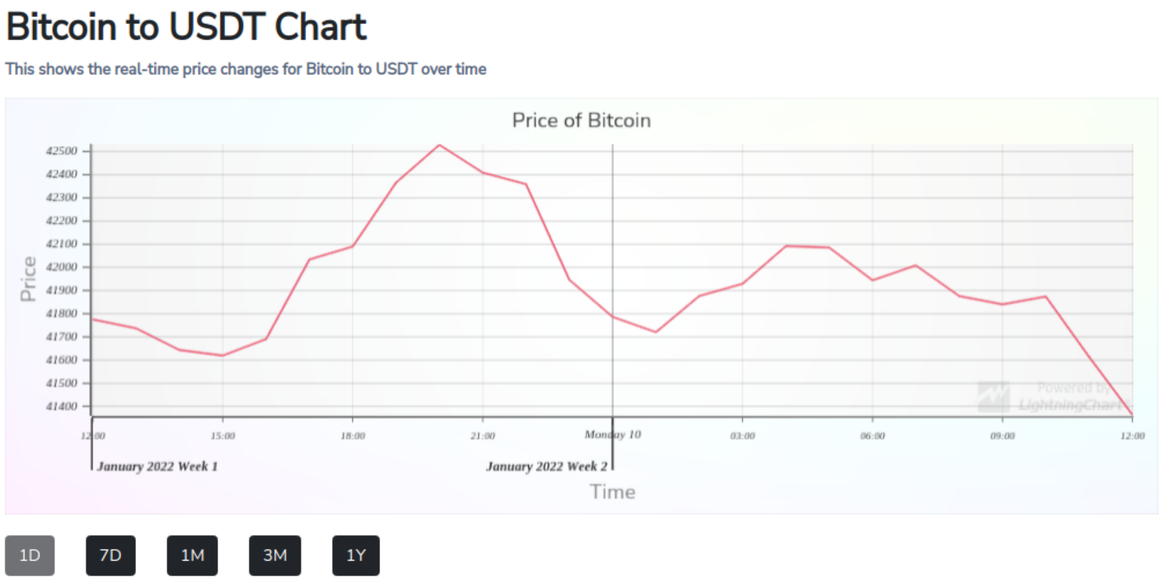
BTC Basis Table

		Spot		
		BINANCE	BITFINEX	COINBASE
Futures	DERIBIT	2.356	-5.269	9.387
	KRAKEN_FUTURES	11.155	4.750	5.555
	OKEX	-1.545	-7.950	-7.145

A.2 Bitcoin Basis History for DERIBIT-BINANCE

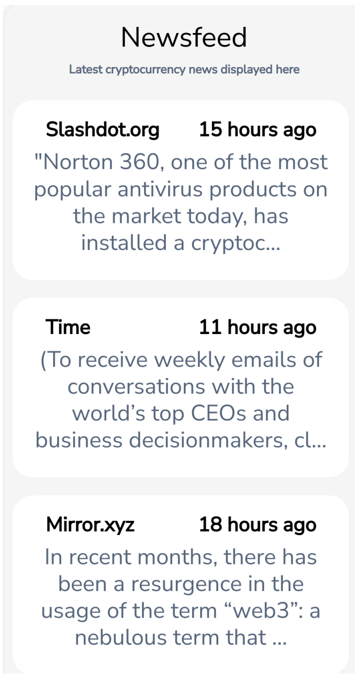


B Price History


Bitcoin to USDT Chart

This shows the real-time price changes for Bitcoin to USDT over time

C News Feed


Newsfeed

Latest cryptocurrency news displayed here

Slashdot.org 15 hours ago

"Norton 360, one of the most popular antivirus products on the market today, has installed a cryptoc...

Time 11 hours ago

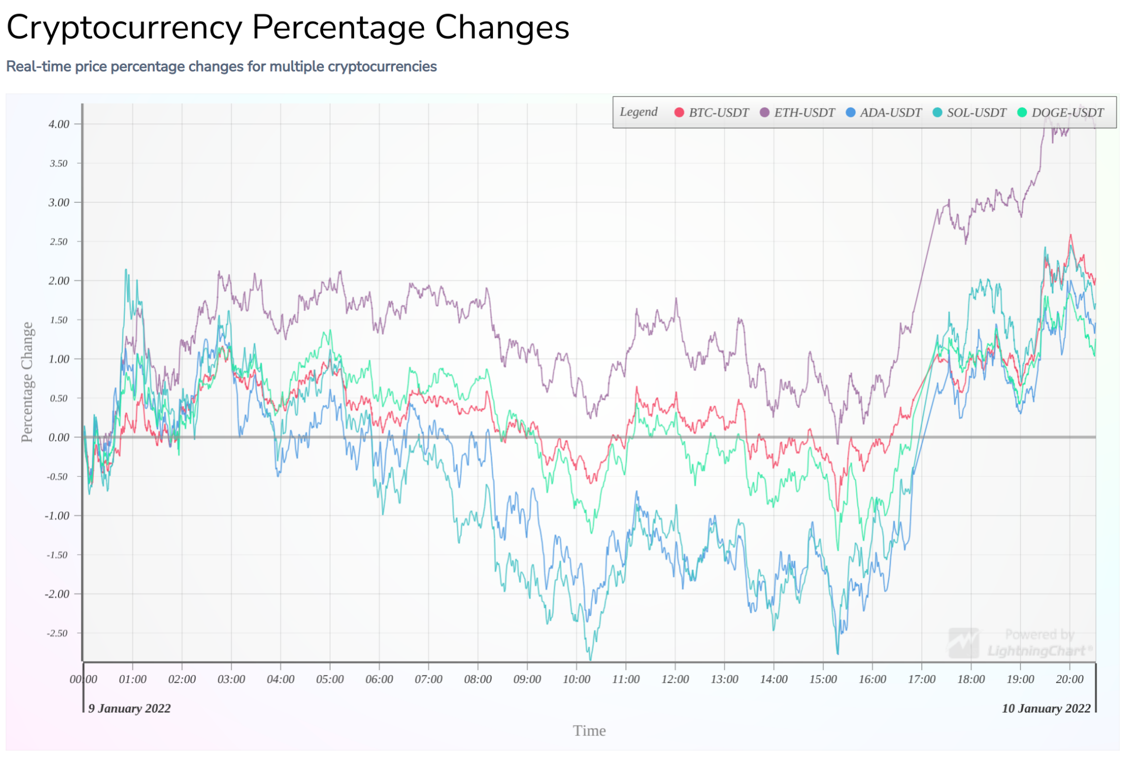
(To receive weekly emails of conversations with the world’s top CEOs and business decisionmakers, cl...

Mirror.xyz 18 hours ago

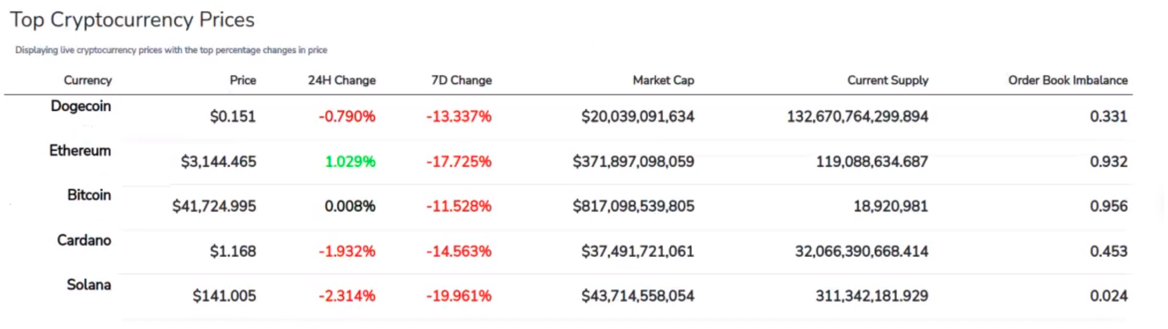
In recent months, there has been a resurgence in the usage of the term “web3”: a nebulous term that ...

D Top Currency Summaries

D.1 Percentage Change of Top Supported Currencies



D.2 Overview of Top Supported Currencies



Top Cryptocurrency Prices

Displaying live cryptocurrency prices with the top percentage changes in price

Currency	Price	24H Change	7D Change	Market Cap	Current Supply	Order Book Imbalance
Dogecoin	$0.151	-0.790%	-13.337%	$20,039,091,634	132,670,764,299.894	0.331
Ethereum	$3,144.465	1.029%	-17.725%	$371,897,098,059	119,088,634.687	0.932
Bitcoin	$41,724.995	0.008%	-11.528%	$817,098,539,805	18,920,981	0.956
Cardano	$1.168	-1.932%	-14.563%	$37,491,721,061	32,066,390,668.414	0.453
Solana	$141.005	-2.314%	-19.961%	$43,714,558,054	311,342,181.929	0.024