TED UNIVERSITY

CMPE 492 Final Report

CryptooFun

Team Members:

Kayra POLAT - 1000306178

Baturalp KIZILTAN - 4456996054

Emrecan ERBAY - 4221160055

Can ŞENGÜN - 1179712534

Supervisor:

Yücel ÇİMTAY

Jury Members:

Tolga Kurtuluş ÇAPIN

Emin KUĞU

Table of Contents

1 Introduction

The infrastructure and formation of cryptocurrencies goes back many years. In 1991, the concept of blockchain entered our world for the first time. Blockchain technology is a recording system technology where data can be stored without the need for any central administrator. In addition, blockchain technology has a highly resistant infrastructure against cyber-attacks. In short, it is a system that eliminates centralization, and everything is open. Blockchain is not just a concept used for cryptocurrencies. Many areas can be created that this infrastructure is compatible with. Today, this work continues. Blockchain is a database system that provides encrypted transaction tracking that is made up of blocks and stored in blocks.

In 2008, a person or community named Satoshi Nakamoto published an article called "Bitcoin: A Peer-to-Peer Electronic Cash System". This article became the basis of cryptocurrencies, which are always on our minds and have become a huge investment. The article detailed the blockchain infrastructure and eliminated the problem of authority (centralization) in between. When we look at the last quarter of 2022, we can clearly see what this article has changed.

Cryptocurrencies have become the most used investment tool in recent years. According to 2022 data, there are currently 300+ million users actively investing in cryptocurrencies. The global crypto market cap is $1.06 trillion as of August 1, 2022. It is in the hands of users to use this investment tool, which has a very bright future, correctly and safely. It is not right to invest in crypto money without understanding the risks and dynamics of the market. At this point, the CryptooFun application appears.

1.1 Purpose of the System

The aim of our project is to ensure that people make minimal losses from their future investments by practicing with our application before investing their money in cryptocurrency exchanges. We will also provide real stock market experience by receiving the data of cryptocurrencies live. Thus, the experiences that people will have, will be more suitable for real life. In addition, there are a lot of cryptocurrency training videos on the market. Users will be able to reinforce their knowledge through our application by applying the investment techniques they learned after watching the tutorial videos. The feature that distinguishes our project from other projects in the market is that we will enable people to compete while improving their knowledge with practice. Thus, we will encourage people to use our app. We will reveal the competitive environment by making various lobbies and publishing the overall balance in the statistics table. Users will be able to increase their balance with the gift starting balance we will give them when they sign up for the application. At the same time, an extra reward balance will be given with the profits they get from the game lobbies they enter. Thus, we will show the 100 people with the most balances in the statistics table.

Blockchain infrastructure will not be the infrastructure of the application we will develop. Our goal here is to create a real market experience for users **as much as possible**.

1.2 System Design Trade-offs

1.2.1 Speed vs Complexity

Since data needs to be received in real time, the speed of data processing and presentation is an important factor. In this case, technologies that provide faster data processing but are more complex can be chosen. However, simpler but slower technologies can offer an easier development process. Given these two aspects, it is more important to show live data in real time. Since users' ability to transact with real-time data is the cornerstone of the project, some of the complexity of the technology to provide and process the data can be sacrificed.

1.2.2 Ease of communication between microservices vs Performance

Using a microservice architecture in the project requires different functions to run on different services. This makes the project more scalable, but communication between services can become complex. It is inevitable that communication between services will affect performance. However, it is the developers' duty to minimize the performance loss. For this we will use the gRPC protocol, which is widely used for communication between microservices majorly due to its high performance.

For this part, it is worth going into detail about the microservice structure. Since each part of the application is embedded in a microservice, there are advantages and disadvantages to this approach. Advantages include the ability to scale each service separately, a problem in one service does not affect other services, and the division of labor within the development team. However, the disadvantages include the fact that each service can use different technologies, which can lead to integration problems, more management and monitoring, and debugging and troubleshooting difficulties.

1.2.3 Scalability vs Design Complexity

The aim of the project is to provide real-time cryptocurrency data, allowing users to make buy and sell transactions. However, if the project becomes popular, it may need to be scalable quickly. Therefore, the scalability of the project should be considered in the early stages of design. However, designing for scalability can increase the complexity of the project.

1.2.4 Flexibility vs Performance

If a problem occurs while using the Binance API, data will continue to be pulled from another cryptocurrency exchange. However, while this increases the flexibility of the application, it may reduce performance. If there is a problem with Binance servers, we will obtain data from another provider while minimizing the performance loss during the transition. However, it is very important that there is no data loss during the migration.

1.2.5 Security vs User Experience

The application allows users to make unreal buy and sell transactions. This can pose a significant security risk between users. Security can be ensured by correct authentication of users and data entry controls. However, these controls can negatively impact the user experience. We want to maximize both security and user experience by using the Identity Provider system for user authentication.

1.2.6 Buy-sell vs Real User Experience in Real Markets

The ability for users to trade with non-real balances allows the app to be used as an educational tool, giving them the opportunity to experience the risks associated with using real money. However, the disadvantages of this approach include the fact that real-time data cannot be manipulated, so the experience may be different from real markets, and users may gain less trading experience compared to trading with real money. The people who will use this app will certainly not be able to manipulate the markets. However, since our target audience is people with a small amount of money, people who do not know the cryptocurrency markets, and people who cannot manipulate even if they trade in real markets because they have a small amount of money, the issue of market manipulation is an issue that can be ignored.

2 Final Architecture

2.1 Overview

The overall software system will be built upon a set of distributed, event-driven, independently deployable, lightweight software services, which is known as microservices architecture. Moreover, synchronous inter-service communication pattern called gRPC were utilized in addition to event-driven communication styles.

A mix of event-driven architecture and request-reply pattern is the communication style that we followed. The mixture satisfies the business needs, because event-driven architecture allows to spread the heavy, long-running business processes over time while request-reply pattern is dedicated for solving synchronous business problems that require an answer immediately.

As a matter of course, each service brings its own executable, which is either a binary file or container image. We decided to fully adopt a container-based solution path for the raised issue via Docker images.

2.2 Project Hierarchy

1. **Web Application** (~/apps/web/)

   The web application is a Next.js project, written with React library in JavaScript. For styling purposes Tailwind CSS framework is utilized. The web app has its own self-contained executable/entry-point and can be deployable independently. Therefore, it's put under "apps/web" directory of the project's root. The app, which is part of the UI subsystem, serves as user-facing side of the project (front-end) and directly affects overall user experience.

2. **Protocol Buffer Definitions** (~/protobuf/)

   The *protobuf* directory includes Protocol Buffers based data structures and gRPC service definitions alongside with a YAML configuration for Buf CLI. The protocol helps to define binary-serializable, cross-platform, language-neutral data structures and services. By that way, we can benefit from multiple programming language interfaces with less hassle, because code-generation tools for Protocol Buffers automatically converts protocol definitions to POJOs and JSONs in our case. Then, we just override the auto-generated interfaces with language-specific implementations to enable gRPC services.

3. **Services** (~/services/)

   As it was described as one of the project's goals, the back-end services are heavily based on microservices architecture. Since we follow monorepo-oriented approach, all services lay under the same source code repository alongside with the web application. We have already mentioned that we might benefit from the use of multiple programming language ecosystems in the previous parts and reports. Thus, we decided to go with Java and Node.js ecosystems. As a part of the monorepo structure, each language/runtime environment has its own root directory. By that way, we can share modules, 3rd party library packages, and commercial off-the-shelf products within the same language environment. For example, common Java packages and Spring Framework dependencies are shared across by defining a Maven root module under the "~/services/java/cryptoofun" directory. Similar concept is also applied to Node.js services via Turborepo based NPM workspaces.

   A. Java (~/services/java/cryptoofun)

   - **genproto:** *genproto* module contains auto-generated Protocol Buffers and gRPC code specific to Java.
   - **messages:** Contains Kafka schemas for commands and events.
   - **Market Data Streaming:** An abstraction layer over external cryptocurrency APIs, e.g., Binance API. The service fetches historical data on demand and provides live data in regular intervals.
   - **Trade Butler:** A receiver for new user orders. Sends orders to the dedicated Kafka topic to be processed later by other service(s).
   - **Order Processing:** Consumes and processes trade orders.
   - **Progression:** Manages achievements, awards, and experience levels for users.

B. Node.js (~/services/nodejs/cryptoofun)

- **genproto:** *genproto* module contains auto-generated Protocol Buffers and gRPC code specific to Node.js (JavaScript and TypeScript).
- **shared:** Shared dependencies and functionalities for services, e.g., JWT middleware
- **Profile:** A service, which is dedicated to manage user profile information. Collabarates with the Identity Provider infrastructure.
- **Cash Wallet:** Manages cash ballance for users.
- **Portfolio:** Manages cryptocurrency holdings of users.
- **Lobby:** Manages creation of lobbies and lobby sessions.
- **Leaderboard:** Maintains leaderboard table(s).

2.3 Subsystem Services

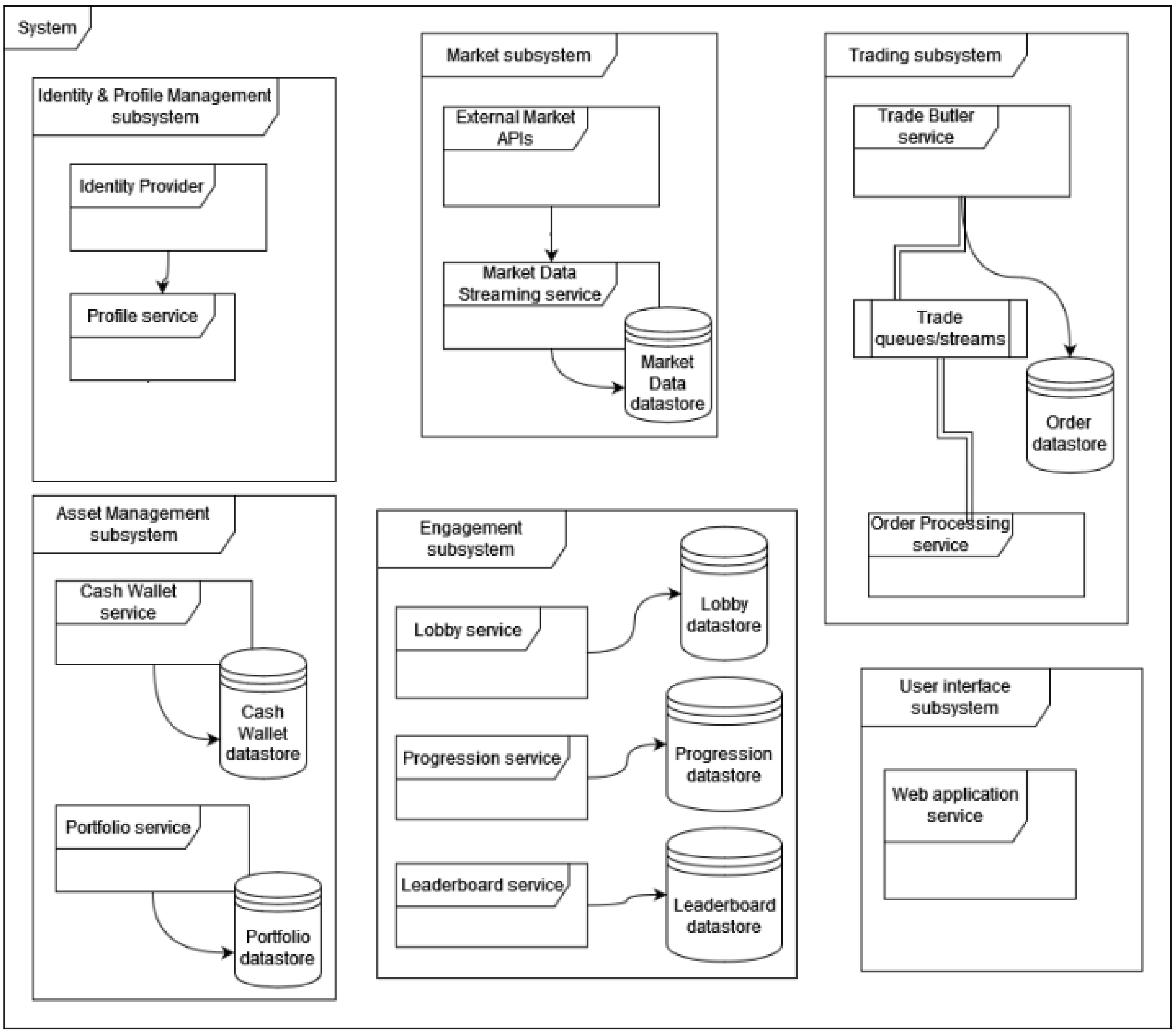


Figure 1: The subsystem map.

2.3.1 Identity & Profile Management subsystem

A subsystem to compose services and subservice elements that are related to management of user identities, accounts, permissions, and profiles together.

- **Identity Provider:** The service, which maintains and manages the identity and account data of users, has capabilities for ensuring cutting-edge security best practices and standards in terms of authentication and authorization.
- **Profile service**: The service that manages the profile specific data, which is differentiated from the identity and access management data, provided by the Identity Provider.

2.3.2 Market subsystem

A subsystem to compose services and subservice elements that are related to financial markets together.

- **External Market APIs:** The external API services that will provide market data for certain financial instruments, such as Binance API.
- **Market Data Streaming service:** The internal service, which will fetch the latest market data from the external market API.
- **Market Data datastore:** Redis

2.3.3 Trading Subsystem

A subsystem to compose services and subservice elements that are related to management of financial assets owned by users together.

- **Trade Butler service:** The service that takes orders and sends them to the order queue.
- **Trade Order datastore:** Apache Cassandra
- **Trade queues/streams:** Apache Kafka
- **Order Processing service:** The service that receives trade orders from the order queue and executes them according to the message properties such as timestamp, symbol, bid/ask price, buy/sell action, wallet, and portfolio references, etc. For proceeding with the execution, the service makes a call to the *Market Data Streaming service* to retrieve the data at the time of execution.

2.3.4 Asset Management subsystem

A subsystem to compose services and subservice elements that are related to management of financial assets together.

- **Cash Wallet service:** As a service, it stores and manages the total wallet balance of a user in the form of cash – not the cryptocurrencies, tokens, or any other securities.
- **Cash Wallet datastore:** CockroachDB
- **Portfolio service:** As a service, it stores and manages the assets which are held by a user.
- **Portfolio datastore:** CockroachDB

2.3.5 Engagement subsystem

A subsystem to compose services and subservice elements that are related to user engagement and user retention together.

- **Lobby service:** The services that will be responsible for storing, creating, removing, and managing lobbies and the related data.
- **Lobby datastores:** CockroachDB for persistency, Redis for lobby processing.
- **Progression service:** The service that will be responsible for managing and storing level/experience progression of each user.
- **Progression datastore:** Apache Cassandra.
- **Leaderboard service:** The service that will maintain leaderboard table in regular interval (every 1 minute) by composing the data received from cash wallet service.
- **Leaderboard datastore:** Redis

2.3.6 User interface subsystem

A subsystem to compose services and subservice elements that are related to user interface together.

- **Web application service:** The backbone service of the user interface subsystem that covers user facing visual interface elements and client interactions for all overall system.

2.4 Data Management

Data plays a crucial role because of the data-driven nature of our system. Therefore, we must utilize meticulously picked solutions for leveraging the data infrastructure. Since the components of a finance application can face with intensive amount of load throughout a day, there may emerge some problems in terms of scalability, availability, speed and consistency in the long run. Besides these non-functional requirements, the possible use cases of datastores must be addressed, and the solutions should be picked accordingly. There are two types of databases in our system: **SQL and NoSQL**. We utilized **CockroachDB** as an SQL solution. On the other hand, **Apache Cassandra** is utilized as NoSQL persistent solution. Furthermore, in-memory databases such as **Redis** is used for data caching and low-latency messaging purposes, however the aforementioned kind of data is not subject to persistency.

2.5 Access control and security

Primarily the Identity Provider (IdP) will be responsible for access control and security, because the chosen IdP will have a set of business capabilities for managing the user identities and users' permissions. OAuth2 security framework is the concept that we are planning to stick with throughout the project since it is based on well-defined international computer security standards. We chose Auth0 as primary IdP solution. The user requests that flow through the restricted internal services, will be authorized by calling the IdP via its SDK. Furthermore, we are willing to stick with an off-the-shelf IdP software instead of building our very own, because these kinds of security products require best-in-class audits by security professionals. Lastly, our service mesh solution in Kubernetes, which is called as Istio, checks access tokens according to our authorization policies in parallel with Auth0.

2.6 Infrastructure and Deployment

For infrastructure & deployment needs, we made use of Kubernetes and Google Cloud Platform.

Kubernetes eases the orchestration and provisioning processes of the CryptooFun services. Otherwise, we could have needed to maintain networking, storage and computing requirements of each microservice independently. In a way, we can say that Kubernetes provides us essential building blocks for CryptooFun's infrastructure with great virtualized and encapsulated abstractions.

Furthermore, Google Cloud Platform (GCP) fulfills the need for large-scale deployments. By nature, CryptooFun is computationally heavy to be able to run on local environments such as laptops and mid-tier desktop devices. Thus, we decided to host whole CryptooFun Kubernetes cluster on a public cloud, i.e., GCP.

Additionally, CryptooFun is dependent to 4 different data platforms: CockroachDB, Apache Cassandra or Cassandra derivatives (ScyllaDB, AstraDB), Apache Kafka, and Redis. In the latest deployment plan, we include only Redis into the CryptooFun Kubernetes cluster as a part of our self-hosting processes. On the other hand, other data platforms are powered by external service providers as the following:

- **Apache Kafka** is powered by Instaclustr [https://www.instaclustr.com/platform/managed-apache-kafka/]
- **Apache Cassandra** is powered by AstraDB [https://www.datastax.com/products/datastax-astra]
- **CockroachDB** is powered by CockroachDB Cloud [https://www.cockroachlabs.com/docs/cockroachcloud/]

However, all the aforementioned data platforms are open-source, and they allow for self-hosting freely. Therefore, it is also possible to include them into the CryptooFun Kubernetes cluster deployments with more compute and storage power, in other words with much more financial cost.

2.7 Used Technologies and Tools

One of the best features of microservice architecture is that it does not depend on a fixed technology. The technologies we use to build our project are in line with today's technologies. Microservices are easier to develop, test, and deploy than monolithic applications. This makes it possible to release new features and bug fixes more frequently. While explaining the technologies used, we will distinguish between front-end, back-end, deployment, and data storages.

2.7.1 Front-end

→ Tailwind: It is a utility-first CSS framework that provides a way to add styles to your web pages without writing any custom CSS. It does this by providing a set of pre-defined classes that you can use to control the layout, typography, colors, and other aspects of your design.

→ React: React is an open-source JavaScript library for building user interfaces. It is one of the most popular JavaScript libraries in the world, and it is used by many large companies, such as Facebook, Netflix, and Airbnb. React is based on the concept of **components**. A component is a reusable piece of code that can be used to create different parts of a user interface. Components are declarative, which means that they describe what the UI should look like, rather than how it should be built. This makes React code more concise and easier to read.

→ Next.js: Next.js is an open-source React framework that enables you to build server-rendered and static web applications. It is built on top of React and Node.js, and it provides a number of features that can help you build better web applications, including Server-side rendering, static site generation, code splitting, bundler, hot reloading, dev server and CLI.

2.7.2 Back-end

→ Express.js: Express.js, also known as Express, is a popular and minimalistic web application framework for Node.js. It provides a robust set of features and utilities for building web applications and APIs. Express.js is designed to be simple and flexible, allowing developers to create web applications quickly and efficiently.

→ Spring Boot: Spring Boot is an open-source framework that simplifies the development of Java applications by providing a streamlined and opinionated approach to building production-ready applications. It is built on top of the popular Spring Framework and aims to minimize the configuration required for setting up Spring applications. Also, it makes it easier to use Java-based frameworks to create microservices.

→ Auth0: Auth0 is an Identity-as-a-Service (IDaaS) platform that provides authentication and authorization solutions for developers and organizations. It offers a comprehensive set of tools and services to handle user authentication and authorization in web and mobile applications. Auth0 simplifies the implementation of user authentication by providing a secure and scalable authentication infrastructure as a cloud-based service. It abstracts the complexities of handling user identity, including features such as user registration, social login, multi-factor authentication (MFA), single sign-on (SSO), and passwordless authentication.

→ gRPC: gRPC (Google Remote Procedure Call) is an open-source framework developed by Google that facilitates communication between client and server applications. It is built on top of the HTTP/2 protocol and uses Protocol Buffers (protobuf) as the interface definition language (IDL) for describing services and message types. It enables efficient and high-performance communication by using a binary serialization format (Protocol Buffers) and leveraging the benefits of HTTP/2. It supports various programming languages, including Java, Python, Go, C++, Ruby, and more, allowing developers to build interoperable and language-agnostic systems.

→ WebSocket: A WebSocket is a persistent bidirectional communication channel that enables two-way communication between a client and a server. It is a TCP-based protocol that is implemented in web browsers and servers. WebSocket's are used for a variety of applications, including real-time chat, gaming, and streaming media.

→ Prisma: Prisma is an open-source database toolkit and Object-Relational Mapping (ORM) tool that simplifies database access and management for modern application development. It provides a type-safe and intuitive API for interacting with databases, allowing developers to focus more on building application logic rather than dealing with low-level database operations.

2.7.3 Deployment

→ Google Cloud Console: The Google Cloud Console is a web-based graphical user interface (GUI) provided by Google Cloud Platform (GCP) for managing and administering various services and resources on the Google Cloud. It serves as a central hub for developers, administrators, and users to interact with and manage their cloud resources.

→ Kubernetes: Kubernetes is an open-source container orchestration platform developed by Google. It provides a robust and scalable framework for automating the deployment, scaling, and management of containerized applications. Kubernetes is designed to handle the complexities of running distributed systems and helps ensure application availability, scalability, and resilience.

→ Istio: Istio is an open-source service mesh platform that provides a comprehensive solution for managing and securing microservices-based applications. It aims to address the challenges associated with communication, observability, and security in distributed systems by providing a dedicated infrastructure layer for managing service-to-service communication.

→ Docker: Docker is an open-source platform that simplifies the process of building, packaging, and deploying applications using containerization. It provides a lightweight and isolated environment, called a container, in which applications and their dependencies can run consistently across different environments.

→ Apache Kafka: Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.

2.7.4 Data Storages

→ CockroachDB: CockroachDB is an open-source distributed SQL database system that is designed to deliver high availability, scalability, and strong consistency across multiple nodes and clusters. It is inspired by Google's Spanner database and aims to provide a global scale, distributed SQL database that can handle large volumes of data and high transactional workloads.

→ Apache Cassandra: Apache Cassandra is an open-source distributed NoSQL database designed to handle massive amounts of data across multiple commodity servers while providing high availability and scalability. It was originally developed by Facebook and later open-sourced and donated to the Apache Software Foundation.

→ Redis: Redis is an open-source, in-memory data structure store that is used as a database, cache, and message broker. It is often referred to as a "data structure server" because it allows the storage and retrieval of data structures like strings, lists, sets, hashes, and more. Redis is designed for high performance and is known for its simplicity, versatility, and low-latency data access.

3 Engineering Solutions

3.1 Global Context

3.1.1 Financial Literacy and Risk Mitigation

- By allowing users to practice and gain experience in a simulated environment before investing real money, the project can contribute to increasing financial literacy and reducing potential losses for individuals globally.
- Greater knowledge and understanding of trading techniques through external instructional videos can be practiced through our app, enabling users to make more informed decisions and potentially minimize the risks associated with cryptocurrency investments.

3.1.2 Access and Inclusion

- By providing a user-friendly web application, the project aims to make cryptocurrency investment practice accessible to a wide range of people globally, regardless of their geographic location or socioeconomic background.
- The availability of the application globally enables individuals from different countries to learn and engage with cryptocurrency investments, fostering inclusivity in the digital financial ecosystem.

3.1.3 Competition and Engagement

- The competitive environment created through various lobbies and the publication of overall balances can motivate users globally to actively participate in the application, increasing engagement and interest in cryptocurrency investments.
- The ranking of the top 100 users with the highest balances in the statistics table creates a sense of achievement and recognition, potentially driving increased usage and competition on a global scale.

3.1.4 Knowledge Sharing and Community Building

- By providing a platform for users to reinforce their knowledge through applying investment techniques learned from tutorial videos, the project encourages knowledge sharing and collaboration among users globally.
- The project's emphasis on community engagement and competition fosters a sense of belonging and encourages users to interact, share strategies, and learn from each other, potentially building a global community of cryptocurrency enthusiasts.

3.1.5 Market Influence

- The project's real-time data integration from cryptocurrency exchanges and the focus on providing a real stock market experience can impact the cryptocurrency market by increasing user engagement, liquidity, and potentially influencing market sentiment.
- As more users gain experience and knowledge through the application, their investment decisions and behaviors may have broader implications for cryptocurrency markets on a global scale.

3.2 Environmental Context

3.2.1 Energy Consumption

The real-time data integration and live cryptocurrency market experience require continuous data retrieval and processing, which can result in increased energy consumption. Depending on the infrastructure and servers used, this could potentially have an environmental impact, particularly if the energy sources powering the servers are not renewable or efficient.

3.2.2 Server Infrastructure

The project's microservices architecture and the need for real-time data may require a robust server infrastructure, including data centers. The construction and maintenance of such infrastructure can contribute to resource consumption, land use, and potentially generate electronic waste if not managed properly.

3.2.3 Data Transfer and Storage

Continuous retrieval and storage of live cryptocurrency data, as well as user data related to transactions and balances, require significant storage and data transfer capabilities. This can result in increased energy consumption and carbon emissions, particularly if the data is hosted on servers located in different geographic regions, requiring long-distance data transfer.

3.2.4 Hardware Impact

The project's requirements for processing real-time data and providing a competitive environment may lead to increased demands for computing hardware, such as servers and personal devices. The production, use, and disposal of such hardware can contribute to environmental impacts, including resource extraction and electronic waste generation.

3.2.5 Sustainable Practices

Incorporating sustainable practices in the project's development and deployment, such as optimizing code for energy efficiency, implementing server virtualization, utilizing renewable energy sources for server infrastructure, and promoting responsible e-waste management, could help mitigate potential negative environmental impacts.

3.3 Societal Context

3.3.1 Increased Financial Literacy

According to a recent study by the World Bank, only 71% of adults worldwide have a basic understanding of financial concepts. This lack of financial literacy can lead to poor investment decisions, which can have a negative impact on people's financial well-being. By providing a safe and realistic environment for people to practice investing in cryptocurrencies, this project could help to increase financial literacy among the general public. This could lead to more informed investment decisions, which could benefit both individuals and the economy as a whole.

3.3.2 Reduced Economic Inequality

The cryptocurrency market is still in its early stages of development, which means that there is a lot of potential for people to make a lot of money. However, the market is also very volatile, which means that there is also a lot of potential for people to lose money. This project could help to reduce economic inequality by providing a platform for people to practice investing and learn how to manage their risk. This could help to level the playing field for people from all walks of life and could give people from underserved communities a chance to build wealth.

3.3.3 Increased Economic Growth

The cryptocurrency market is a new and growing industry, which means that there is a lot of potential for economic growth. This project could help to boost economic growth by providing a platform for people to invest in cryptocurrencies and by helping to increase financial literacy among the public. This could lead to increased investment in the cryptocurrency market, which could lead to new businesses being created and jobs being created.

3.3.4 Improved Financial Inclusion

The cryptocurrency market is a global market, which means that it has the potential to improve financial inclusion around the world. By providing a platform for people to invest in cryptocurrencies, this project could help to make financial services more accessible to people in developing countries and other underserved communities. This could help to improve the lives of millions of people around the world.

4 Contemporary Issues

1. **Market Manipulation and Insider Trading:** The cryptocurrency market has faced concerns about market manipulation and insider trading. Coordinated efforts to manipulate prices, spread false information, or exploit knowledge asymmetry can lead to significant losses for investors. It is important for users to understand these risks and be cautious of suspicious activities while trading.
2. **Regulatory Challenges and Compliance:** The regulatory landscape surrounding cryptocurrencies is still evolving, with different countries and jurisdictions implementing various regulations. Compliance with legal requirements, such as Know Your Customer (KYC) and Anti-Money Laundering (AML) regulations, can be complex for cryptocurrency exchanges and trading platforms. Ensuring that your application adheres to relevant regulations and promotes compliance among users is crucial for long-term success.
3. **Security and Hacking Risks:** The security of cryptocurrency exchanges, wallets, and trading platforms remains a critical issue. Hacking incidents, phishing attacks, and thefts have resulted in substantial financial losses for users. Implementing robust security measures, such as encryption, multi-factor authentication, and regular security audits, is essential to protect user funds and personal information.

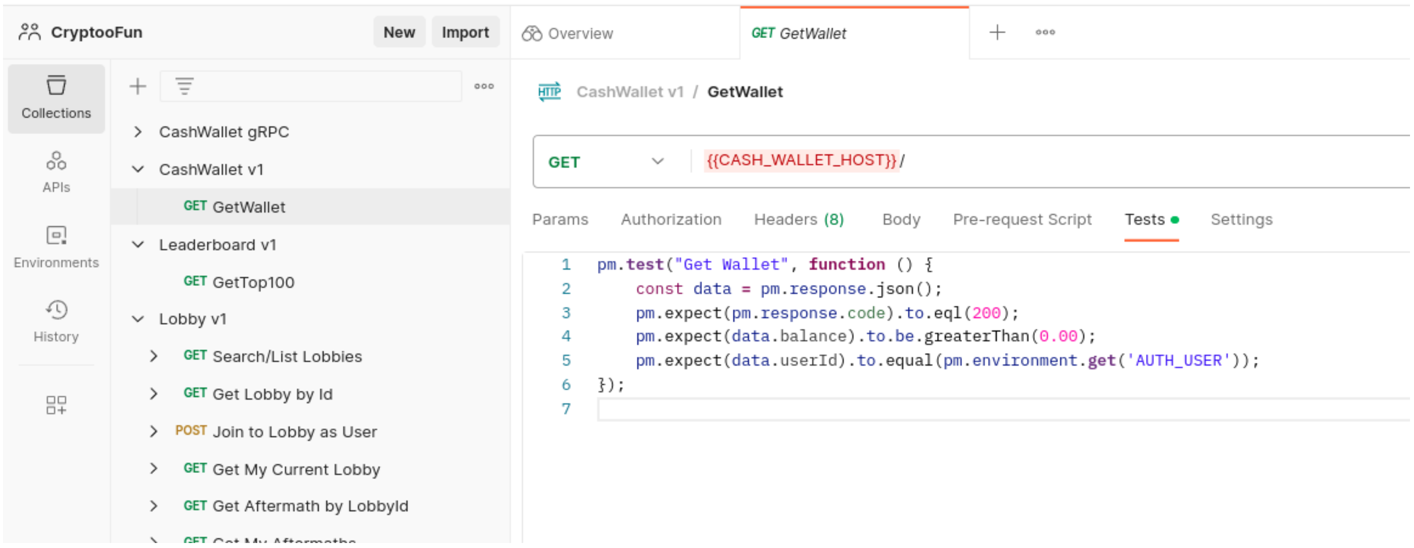
4. **Market Volatility and Risk Management:** Cryptocurrencies are known for their price volatility, which can lead to significant gains or losses in short periods. Managing risk and employing effective risk management strategies are important skills for cryptocurrency traders. Educating users about risk management techniques, setting realistic expectations, and emphasizing the importance of diversification can help mitigate potential losses.
5. **Lack of Investor Protection:** Unlike traditional financial markets, cryptocurrencies often lack the same level of investor protection and recourse mechanisms. In cases of fraud, hacking, or exchange failures, recovering lost funds can be challenging. Educating users about the risks and encouraging them to use reputable exchanges and wallets with robust security measures can help minimize potential losses.
6. **Emotional Decision-Making and Speculative Behavior:** The cryptocurrency market can be highly emotional and driven by speculative behavior. Users may be influenced by fear of missing out (FOMO), social media hype, or panic selling during market downturns. Promoting rational decision-making, providing educational resources on investment psychology, and encouraging a long-term investment mindset can help users avoid impulsive and emotional trading decisions.
7. **Environmental Impact:** Cryptocurrency mining, especially for proof-of-work cryptocurrencies, requires significant computational power and energy consumption. The carbon footprint of cryptocurrencies has raised concerns about their environmental impact. Encouraging users to consider sustainable alternatives, such as proof-of-stake cryptocurrencies or supporting eco-friendly initiatives within the industry, can help address these environmental challenges.

By addressing these contemporary issues through educational resources, security measures, risk management strategies, and fostering responsible trading practices, this project can contribute to a safer and more informed cryptocurrency trading environment for users.

5 Testing Results

5.1 Integration Testing

All integration tests are implemented and successfully completed in Postman according to the Test Plan Report. The code that has been written for testing the HTTP GET endpoint of Cash Wallet service, is displayed in Figure-2. All other integration tests have similar structure too. The Postman collection can be downloaded from the CryptooFun's git repository. [https://github.com/CryptooFun/cryptoofun/tree/main/postman/collections].



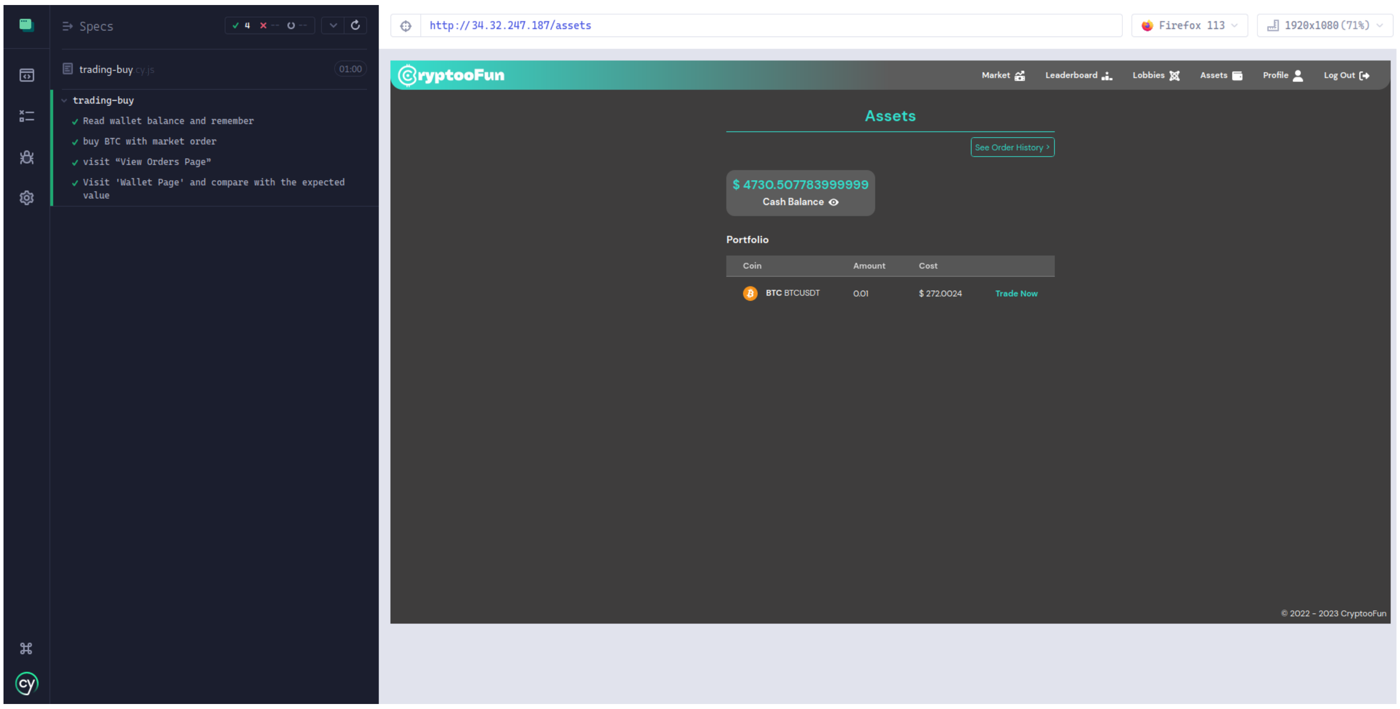
```
pm.test("Get Wallet", function () {
    const data = pm.response.json();
    pm.expect(pm.response.code).to.eql(200);
    pm.expect(data.balance).to.be.greaterThan(0.00);
    pm.expect(data.userId).to.equal(pm.environment.get('AUTH_USER'));
});
```

Figure 2

5.2 End-to-End Testing

The end-to-end (E2E) tests can be found under the web app module's Cypress test directory: "<project-root>/apps/web/cypress/e2e/". The tests are successfully executed with Cypress test framework as they were described in the Test Plan Report. An example run for "Trade (buy) Bitcoin test" is shown in Figure-3.

Note: Tests were run against the live web app instance, which was deployed on Google Cloud, with other dependent live services. Auth0 and Cypress configurations were tweaked accordingly.



```
trading-buy.cy.js                                   01:00
trading-buy
  ✓ Read wallet balance and remember
  ✓ buy BTC with market order
  ✓ visit "View Orders Page"
  ✓ Visit 'Wallet Page' and compare with the expected
    value
```

http://34.32.247.187/assets

Assets

$ 4730.507783999999
Cash Balance

Portfolio

Coin	Amount	Cost	
BTC BTCUSDT	0.01	$ 272.0024	Trade Now

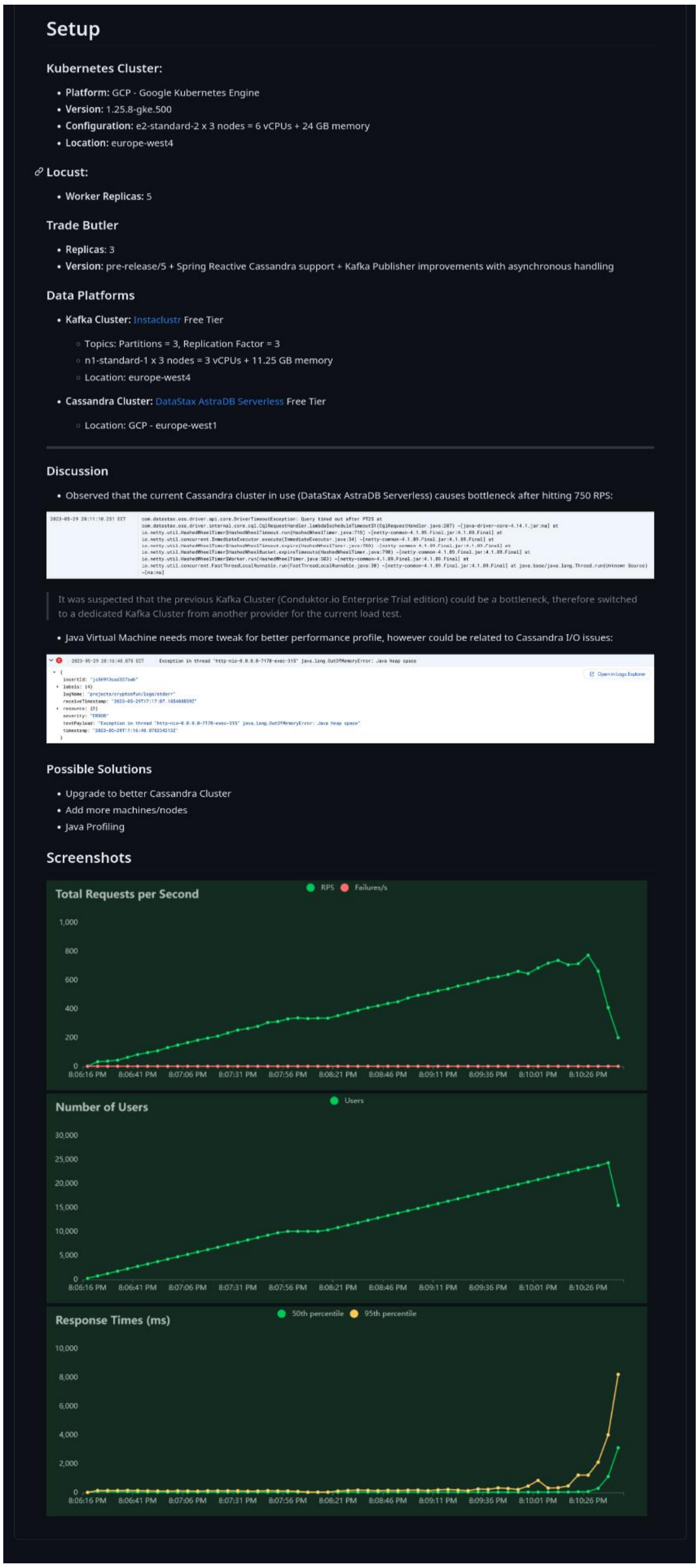
Figure 3

5.3 Load Testing

Load tests were carried out successfully according to the Test Plan Report as well. The test results helped us to identify specific issues that may happen when the services are under heavy load. Details about test setups and results were discussed in Figure-4 and Figure-5.

Note: Graphs were created via Locust load testing framework automatically during testing:

- **Trading (create market order) functionality:**



Setup

Kubernetes Cluster:

- **Platform:** GCP - Google Kubernetes Engine
- **Version:** 1.25.8-gke.500
- **Configuration:** e2-standard-2 x 3 nodes = 6 vCPUs + 24 GB memory
- **Location:** europe-west4

Locust:

- **Worker Replicas:** 5

Trade Butler

- **Replicas:** 3
- **Version:** pre-release/5 + Spring Reactive Cassandra support + Kafka Publisher improvements with asynchronous handling

Data Platforms

- **Kafka Cluster:** Instaclustr Free Tier
  - Topics: Partitions = 3, Replication Factor = 3
  - n1-standard-1 x 3 nodes = 3 vCPUs + 11.25 GB memory
  - Location: europe-west4
- **Cassandra Cluster:** DataStax AstraDB Serverless Free Tier
  - Location: GCP - europe-west1

Discussion

- Observed that the current Cassandra cluster in use (DataStax AstraDB Serverless) causes bottleneck after hitting 750 RPS:

```
2023-05-29 20:11:10.251 EET    com.datastax.oss.driver.api.core.DriverTimeoutException: Query timed out after PT2S at
                               com.datastax.oss.driver.internal.core.cql.CqlRequestHandler.lambda$scheduleTimeout$1(CqlRequestHandler.java:207) ~[java-driver-core-4.14.1.jar:na] at
                               io.netty.util.HashedWheelTimer$HashedWheelTimeout.run(HashedWheelTimer.java:715) ~[netty-common-4.1.89.Final.jar:4.1.89.Final] at
                               io.netty.util.concurrent.ImmediateExecutor.execute(ImmediateExecutor.java:34) ~[netty-common-4.1.89.Final.jar:4.1.89.Final] at
                               io.netty.util.HashedWheelTimer$HashedWheelTimeout.expire(HashedWheelTimer.java:703) ~[netty-common-4.1.89.Final.jar:4.1.89.Final] at
                               io.netty.util.HashedWheelTimer$HashedWheelBucket.expireTimeouts(HashedWheelTimer.java:790) ~[netty-common-4.1.89.Final.jar:4.1.89.Final] at
                               io.netty.util.HashedWheelTimer$Worker.run(HashedWheelTimer.java:503) ~[netty-common-4.1.89.Final.jar:4.1.89.Final] at
                               io.netty.util.concurrent.FastThreadLocalRunnable.run(FastThreadLocalRunnable.java:30) ~[netty-common-4.1.89.Final.jar:4.1.89.Final] at java.base/java.lang.Thread.run(Unknown Source)
                               ~[na:na]
```

> It was suspected that the previous Kafka Cluster (Conduktor.io Enterprise Trial edition) could be a bottleneck, therefore switched to a dedicated Kafka Cluster from another provider for the current load test.

- Java Virtual Machine needs more tweak for better performance profile, however could be related to Cassandra I/O issues:

```
2023-05-29 20:16:48.878 EET    Exception in thread "http-nio-0.0.0.0-7170-exec-315" java.lang.OutOfMemoryError: Java heap space
{
  insertId: "js56913oad327aeb"
  labels: {4}
  logName: "projects/cryptoofun/logs/stderr"
  receiveTimestamp: "2023-05-29T17:17:07.105408059Z"
  resource: {2}
  severity: "ERROR"
  textPayload: "Exception in thread "http-nio-0.0.0.0-7170-exec-315" java.lang.OutOfMemoryError: Java heap space"
  timestamp: "2023-05-29T17:16:48.878254213Z"
}
```

Possible Solutions

- Upgrade to better Cassandra Cluster
- Add more machines/nodes
- Java Profiling

Screenshots

Figure 4

- **Lobby Search Functionality**

Setup

Kubernetes Cluster:

- **Platform:** GCP - Google Kubernetes Engine
- **Version:** 1.25.8-gke.500
- **Configuration:** n1-standard-2 x 3 nodes = 6 vCPUs + 22.5 GB memory
- **Location:** europe-west4

Locust:

- **Worker Replicas:** 5

Lobby Service

- **Replicas:** 3
- **Version:** pre-release/7

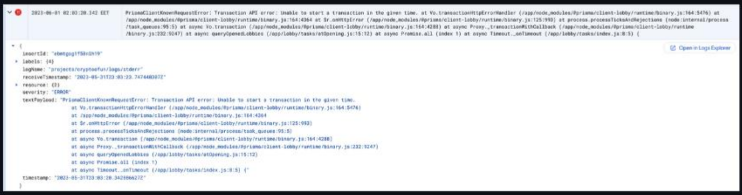
Data Platforms

- **CockroachDB Cluster:** Cloud Serverless Free Tier
  - Location: GCP - europe-west1
- **Redis Cluster:** Self-hosted within the same K8S cluster (1 master + 3 replicas via Helm)

> ℹ️ Redis does not have a significant impact on this test case!

Discussion

Lobby Service has been able to hit only up to ~350 RPS with 3 instances. The unexpected spike of high response times (ms) happened after Prisma ended up with multiple transaction errors like that:



```
2023-06-01 02:03:20.342 EET    PrismaClientKnownRequestError: Transaction API error: Unable to start a transaction in the given time. at Vo.transactionHttpErrorHandler (/app/node_modules/@prisma/client-lobby/runtime/binary.js:164:5476) at
                               /app/node_modules/@prisma/client-lobby/runtime/binary.js:164:4364 at $r.onHttpError (/app/node_modules/@prisma/client-lobby/runtime/binary.js:125:993) at process.processTicksAndRejections (node:internal/process
                               /task_queues:95:5) at async Vo.transaction (/app/node_modules/@prisma/client-lobby/runtime/binary.js:164:4288) at async Proxy._transactionWithCallback (/app/node_modules/@prisma/client-lobby/runtime
                               /binary.js:232:9247) at async queryOpenedLobbies (/app/lobby/tasks/atOpening.js:15:12) at async Promise.all (index 1) at async Timeout._onTimeout (/app/lobby/tasks/index.js:8:5) {

{
  insertId: "ebmtgog1f50n1h19"
  labels: {4}
  logName: "projects/cryptoefun/logs/stderr"
  receiveTimestamp: "2023-05-31T23:03:23.747448397Z"
  resource: {2}
  severity: "ERROR"
  textPayload: "PrismaClientKnownRequestError: Transaction API error: Unable to start a transaction in the given time.
                    at Vo.transactionHttpErrorHandler (/app/node_modules/@prisma/client-lobby/runtime/binary.js:164:5476)
                    at /app/node_modules/@prisma/client-lobby/runtime/binary.js:164:4364
                    at $r.onHttpError (/app/node_modules/@prisma/client-lobby/runtime/binary.js:125:993)
                    at process.processTicksAndRejections (node:internal/process/task_queues:95:5)
                    at async Vo.transaction (/app/node_modules/@prisma/client-lobby/runtime/binary.js:164:4288)
                    at async Proxy._transactionWithCallback (/app/node_modules/@prisma/client-lobby/runtime/binary.js:232:9247)
                    at async queryOpenedLobbies (/app/lobby/tasks/atOpening.js:15:12)
                    at async Promise.all (index 1)
                    at async Timeout._onTimeout (/app/lobby/tasks/index.js:8:5) {"
  timestamp: "2023-05-31T23:03:20.342866627Z"
}
```

Open in Logs Explorer

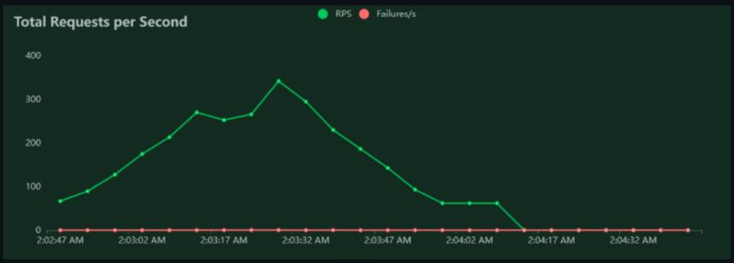
We suspect that either Prisma's engine is unable to handle too many open transactions under heavy load (see a similar issue while benchmarking) or the serverless CockroachDB cluster is incapable for high load and causes transaction timeouts.

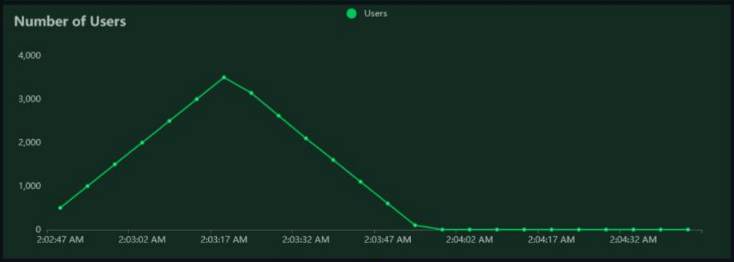
However, the aforementioned transaction issues were not directly related with the search endpoint, therefore no HTTP GET query failed for end users.

Possible Solutions

- Switch to another ORM (and/or write raw SQL queries) as an alternative to Prisma
- Upgrade to a better CockroachDB cluster
- Add more machines/nodes

Screenshots





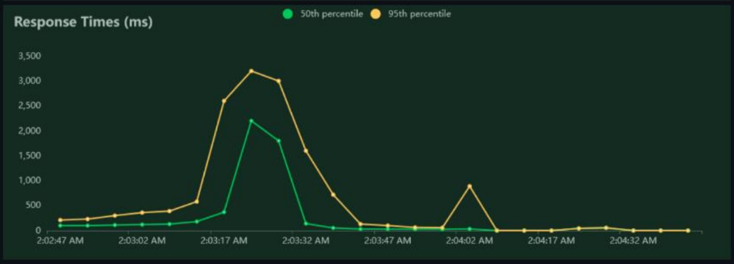


Figure 5

6 References

- Links have been added to the names of all technologies in the technologies and tools section. When clicked, the relevant resources can be accessed.
- Websocket Market Streams – Binance API Documentation (binance-docs.github.io)
- Fernandez, T. (2022, July 15). What is monorepo? (and should you use it?). Semaphore. Retrieved December 25, 2022, from https://semaphoreci.com/blog/what-is-monorepo
- Microsoft Docs Contributors. (2022). Interservice Communication in microservices - azure architecture center. Interservice communication in microservices - Azure Architecture Center | Microsoft Learn. Retrieved December 25, 2022, from https://learn.microsoft.com/en-us/azure/architecture/microservices/design/interservice-communication
- Amazon Docs Contributors. (2022). WebSphere Business Integration Pub/Sub Solutions. Amazon. Retrieved December 25, 2022, from https://aws.amazon.com/pub-sub-messaging/