TED UNIVERSITY

CMPE 491 High Level Design 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

In this report, the purpose of the application and the design objectives are explained. Also, Proposed software architecture and its subtitles and Subsystem services are explained in detail.

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.

CF

CryptooFun

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 leagues 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, as well as their actual virtual balance with daily login rewards. 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 20 people with the most balances in the statistics table. In addition, people will be able to see how they rank in their profiles.

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 Design goals

In this document, where technical details about the project are given, it would not be right to be limited to these only. Along with the technical details, the design targets of the system should also be determined. The following are related to what requirements the system should have, namely our design goals.

1.2.1 Performance

The system must have a very fast response time. It is not a problem if there will be a slight slowdown in the system when too much work is overloaded on the system. However, how late users get their work done reduces the reliability of the application proportionally. There should be no loss of speed during both the registration and login processes. All procedures performed by users should be carried out as quickly as possible. The system must be built on an efficient database. The system should not make mistakes while using real-time data and should be able to reduce the possible delay to the shortest level. The increase or decrease in the rate of increase or decrease of cryptocurrencies should not affect the system badly. Because users need to trade in a high-performance environment.

1.2.2 Reliability

Real-time cryptocurrency data transfer should work 24/7. In this way, users should be able to trade with their virtual balances as if they were real. The failure rate of the system should be at very low levels. Errors that may occur independently of the user should be minimized.

1.2.3 Security

Data security is one of the most important goals of the application. While keeping the private information of the users in the database, it is most important that the access to them is prevented by external people. Although users have virtual balances, the system must ensure the security of these balances. If different users access each other's virtual balances, the application is useless. Both the general security of the system and the security of the cryptocurrencies that users have virtually must be at a high level.

1.2.4 Usability

The application interface should be very comfortable for users. Since our target audience will be people who have never traded, users should not get lost in the application. For each module, there should be a brief explanation at the first stage. In this way, users can easily understand each module when they encounter it. The application interface should be in an attractive design for the users. Interface design is a very important factor for usability.

1.2.5 Privacy

Although it has been a virtual environment, it must exist with a high level of privacy, just like a real cryptocurrency exchange. Due to the law on the protection of personal data, users' password data shall be stored in an encrypted manner. The system shall ensure the confidentiality of the data as a priority.

1.2.6 Durability

The system must be prepared for overload. It is very important to provide durability in cases where more than one user is operating. At the same time, the transmission of real-time cryptocurrency data via Binance or other exchanges must be based on a solid infrastructure.

1.2.7 Supportability

The system should be ready for new updates. Basically, data will be transferred via Binance's API, but in case of a possible connection interruption, data transfer should be continued via another cryptocurrency exchange or another site that displays the prices of cryptocurrencies.

1.3 Definition, acronyms, and abbreviations

- **Binance**: A Cryptocurrency Exchange which has the highest trading volume.
- **IdP:** Identity provider.
- **RPC:** Remote procedure call.
- **gRPC:** An open-source remote procedure call framework by Google.
- **Pub-Sub:** PubSub or Pub/Sub or Publisher/subscriber. An asynchronous interservice communication style.

- **Message:** Commands or events. In the report, it has taken into account as asynchronous only and should not be confused with synchronous remote procedure calls such as gRPC messages. Recipients have to subscribe to the messages via Pub-Sub mechanisms over queues or streams.
- **Command:** A type of message, which has one and only one recipient for triggering a remote procedure call in an asynchronous way, can be sent by various subsystems and/or services.
- **Event:** A type of message that originates from one and only one service but may have zero or multiple recipients.

1.4 Overview

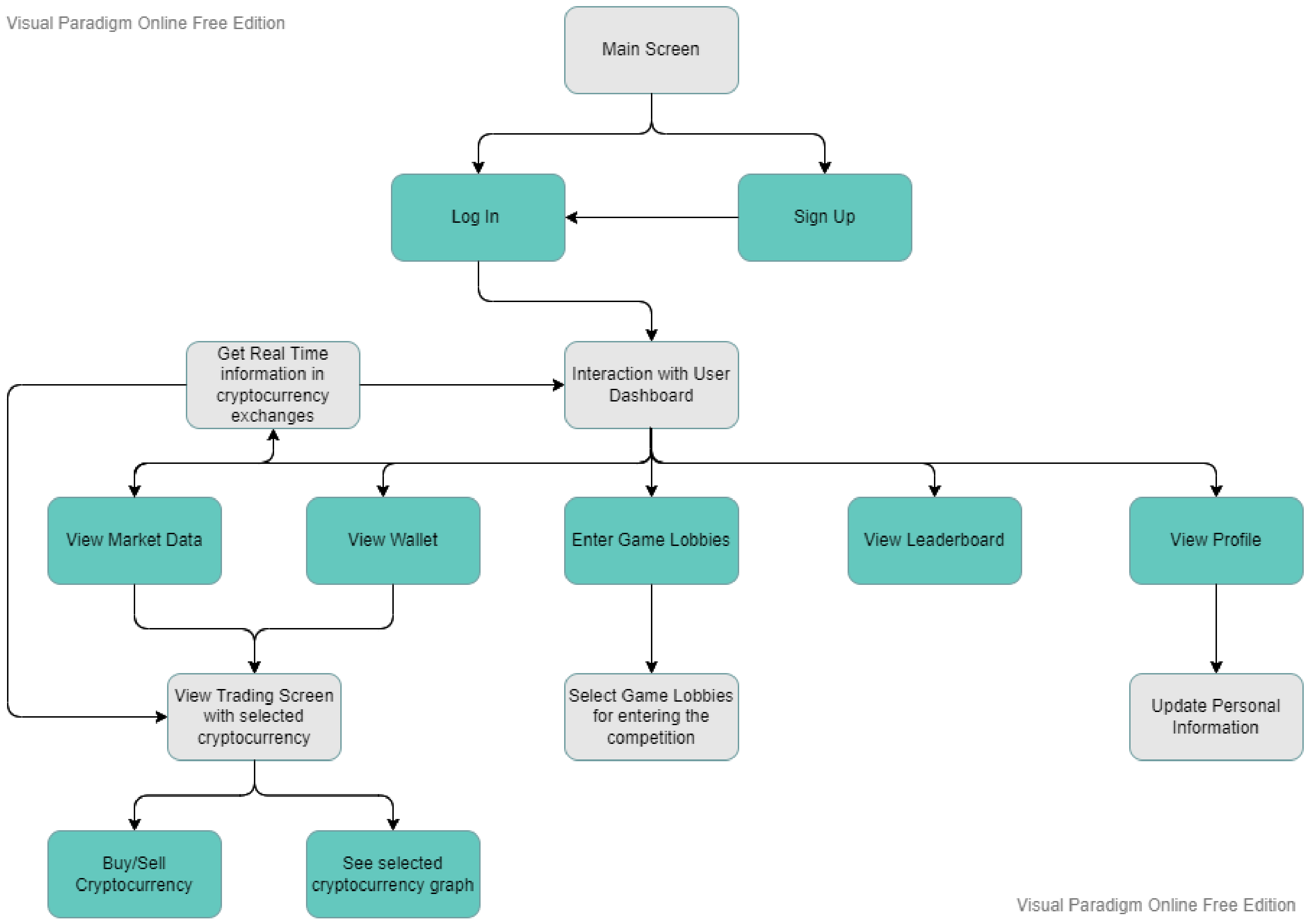
Visual Paradigm Online Free Edition
Main Screen
Log In
Sign Up
Get Real Time information in cryptocurrency exchanges
Interaction with User Dashboard
View Market Data
View Wallet
Enter Game Lobbies
View Leaderboard
View Profile
View Trading Screen with selected cryptocurrency
Select Game Lobbies for entering the competition
Update Personal Information
Buy/Sell Cryptocurrency
See selected cryptocurrency graph
Visual Paradigm Online Free Edition

Figure 0: *The stages is needed for user interaction*

As seen in the Figure 0, the general features and structure of our application are as follows. After the user logs in, or signs up, the main screen welcomes him. This screen is a screen that introduces the application and displays some cryptocurrencies. On this screen, there is access to all the features of the application. As seen in the picture, the most important process to ensure the sustainability of the application is real-time information retrieval. The user can view the market data and make transactions on any cryptocurrency. Can see the leaderboard among users. By entering game lobbies, you can participate in trade competitions according to their conditions, etc. As a result, the outline of the application and the roadmap to be followed are as follows. In other parts of the report, the comprehensive technical details of the application and the infrastructure of the system are explained in detail.

2 Current Software Architecture

In the current status, the whole software system is placed on a monolithic web application for prototyping and demo purposes. Naturally, the application is not feature-complete and demonstrates some authentication capabilities and real-time market data updates on a user interface with mock-ups. In the later stages, the features will be split into various subsystems as they described under the *Proposed software architecture* section for addressing concerns related to non-functional requirements, e.g., reliability, redundancy, performance, availability, etc. via distributed system architectures and techniques.

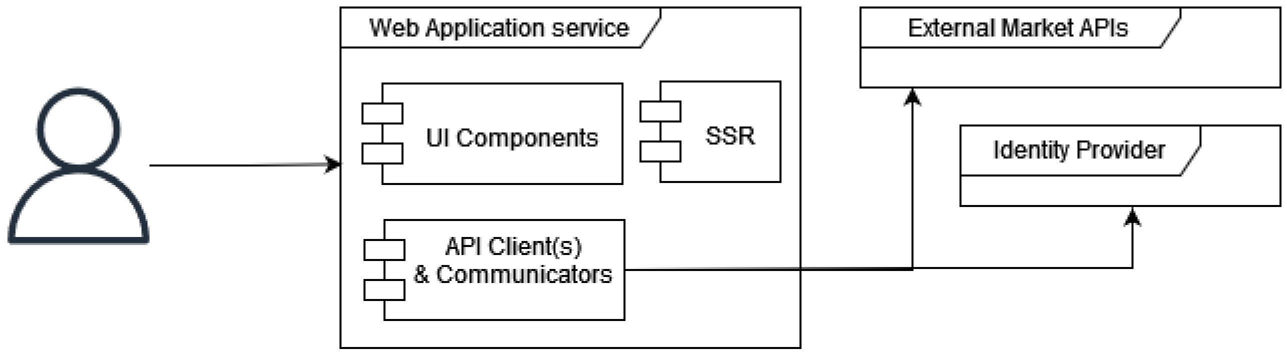

Figure 1: The current software architecture.

3 Proposed software architecture

3.1 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. Time to time, the philosophy that we follow may differ per service basis, because the nature of microservices architecture enables us to opt for different software design paradigms depending on requirements of each service, individually.

Microservices architecture is only one side of the coin. It is also crucial to consider additional details such as communication styles & paradigms between service boundaries, as well. A mix of event-driven architecture and request-reply pattern is the communication style that we will follow. 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.

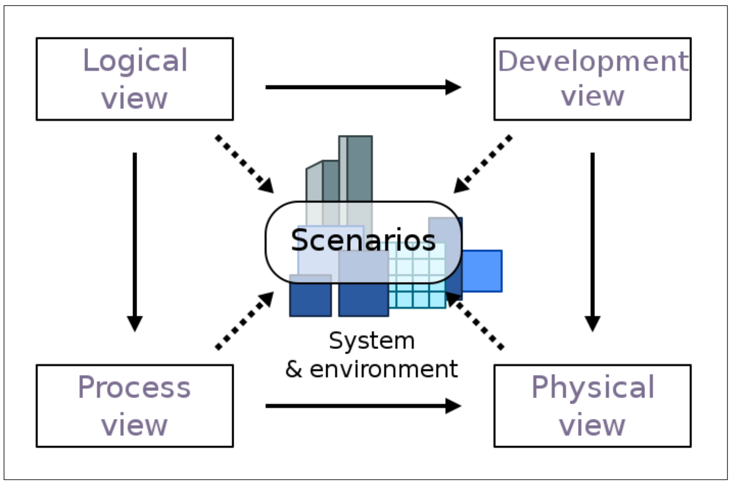

Figure 2: *4+1 Architectural View Model Illustration. The image is taken from https://en.wikipedia.org/wiki/File:4%2B1_Architectural_View_Model.svg*

Additionally, development view of the software is another important aspect to consider as it is depicted in the renowned diagram of *4+1 Architectural View Model* (Figure-1). Developing a software solution, which spans across diverse sets of distributed services, requires a different approach in the development process. Altogether, deciding on the right Git repository strategy might be knotted. We opted for relying on mono-repo strategy instead of creating a different repository for each microservice, since it will be beneficial in terms of improved collaboration, better code consistency, and simpler dependency management **[1]**.

3.2 Subsystem Decomposition

When business requirements are considered, detecting and inspecting the right business domains is vital. Therefore, splitting business domains correctly guides decomposing subsystems accordingly. That may sound like we are following the renowned *domain-driven design* approach; however, we are not planning to totally act on it. Even so, considering the boundaries of business domains helped us to detect subsystems easier.

In the problem space, there are two business domains that we have detected: *identity & profile management* and *market & trade activities*.

An identity provider solution will be dedicated to covering *identity & profile management* domain, herewith a single subsystem will be responsible for that issue as well. In the *Access control and security* subsection, further details about identity provider and the domain itself are discussed in detail, besides *Subsystem services* section covers the in-depth view of the subsystem.

On the other hand, *market & trade activities* domain spans across multiple subsystems: *trading subsystem*, *market subsystem*, *asset management subsystem*, and *engagement subsystem*. Each subsystem is explained under *the Subsystem Services* section, including the subservices they cover.

Moreover, *user interface (UI)* will be taken into account as a separate subsystem since it is a monolith organism, which spans across various subsystems.

3.3 Hardware/software mapping

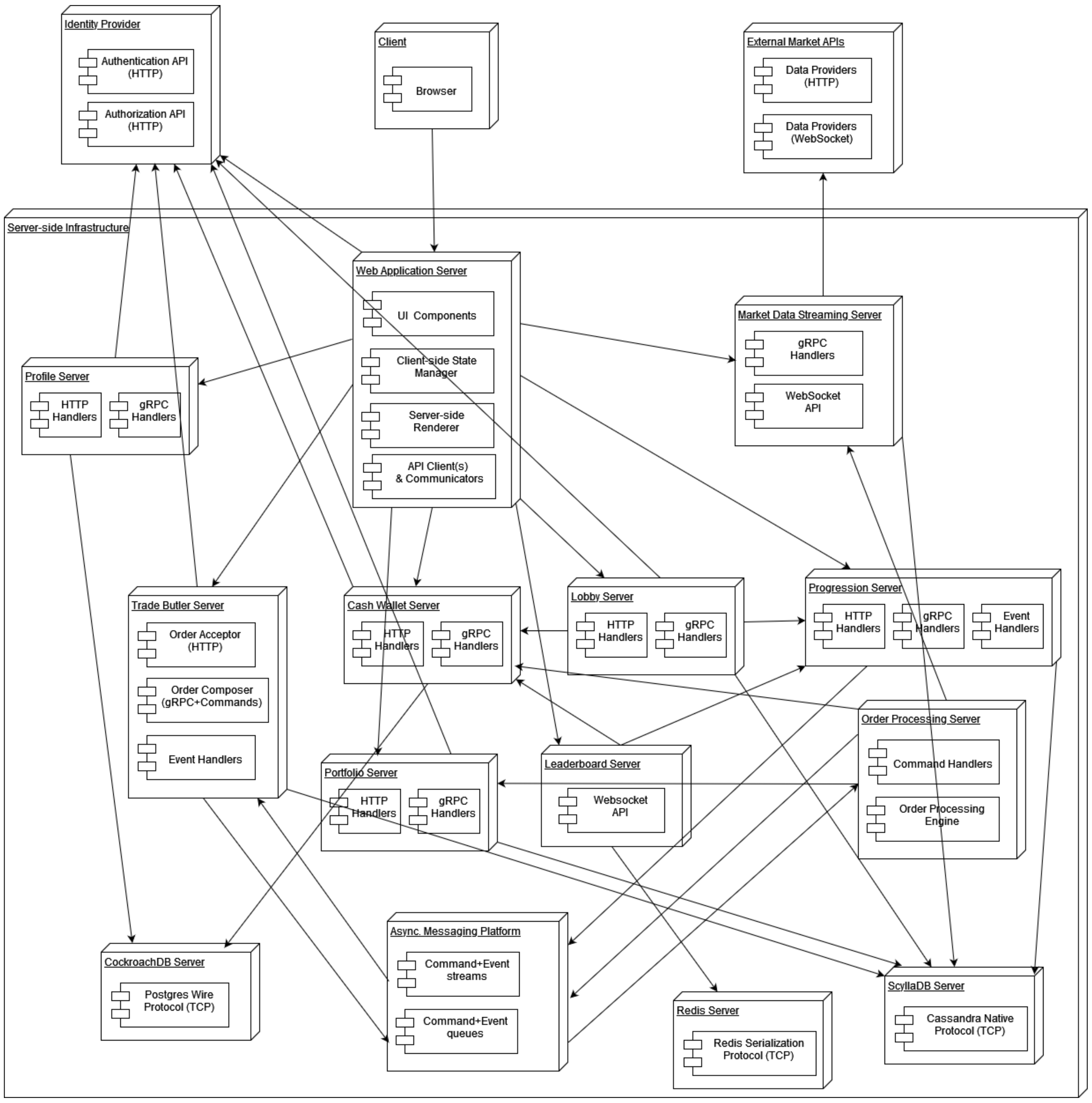

Figure 3: The inter-dependency map for hardware (both bare metal & virtualized resources) and software components.

3.4 Persistent 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. We separated the use cases into two groups: *identity & profile related features* and *features at the heart of market & trade activities*. The features which are at the heart of market & trade activities are subject to low latency and high throughput characteristics, therefore, a NoSQL solution must be dedicated to leverage that scale of intensive data. **ScyallaDB** is our go-to solution for this problem. On the other hand, a traditional ACID-compliant SQL database would be more than enough for identity & profile related features as well as other banal use cases that may emerge in the future. **CockroachDB** is dedicated to solving these kinds of tasks. Lastly, in-memory databases such as Redis will be utilized for data caching purposes, however the aforementioned kind of data is not subject to persistency.

3.5 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. Auth0 and Ory are our prominent candidates as IdP solutions. 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.

3.6 Global software control

The omnidirectional communication between subsystems and services must be considered as a part of the global software control because it describes how subsystems handle information exchange and data synchronization as a response to the initiated global triggers such as user and interservice requests. As it was mentioned in the Microsoft's post about microservices architecture **[2]**, the communication between services or subsystems can be leveraged either in synchronous or in asynchronous ways. In our proposed system architecture, both communication methods will be utilized.

The synchronous way will be responsible for user and/or service requests, which expects immediate response. These kinds of requests cover mostly short-running business processes that do not require that much time and computing resources, e.g., user authorization, CRUD operations. gRPC will be dedicated as a solution to this problem.

On the other hand, complex business domain operations, which require decent computing resources and time, need to be handled in an asynchronous way. There are several styles to make that happened and we are planning to stick with *publisher/subscribe* type of messaging technique as a part of the *event-driven architecture* as it is defined well in Amazon's post **[3]**. One of the battle-tested, high-quality event streaming/queue platforms such as Apache Kafka®, RabbitMQ, NATS and Redpanda, will be utilized in the context of this problem. There sometimes occur short-term, system-wide eventual consistency related problems, because a piece of data will be owned by a particular subsystem or service. Therefore, if another subsystem wants to make use of that piece of data, the synchronization may take time. Ultimately, the addressed issue is not a complete defect, it is a requirement for handling long-running business processes in a performant and elegant way. Either user or another service may trigger this behavior in the context of global software control.

3.7 Boundary conditions

3.7.1 Start-up Condition

The start-up (initialization) conditions can be grouped into two sections: *user perspective* and *system perspective*.

From the *user perspective*, the start-up takes place in a web browser environment, which is the sole requirement for client application. Some real-time market data properties can be accessed without any authorization, but authentication will be required for further activities such as trading and joining lobbies. If user has been created an account previously and logged into the application, the rest of operations like session management, fresh data updates, state management, etc. will be managed by the client web application automatically.

From the *system perspective*, the initial market data synchronization must be handled before proceeding with any operations. In parallel, the clients must be initialized for handling database, IdP, gRPC, and messaging platform connections.

3.7.2 Error Condition

If any error occurs related to synchronous global software control instruments, an error message is returned to the actor immediately. The actor is either a user or a subsystem component. Ultimately, the related subsystem can return the end result to the end-user without any retry.

In case of an error related to asynchronous global software control instruments, errors must be handled in a more sophisticated way. Because of the asynchronous nature of the mentioned instruments, it is not possible to return the error immediately to the end-user immediately. Therefore, certain retry mechanisms will be utilized. Moreover, an *event store* will be delegated to the services, which may need to handle such issues in an elegant way. Since the event store maintains the history and set of states for the handled and unhandled commands & events, the failed ones can be retried automatically without causing any system-wide issue. On the other hand, a service may be terminated suddenly due to a fatal error. Thanks to the fault tolerant structure of the system, the unhandled events would be processed when the service is back online.

3.7.3 Shutdown Condition

From the *user perspective*, the shutdown condition is just a one-click away. Activating the exit button from client web application interface will terminate the user session properly by maintaining the latest user state at the backend.

From the *system perspective*, the shutdown condition must be handled gracefully. Services execute set of operations for closing the open network connections, etc.

4 Subsystem services

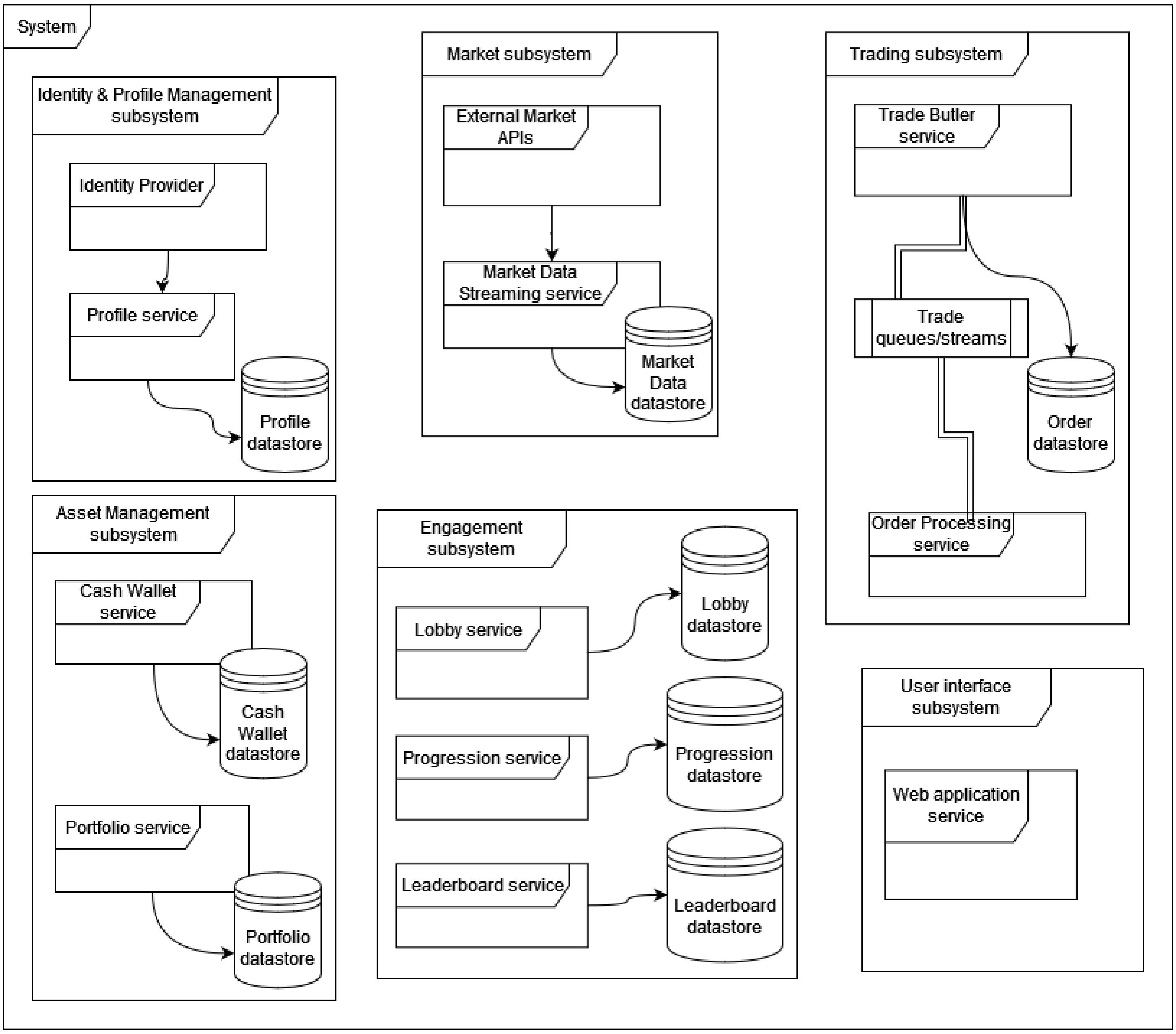

Figure 4: The subsystem map. Represents the high-level logical view within a subsystem.

4.1 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 stores and manages the profile specific data, which is differentiated from the identity and access management data, provided by the Identity Provider.
- **Profile datastore**

4.2 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 services regularly, feeds *Trading subsystem* with the properly organized market data.
- **Market Data datastore**

4.3 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. Within the process, properties like wallet balance are checked.
- **Order datastore**
- **Trade queues/streams**
- **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.

4.4 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**
- **Portfolio service:** As a service, it stores and manages the assets which are held by a user. Except the main portfolio, a user may have multiple additional portfolios at a time, too, for instance a portfolio per lobby is created by the system on behalf of user.
- **Portfolio datastore**

4.5 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 datastore**
- **Progression service:** The service that will be responsible for managing and storing achievements, awards, and level/experience progression of each user.
- **Progression datastore**
- **Leaderboard service:** The service that will maintain leaderboard table in near real-time or term-based constraints by composing (sort, filter, classify) the data received from other subsystems.
- **Leaderboard datastore**

4.6 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, which is based on a SSR (server-side rendering) based web application framework, covers user facing visual interface elements and client interactions for all overall system. It will make communication with the other subsystems for fetching and pushing the client data via predefined communication protocol standards such as HTTP API, gRPC, etc. Serves in both browser (client) and server (backend) environments.

5 References

[1] 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

[2] 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

[3] Amazon Docs Contributors. (2022). *WebSphere Business Integration Pub/Sub Solutions*. Amazon. Retrieved December 25, 2022, from https://aws.amazon.com/pub-sub-messaging/

Wikimedia Foundation. (2022, December 21). *Blockchain*. Wikipedia. Retrieved December 25, 2022, from https://en.wikipedia.org/wiki/Blockchain