TED UNIVERSITY

CMPE 491 Project Analysis 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

Cryptocurrencies have become the most used investment tool in recent years. 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.

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.

2 Proposed System

2.1 Overview

Due to heavyweight nature of the business domain, the system benefits from event-driven microservices architecture. As a part of this decision, it's planned to leverage the cutting-edge event-streaming and queue platforms such as Apache Kafka®, RabbitMQ, NATS and Redpanda considering publisher-subscriber style patterns in mind. The final decision will be made after conducting some experiments and reviewing real-life benchmarks among these options and other alternatives.

The concept of being event-driven is crucial to reduce the load over server-side services, especially the load caused by long-running/heavy business processes, e.g., processing trade orders, near real-time ingestion of market data, or regular leaderboard updates in batch. However, there are also cases that do not require sophisticated solutions. As much as event-driven architecture helped out, it might also make development, provisioning, and troubleshooting processes complicated. When the business requirements such as time to market (TTM), developer experience (DX), and software maintainability are considered, preferring the traditional request-response pattern can be cheaper and beneficial for some inter-service communication needs as an alternative to sophisticated nature of event-driven protocols. There are certain light-weight processes in our system like dealing with basic account & profile data and it is considerable to impose the aforementioned request-reply pattern in such cases. For that purpose, it's planned to opt for gRPC – a modern RPC (Remote Procedure Call) framework (internal communications). Furthermore, conventional HTTP APIs will be dedicated for external communications between the client and server-side parties.

For authentication and authorization needs, we decided to rely on battle-tested identity provider (IdP) solutions better and healthier. Auth0 and Ory are prominent candidates we consider because of their proven capabilities in production environments.

Data holds an important place for the overall system. Therefore, opting for right datastore solutions is crucial to create an application, which is reliable, fast, consistent and provides a great user experience. We believe that ACID-compliant traditional relational databases are still prominent for transactional needs. CockroachDB – a PostgreSQL wire compatible, distributed, yet ACID-compliant database – will help us to power up the account & profile related features. Furthermore, making use of CockroachDB for other features

is also considerable if requires. Beyond that, the advantages of NoSQL database solutions are undeniable facts for certain workloads that might require storing enormous scale of data. ScyllaDB – a NoSQL, wide-column datastore as an alternative to renowned Apache Cassandra – will bring our data-intensive features, e.g., trading and lobby, to life with its high throughput and low latency characteristics. Additionally, Redis – a popular in-memory key-value database – will be utilized for query caching purposes thanks to its ultra-fast nature.

The user will reach out to the server-side systems via a web-based client application. We are going to put Next.js into action for building up the web application. Next.js, which is a popular and battle-tested web framework, can handle both client and server related concerns and supports modern concepts like SSR (server-side rendering). When comes to the back-end part of our proposed system, we are freer to roam around various programming languages and tech stacks since there is no such a restrictive environment like browser at the server-side. Instead, we can take advantage of different kind of tools and technologies thanks to the loosely coupled and independent nature of microservices. Thus, even if we are dedicating mostly on Go language and Node.js runtime, it is possible to utilize other programming languages and platforms according to our needs by time.

As we mentioned before in previous reports, market data will be fetched from a certain set of APIs hosted by cryptocurrency data providers, e.g., Binance. These APIs are important external dependencies for the system. The application may live without them for a while, but it would be meaningless devoid of fresh market data continuously in near real-time.

2.2 Functional Requirements

2.2.1 Log In

- REQ-1: The user who enters e-mail and password shall be able to press the log in button.
- REQ-2: The system should be able to authenticate and authorize the user.
- REQ-3: System should give an output which is “Your password or e-mail is wrong, please try again...”, when user give a wrong information.
- REQ-4: If the user forgets his password, he/she should be able to renew his password from the password reset e-mail sent to him/her by clicking the "Forgot your password?" button.

2.2.2 Sign Up

- REQ-1: Users shall be able to create an online account.
- REQ-2: While creating an account, if there is a mismatching between two passwords, system should give an error message which is “The first password you entered does not match the second password.”
- REQ-3: The system shall be able to protect user information in secure way.

2.2.3 Dashboard

- REQ-1: The system should be able to display the cryptocurrencies with the largest market volume under the "Popular Cryptocurrencies" heading on the user dashboard.
- REQ-2: Users should be able to go to the trading screen of that cryptocurrency by clicking on any of the popular cryptocurrencies.

2.2.4 Market Screen

- REQ-1: The system should be able to show live cryptocurrency data to the user with the minimum possible delay.
- REQ-2: User shall be able to search any cryptocurrencies with search bar.
- REQ-3: Users should be able to go to the trade screen of the relevant cryptocurrency by clicking the "Trade" button next to each cryptocurrency.
- REQ-4: Users should be able to buy the cryptocurrencies they want to their favorites.

2.2.5 Trade

- REQ-1: Users should be able to trade with their virtual balances.
- REQ-2 The system should be able to display real-time graphics of related cryptocurrency.
- REQ-3: The user's favorites should be displayed on this screen by the system and the user should be able to open the trade screen of the relevant cryptocurrency by clicking on one of their favorite cryptocurrencies.

2.2.6 Game/Trade Lobbies

- REQ-1: Users should be able to enter trading competition lobbies.
- REQ-2: The system should be able to award a prize to lobby winners.
- REQ-3: Users should only be able to enter the lobbies they meet the rules in accordance with the rules of the lobbies.

2.2.7 Leaderboard

- REQ-1: A ranking table among users should be displayed by the system.
- REQ-2: Users should be able to see who has the most virtual balance or who has made the most profit.
- REQ-3: Users should be able to see what rank they are among those who use the app.

2.2.8 Virtual Balance/Wallet

- REQ-1: The system should be able to define virtual balance to users.
- REQ-2: The system should be able to calculate the profit and loss of the users and update the balance of the user.
- REQ-3: Users should be able to see how much money they have in their virtual balance and what cryptocurrencies they have.
- REQ-4: Users should be able to go to the trade screens of their cryptocurrencies with the "Trade" button on this screen.

2.2.9 Profile

- REQ-1: The system should provide the opportunity for users to view their profiles and change some of their information.
- REQ-2: The system should be able to show a small image of the leaderboard screen on this screen. (The person's rank and others near them)

2.3 Nonfunctional Requirements

2.3.1 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.

2.3.2 Security

Data security is the most important requirement 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.

2.3.3 Usability

- The System should be user friendly.
- Users should not experience confusion while using the application.
- The graphical interface of the application should be understandable and interesting.
- The performance of the application should be at high levels so that the user does not force the application.

2.3.4 Reliability

- The system must work 24 hours a day. The user should be able to use this application whenever he/she wants.
- The failure rate of the system should be at very low levels. Errors that may occur independently of the user should be minimized.

2.3.5 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.

2.3.6 Durability

- The system must be prepared for overload.
- The system should provide lifetime use.

2.3.7 Supportability

- The system should be prepared for possible updates.
- The system should be maintainable.

2.4 Pseudo requirements

- Operating environment must be Linux based.
- The system should have enough storage capacity to be able to operate healthy.
- Application must have active internet connection.
- At least one of the external Crypto APIs must be operational.

2.5 System models

2.5.1 Scenarios

Scenario: Sign Up

Actors: User, Server, Identity Provider

1) User clicks to Sign Up button.

2) Enters the registration information.

3) Agrees Terms of Use and Privacy Policy.

4) Submits the registration information to the server.

5) Server communicates with the identity provider and creates an account behalf of user.

6) Identity provider sends a verification email to user with expiration.

7) If user verifies the email successfully, the account creation process is completed.

Scenario: Log In

Actors: User, Server, Identity Provider

1) User clicks to Log In button.

2) Fills up the login information.

3) Optionally, selects the Remember Me box.

4) Submits the login information and sends to the server.

5) Server verifies user credentials by communicating with the identity provider.

6) If credentials are valid, returns the ID & access tokens to user.

7) User fetches dashboard details using the access token and is forwarded to dashboard page.

Scenario: View Overall Market Details

Actors: User, Server

1) User visits Markets page.

2) A real-time socket connection (e.g., WebSocket protocol) is created between user and server.

3) Server pushes the latest aggregated market data continuously to user over the socket.

Scenario: Search Cryptocurrencies from Market List

Actors: User, Server

1) User visits Markets page.

2) Types keywords to search currencies.

3) Server queries the results, then returns the filtered results that are appropriate to search query.

Scenario: Trade Currencies

Actors: User, Server

1) User clicks to Trade button from Markets page and the page is forwarded to trading view.

2) User views the latest trading details of the selected parity and optionally changes the parity from the list.

3) User can either buy or sell certain number of cryptocurrencies by typing the amount into the boxes or adjusting via slider.

4) Server enqueues the demand and processes when the server load is appropriate to process the order.

5) After the process, the wallet balance of user is tweaked by server.

Scenario: View Leaderboard

Actors: User, Server

1) User visits Leaderboard page.

2) Server queries the top 100 players with some details and returns to user. By default the leadership is ordered by Total Wallet Balance. If user is not ranked among top 100 players, user's rank is appended as an extra.

Scenario: View Wallet Balance

Actors: User, Server

1) User queries the wallet information from server.

2) Lists the coins with the owned amount more than 0 at the top.

3) Optionally, user can click to Trade button and is forwarded to the selected coin's trading page.

Scenario: Change Username

Actors: User, Server

1) Visits Profile Settings.

2) Users write the new username and clicks to Rename button.

3) Server validates the request and updates username globally.

Scenario: Show Profile Details

Actors: User, Server

1) Clicks to Profile icon from upper-right corner and is forwarded to Profile page.

2) Profile details such as username, profile photo, real name, the rank of user is retrieved from server.

Scenario: List Trading Lobbies

Actors: User, Server

1) User visits Lobbies page.

2) Fetches the available lobbies from server and lists in a board view.

Scenario: Change Email

Actors: User, Server

1) Visits Profile Settings.

2) User writes the new email and submits.

3) Server validates the request and sends a verification email with expiration.

4) After the successful verification of email, the email is updated globally.

Scenario: Change Password

Actors: User, Server

1) Visits Profile Settings.

2) User writes the old password for changing the current password.

3) After validation of the old password, the boxes to type new password appear.

4) User fills up the new passwords and submits.

5) Server validates the request and sends a verification email with expiration.

6) After the successful verification of email, the password is updated globally.

Scenario: Ingest Real-time Market Data

Actors: Server, Crypto APIs (e.g., Binance)

1) Server calls the predetermined crypto APIs for fetching the latest market data continuously.

2) Crypto APIs returns the results in the format they specified.

3) Server filters the results via various adapters depending on the external crypto API.

4) Server aggregates the results from these sources, then caches and saves to the databases.

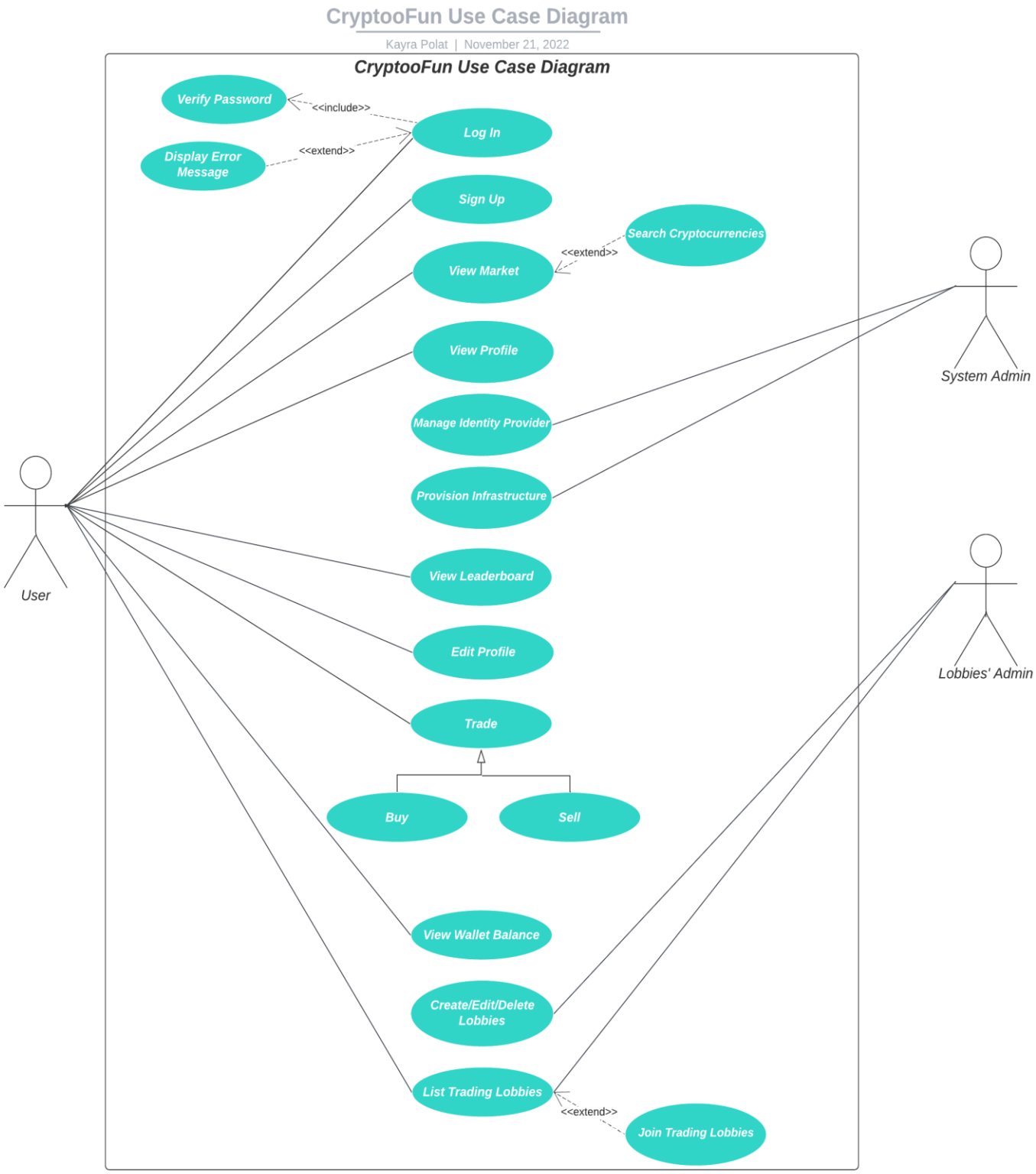
Scenario: Refresh Leaderboard Ranks

Actors: Server

1) Server aggregates all profiles in batch and filters out for ranking purposes.

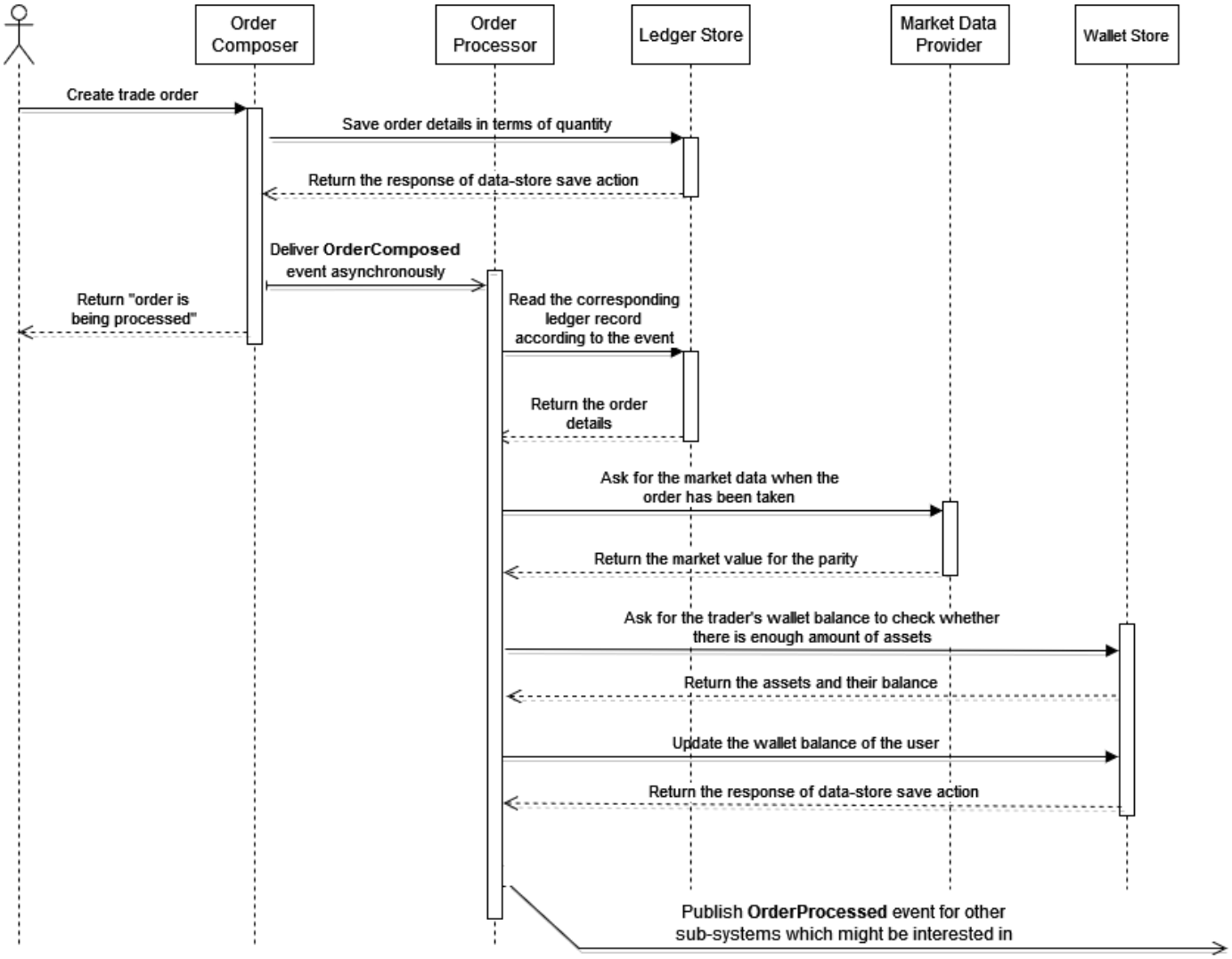
2) After the ranking query is done, the result’s cached and saved to an in-memory datastore.

2.5.2 Use case model

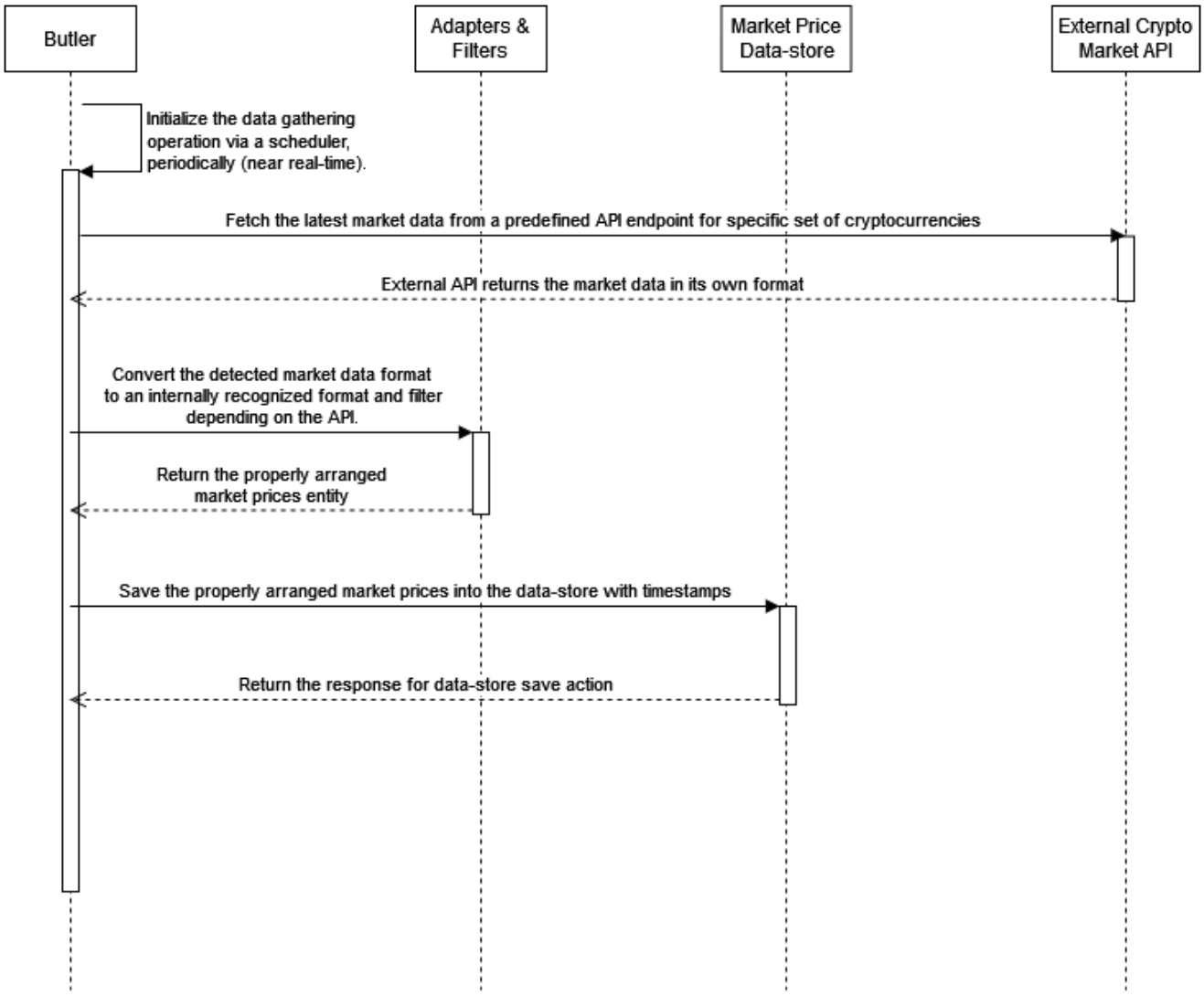


2.5.3 Dynamic models

2.5.3.1 Sequence Diagram for Trade Currency

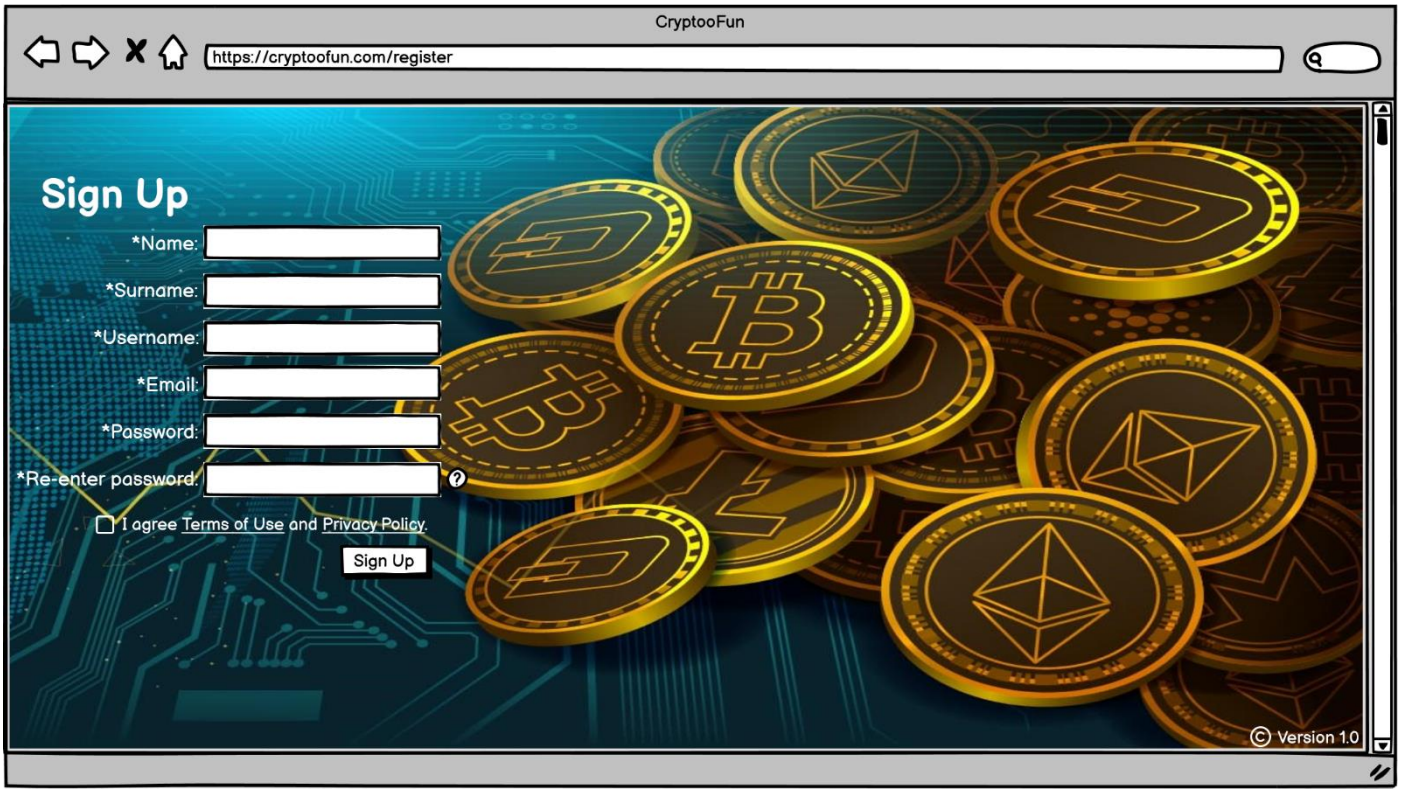


2.5.3.2 Sequence Diagram for Ingest Market Data

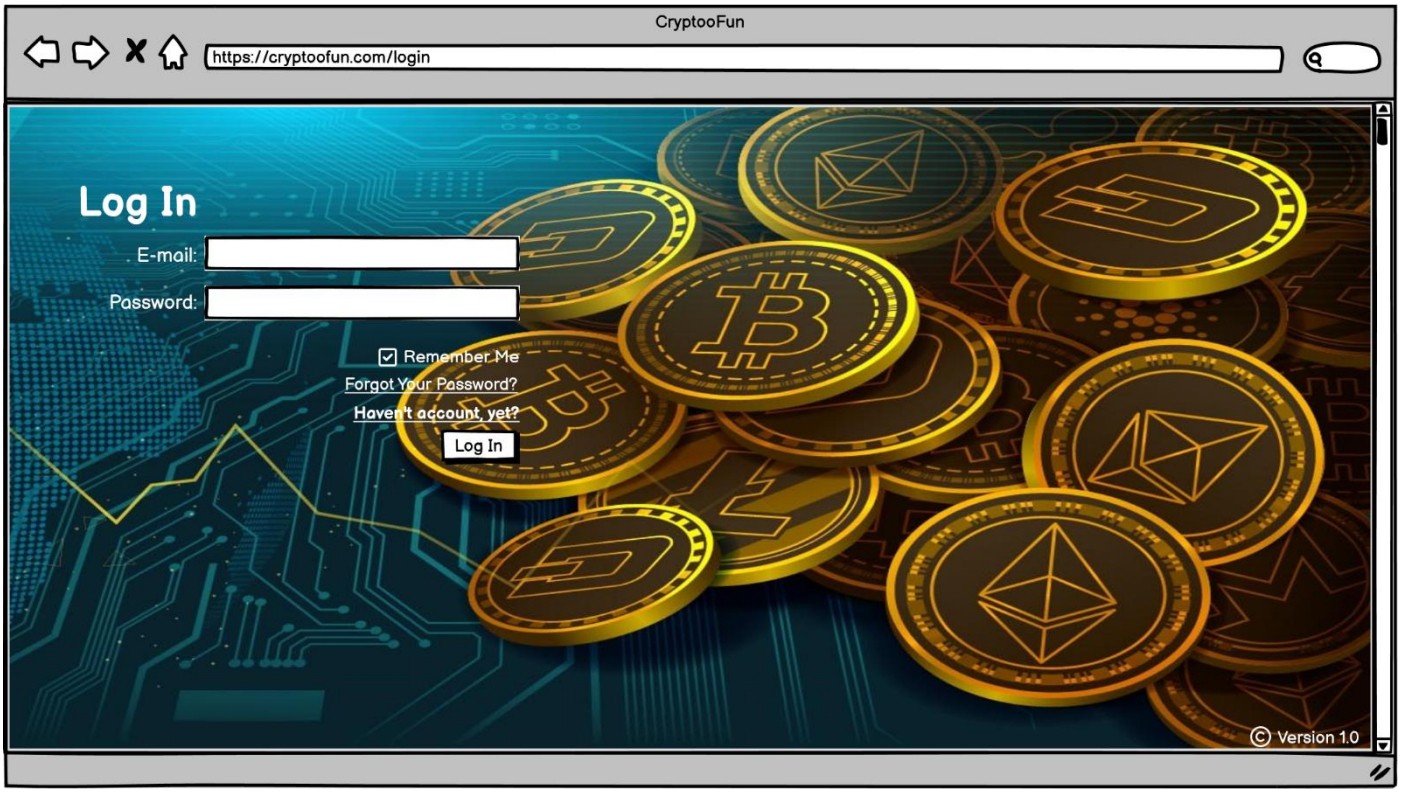


2.5.4 User interface (Mock-Up)

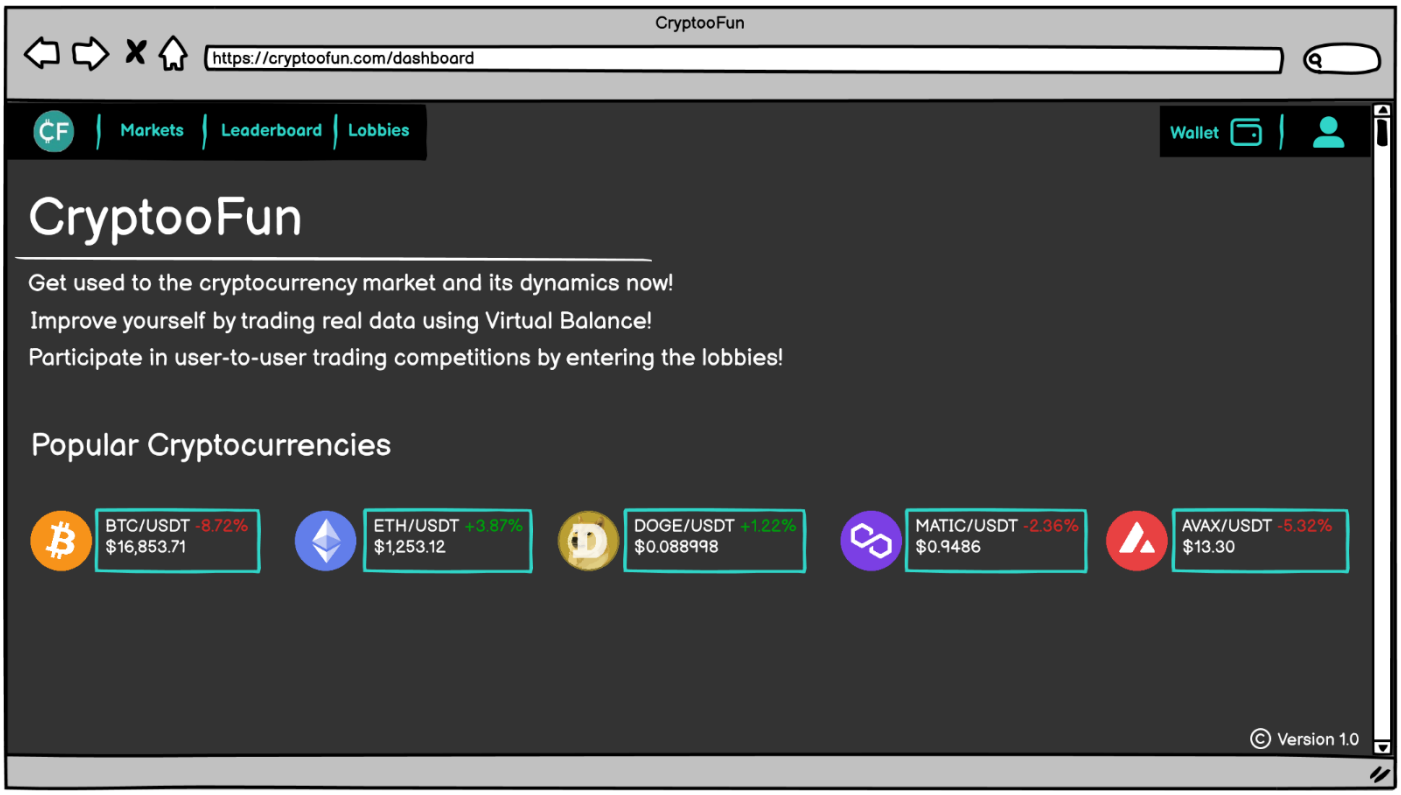
[1] Home Screen



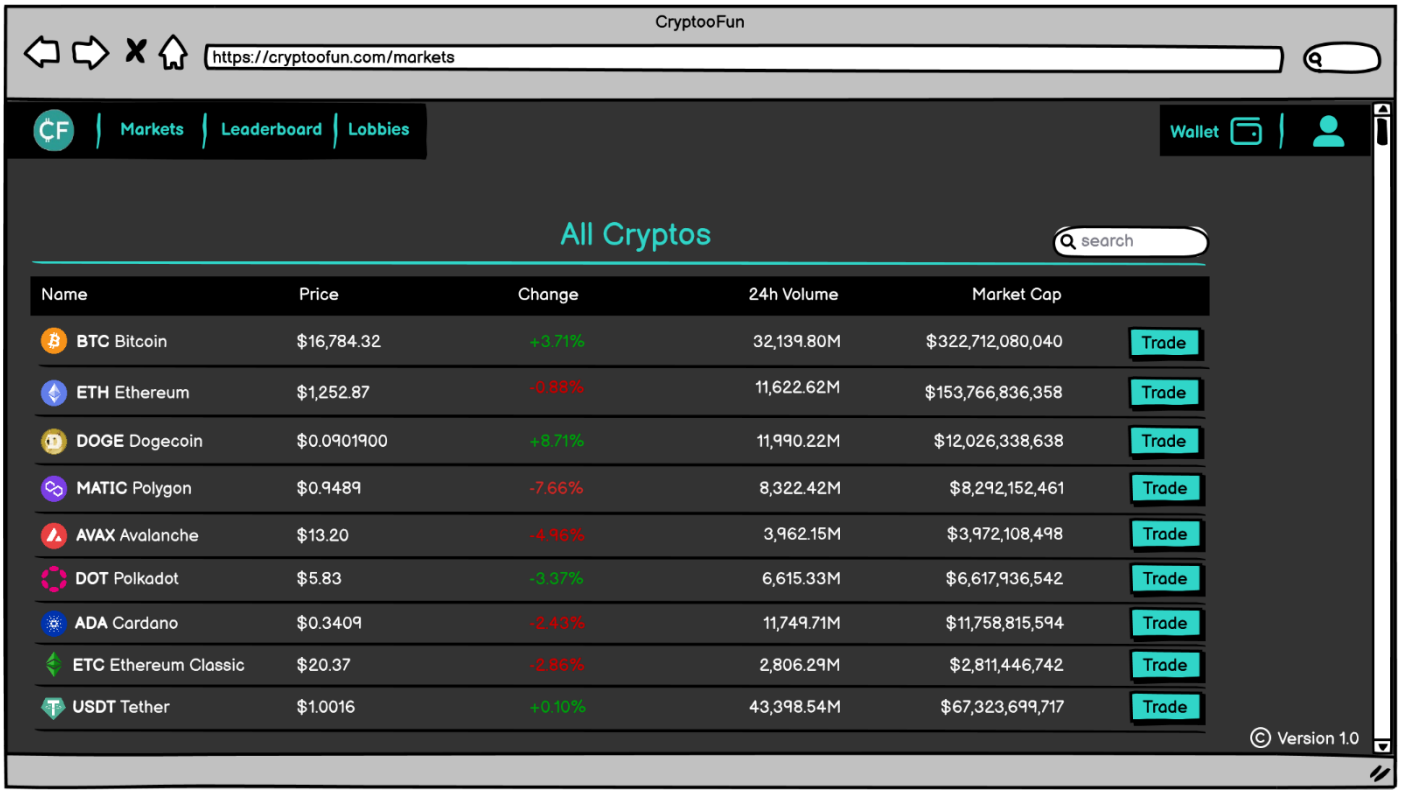
[2] Sign Up Screen



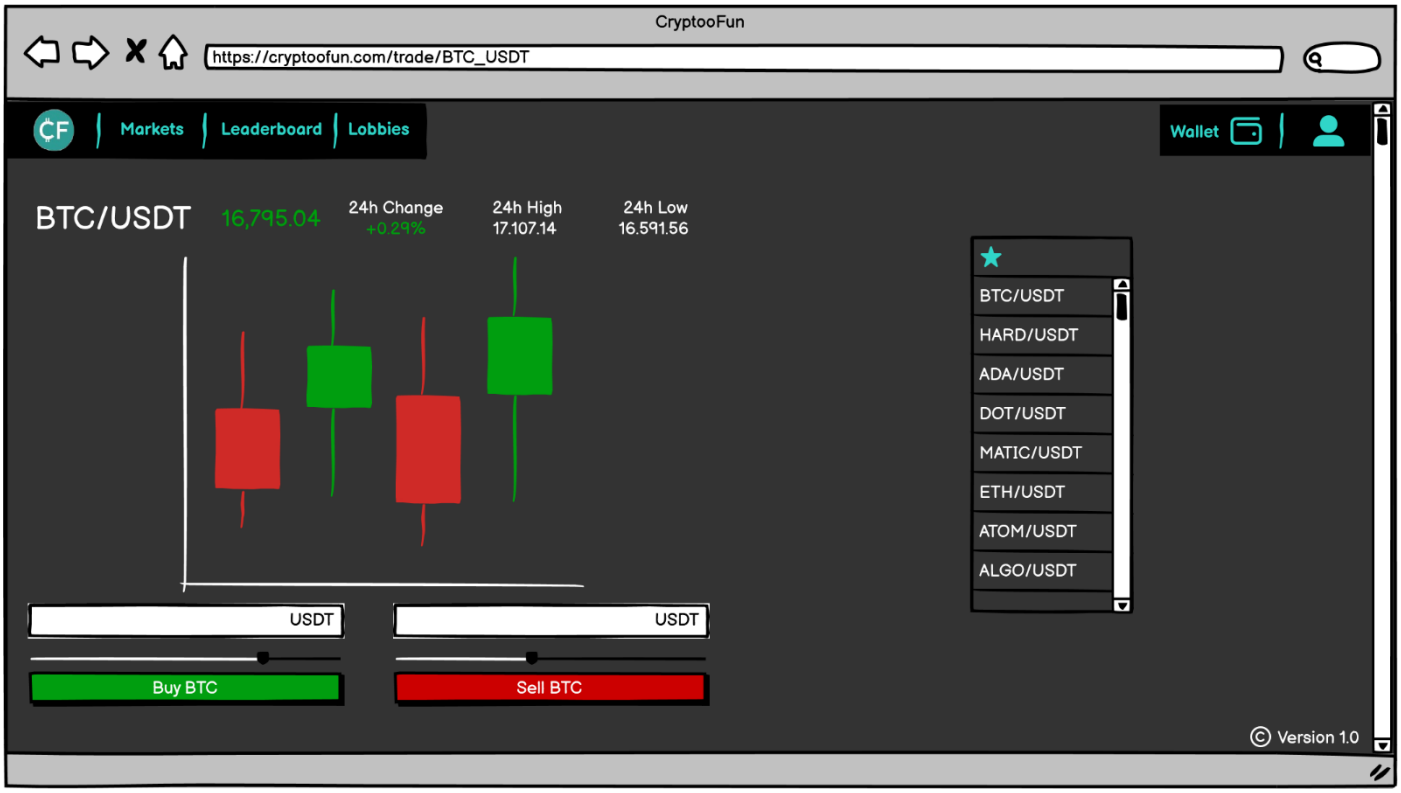
[3] Log in Screen



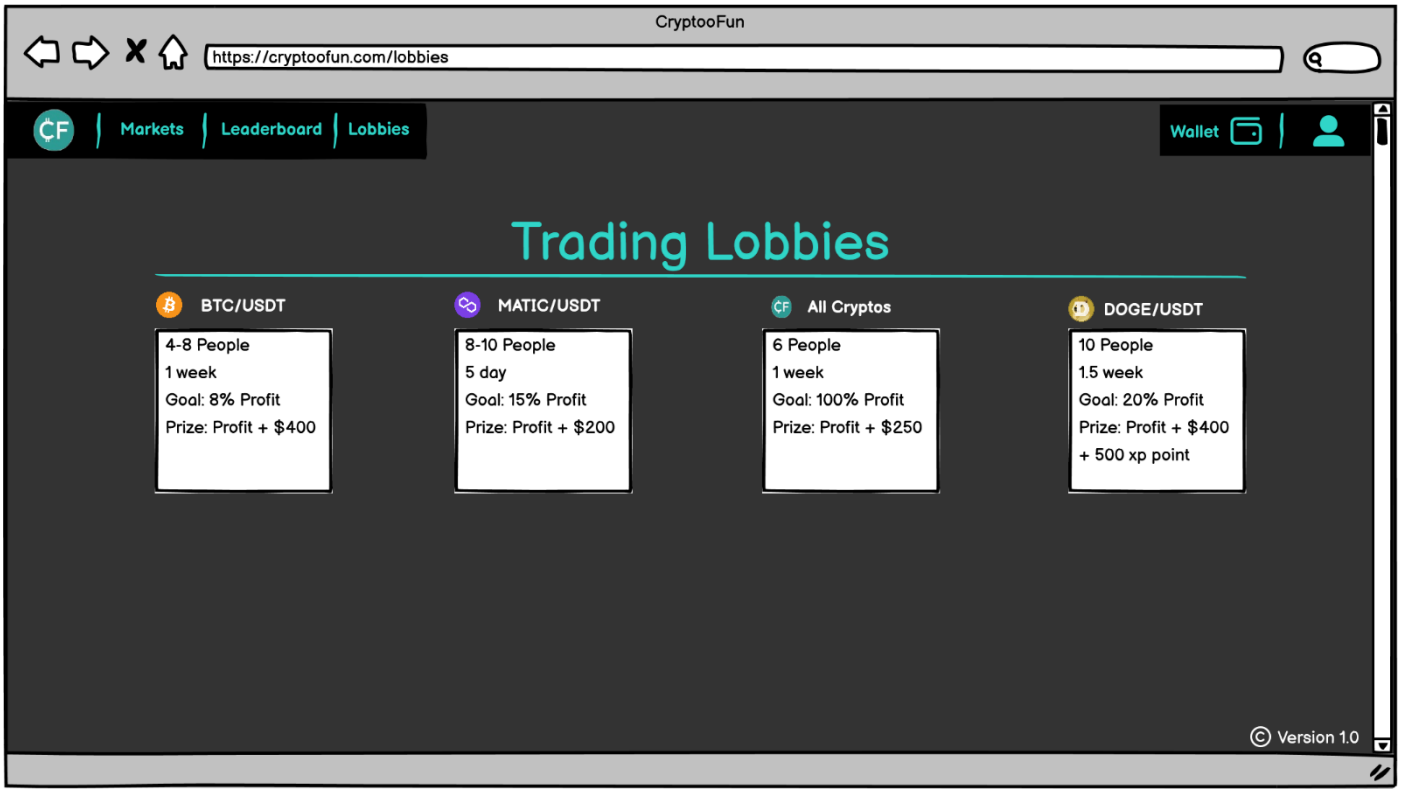
[4] User Dashboard



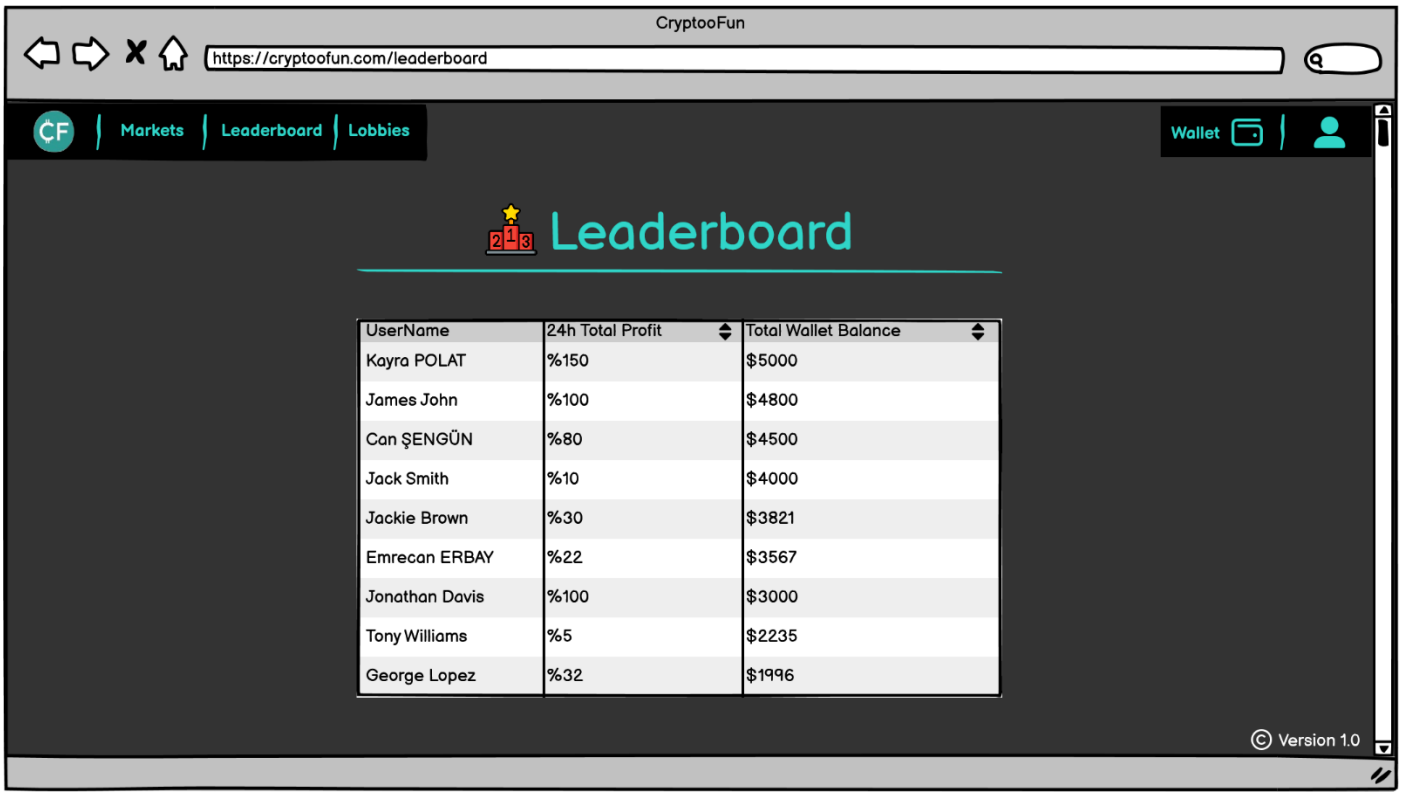
[5] Market Screen



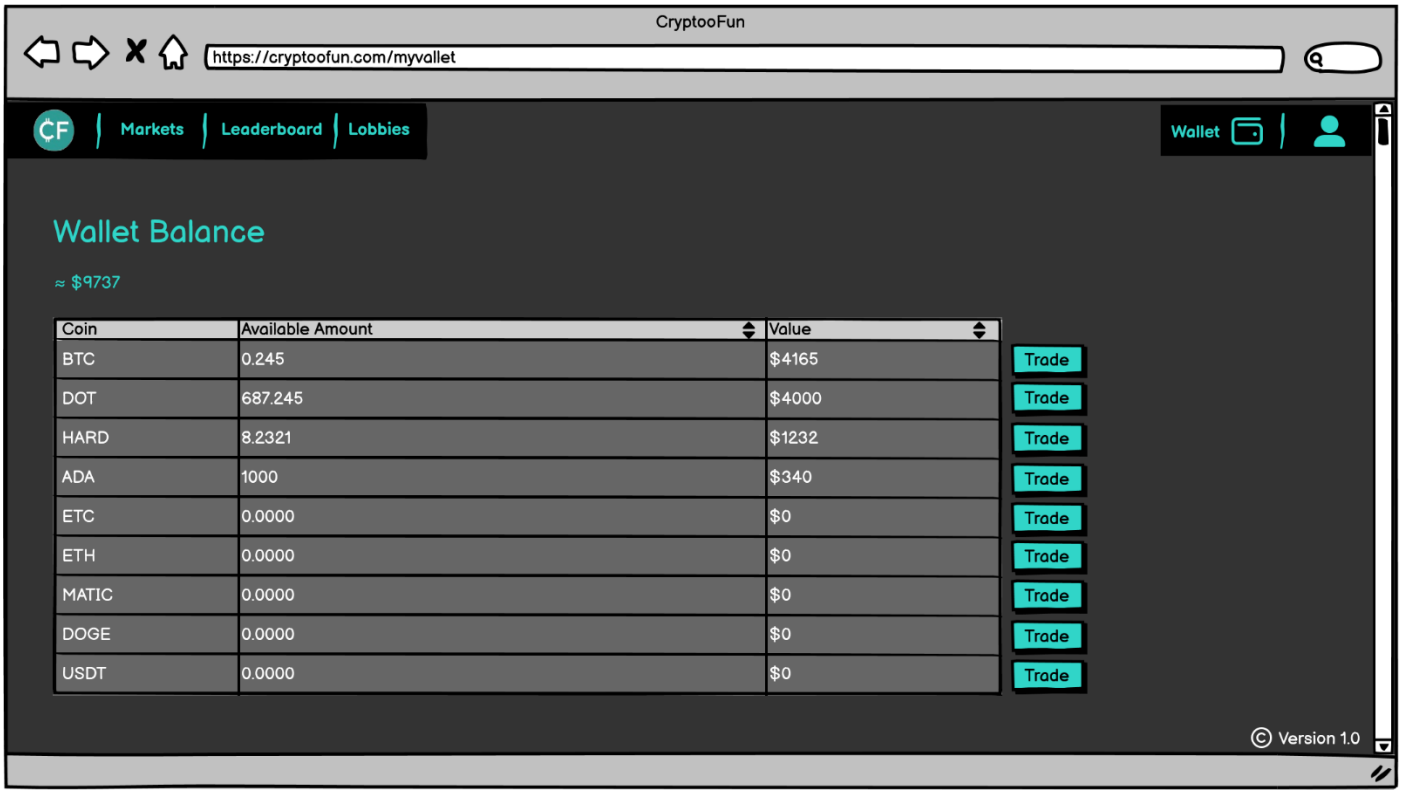
[6] Trade Screen (In this case BTC/USDT)



[7] Lobbies Screen

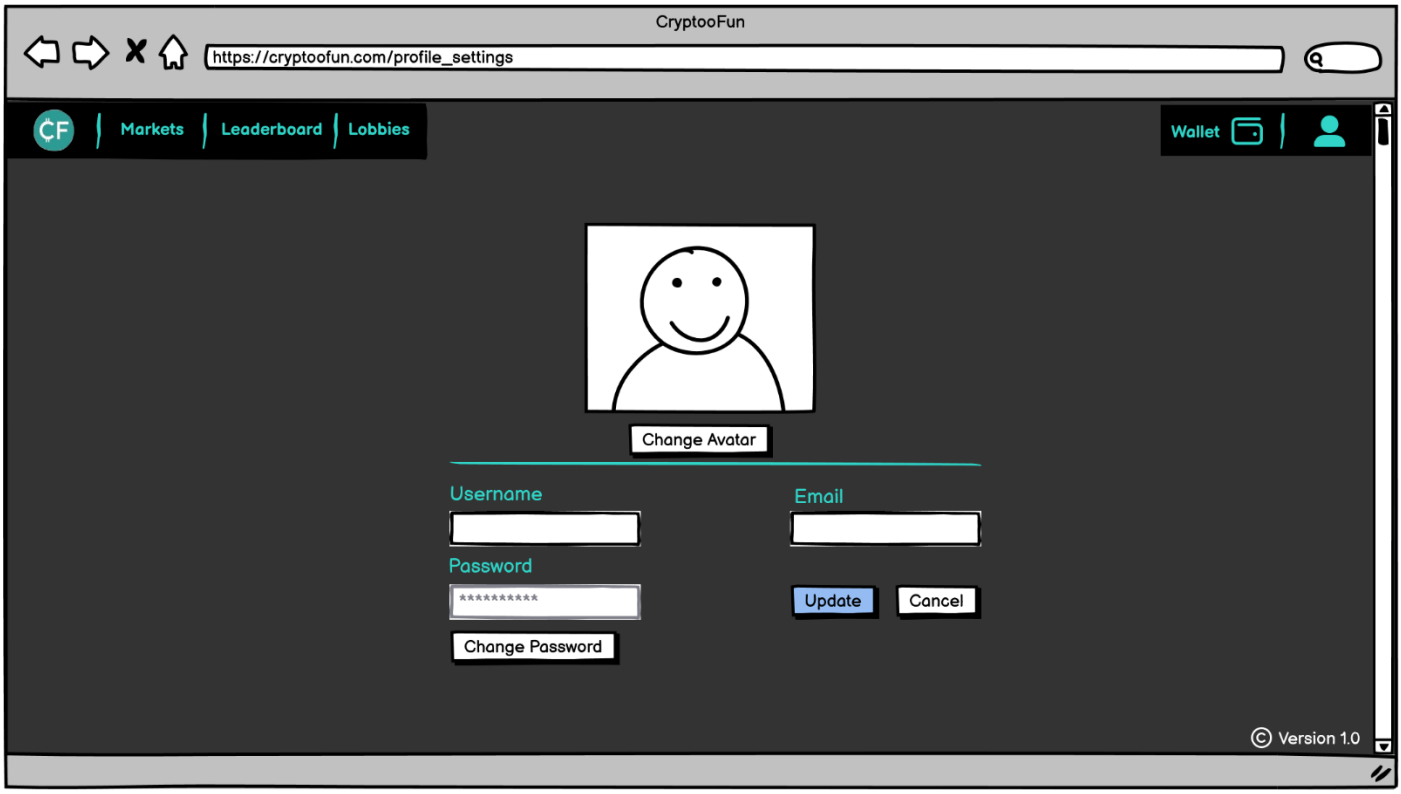


[8] Leaderboard Screen



[9] Wallet Screen

[10] Profile Screen



[11] Profile Settings Screen