IT5007: Final Project MazeSoba

Team Members
Ding Ming (A0214574E)
Chen Yixun (A0245243L)

Project Overview

Problem statement

With nearly 16% of Singaporeans owning cryptocurrency, it has become more important than ever to be able to buy and sell cryptocurrency easily, and to have the option to earn interest on long term investments. Many existing centralized crypto exchanges either operate on a traditional Central Limit Order Book (CLOB) model which can be daunting to the average layperson, or utilize a Broker-style model and charge exorbitant settlement fees. This proposal is for a centralized off-chain exchange utilizing the Automated Market Maker (AMM) system, modeled after similar decentralized exchanges but as of yet not seen in centralized exchanges. The AMM model presents a **simplified interface to consumers** (Buy/Sell), enables **lower trading fees** (no external settlement), and allows long term holders to **earn interest** via lending to liquidity pools.

With cryptocurrency being so pervasive in modern society, it is unlikely that exchanges are going anywhere in the next decade. As the Robin Hood app in the US has proven, there will always be strong demand if you provide a consumer friendly interface to the financial markets.

Challenges

The main challenge at the forefront of consumer's minds is that of security. How can one hold custody of crypto assets safely? In the centralized exchange model, this is done via a combination of "Hot" wallets (cryptocurrency wallets in active use by production servers), and "Cold" wallets (cryptocurrency secured by offline wallets). Access to private keys via rogue employees or spear phishing attempts are also potential attack vectors, and these can be mitigated either via multi-signature wallets or secure multiparty computation techniques where private keys can be secured in physically separate locations, and only with access to multiple keys can one access the wallet.

In Singapore's climate, another challenge would be regulatory compliance. While the Monetary Authority of Singapore (MAS) is generally friendly to cryptocurrency related ventures, securing regulatory approval to operate an exchange may be difficult. Various regulations such as Know Your Customer (KYC) and Anti Money Laundering (AML) laws will need to be followed.

Technical challenges

While the relative complexity of implementing the AMM model is much less compared to the CLOB, there are a number of disadvantages. If the commonly used constant product AMM formula is used, in situations of poor liquidity, slippage when trading can be extremely high. Newer on-chain exchanges try to solve this by either implementing concentrated liquidity algorithms (c.f. UniswapV3) or by implementing special formulas such as the stableswap invariant used by Curve.fi, which in a nutshell decreases slippage for traders while improving capital efficiency for liquidity providers (LPs), with the trade off of amplifying impermanent loss (IL).

CLOBs generally do not suffer much from this slippage issue as liquidity is concentrated around the best bid/offer book prices by the very nature of end users placing bids and orders around current market price.

Lastly, as market volume grows, transaction throughput will also be a major challenge. This is largely due to the fact that the calculation cannot be parallelised, and all orders for a given pair must be executed sequentially.

Competition analysis

Coinhako
A truly Singapore-native cryptocurrency exchange that does not provide a CLOB but instead a simplified buy/sell interface. Unclear what settlement model they utilize. Probably the most lay-person friendly Singaporean exchange with extremely easy SGD on-ramps (including GrabPay). Recently released interest-bearing products, though again unclear where yield comes from, and with relatively low APYs for the crypto space. Charges relatively high fees to the end user.

FTX
A relatively new (2 years old) global cryptocurrency exchange that has quickly grown to become one of the largest exchanges in the world. Geared towards more advanced users, FTX uses the traditional CLOB model and offers a large range of spot and derivatives. On-ramping for Singaporeans is comparatively difficult relative to Coinhako, but they do accept USD debit card deposits or a variety of fiat bank wires including SGD.

DyDx
A truly unique cryptocurrency derivatives exchange, DyDx operates a hybrid decentralized/centralized model. Asset management, custody, and settlement are all on-chain, and yet their interface and API are that of a centralized exchange with none of the pains of the average decentralized exchanges (e.g. high gas fees). While it is the most difficult to on-ramp for the layperson and thus serves a different target audience, they are a potential “competitor” in the sense that the hybrid on-chain/off-chain model would be the gold standard for this project to transition to in the future.

Features

Project Layout

This project uses the `yarn` package manager and its workspaces feature to segregate the client and server projects. Two projects are defined: `@soba/client` and `@soba/server`. These projects leverage Typescript, which is a superset of Javascript that introduces a structural type system, to reduce run-time bugs by catching common errors at compile time.

The client React Single Page Application (SPA) is bootstrapped with the `create-react-app` template. The server project was built from the ground up with `express.js` and containerised with Docker.

Frontend Features

Theming and Design

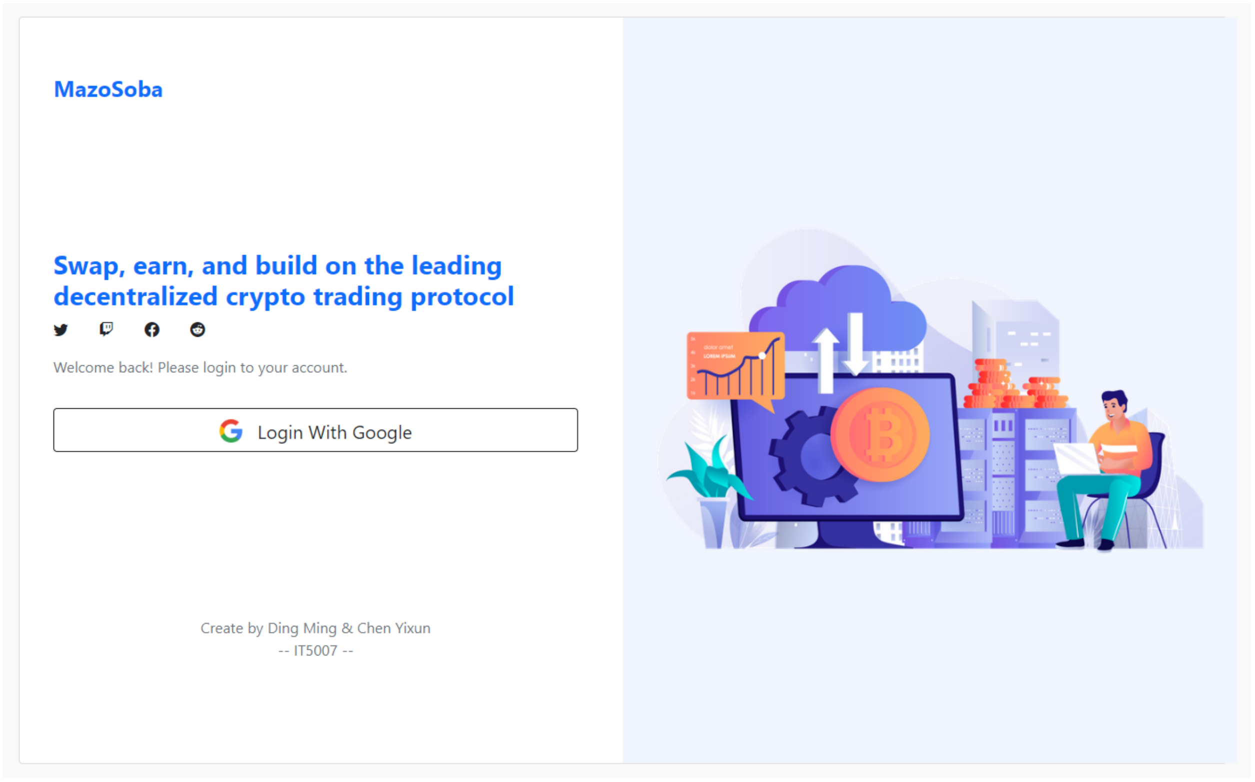
The application is designed with a theme of bootstrap primary blue (`#0d6efd)` mix with light shades of blue and gray. The default bootstrap font is used.

All the pages in the application are made up of a Navbar on top to handle the navigation between pages and a console at the center of the page where the user carries out different operations.

Login Page Features

This page handles user authentication.

1. Login in with Google email
    a. Any user can login and use this application through a 1 click google sign in.
    b. Once the user authenticates successfully, his/her credentials will be saved in local storage and persist between sessions. Subsequent visits to the website will not require logging in again.
    c. Authentication is handled via Firebase via means of JSON Web Tokens (JWT). This allows for easy front-end only authentication and stateless backend code, as handling of JWT tokens is done by Firebase.
    d. Clickable social media icons



MazoSoba

Swap, earn, and build on the leading decentralized crypto trading protocol

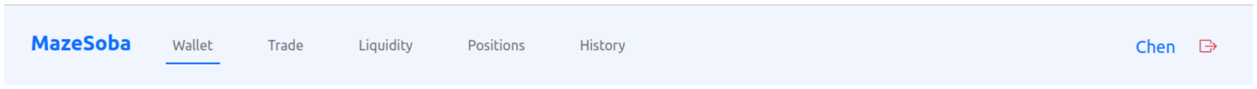
Welcome back! Please login to your account.

Login With Google

Create by Ding Ming & Chen Yixun
-- IT5007 --

Navbar Features

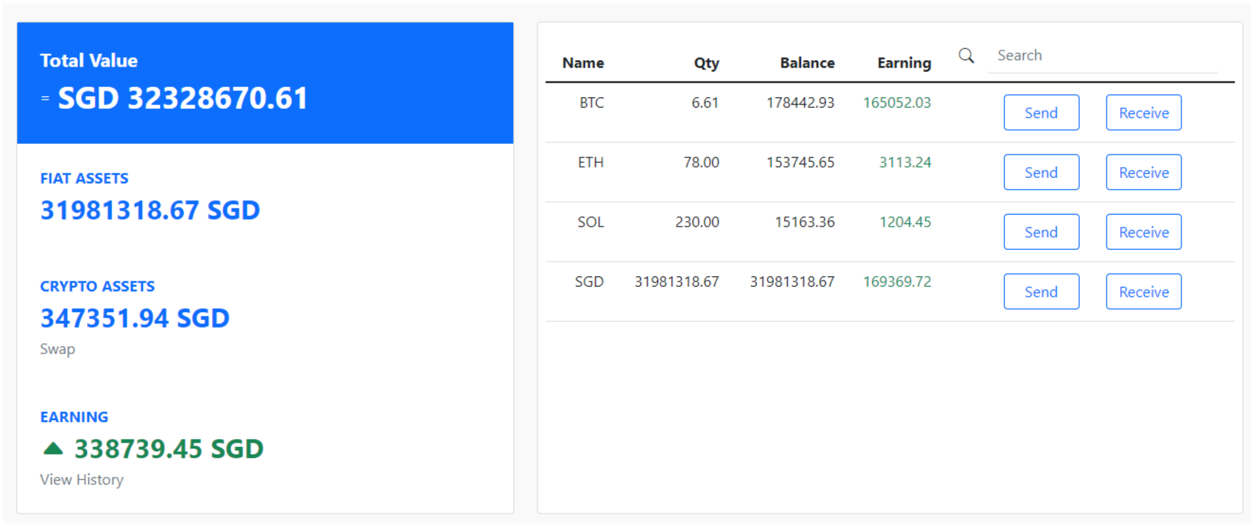
This component handles page navigation.



1. Click on the logo to return to homepage.
2. Tabs to navigate between pages. The active tab is underlined.
3. Display the currently logged in user.
4. A logout button to end current session and clear local storage

Wallet Page Features

The wallet page is split into 2 main containers. On the left, there is a console to display current user's fiat assets, crypto assets and earnings. On the right, there is a table to display the currency that the user holds.


Total Value

= SGD 32328670.61

FIAT ASSETS

31981318.67 SGD

CRYPTO ASSETS

347351.94 SGD

Swap

EARNING

▲ 338739.45 SGD

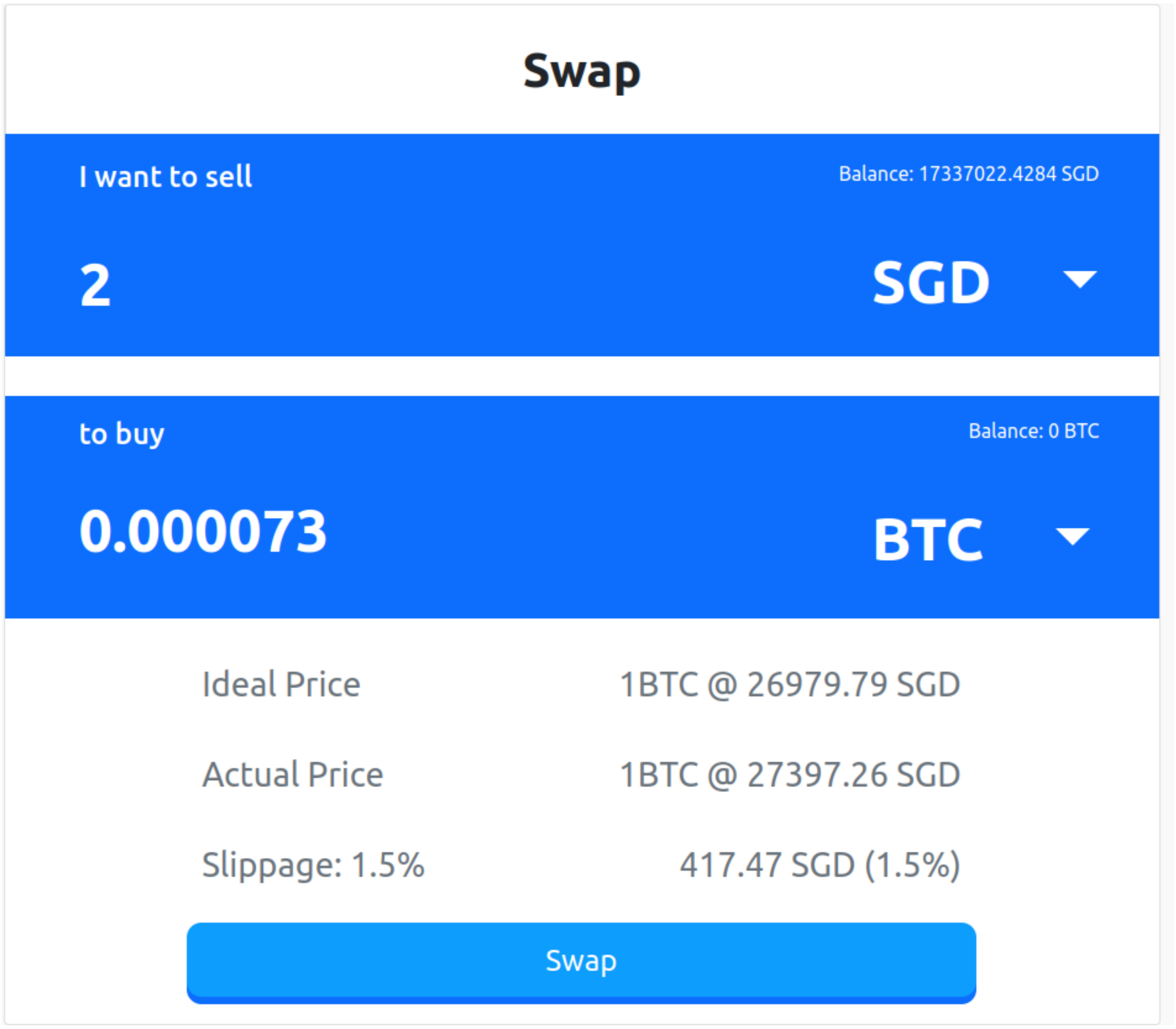
View History

Name	Qty	Balance	Earning		
BTC	6.61	178442.93	165052.03	Send	Receive
ETH	78.00	153745.65	3113.24	Send	Receive
SOL	230.00	15163.36	1204.45	Send	Receive
SGD	31981318.67	31981318.67	169369.72	Send	Receive

1. When users log in for the first time, a welcome modal will pop out that offers you some quick funds to allow easy exploration of the UI.
2. This page is fully responsive, its layout will change based on different screen sizes
3. Wallet console (on the left)
   a. Quick navigation buttons to the specific functions in the wallet console
4. Wallet table (on the right)
   a. List of available currency
   b. Y Overflow is control by a scroll bar
   c. A search input field to filter the currency by name
   d. Nested send and receive buttons to open a modal that contains information/steps on how to receive and send currency
5. Terminology
   a. Total Value
      i. Total fiat value of all assets the user controls
   b. Fiat assets
      i. Value of the fiat assets that make up the total value
   c. Crypto assets
      i. Value of the crypto assets that make up the total value
   d. Earning
      i. Value of the crypto+fiat assets that the user has currently staked in liquidity pools

Trade Page Features

This page handles the buy and sell of funds from one currency to another. *currently only supports the exchange of crypto tokens with SGD, vice versa.



Swap

I want to sell

Balance: 17337022.4284 SGD

2 SGD

to buy

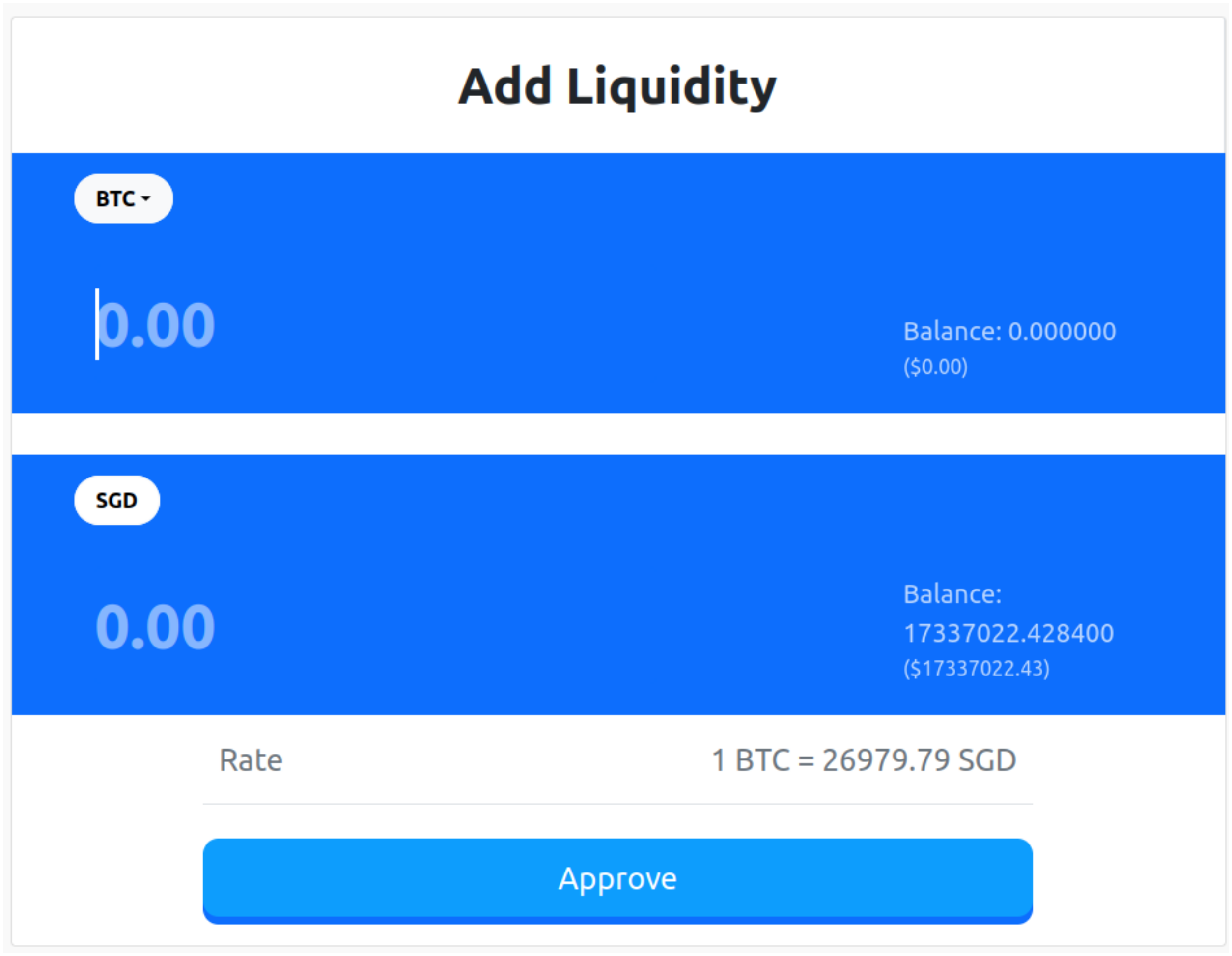
Balance: 0 BTC

0.000073 BTC

Ideal Price	1BTC @ 26979.79 SGD
Actual Price	1BTC @ 27397.26 SGD
Slippage: 1.5%	417.47 SGD (1.5%)

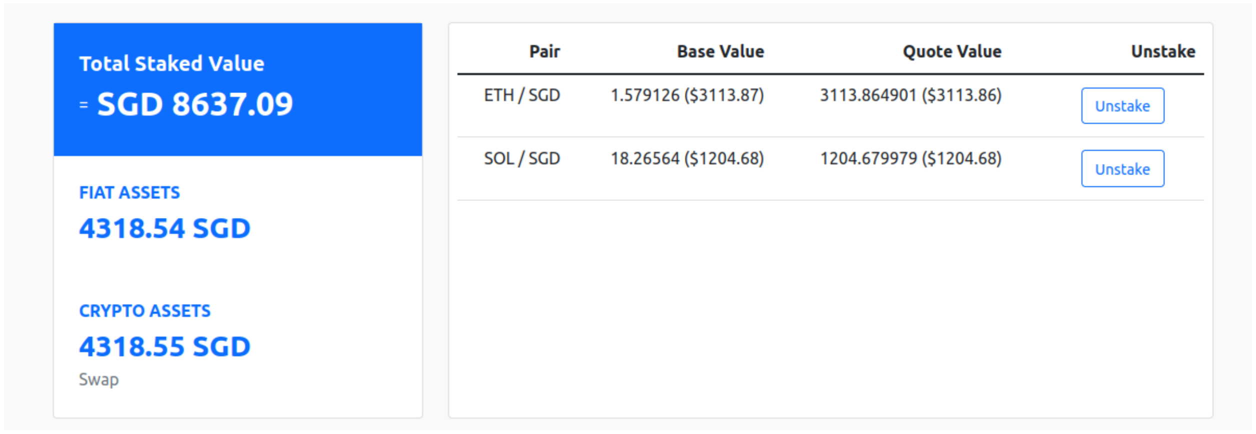
1. Dynamic quotation of price and exchange rate, once the upper input field is filled, a query is made to the back end to calculate the output amount, ideal price, actual price, and the associated slippage
    a. Quotation changes if the exchange to currency changes
2. Dynamic display of price, slippage and confirmation button
3. Dynamic filtering of coin dropdowns based on any given selections
4. Failed and success exchange transaction notice

Liquidity Page Features



In order for the user to earn interest on their idle crypto assets, we also allow users to stake their idle assets into the exchange’s liquidity pools, earning trading fees made by other end users. The boxes are separated into Base Currency (top) and Quote Currency (bottom), where the quote is always SGD and the base is any available crypto asset. Liquidity must be provided in a 50:50 ratio to avoid impermanent loss, and the UI calculates this for you and fills in the bottom field accordingly.

Positions Page


Total Staked Value
≈ SGD 8637.09

FIAT ASSETS
4318.54 SGD

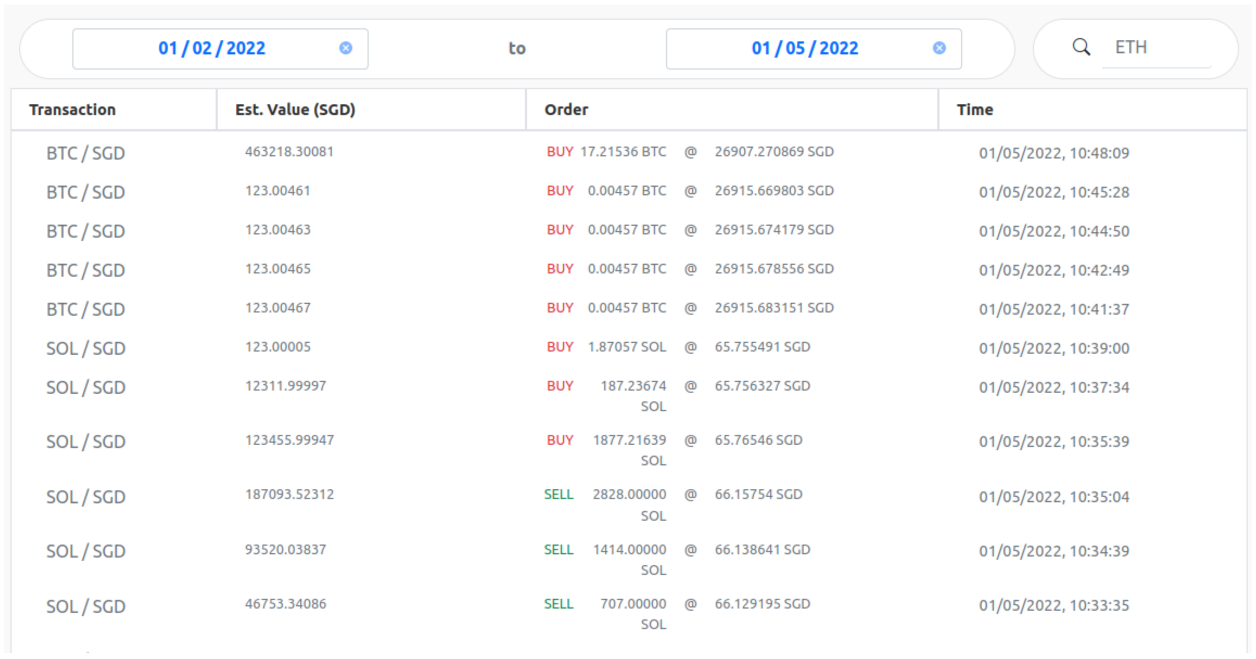
CRYPTO ASSETS
4318.55 SGD
Swap

Pair	Base Value	Quote Value	Unstake
ETH / SGD	1.579126 ($3113.87)	3113.864901 ($3113.86)	Unstake
SOL / SGD	18.26564 ($1204.68)	1204.679979 ($1204.68)	Unstake

This page displays the assets that the user currently has staked in liquidity pools. It also implements an Unstake button to allow users to remove their currencies from the liquidity pools and back into their wallet.

History Page Features

This page displays past transaction history


01 / 02 / 2022 to 01 / 05 / 2022 ETH

Transaction	Est. Value (SGD)	Order	Time
BTC / SGD	463218.30081	BUY 17.21536 BTC @ 26907.270869 SGD	01/05/2022, 10:48:09
BTC / SGD	123.00461	BUY 0.00457 BTC @ 26915.669803 SGD	01/05/2022, 10:45:28
BTC / SGD	123.00463	BUY 0.00457 BTC @ 26915.674179 SGD	01/05/2022, 10:44:50
BTC / SGD	123.00465	BUY 0.00457 BTC @ 26915.678556 SGD	01/05/2022, 10:42:49
BTC / SGD	123.00467	BUY 0.00457 BTC @ 26915.683151 SGD	01/05/2022, 10:41:37
SOL / SGD	123.00005	BUY 1.87057 SOL @ 65.755491 SGD	01/05/2022, 10:39:00
SOL / SGD	12311.99997	BUY 187.23674 SOL @ 65.756327 SGD	01/05/2022, 10:37:34
SOL / SGD	123455.99947	BUY 1877.21639 SOL @ 65.76546 SGD	01/05/2022, 10:35:39
SOL / SGD	187093.52312	SELL 2828.00000 SOL @ 66.15754 SGD	01/05/2022, 10:35:04
SOL / SGD	93520.03837	SELL 1414.00000 SOL @ 66.138641 SGD	01/05/2022, 10:34:39
SOL / SGD	46753.34086	SELL 707.00000 SOL @ 66.129195 SGD	01/05/2022, 10:33:35

1. Transaction history filtered by date range.
    a. By default, last 3 months transactions are shown
2. The transactions are ordered by transaction date
3. Filtering of transactions history by currency symbol
4. Different color signals for buy and sell orders
5. Displays transaction quantity and the price for each transaction

Backend Features

The backend consists of two main parts, an `express.js` based REST endpoint, and an SQL (MySQL) database running on Google Cloud SQL. The server is built with typescript, and in the post-build script, non-typescript files such as SQL files are copied over to the build directory with `shell.js`.

Initialisation

The entry point to the server is `index.ts` in the `src` folder. In production, this would have been transpiled to an `index.js` Javascript file in the `dist` folder. All necessary initialisation values such as database host, port, credentials files, etc are read via environment variables. In local development, these are injected via the `dotenv` package and a local `.env` file in the project root. In production, these are injected by the Google Cloud Run environment.

Authentication

We leverage Firebase to perform all user authentication by means of JSON Web Tokens (JWT). This allows the server to be completely stateless. If user load grows high, it would be trivial to spin up parallel containers due its stateless nature. A custom middleware `isLoggedInMiddleware` and `isAdminMiddleware` defined in `src/auth.ts` help to guard all endpoints that require authentication. These middleware verify the JWT transmitted via HTTP Bearer Authentication headers. If valid, it attaches the decoded user token to the request object and continues down the middleware stack. If invalid, it ends the request and returns a HTTP 401 Unauthorized response.

Datastore

The back end interfaces with a MySQL database, hosted by Google Cloud SQL. Authentication with the backend is done via Google Service Accounts; in production, the `node.js` app sees this as a Unix socket mounted at `/cloudsql`. In development, the database can be connected via said Unix socket or tcp at localhost via Google `cloud_sql_proxy`.

All table schema files are found under `src/model/sql/*.sql`.

Routing

As Firebase handles all front-end related endpoints, the back-end only needs to handle the REST api endpoints. These endpoints are defined in `index.ts`. We define the following endpoints:

```
- GET  /api/debug/initialise
    - Admin only
    - Sets up the database and some initial balances
- GET  /api/debug/funds
```

```
        -  Admin only
        -  Gives funds to the currently logged in admin for development
           purposes
    -  POST /api/airdrop
        -  Allows new users to claim free funds for testing
    -  GET  /api/wallet
        -  Returns the user’s entire balance, including staked tokens
    -  GET  /api/getStaked
        -  Returns the pools the user has currently staked, and the balance
           of Liquidity Provider (LP) tokens, which represents the user’s
           share of the underlying liquidity pool
    -  GET  /api/getLiquidityValue
        -  Returns the value of the user’s staked tokens in base and quote
           currency
    -  GET  /api/prices
        -  When given a list of currency symbols via URL query parameters,
           returns a list of ideal prices for these assets.
    -  POST /api/swap
        -  Performs a swap for the user
    -  GET  /api/history
        -  Returns the transaction history for the user
    -  POST /api/withdraw
        -  Submits a withdrawal request for the user
    -  GET  /api/deposit
        -  Returns the blockchain address that a user should send funds to
           to deposit onto the exchange
    -  POST /api/stake
        -  Allows the user to stake tokens to provide liquidity
    -  POST /api/unstake
        -  Allows the user to unstake LP tokens
    -  GET  /api/quote
        -  Fetches a quote for a swap when provided parameters via URL query
           params
    -  GET  /api/pairs
        -  No authentication required
        -  Returns a list of all pairs traded on the exchange
```

Features

Airdrop

The database keeps track of which users have claimed an arbitrary airdrop, allowing the back end to disburse funds as a once-off.

Deposit / Withdraw

To deposit, users send funds to the crypto address provided by the REST endpoint. The address would theoretically be scraped regularly to check for incoming transactions. For withdrawals, the user submits withdrawal requests which are stored in the database with a PENDING status. Admins would then verify the withdrawal requests before allowing them through.

Matching Engine

The exchange matching engine utilises the Automated Market Maker (AMM) model to perform its trades. This model allows the trade to be near stateless, only requiring the data store to store balances and liquidity pool reserves, rather than complicated data structures of end user orders. Matching is done against the pool rather than other users. In order to ensure atomicity, all transactions leverage SQL and a traditional RDBMS (MySQL) to ensure ACID compliance. We ensure that users have enough balance, or that pools have enough liquidity, via means of MySQL Check Constraints (`CHECK (amount >= 0)`). Should these constraints be violated, the transaction is rolled back and the appropriate error message shown to the user.

The AMM follows the battle tested UniswapV2 constant product formula `x*y=k` algorithm to calculate all trades.

Deployment

Configuration files in the root allow the project to be deployed in the cloud. This project leverages three Platform-as-a-Service technologies: Firebase, Google Cloud Run, Google Cloud SQL.

Firebase

The front end is a SPA, so deployment to Firebase is trivial. The route and build path definitions are defined in `firebase.json` in the root. The admin secrets to the Firebase Service Account (which authenticates against both Firebase Auth and the database) is stored in `firebase-admin.json` in the project root. This file is not in source control, and in production is supplied by the Google Cloud Run environment. Deployment is currently done manually via running `firebase deploy` in the terminal.

Google Cloud Run

Cloud Run allows running of any arbitrary Docker image. This project is dockerised by means of the Dockerfile in the project root. The back-end is given a service id of `sobaapi`, and routing from the front end to the back end is handled by Firebase, as defined in firebase.json. Deployment is currently done manually via running `gcloud run deploy sobaapi --source .` in the terminal

Google Cloud SQL

Cloud SQL manages our MySQL instance. Only the back-end talks with the database, which performs requests on behalf of the front-end SPA. Authentication is handled via service accounts defined in `firebase-admin.json`. These service accounts are given the appropriate roles in the Google Cloud Console GUI.

Improvement Plan

Immediate improvements that could be considered would be making the UI mobile responsive. Currently the UI is designed for a desktop application, and the user experience suffers on mobile.

Other improvements could be the inclusion of the concept of “locked” balances on the backend. As an example, currently it is possible for the user to submit multiple withdrawal requests without the relevant balance being “locked”, i.e. the user is still able to trade. In practice this is not an issue as the request would still be verified before being processed, but it does not make for a very good user experience. Another improvement would be the inclusion of withdrawal request history lists.

Another feature for future improvement would be an overview page of all available liquidity pools and their associated statistics such as Total Value Locked, APY, etc. At the moment, it is impossible for a user to gauge what kind of interest they could earn via staking their currency. Viewing all pools also is not a great user experience as there is no overview page.

For the matching engine, we expect performance to degrade at high transaction volumes due to the high usage of joins in the queries. Should this prove to be an issue, one option would be to move to a high speed key-value store that still supports transactions such as Redis.