Faculty of Computer
Science and Engineering

University of Ss. Cyril
Methodius

Requirements Specification for Stock Market Analytics

An app for analysis and prediction of the stock market

Submitted by:
-Matej Josifov 221114
-Mihail Gjorgjievski 221242

Under the supervision of:
Dr. Antovski Ljupcho

1. FUNCTIONAL REQUIREMENTS

1.1 This application is accountable for gathering financial data from different sources like Yahoo Finance and Alpha Vantage.

1.2 This application is processing the data in real-time using Redis for the streaming of the data through different channels and Celery for their processing and Celery Beat for scheduling the Celery tasks.

1.3 This application will save the data in a relational database such as PostreSQL.

1.4 This application enables fast access to the data for analysis and visualization.

1.5 This application is analyzing the data using algorithms of machine learning and statistical analysis using the library Pandas.

1.6 This application is using services and generates trends, predictions and logs based on the historic data and real-time data.

1.7 This application is providing a Web-interface for a better user-friendly environment and is visualizing the data using a library like plotly.

1.8 This application is communication with the Data Analysis Service to display the analyzed data in the form of graphs and tables.

1.9 This application uses Python as the choice of work environment.

1.10 This application is fully Docker containerized and is hosted on the domain xxx

1.11 The user will have the ability to choose for what company to view the historical graphed data and later will be given a choice to generate trends and analyze them.

1.12 The landing page for the viewers will contain the most active stocks for the companies as well as the top gainers and top losers of the day and that statistic will be constantly updated.

1.13 This application will process the data in a batch-fetching structure from the finance library and it will populate the postgreSQL database with 10-years of data.

2. NON FUNCTIONAL REQUIREMENTS

2.1 Performance

This application will use redis in-memory caching on the data that is used frequently by the frontend service or a certain company is being queried multiple times and it will be added to the cache. This data will update quickly and have a short TTL (Time to live) as we will pull data from yfinance every 10 seconds and update the daily trends in the optimal time (yahoo has an internal limit of 2000 requests/hour).

2.2 Scalability and reusability

This application will pull and process the daily data for all companies in a batch and it will update the best/daily trends in an interval of 60 seconds and for every next pull the current data in redis will be stored in the postgreSQL database to optimize requests.

Workflow of the app to optimize scalability:

- User makes a request for the analysis of data for a given company.
- Concurrent thread accepts the request by a given user, after it checks the redis-database for a hit on a given company ticker.
- If there is a hit on the historical data in the redis-database we will return that data, and if there isn't a hit in the redis database we will return the data from the PostgreSQL database.
- A real-time processing service pulls the data from the cache memory and makes a dataframe on it using machine learning and calculates trends and common scenarios.
- The frontend gets the data and returns a view to the user with the historical data of a current stock for the company and the generated trend by a real-time processing service.

2.4 Maintainability

This application will be divided on multiple different services (some of them real-time services) for which there will be a different docker-container so the app will be easier to manage and maintain.

2.5 Configurability

This application will be configured in such a way that the redis database will be linked on a different port as from the postgreSQL database, and they will communicate with the Data ingestion service to retrieve data for the user on which they will not have to configure anything on their part.

2.6 Availability and scalability

The application should work concurrently for 1000 requests in a given moment and the server should be available 99.9% of the time.

3. CONTEXTUAL REQUIREMENTS

3.1. Regulatory & Compliance Requirements

- Data access limit : 2000 requests/hour from yahoo so the app will update every 10 seconds
- Stock market regulations : ensure the system won't violate financial regulations upon displaying trends or recommendations .

3.2. Operational & Deployment Requirements

- Cloud and deployment : the system should be hosted on cloud services like AWS or Azure
- Scalability and performance : the system should support multi concurrent users requesting historical stock data and should hold good on spikes in requests because of the caching (redis) and the real-time processing services adding the data to the cache.
- Resilience and fault tolerance : the system should fall back on the data it has saved in the database upon failure fetch from the yahoo API and it should implement retry mechanisms on failed fetch
- Generating trends and predictions : the system will use a real-time processing service with Apache kafka and Apache spark and return the given predictions to the user.

3.3. User & Stakeholder Requirements

- Target audience : Investors, traders and people who often look at the stock market and try to predict/analyze it.
- User accessibility and UI design : the users will have a friendly looking target page with the best/worst trends in the given day, after which they will have the option to select a given company.
- After selecting a given company the user will be able to generate predictions and trends on a given company once every 30 minutes.

3.4. Subscription business model (optional)

- Free basic plan : users can still generate trends but only once every hour.
- Premium customers: will follow the trend of premium customers if you want to generate more than 1 trend in an hour or get a notification for a prediction on a stock.

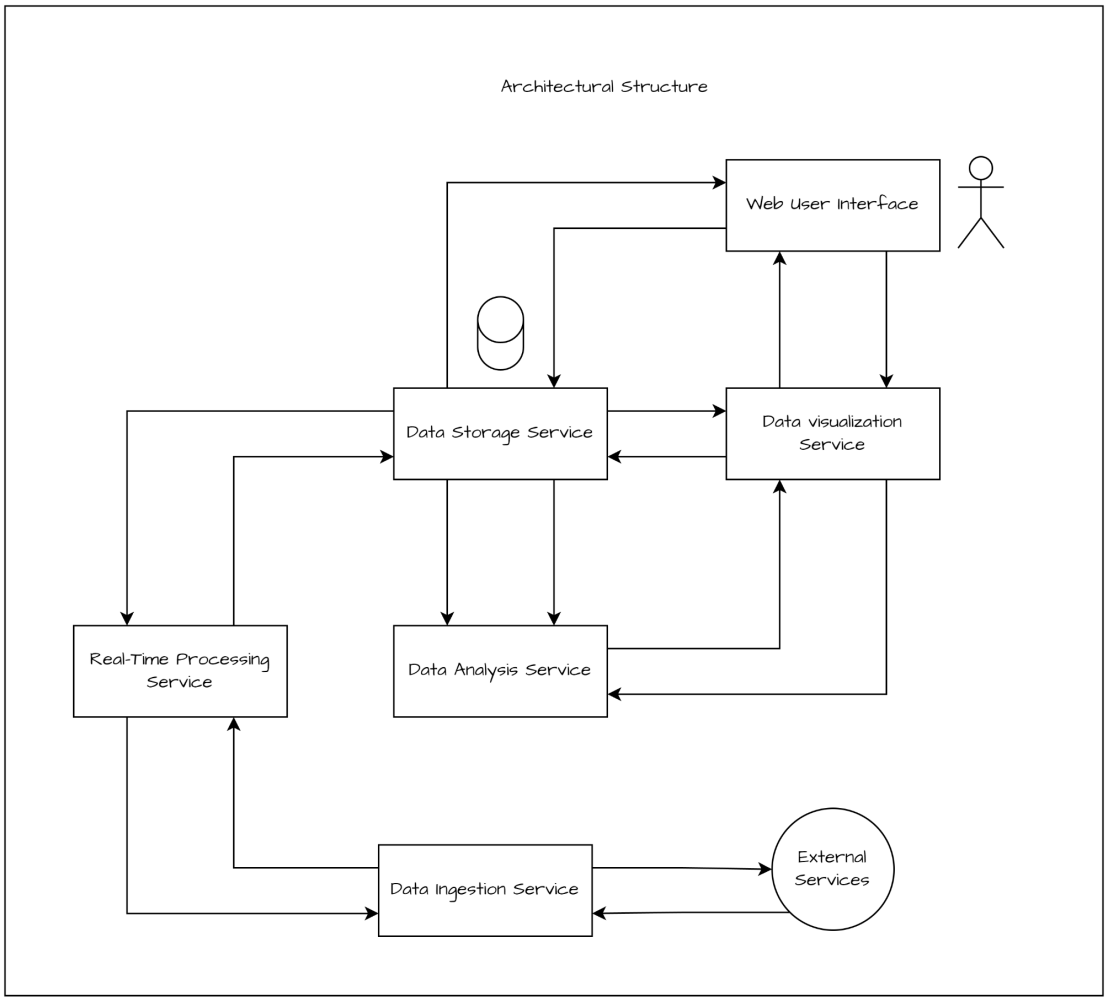
3.5. Organizations of teams

- The application is being worked on by 2 team members Mihail Gjorgjievski and Matej Josifov.
- Both members have worked on production projects and commercially distributed games and are currently studying at the Faculty of computer science and engineering at the University of Ss. Cyril and Methodius.
- Matej is based in Kochani, currently living in the municipality of Kisela voda,Skopje.
- Mihail is based in Sveti Nikole, currently living in the municipality of Aerodrom,Skopje.

4 SCHEDULE REQUIREMENTS

This app will be finished before 20th March 2025.

5. CONCEPTUAL ARCHITECTURE



6. USER USE CASES

6.1 Personas and general use case descriptions

Persona:
John, 25 years old

Regularly trades and likes to keep an eye on stock prices. When he uses the apps for trading he cannot analyze and predict in detail what the future of the stock market will be so he uses this system for analysis and predictions.

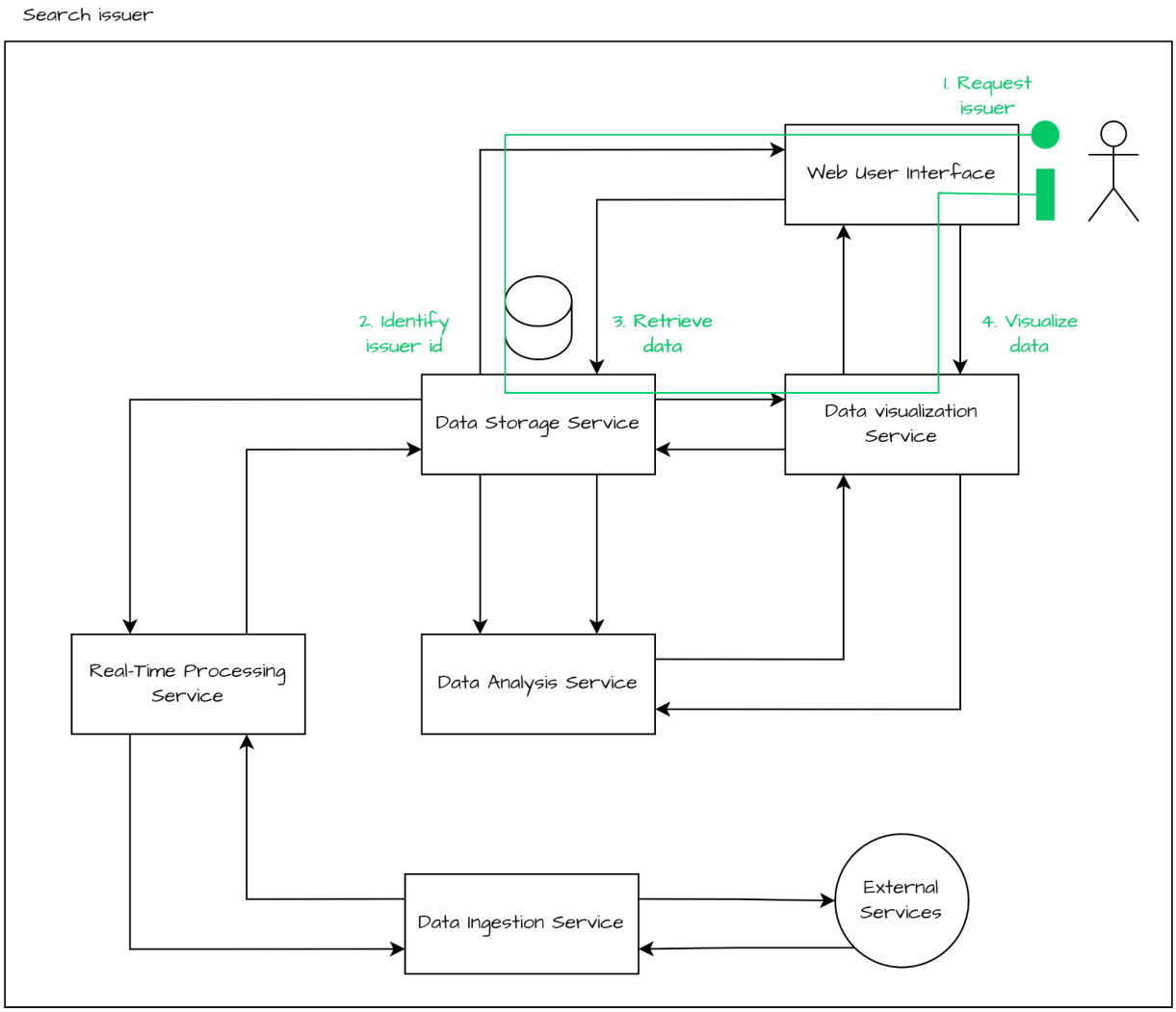
For all his trades he uses his mobile phone which is equipped with chrome browser.

Name: Search route
Aim: User is searching for a specific issuer
Actors: User, System
Precondition: None
Trigger: Users trigger a new search
Includes:
1 User starts a new search
2 System shows results
3 User specifies the preferred choice
Extensions:
None

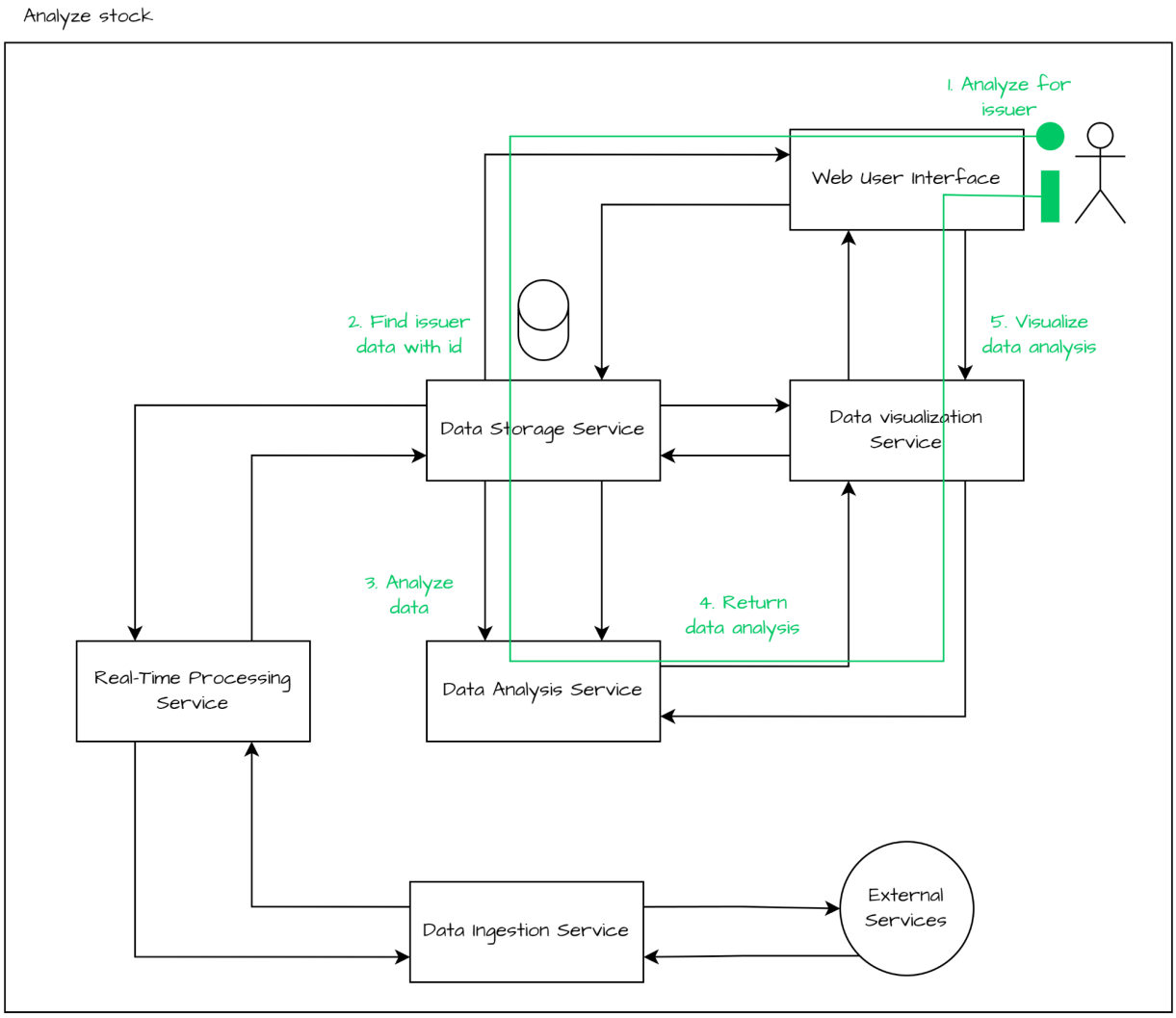
Name: Analyze stock
Aim: User is analyzing a specific issuer
Actors: User, System
Precondition: User has found desired issuer
Trigger: Users triggers a new analyze process
Includes:
1 User clicks analyze
2 System shows results
Extensions:
None

Name: Predict action
Aim: User is predicting if he should trade stocks from a specific issuer
Actors: User, System
Precondition: User has found desired issuer
Trigger: Users triggers a new prediction process
Includes:
1 User clicks predict
2 System shows results
Extensions:
None

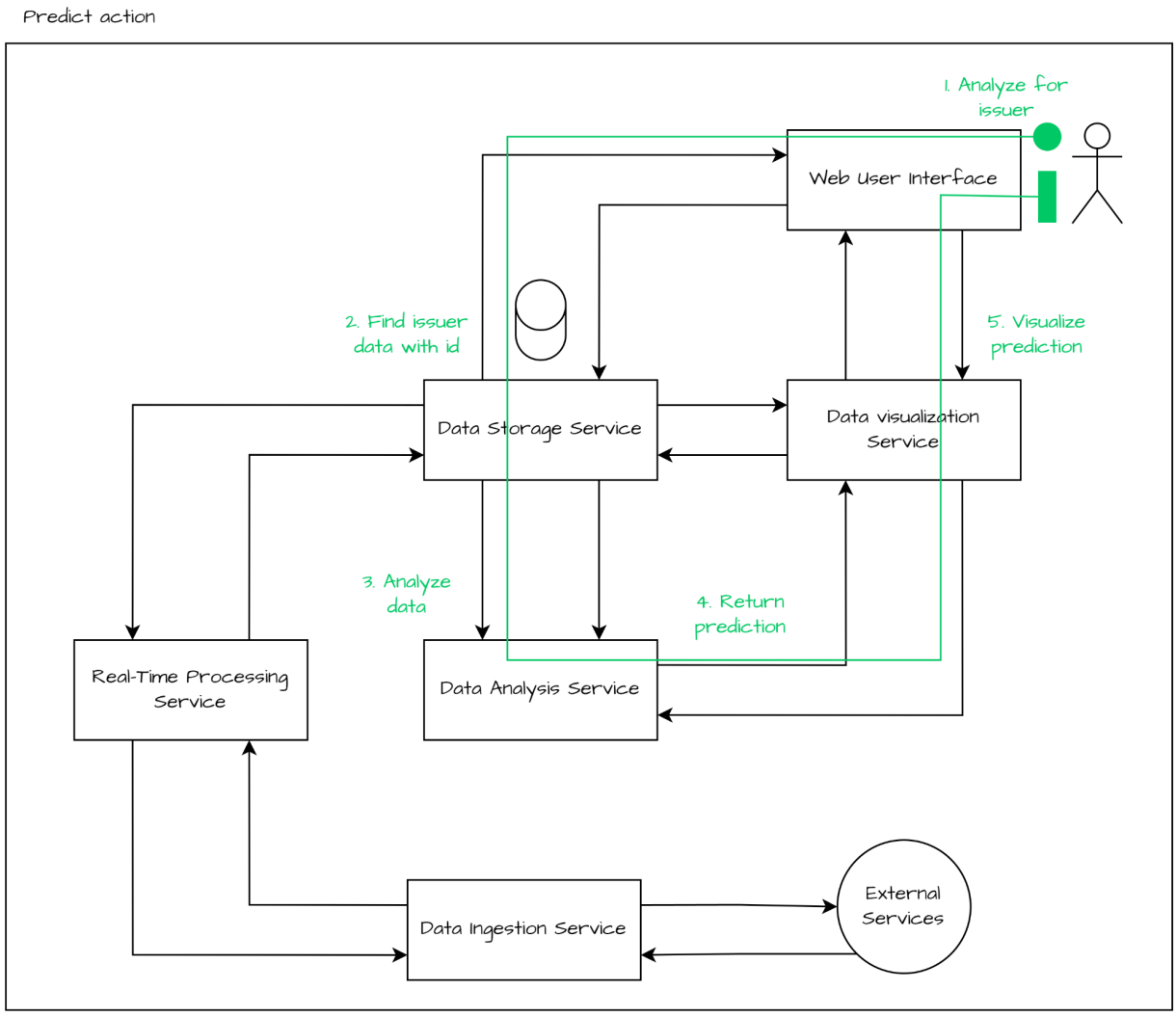
6.2 Behavioural exploration of use case search issues



6.3 Behavioural exploration of use case analyze stock



6.4 Behavioural exploration of use case predict action



7.KEY CONCEPTS AND CATEGORIZATION

7.1 Key Concepts

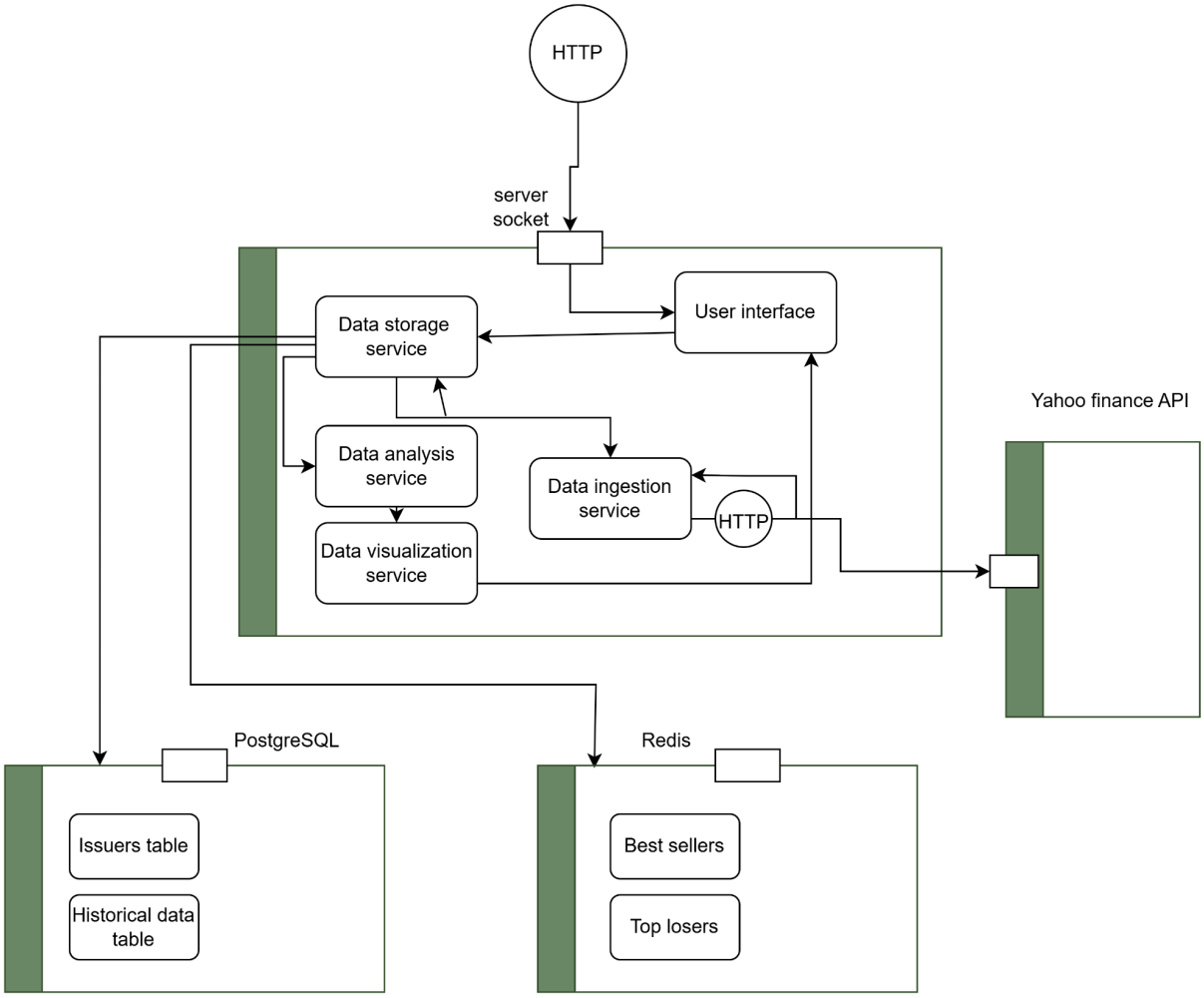
- Review a stock - Abstract concept
- Calculate SMA - Abstract concept
- Calculate EMA - Abstract concept
- Make Prediction - Abstract concept
- Generate trend - Abstract concept
- Generate helping flag - Abstract concept
- Stock entry - Data
- PostgreSQL database - Data
- Redis database - Data
- Company codes/tickers - Data
- Get stock data - Function
- Filter data - Function
- Process data - Function

7.2 Categorization

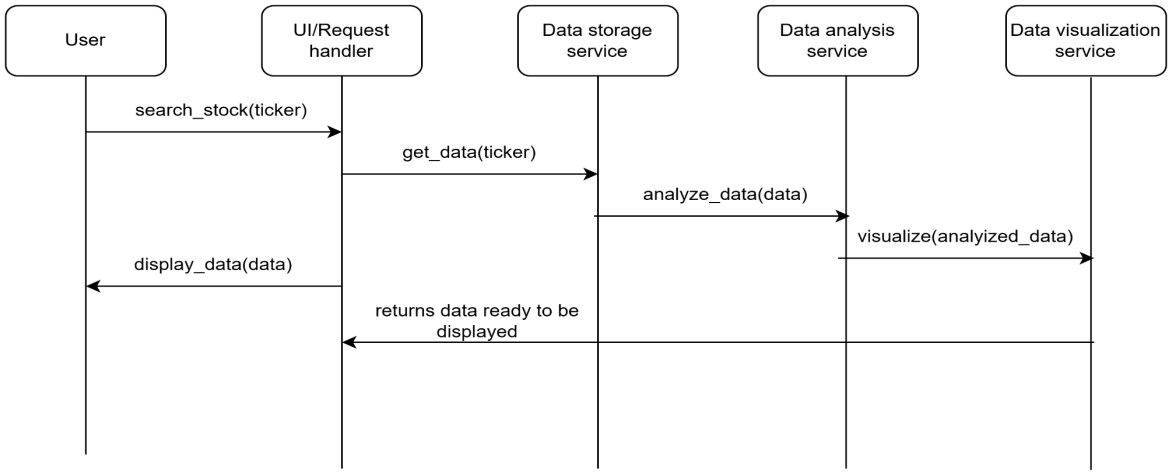
Data	Function	Abstract concept
PostgreSQL	Get stock data	Review a stock
Redis	Filter data	Calculate SMA
Stock entry	Process data	Calculate EMA
Company code/ticker		Make prediction
		Generate trend
		Generate helping flag

8. IMPLEMENTATIONAL ARCHITECTURE

8.1 Interface diagram



8.2 Application sequence diagram



8.3 Data storage service sequence diagram