Amuldeep Dhillon, Hamza Sayyid, Kevin Kuo, Meerza Ahmed, Paul Algozzini, Sarah Yeun, Timothy Lin

Professor Jae Young Bang

CSCI 578: Software Architectures

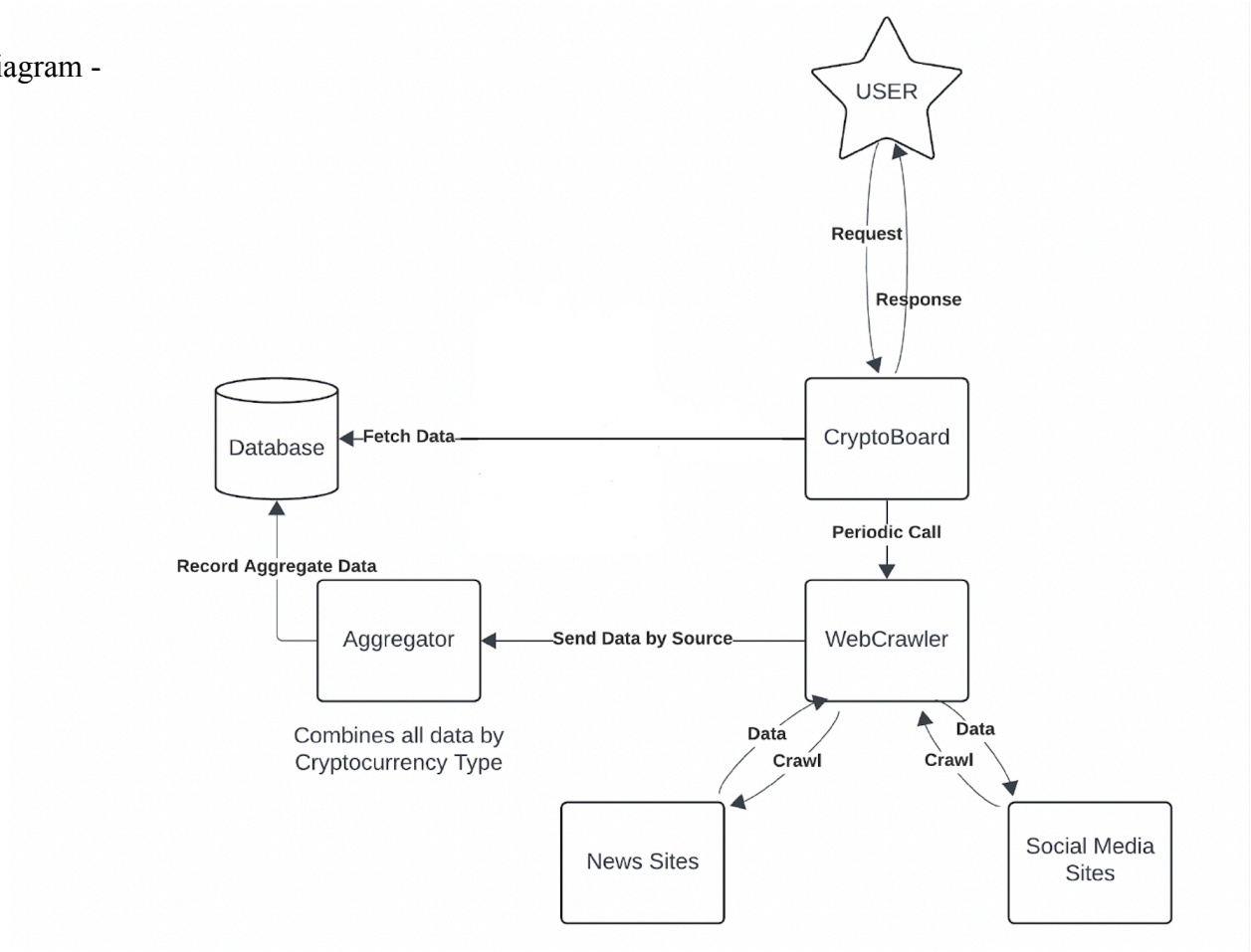
5 December 2024

CSCI 578 Team Project

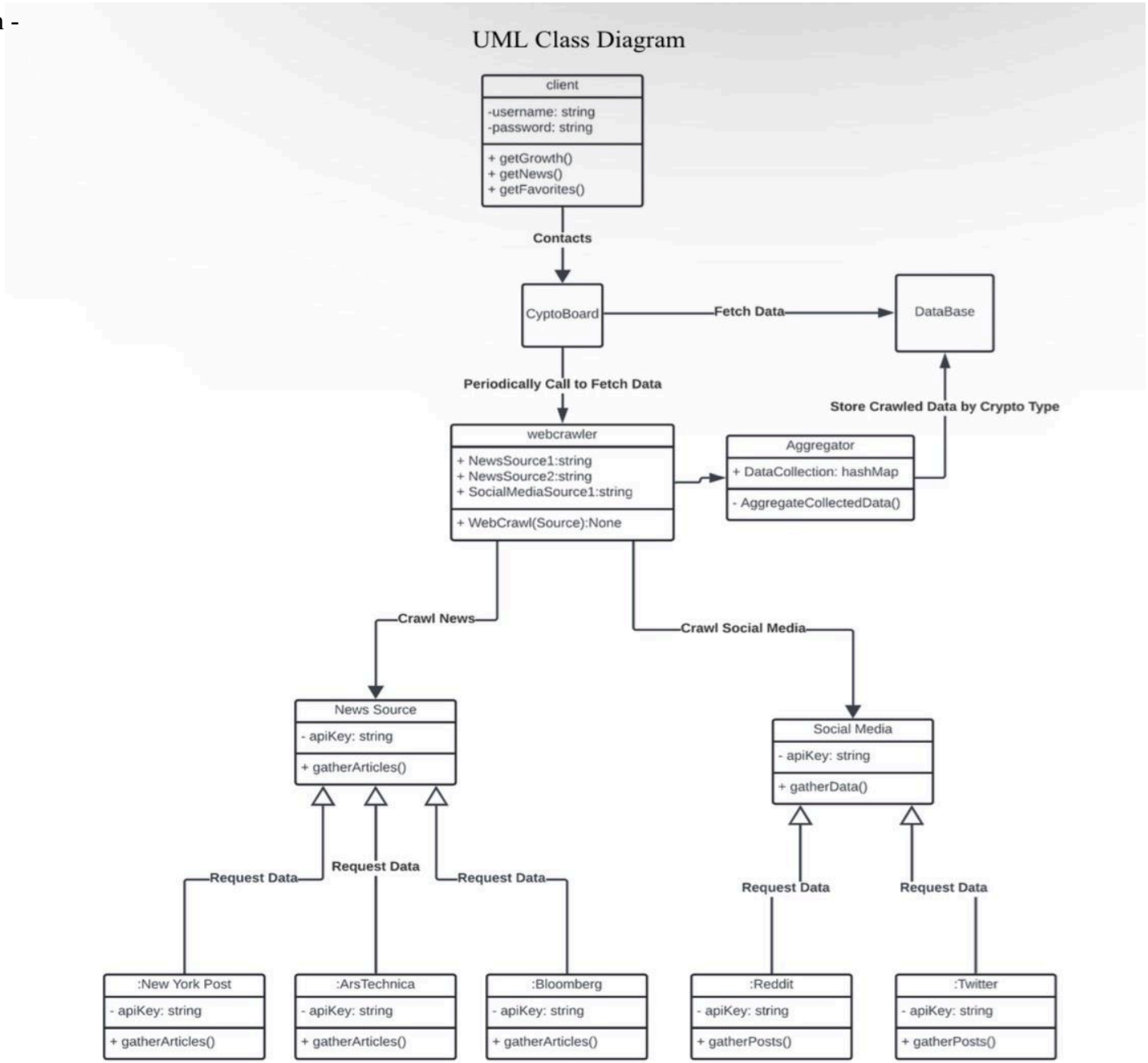
PART 1

UML Diagrams:

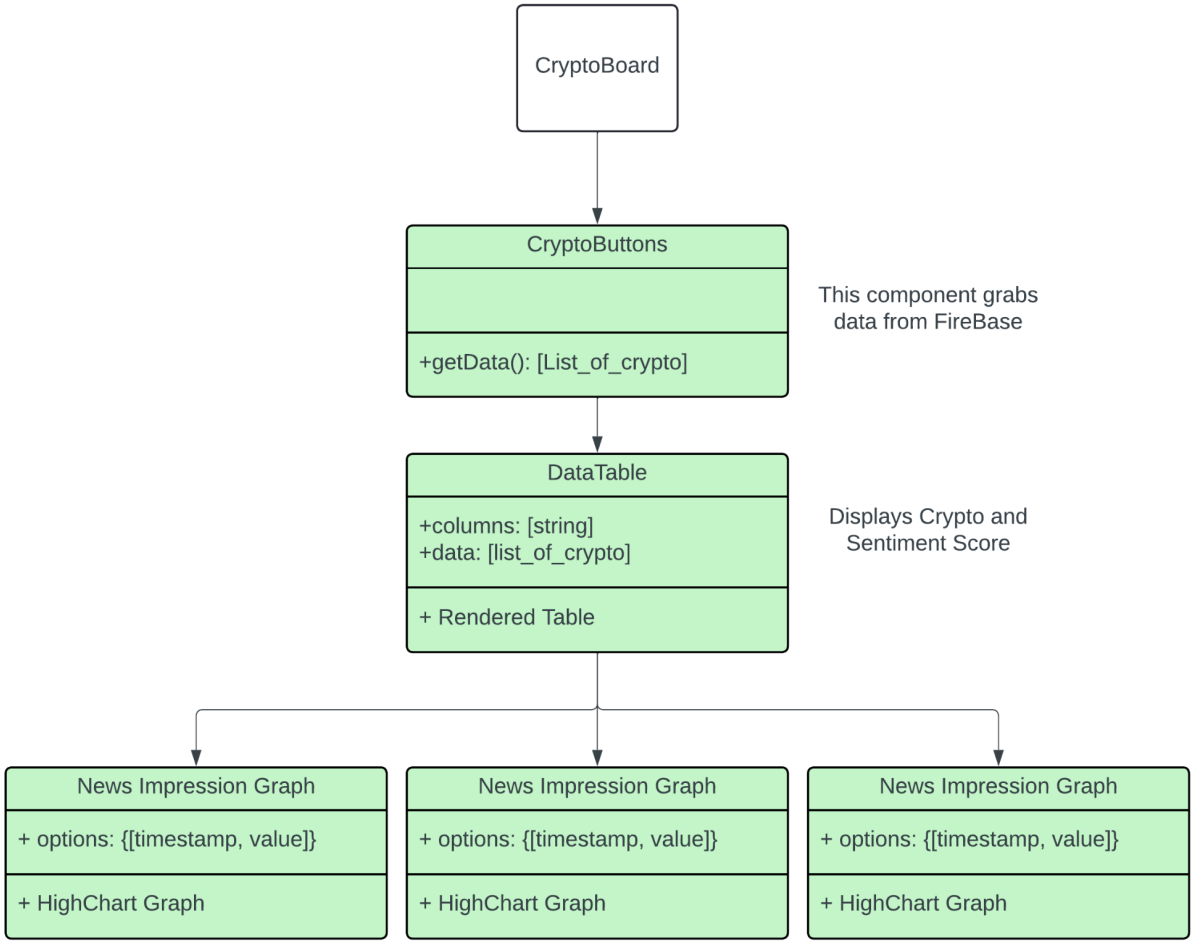
Component Diagram -



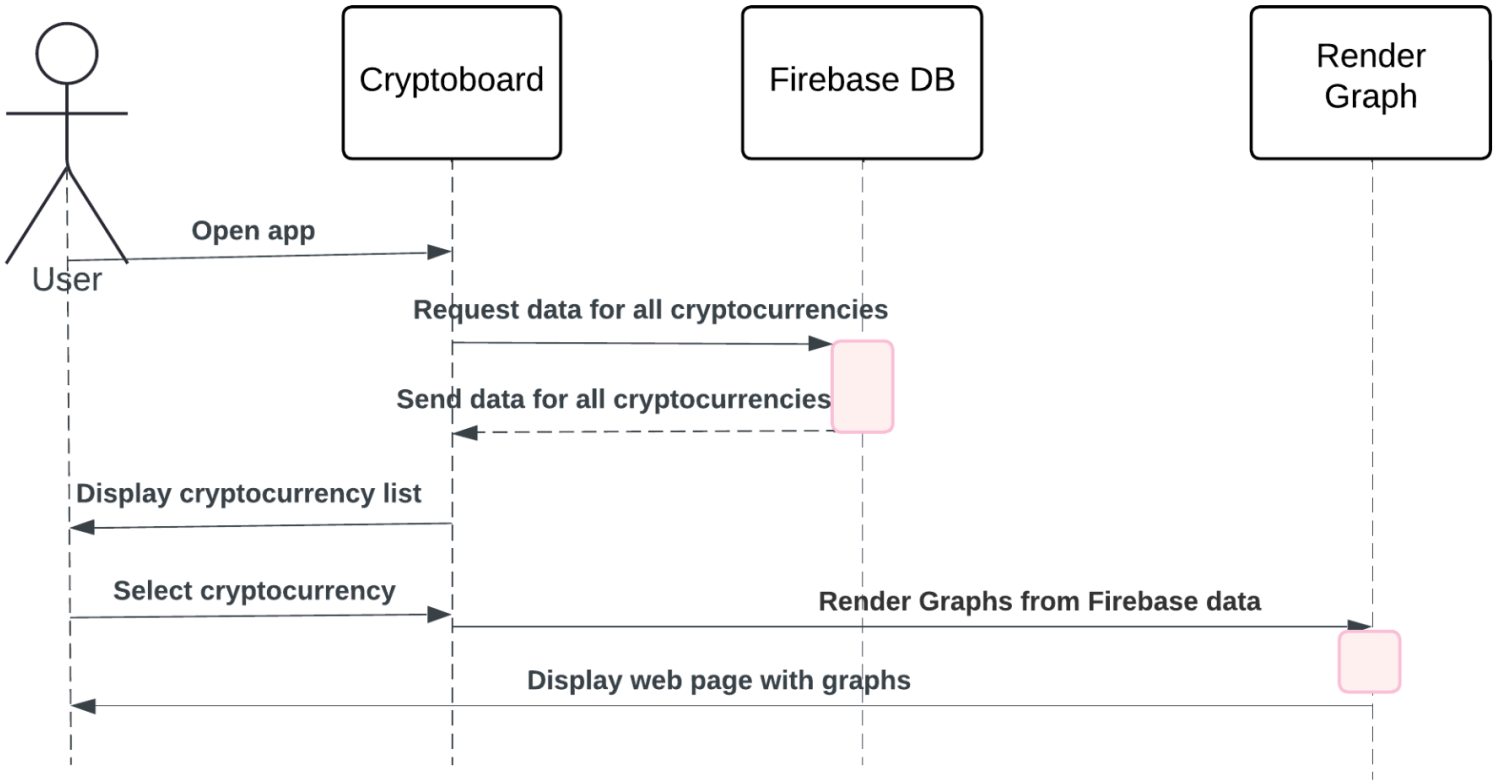
Class Diagram -



Front End Diagram -



Front End Sequence Diagram -



Our architectural design features a web scraping module with three scrapers: two built using Scrapy to crawl news sites and one using PRAW to scrape Reddit. These scrapers operate asynchronously in the background, collecting data on cryptocurrencies in the form of links and associated text. The scraped data is passed through a one-way data pipeline to the aggregator module. This module processes each data item by performing sentiment analysis and then stores the analyzed results in Firebase. The front end retrieves these sentiment analysis results from Firebase to dynamically populate the user interface, ensuring users have up-to-date and relevant insights. This structure enables seamless integration between the scraping, processing, and display components of the system.

Rationale for Architectural Design and Capability Subset:

When choosing which architectural design to proceed with among the team members, we decided to follow through with this design for the following reasons:

1. Clear Separation of Concerns - The selected architectural design clearly separates data collection, aggregation, and presentation. Data collection is handled by a web crawler that crawls through news sites and social media sources for text information about relevant cryptocurrencies. Data aggregation is handled by a data aggregator module that parses through the collected text information and performs sentiment analysis to generate a score for how favored cryptocurrencies are. Lastly, data presentation is handled by a React front-end interface allowing users to display stats and other information about their selected cryptocurrencies.

2. Periodic Data Retrieval - This design contains a web crawler component that periodically collects data from external sources such as news sites and social media platforms. This decouples data fetching from user requests, reducing the load on external APIs and ensuring fresh data is available when users query the dashboard.

3. Aggregation for Data Integrity - The design contains an aggregator that consolidates the data from the web crawler about various cryptocurrencies, ensuring consistent and unified data representation. This reduces redundancy and prevents inconsistencies if data is fetched and displayed directly from multiple sources.

4. Caching for Performance - This design includes caching layers, ensuring low-latency responses to user queries. We thought this was particularly important for a crypto dashboard, where users expect real-time or near-real-time updates.

5. React Components for Simplicity, Scalability, and Reusability - The design's use of React for the front-end interface emphasizes reusability and scalability. This, combined with React's ease of use and flexibility convinced us it would be well-suited for our project. We used Vite to create a quick workflow with hot module reloading and faster bundle times, Tailwind CSS to simplify the CSS process, with Shadcn acting as a useful UI component library. We used ChatGPT to figure out how to place different Highchart Graph components near each other.

6. Database for Persistence - This design outlines a database we can use to pull data we have previously processed and stored already, reducing instances of re-crawling and re-processing of data.

Although different team members' designs had various overlapping components, we felt this architectural design was the most comprehensive and feasible to implement given the timeline, our members' tech stack experiences, and the CryptoDashboard features we wanted to support.

PART 2

Departures from Architectural Design

The following are deviations from the original architectural design that were made during implementation:

1. We added sentiment analysis by running each article through an AI to calculate a sentiment score per article. We then used these values as a way to rank different crypto currencies and showed the sentiment over time graph.
2. We deviated from using a cache to allow both the front-end and back-end to query Firebase directly for the following reasons:
    - **Faster Development**: Avoiding cache complexities enabled quicker iteration and focus on core functionality.
    - **Real-Time Capabilities**: Firebase's real-time updates are more effectively utilized without cache-induced delays.

- **Low Initial Traffic**: Direct queries are manageable at this stage, making a cache unnecessary for performance or cost efficiency.

3. News sites such as NYTimes and NYPost were excluded from data crawling due to restrictive web crawler policies. Instead, we focused on crypto websites with more permissive policies. Additionally, our scraping was limited to free news sources without paywalls, excluding major networks like NYTimes, WSJ, Business Insider, and Bloomberg. Social media platforms posed even stricter restrictions, so scraping was limited to Reddit using its old web UI.
4. We removed the ability to login or have authentication as implementing these ideas would take a lot more time and effort, and we focused on implementing the three graphs for better user experience.

PART 3

GitHub Repo - Guidelines for how to run our Cryptoboard are in the README

The ZIP file for the source code is also in the submission.

Note: The docker image was too large to upload to the assignments portal. Please download from the following link:

https://drive.google.com/file/d/1dNZOKhNZTYXG1iwHFdTgUdlDMKyGVH9r/view

PART 4

The video is in the ZIP file of our submission.