

Java Juggernauts
Crypto News app

Design Document

Antti Santala
Veeti Salminen
Onni Rinne
Valtteri Rinne

Contents

1.	Project idea	3
1.1.	App idea.....	3
1.2.	Use cases and requirements.....	3
2.	Technologies	3
2.1.	Tools, programming language and framework	4
2.2.	Components	4
3.	Self-evaluation for midterm submission.....	6
3.1	Original plan and design	6
3.2.	Current design and quality	7
3.3.	Anticipated changes.....	7
4.	Group member contributions	8
5.	Use of AI	8
5.1.	UI prototype (code)	8
5.2.	Project planning and Desing document.....	9
5.3.	Midterm AI usage	9
	Document.....	9
	Code	9

1. Project idea

In this chapter we introduce the main idea of the project application and the use cases and requirements.

1.1. App idea

The application is a Java-based cryptocurrency information service that allows users to easily follow the price development of the five largest cryptocurrencies in the market. The app features a dashboard screen that includes three sections: crypto list, crypto detail view with price and volume charts as well as basic information, and a section for major current crypto-related news and adoption topics. The app can be used to follow five top coins, currently: Bitcoin, Ethereum, Tether, XRP and BNB. User is able to select a crypto token, that is displayed in the detail section with historical price and volume data charts. Time frame of the chart can be adjusted. Alongside the chart, the application displays news specifically related to the selected token, allowing users to monitor both market data and relevant external factors in one place. The news section also allows toggling the section to show news about all of the featured cryptos in one place. The goal of the service is to provide a simple and accessible way to stay updated on the most important cryptocurrencies while offering a smooth and interactive user experience.

1.2. Use cases and requirements

The primary use case is to enable users to monitor current and historical cryptocurrency data and stay informed about crypto-related news. Users open the app, browse current crypto news on the side section, select a cryptocurrency from the top 5 list, and then view a price chart and related news. The app focuses on simplicity and fast access to the most relevant market information.

Key Functional Requirements:

- Displaying 5 largest cryptocurrencies by market capitalization
- Show current price for the 5 cryptocurrencies
- Show historical price and volume data up to 365 days for a selected cryptocurrency
- Price history chart with adjustable time range
- Volume chart with adjustable time range
- Show global crypto news
- Show crypto-specific news for the selected cryptocurrency

Non-Functional Requirements:

- Performance: Quick data loading and chart rendering
- Avoid unnecessary API calls
- Usability: Minimalistic UI
- Compatibility: Working on common desktop platforms running Java

2. Technologies

In this chapter we introduce the technologies and the relationships between components used by the application.

2.1. Tools, programming language and framework

The application will be developed using Java as the main programming language due to its wide libraries, cross-platform support, and familiarity within the team. The graphical user interface will be implemented using JavaFX, which enables building interactive views such as the dynamic price charts. Development will be done primarily using Visual Studio Code, supported by JavaFX extensions and build system support. GitLab will be used for version control and collaboration, branching workflow, and continuous integration features if needed. Team communication will take place via Microsoft Teams and Telegram to ensure efficient coordination during development.

For real-time cryptocurrency pricing and historical chart data, we will integrate the CoinGecko KeylessApi, a reliable, public API widely used for crypto market data. For news retrieval, we will use Google SERP API, which enables executing automated search queries and retrieving news articles related to either general cryptocurrency topics (on the home page) or a specific token selected by the user. JSON responses from both APIs will be parsed with a suitable Java JSON library such as GSON or Jackson.

2.2. Components

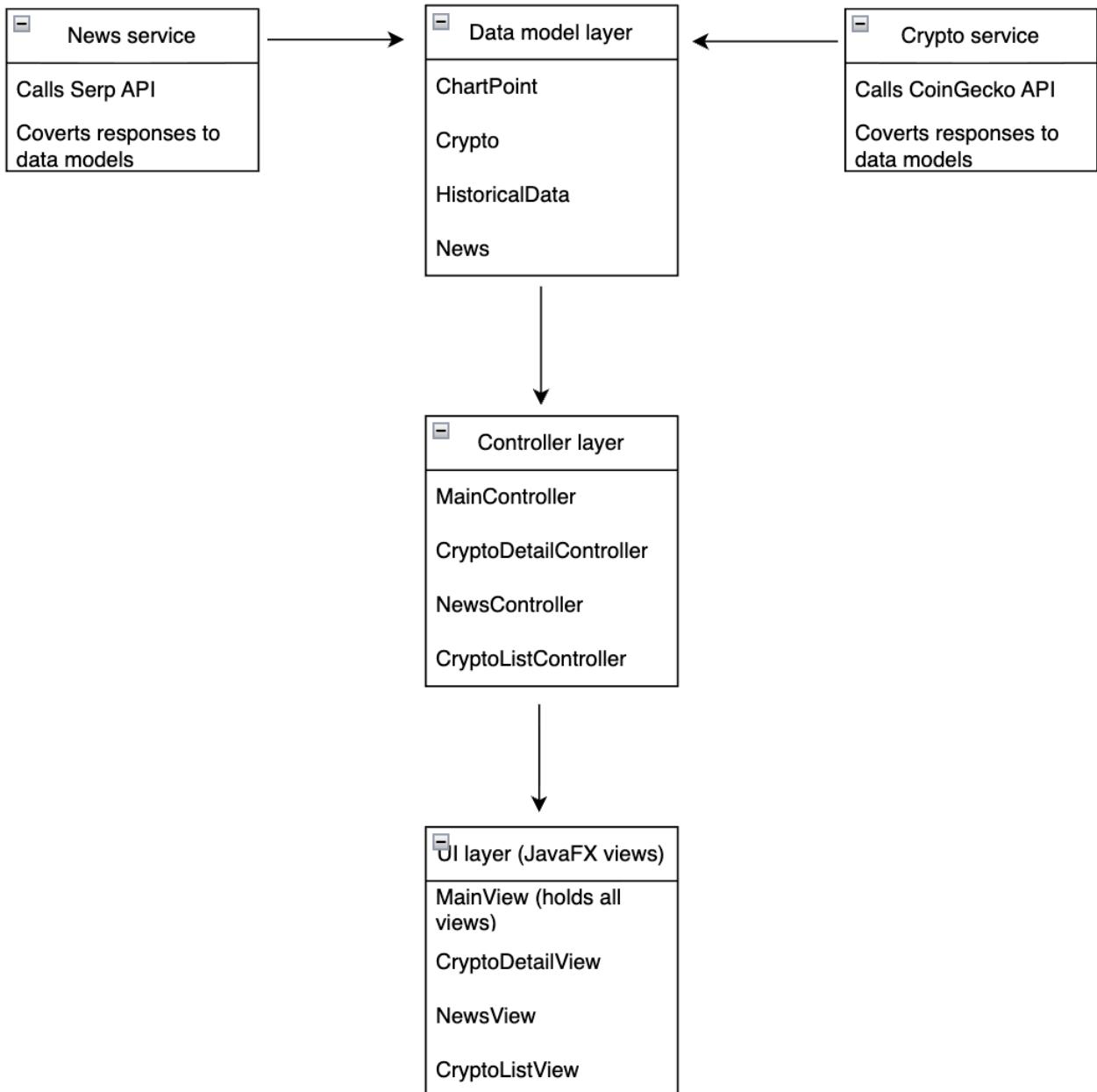
The application will be structured according to the Model-View-Controller (MVC) architecture, which separates UI presentation, business logic, and data handling. This improves many software quality metrics, such as maintainability, testability, and scalability.

Planned components for the application:

Component	Purpose	Interactions
App	Main entry point for the JavaFX application. It is responsible for instantiating all the major components (Models, Views, Controllers, Services) and wiring them together to form the complete, functional application	Initializes services, connects controllers to views
CryptoService	Fetches, caches and potentially polls crypto related data from CoinGecko API and parses responses	Sends parsed data to CryptoListController and CryptoDetailController. Returns model objects (List<Crypto>, HistoricalData)
SerpAPINewsService	Fetches crypto-related general and token-specific news from SerpAPI and parses responses	Sends parsed data back to NewsController and returns model objects (List<News>)
Crypto	Represents a single cryptocurrency. It holds all relevant data like its ID, name, symbol, price, 24h change, market cap, etc	Passed to CryptoListView and CryptoDetailView to be displayed to the user
HistoricalData	A container object that holds a list of ChartPoints, representing the price/volume history of a cryptocurrency over a specific period	Passed to the CryptoDetailView to render the chart

ChartPoint	Represents a single data point on a time-series chart, containing a timestamp, a price, and a volume	Held within HistoricalData model, used in CryptoDetailView to create data points for chart plotting
News	Represents a single news article, holding its title, source, link, summary, and date	Passed to Newsview for news display
MainController	The coordinating controller. It manages the high-level application flow and orchestrates the interactions between the other, more specialized controllers. It maintains the state of the selected crypto	Hold references to all the other controllers. Handles events from CryptoListView and NewsView and delegates tasks to CryptoDetailController and NewsController
CryptoListController	Manages the logic for the crypto list. Its primary responsibility is to load the initial list of top cryptocurrencies	Calls CryptoService to get the top 5 cryptos, updates CryptoListView and forwards events to MainController
CryptoDetailController	Manages the logic for the CryptoDetailView. It is responsible for fetching and providing the historical chart data based on user selections	Receives commands from MainController, handles events from CryptoDetailView, calls CryptoService to get historical data and updates CryptoDetailView
NewsController	Manages the logic for the news view. It fetches news articles based on the application's current state (either general news or news for a selected crypto)	Receives commands from MainController, calls the news service to get news data and updates NewsView
MainView	The root UI container. It places the crypto list on the left, the detail view in the center, and the news feed on the right	Holds all of the other views
CryptoListView	The left sidebar UI. It displays a searchable, scrollable list of top 5 cryptocurrencies	Receives data from CryptoListController and notifies the controller when a crypto is selected
CryptoDetailView	The central panel UI. It displays the detailed information (price, stats) and historical charts (price, volume) for the currently selected cryptocurrency	Receives data from CryptoDetailController and notifies the controller when a time interval changes or volume / price toggle changes
NewsView	The right sidebar UI. It displays a list of news articles, either general crypto news or news related to specific crypto	Receives data from NewsController and notifies MainController when toggle between general / selected changes

Diagram for the component interactions, arrows indicating the direction of data flow through the layers:



3. Self-evaluation for midterm submission

In this chapter, we will evaluate how well our design has supported the implementation and how well we are anticipating it to support the implementation of remaining functionality.

3.1 Original plan and design

We have mostly been able to stick to the original design for the most part. Our project structure is now MVC with the additional service layer, as was initially planned. However, the project structure did not fully correspond to the MVC structure at one point and had to be refactored for the midterm submission to follow MVC more strictly. We had to create the controller layer, which separates business logic from the views and works as a bridge between the services and views, passing data

models. The views were refactored to be independent and to only be responsible of rendering the UI. This was done as the MVC structure allows the project to have clear separation of concerns, leading to better testability, component reusability, maintainability and scalability. Another change in design was to refactor the CryptoListView into its own view and controller as it was formerly created in MainView. This was likewise done to obtain a clear separation of concerns and good modularity.

We also added a second graph of crypto volumes, in addition to the price graph we already had, to meet the requirements of the group assignment. This volume bar graph was not initially part of our design plans but since we learned we needed another visual representation of data, we quickly decided that a volume bar graph would suit the application and complement the crypto detail data well.

Implementing features has been quite straightforward based on our original plan. We are able to display details about the five largest crypto coins using CoinGecko API and get news about them using the SERP API, as was initially planned. However, there have also been some minor changes to the plan and requirements. We found out that CoinGecko has rate limits so it seems like the requirement of showing live price could be difficult. Similarly, quick crypto data loading and chart rendering are not currently possible with the rate limits as changing the intervals on the crypto chart quickly consumes API calls, leading to the chart not updating once the rate limit has been reached. We could try caching the data locally to reduce the number of API calls and try polling the live data every X seconds/minutes to overcome these challenges.

Original app idea and requirements were updated to match the current implementation choices and plan for the rest of the implementation. The description about the main view and its sections was updated and cleared up. Requirement about chart zoom and plan on websocket usage were removed due to API limitations and design decisions. We will be exploring polling for the final submission.

3.2. Current design and quality

As previously stated, our current design that follows the MVC architecture has many benefits that improve the quality of the software, including:

- Maintainability: clear structure makes it easy to locate and fix bugs
- Testability: modular design makes testing separate components easier
- Reusability: data model like Crypto can be used by multiple views and controllers
- Scalability: it is easy to add new features as the existing structure can be utilized
- Reliability: changes or errors in one layer is less likely to break others
- Modifiability/flexibility: technologies, APIs or UI can often be changed without changing other logic

3.3. Anticipated changes

We are anticipating some changes to the design and plan as we start implementing the remaining features even though the architecture is already solid and we have most features implemented. We will be creating unit tests for the most critical functionality like data parsing of the APIs and data formatting of the model classes. We likely need to refactor the services to be more testable as currently they use hard coded and static dependencies, making it difficult to mock HTTP responses and test without real API calls. The refactor will likely be done by extracting the interfaces and

implementing dependency injection. This way we can mock services and HTTP clients easily and test without actual API calls.

Apart from the tests, we will be refining the existing logic and structure of the application. We will likely implement crypto data caching to reduce the CoinGecko API calls and to make chart rendering smoother. Caching will also allow us to potentially do polling of price / volume data to update charts because of reduced API calls.

4. Group member contributions

	Antti	Veeti	Onni	Valtteri
Prototype	Created design document and planned the project idea.	Finalizing document. Tweaking the app architecture plan.	Initial UI layout code and components code	UI prototype modifications and finalization (crypto list, news toggle), created README
Midterm	Created API services for CoinGecko and SerpAPI and integrated them into the data handling and UI to display live data. I removed the mock data and added a dedicated handler file for the APIs	Implemented the historical data part of the crypto service and connected data to the chart. Implemented volume chart. Removed unused parts of the app and updating readme for midterm submission. Updating app idea and requirements.	Refactor to separate crypto list logic and UI from main view and main controller by adding a separate view and controller for the list. Update document for midterm (updated component table and graph).	Refactor project structure into MVC with services (created controller layer). Update document for midterm (self-evaluation section, minor updates across the document)

5. Use of AI

Artificial intelligence was used during the planning phase of the project to support documentation quality and clarify component relationships. The idea, API selection and technological choices were created independently by the project team, but AI was utilized to improve the structure and clarity of the design documentation. AI was also used in the coding phase.

5.1. UI prototype (code)

The prototype UI code was generated with the assistance of Copilot AI tool. The AI tool was used to produce initial UI layout code and UI components based on the selected app idea and our initial design plans. All generated code was reviewed, and components including the chart and news section were updated by modifying colors, fonts and sizes. New UI components and logic were also introduced including adding the left sidebar and a news filter for selected cryptos. These changes were all designed by us but generated with the AI tool iteratively.

5.2. Project planning and Desing document

The design document content was partially produced with the assistance of generative AI tools. The AI was used during the planning phase to support structuring the project concept and improving documentation quality. The project idea, the selected APIs, and the overall technology stack were created independently by the team, and AI was not used to make any decisions regarding features, architecture, or implementation.

AI assistance was used in mapping the relationships between application components based on our own descriptions of the system idea. The tool suggested an initial component breakdown and data flow structure within the MVC architecture. All final design decisions were made by the team, and components were further refined and renamed where necessary to better match the actual implementation plan.

AI was also used in drafting and improving the wording of functional and non-functional requirements based on our own notes. The original text for these sections was written in Finnish and translated into English using the AI tool. Additionally, some paragraphs were rewritten for better clarity and readability, while preserving our intended meaning and content. All documentation produced with AI support was reviewed and updated by the team before inclusion in this document.

5.3. Midterm AI usage

Document

In the component table, all the purposes and interactions were first generated with AI and then the responses were modified a little by changing wording, excluding certain points and adding some things the AI did not mention. AI was also used to get the data flow arrow directions in the component interactions diagram.

AI was slightly used in the self-evaluation section. It was used to explore solutions for reducing the GoinGecko API calls, which are mentioned at the end of original plan and design section. AI was also utilized to explore some of the quality benefits of the MVC structure, seen in the current design and quality section. Finally, AI was used to find out how the services can be unit tested easier.

Code

All of the refactoring (creating controller layer and separating crypto list from main view) mentioned in the self-evaluation chapter was generated with Copilot AI tool, in accordance with our own modification plans and goals. The code was then manually reviewed and modified where it was necessary.

In the API integration phase, the API calls for CoinGecko and SerpAPI were implemented manually, but Copilot AI was used to improve the handling of the returned data. AI helped generate the initial logic for parsing, formatting, and replacing the existing mock data with live API responses. A dedicated API

handler file was created and AI helped rearrange the code. All API parameters, request structures and data mappings were manually reviewed and corrected to ensure compatibility with the final UI implementation. Connecting the fetched data to the existing mocked graph was mainly implemented with Copilot with giving it granular tasks to get to the end goal of 2 graph options to choose from. Copilot was also used in debugging issues with the chart implementation, but also required manual intervention.