CryptoAdvisor - Implementation Project Final Report

Prepared by
Yusuf Ghani,
Anjali Viswan,
Nimrah Ahmed,
Minahil Rahimullah,
(Group 2 - CS 440)
at the
University of Illinois Chicago

December 2025

Table of Contents

I Project Description

1 Project Overview

CryptoAdvisor is a comprehensive personalized recommendation investment platform that provides users tailored towards stock and cryptocurrency based on user preferences. The system combines:

- Real-time market data accurate to the minute
- Previous user preferences
- Artificial intelligence / machine learning–style recommendation logic

to deliver a personalized feed of:

- Investment opportunities (stocks and cryptocurrencies)
- News articles
- Community discussions in forums

The application supports both cryptocurrency and stock market investment, allowing users to choose:

- Investment Strategies (Day Trading vs. Long-Term)
- Industries of interest
- Asset types (crypto vs. stock vs. both)

1.1 Key Features

- **User authentication and profile management**
- **Personalized investment recommendations** (stocks and cryptocurrencies)
- **Real-time price fetching** from external APIs:
    - Alpha Vantage (Stock API)
    - CoinGecko (Crypto API)
- **News aggregation** from MarketAux API
- **Community forums** for user discussions
- **Investment preference management** (asset types, industries, investment strategies)
- **Support for both Day Trading and Long-Term investment strategies**

1.2 Technology Stack

- **Backend**: Node.js, Express.js, MySQL (mysql2, hosted on AWS RDS)
- **Frontend**: JavaFX (Java)
- **APIs**:
    a. Alpha Vantage (Stock API)
    b. CoinGecko (Crypto API)
    c. MarketAux (News API)
- **Authentication**: JWT (JSON Web Tokens)

1.3 Objectives

1. **Personalization**: Deliver investment recommendations tailored based on user preferences.

2. **Real-time Data**: Integrate external APIs to fetch current prices and updated news.
3. **User Experience**: Provide an easy-to-use and intuitive interface in JavaFX.
4. **Community Engagement**: Allow users to discuss investments through forums.
5. **Scalability**: Design a robust backend architecture that can handle multiple concurrent users.

1.4 Scope

In Scope:

- User registration and authentication
- Investment preference management
- Personalized recommendation generation
- Real-time price fetching
- News aggregation
- Forum discussions
- Day Trading and Long-Term investment strategies

Out of Scope (for this version 1.0):

- Real-time trading execution
- Portfolio management
- Payment processing
- Advanced charting and technical analysis
- Mobile application

1.5 System Architecture Overview

CryptoAdvisor follows a **client–server architecture** with clear separation between frontend and backend.

High-Level Architecture:

- **JavaFX Client (Frontend)**
    - Handles UI, user interactions, token storage
    - Communicates with backend over HTTP/REST with JWT-based authorization
- **Express Server (Backend)**

- Exposes REST API endpoints under http://localhost:3000 (can be reconfigured)
- Performs authentication, preference management, recommendation generation, news aggregation, and forum handling

- **MySQL Database (AWS RDS)**
  - Stores users, preferences, recommendations, forum posts, and replies

- **External APIs**
  - Alpha Vantage – Stock prices
  - CoinGecko – Crypto prices
  - MarketAux – News articles

Front-end and back-end communicate via JSON over HTTP, with JWT tokens passed in the Authorization: Bearer <token> header.

1.6 Major Components

Backend Components:

1. **Authentication Module**
   - Handles user registration, login, and JWT token management.
2. **Preferences Module**
   - Manages user investment preferences (asset types, industries, cryptocurrencies, investment type).
3. **Recommendation Engine**
   - Generates personalized recommendations based on user preferences and real-time prices.
4. **Price Fetcher**
   - Integrates with external APIs (Alpha Vantage, CoinGecko) to fetch real-time prices, with caching and fallback.
5. **News Aggregator**
   - Fetches and filters news articles from MarketAux API based on user preferences and asset type.
6. **Forum Module**
   - Manages forum posts and replies, including author information and timestamps.

Frontend Components:

1. **LoginScreen** – User authentication interface.
2. **RegisterScreen** – New user registration.
3. **HomeScreen** – Main feed displaying recommendations, news, and forums.
4. **PreferencesScreen** – User preference management.
5. **NewsScreen** – Dedicated news viewing interface.
6. **ForumsScreen** – Forum browsing and interaction.

---

2 Project Domain

CryptoAdvisor lies in the retail investing and financial technology (FinTech) domain.

- It targets individual investors (not institutional traders).
- It supports stocks and cryptocurrencies.
- It uses news and community discussion**s** to allow these investors to discuss with others their strategies and inform them of what worked for them and what did not.

Key aspects of the domain:

- **Assets**: publicly traded stocks (via Alpha Vantage) and cryptocurrencies (via CoinGecko).
- **Investment Strategies**:
    - Day Trading – short-term, high-frequency, volatility-based investing.
    - Long-Term – multi-month or multi-year holdings, with focus on fundamentals and stability.
- **User Preferences**:
    - Preferred asset type).
    - Investment type (Day Trade, Long-Term).
    - Industries of interest (e.g., Technology, Finance, Energy).
    - Preferred cryptocurrencies (up to 5).

The project's goal is not to execute trades, but to **recommend**, **inform**, and **support discussion**, leaving actual trading to future implementations and for now, other platforms.

---

3 Relationship to Other Documents

This report sits alongside several other project artifacts:

- **Group 15 Final Development Report**
    - Contains the original report off which this project was based off of
- **SE Coding_Project_Report_Template v2.0.pdf**
    - The structure of this report has been aligned with this template
- **Backend documentation (README.md in backend/)**
    - Contains instructions on how to install dependencies, set environment variables, and run the server.
    - Coordinates with Section 11 (Deployment) of this report.
- **Frontend documentation (README.md in frontend/)**
    - Provides instructions on running the JavaFX client through Maven or scripts.
- **Database schema (schema.sql)**
    - Contains the SQL definitions for users, user_preferences, recommendations, forums, and forum_replies. So
    - Corresponds to Sections 5.1 and 7 (Database Design).
- **External API Documentation** (linked in Appendix E)
    - Alpha Vantage, CoinGecko, MarketAux documentation used to implement price and news fetching.

This report summarizes design, implementation, testing, and issues at a higher level and connects to those more detailed artifacts.

---

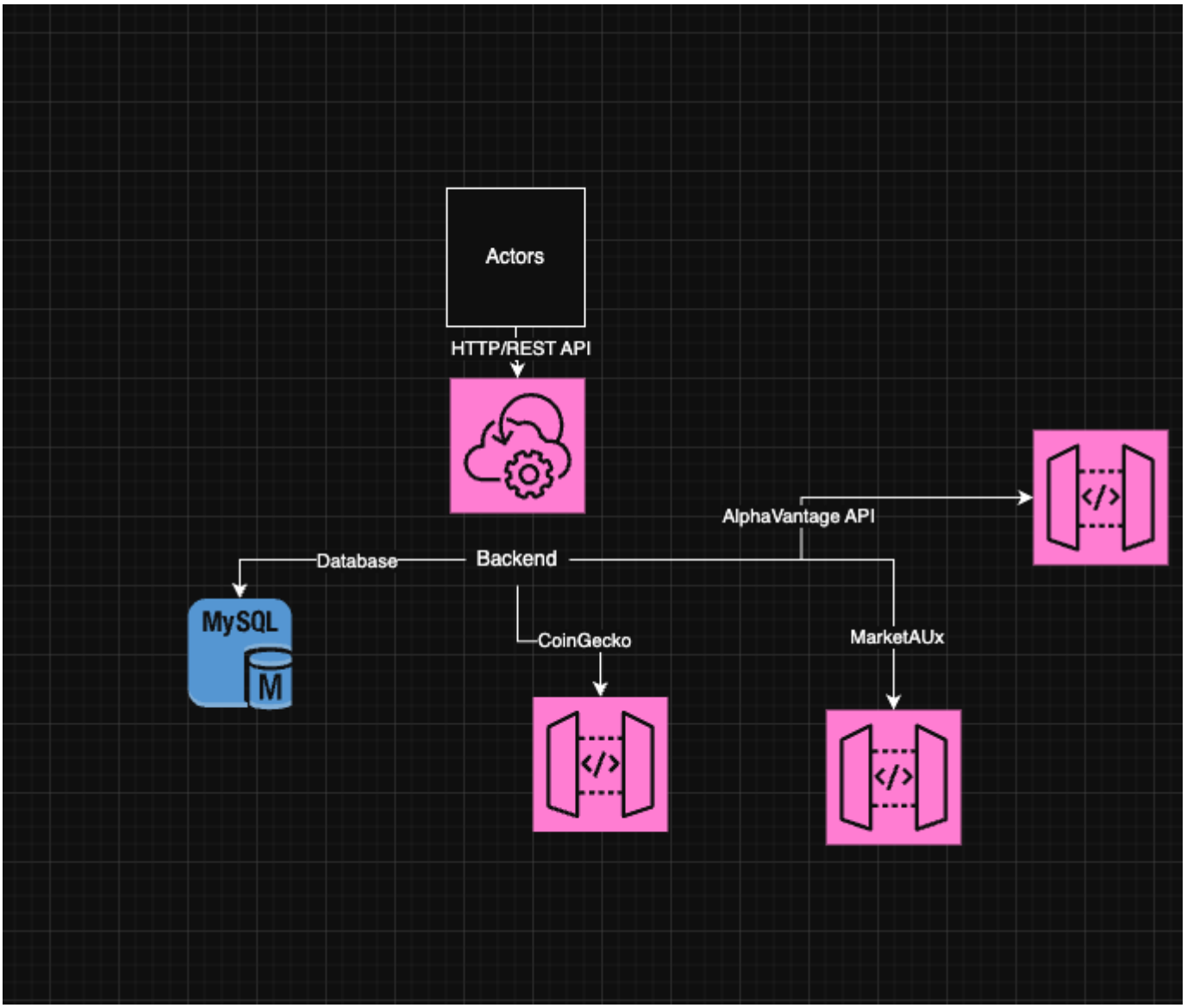
4 Naming Conventions and Definitions

4a Definitions of Key Terms

- **CryptoAdvisor**: The personalized investment recommendation platform developed in this project.
- **User**: An individual registered in the system, with stored preferences and authentication credentials.
- **Day Trade**: Short-term trading strategy focused on intraday price moves and high volatility.
- **Long-Term**: Investment strategy focused on holding assets for longer periods (months/years).
- **Preferred Asset Type**: The user's selection of crypto, stocks, or both.
- **Investment Type**: The user's choice of Day Trade or Long-Term.

- **Industries**: User-selected sectors like Technology, Finance, Energy; stored as JSON.
- **Cryptocurrencies**: Coins like bitcoin, ethereum, solana; stored as JSON.
- **Recommendation**: A database record containing an asset, current price, confidence score, reasoning, prediction message, and news summary.
- **AI Predictor**: The internal logic that generates reasoning and prediction messages based on data and preferences.
- **JWT (JSON Web Token)**: Token used to authenticate requests to the backend after login.
- **Alpha Vantage**: External stock API used for real-time stock prices.
- **CoinGecko**: External crypto API used for real-time cryptocurrency prices.
- **MarketAux**: External news API used for financial news aggregation.
- **Forum Post**: A discussion topic created by a user, stored in the forums table.
- **Forum Reply**: A reply associated with a specific forum post, stored in forum_replies.

4b UML and Other Notation Used in This Document



4c Data Dictionary for Included Models

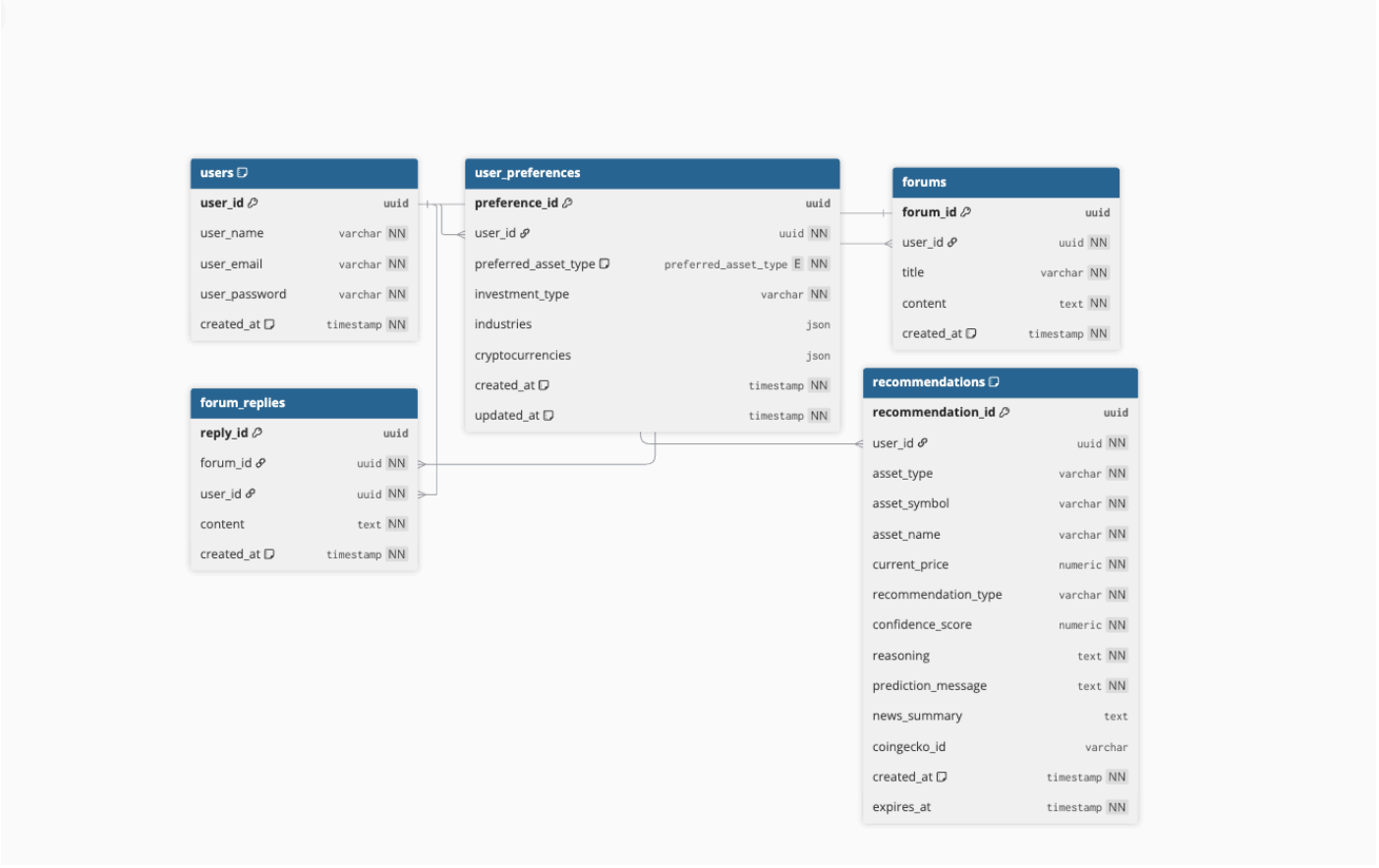


Table: users

- user_id (PK) – UUID, primary key.
- user_name – string, unique, used for display and uniqueness in forums.
- user_email – string, unique, used for login.
- user_password – hashed password.
- created_at – timestamp of account creation.

Indices:

- user_email – indexed for fast login lookups.
- user_name – indexed for uniqueness and quick search.

Table: user_preferences

- preference_id (PK) – UUID.
- user_id (FK) – refers to users.user_id.
- preferred_asset_type – ENUM: 'crypto', 'stocks', 'both'.

- investment_type – string: "Day Trade" or "Long-Term".
- industries – JSON array of strings (e.g., ["Technology", "Finance"]).
- cryptocurrencies – JSON array of strings (e.g., ["bitcoin", "ethereum"]).
- created_at – timestamp.
- updated_at – timestamp.

Table: recommendations

- recommendation_id (PK) – UUID.
- user_id (FK) – refers to users.user_id.
- asset_type – 'stocks' or 'crypto'.
- asset_symbol – e.g., "AAPL" or "BTC".
- asset_name – human-readable name (e.g., "Apple Inc.").
- current_price – numeric value (stock or crypto price).
- recommendation_type – "Day Trade" or "Long-Term".
- confidence_score – numeric (e.g., 0–100).
- reasoning – text explaining the rationale.
- prediction_message – user-facing AI Predictor output.
- news_summary – brief summarized news related to the asset.
- coingecko_id – optional; used if the asset is a crypto.
- created_at – timestamp.
- expires_at – timestamp for 24-hour expiration.

Indices:

- user_id, asset_symbol, expires_at.

Table: forums

- forum_id (PK) – UUID.

- user_id (FK) – refers to users.user_id.
- title – discussion title.
- content – full content of the post.
- created_at – timestamp.

Table: forum_replies

- reply_id (PK) – UUID.
- forum_id (FK) – refers to forums.forum_id.
- user_id (FK) – refers to users.user_id.
- content – reply content.
- created_at – timestamp.

II Project Deliverables

5. Design – Release 1 (First Implementation Scenario)

This section describes the system design as it was **implemented in Release 1**, corresponding to the scenario:

> "Register, Set Preferences and View Recommendations/Forums"

In this release, the focus was on delivering:

- A **working account system** (register, login, logout).
- A **profile/preferences system** (preferred companies/industries).
- A **home feed** that reacts to user preferences (v1.0 behavior).
- A **forum** where users can view, create, and reply to discussion posts.

This Scenario is a combination of some of the scenario names from the Group 15 development report, namely login/register, select preferences and enter forum. The recommendations here was a basic v1.0 and did not cover that scenario.

5.1 Database Design – Release 1

Release 1 used the following tables in the database:

- **users**

    - Used for account creation and login.
    - Stores user_id, user_name, user_email, user_password, created_at.
- **user_preferences**
    - Stores each users company/industry preferences from the initial questionnaire.
    - Linked to users via user_id.
- **forums**
    - Stores user-created forum posts.
    - Linked to users via user_id so each post has an author.
- **forum_replies**
    - Stores replies to forum posts.
    - Linked to forums via forum_id and to users via user_id.
- **recommendations**
    - In Release 1, this table is not used.

Indexes and constraints in Release 1 focused on:

- Fast login and user lookup via users.user_email and users.user_name.
- Quick preference lookup via user_preferences.user_id.
- Efficient forum loading via forums.user_id, forums.created_at, and forum_replies.forum_id.

5.2 API Design – Release 1

Authentication

- POST /api/auth/register
    - Creates an account from the **"Make account"** screen.
- POST /api/auth/login
    - Authenticates a user from the login screen.
    - Returns a JWT token stored by the client.

Preferences

- POST /api/user/preferences

    - Saves questionnaire results such as preferred companies/industries.

- GET /api/user/preferences
    - Retrieves saved preferences to display on the settings page and to drive the v1.0 home feed.

Forums

- GET /api/forums
    - Returns a list of forum posts the user sees when navigating to the Forums tab.
- POST /api/forums
    - Creates a new forum post.
- GET /api/forums/:forumId/replies
    - Returns replies inside a specific forum thread.
- POST /api/forums/:forumId/replies
    - Adds a reply to a specific discussion thread.

Home / Initial Feed

- In Release 1, the home feed is primarily driven by simple logic and/or mocked data using the saved preferences (for example, v1.0 news or example recommendations), not by a full algorithm.
- The API supports returning basic recommendation/news items that reflect the user's preferences so that the UI can demonstrate that preferences matter.

5.3 Frontend Design – Release 1

LoginScreen

- Shows username/email and password fields.
- Provides a **"**Make accoun**t"** / Register link.
- Handles invalid login feedback with an error message such as **"**Wrong Password**"** and prompts the user to try again while clearing the input

RegisterScreen

- Collects username, email, and password to create an account.
- On success, redirects the user back to the login screen to sign in.

Preferences / Questionnaire Screen

- Presented the first time a user logs in.
- Shows a short set of questions about:
    - Preferred companies or sectors/industries.
- Answers are saved to user_preferences via POST /api/user/preferences.

HomeScreen (v1.0)

- Displays **basic or demo recommendations/news** influenced by the user's preferences ("v1.0 feed").
- Serves primarily as a **proof-of-concept** that:
    - Preferences are stored correctly.
    - The content on the Home page changes based on those preferences.

SettingsScreen

- Allows the user to:
    - Change preferences (industries, strategies, etc.).
    - Log out of the app, which clears the stored token and returns to the login screen.

ForumsScreen

- Shows both recommended and general discussion posts.
- Allows the user to:
    - Scroll through multiple discussion posts.
    - Open a forum thread and read replies.
    - Reply to a forum with their own comments.
    - Create new forums/topics to ask specific questions.

Release 1:

- **Account system:** register, login, logout.
- **Profile/preference storage:** via the initial questionnaire and settings page.
- **Home screen (v1.0):** shows content influenced by preferences (even if partly mocked).
- **Forums:** browsing, creation, and replies.

- **Settings:** updating preferences and account details.

6. Implementation – Release 2 (Second Implementation Scenario)

This section describes what was implemented in **Release 2**, corresponding to the scenario:

> "A Personalized Recommended Feed based on User Preferences"

This was mainly based on the last 2 scenarios presented by group 15 in which it was generate recommendations and view recommendations with a real algorithm now.

6.1 Database Usage – Release 2

Release 2 continues to use all Release 1 tables and expands their role:

- **user_preferences**
    - Now actively drives the **recommendation algorithm**, including:
        - Investment type (Day Trade vs Long-Term).
        - Up to three industries (e.g., Technology, Healthcare, Energy).
- **recommendations**
    - Becomes a **central table** in Release 2.
    - Stores algorithm-generated, preference-based recommendations with:
        - asset_type (stock/crypto).
        - asset_symbol, asset_name.
        - current_price.
        - recommendation_type (Day Trade / Long-Term).
        - confidence_score, reasoning, prediction_message.
        - created_at, expires_at (~24 hours after creation).
- **pricing / external data usage**
    - Stock prices are now **fetched dynamically** from an API and displayed to the user.
    - Links to external charting sites (e.g., Yahoo Finance, CoinGecko) are provided for visualization.
- **forums** and **forum_replies**
    - Continue to be used for community features that remain accessible from the Home tab.

Recommendations

- GET /api/recommendations

    - Fetches recommendations from the recommendations table, filtered by user preferences.
    - Example for Jane:
        - Long-Term investments.
        - Industries: Technology, Healthcare, Energy.
    - Returns entries such as:
        - AAPL – Apple Inc. (Long-Term, Technology).
        - PFE – Pfizer Inc. (Long-Term, Healthcare).
        - XOM – ExxonMobil Corp. (Long-Term, Energy).
- POST /api/recommendations/refresh
    - Triggers recomputation of recommendations based on the latest market data and the current user preferences.
    - Used when the user refreshes their recommendations (e.g., on the Home page).

News / Articles

- GET /api/news
    - Retrieves articles relevant to the user's portfolio and industries.
    - Used on the Articles/News page to show:
        - Title.
        - Summary/intro.
        - Source and URL.
    - Filters content by:
        - Asset type (stocks/crypto).
        - Industries in user preferences.

Price / Market Data (for demo and algorithm support)

- GET /api/stocks/price/:symbol
    - Fetches or simulates stock prices used by the recommendation algorithm and UI.
- GET /api/crypto/price/:id
    - Fetches or simulates cryptocurrency prices used by the algorithm and UI.

6.3 Backend Implementation – Release 2

Release 2 implements the **recommendation engine v1.0** and the data flow described in the second scenario:

1. **Retrieve user preferences**
    - The backend reads the user's investment_type (Day Trade / Long-Term) and their selected industries from user_preferences.
2. **Fetch market data**
    - The system queries external APIs (or mock services) to obtain current or sample market data for:
        - Stocks in relevant industries.
        - Cryptocurrencies, when applicable.
3. **Filter and score assets**
    - For Long-Term: Return stocks given to us by the API
    - For Day Trade: Return stocks given to us by the API
    - Filter by the user's chosen industries.
    - Assign confidence_score and generate short reasoning and prediction_message.
4. **Store recommendations**
    - The system writes recommendation objects into the recommendations table with:
        - created_at timestamp.
        - expires_at 24 hours later(refresh recommendations after 24hrs in backend)
5. **Serve recommendations**
    - GET /api/recommendations returns a subset of recommendations for display on the Home page.
    - POST /api/recommendations/refresh clears or expires any stale recommendations and regenerates them using updated market data.
6. **Serve news**
    - GET /api/news uses the same preferences to select relevant stock/crypto articles for the Articles page.

6.4 Frontend Implementation – Release 2

HomeScreen (Personalized Feed)

- After Jane logs in and has preferences set:
    - The client calls GET /api/user/preferences to confirm her investment type and industries.
    - The client calls GET /api/recommendations to load her personalized recommendations.
- The Home screen now shows:
    - Stock/crypto symbol (e.g., AAPL).
    - Asset name (e.g., Apple Inc.).
    - Investment type tag (e.g., Long-Term).
    - Industry tags (Technology, Healthcare, Energy).
    - Current price (from price endpoints or embedded market data).
- The Home screen includes a Refresh control that:
    - Calls POST /api/recommendations/refresh.
    - Then re-calls GET /api/recommendations to update the list.

PreferencesScreen (Release 2 Behavior)

- The Preferences screen is aligned with Scenario #2:
    - Allows selecting investment type: Day Trade or Long-Term.
    - Allows selecting up to three industries (e.g., Technology, Healthcare, Energy).
- When Jane clicks Save:
    - The client sends the new configuration to POST /api/user/preferences.
    - The backend may clear or recompute recommendations so that the new preferences immediately influence the feed.

Articles/NewsScreen

- The Articles/News screen:
    - Calls GET /api/news.
    - Displays a list of news items with:
        - Title.
        - Summary/introduction.

        - Source.
        - A clickable link that opens the full article in the browser.
- Articles are filtered by Jane's preferences:
    - Stocks/crypto that match her industries or portfolio.

ForumsScreen (Release 2 Enhancements)

- Continues to provide:
    - Listing of all forum posts.
    - Ability to open threads and read/post replies.
    - Ability to create new forum posts.
- May now highlight:
    - "Recommended" forums or discussion boards related to Jane's industries or assets.
    - Boards she follows or that are new and relevant to her portfolio.

Added in Release 2:

- A **persistent recommendation pipeline** from:
    - user_preferences → external API/market data → recommendations table → personalized Home feed.
- **Articles/News screen** tied to user industries and asset types.
- **24-hour recommendation cycles** plus a manual refresh button.
- Dynamic price fetching and links to external charts for deeper exploration of recommended stocks/cryptos.

7 Comparison with Original Project Design Document

The implemented system is very close to the **planned design:**

Matched with Original Design:

- Client–server separation (JavaFX frontend and Express backend).
- Use of MySQL with clear ER structure (users, preferences, recommendations, forums, replies).
- JWT-based authentication.

- Real-time data via Alpha Vantage and CoinGecko.
- News aggregation via MarketAux.
- Support for both Day Trading and Long-Term strategies.
- Forum module for community interaction.

Simplifications / Deviations:

- **No real trading/execution:**
    - Real-time trading execution remains out of scope; the app focuses on recommendations and information only.
- **No full ML/AI model deployment:**
    - The AI-Predictor uses rule-based logic rather than a fully trained machine learning model(this is something for future implementations).
- **Advanced charting omitted:**
    - Advanced internal charting tools and technical indicators are not implemented; instead, external chart websites (Yahoo Finance, CoinGecko) are opened.

III Testing

8 Items to be Tested

Testing covered both backend and frontend, as well as integration between them.

Backend Items:

1. **Authentication module**
    - Registration
    - Login
    - Token validation
    - Profile update
    - Account deletion
2. **Preferences module**
    - Saving preferences (asset type, investment type, industries, cryptos).
    - Updating preferences.

- Correct retrieval of user preferences.

3. **Recommendation engine**
    - Generation logic for:
        - Crypto-only users.
        - Stock-only users.
        - Both assets.
        - Day Trade vs Long-Term Recommendations.
    - Expiration after 24 hours.
4. **Price fetching**
    - Stock prices via Alpha Vantage.
    - Crypto prices via CoinGecko.
    - Caching behavior.
    - Fallback behavior if APIs fail or rate limits hit(use fallback prices).
5. **News aggregation**
    - News fetching from MarketAux based on asset type.
    - Correct structure and fields (title, summary, URL, source, date).
6. **Forum module**
    - Creating posts.
    - Listing posts.
    - Creating replies.
    - Listing replies per forum.

Frontend Items:

1. **Navigation**
    - Login → Home → Preferences → News → Forums → Logout.
    - Error messages and transitions.
2. **Data display**
    - Recommendations rendering (stock vs crypto sections, price formatting).

- News article display (titles, summaries, sources).
- Forum posts and replies display.

3. **User interactions**
    - Setting preferences and saving them.
    - Creating forum posts and replies.
    - Opening external links (Yahoo Finance / CoinGecko / news URLs).
4. **Error handling / UX**
    - Connection errors.
    - Invalid token handling (e.g., expired tokens).
    - Empty states when no content is available.

---

9 Test Specifications

Testing was performed manually with structured scenarios.

Authentication Test Cases

- **Test A1 – Register Valid User**
    - Input: unique username, valid email, valid password.
    - Steps:
        - Call POST /api/auth/register.
        - Check HTTP status and body.
    - Expected:
        - HTTP 201.
        - {"message": "User registered successfully", "token": "...jwt..."}.
- **Test A2 – Register with Duplicate Email**
    - Input: same email as an existing user.
    - Expected:
        - HTTP 4xx.
        - Error message about email already in use.

- **Test A3 – Login Valid Credentials**
    - Input: correct email and password.
    - Expected:
        - HTTP 200.
        - Response with message, token, and user object.
- **Test A4 – Login Invalid Credentials**
    - Input: correct email, wrong password OR non-existing email.
    - Expected:
        - HTTP 401/403 with a clear error message.
- **Test A5 – Profile Update**
    - Headers: valid Authorization: Bearer <token>.
    - Body: updated email/username/password.
    - Expected:
        - User record updated in DB.
        - Response indicates success.

Preferences Test Cases

- **Test P1 – Set Preferences**
    - Asset type: "both".
    - Investment type: "Day Trade".
    - Up to 3 industries.
    - Up to 5 cryptocurrencies.
    - Expected:
        - HTTP 201 with "message": "Preferences saved successfully".
        - "recommendationsCleared": true.
- **Test P2 – Get Preferences**
    - Headers: valid token.

- Call GET /api/user/preferences.
- Expected:
    - Response contains the exact latest saved preferences.

Recommendations Test Cases

- **Test R1 – Get Recommendations After Setting Preferences**
    - Steps:
        - Set preferences.
        - Call GET /api/recommendations.
    - Expected:
        - List of recommendations with all fields (asset_type, symbol, price, confidence_score, reasoning, prediction_message, news_summary, created_at, expires_at).
- **Test R2 – Refresh Recommendations**
    - POST /api/recommendations/refresh.
    - Expected:
        - Message "Recommendations refreshed successfully!".
        - Old recommendations cleared or replaced.

News Test Cases

- **Test N1 – Get News for AssetType ‘crypto’**
    - Preferences set to crypto-only.
    - Call GET /api/news.
    - Expected:
        - news array with "type": "crypto" and relevant titles/summaries.
- **Test N2 – Get News for AssetType ‘both’**
    - Preferences set to both.
    - Expected:
        - Mixed news items labeled by type (crypto/stocks).

Forum Test Cases

- **Test F1 – List Forums**
    - Call GET /api/forums.
    - Expected:
        - List of forum posts with forum_id, user_id, title, content, author_name, created_at.
- **Test F2 – Create Forum Post**
    - Auth header + body with valid title/content.
    - Expected:
        - Post appears in GET /api/forums.
- **Test F3 – Replies**
    - Call GET /api/forums/:id/replies to view replies.
    - Call POST /api/forums/:id/replies to add a reply.
    - Expected:
        - Replies stored and displayed correctly with author_name and created_at.

Frontend Test Scenarios

- **Scenario F-Flow1 – Full User Journey**
    - Register → Login → Set Preferences → View Recommendations → View News → View Forums → Logout.
    - Expected:
        - No crashes, correct navigation, correct data display.
- **Scenario F-Error1 – Invalid Token**
    - Use an expired/invalid token.
    - Expected:
        - Backend returns error, frontend displays user-friendly message, possibly redirects to login.
- **Scenario F-Display1 – Price Formatting**
    - Stocks and crypto with very small or large prices.

    - Expected:
        - Proper formatting that handles small crypto prices (10^-4 range) gracefully.

---

10 Test Results

Overall, manual tests showed:

- **Authentication**: Passed for valid paths; correctly rejected invalid credentials.
- **Preferences**: Correctly stored, retrieved, and updated per user; typical limits (3 industries, 5 cryptos) respected.
- **Recommendations**:
    - Generated correctly based on preferences.
    - Distinction between Day Trade and Long-Term visible in reasoning/prediction messages.
    - Recommendations were timestamped with expires_at 24 hours later.
- **Price Fetching**:
    - Worked correctly under normal conditions.
    - When Alpha Vantage rate limits were hit, fallback logic used cached/hardcoded values.
- **News**:
    - News articles loaded with correct titles, summaries, URLs, and sources.
    - Asset type filtering behaved as intended.
- **Forums**:
    - Posting and replying worked and were reflected in the UI.

Some minor UX issues (e.g., occasionally vague error messages) were observed but did not break core flows.

---

11 Regression Testing

After significant changes (e.g., introducing new endpoints or refactoring backend logic), the following regression steps were repeated:

- Re-run **A1–A4** to validate authentication.
- Re-run **P1–P2** to confirm preference handling was unaffected.

- Re-run **R1–R2** to ensure recommendations still generate correctly.
- Spot-check frontend flows (**F-Flow1**) to verify navigation and display still behaved as expected.

Regression testing helped catch a few small issues (e.g., token parsing inconsistencies) which were then resolved.

IV Inspection

12 Items to be Inspected

Backend Files:

- server.js – Overall server setup, routing, middleware.
- Controllers/handlers for:
    - /api/auth/register
    - /api/auth/login
    - /api/auth/update
    - /api/auth/delete
    - /api/user/preferences
    - /api/recommendations, /api/recommendations/refresh
    - /api/news
    - /api/forums, /api/forums/:id/replies
    - /api/stocks/price/:symbol
    - /api/crypto/price/:id

Frontend Files:

- CryptoAdvisorApp.java (app bootstrap and navigation).
- UI screens:
    - LoginScreen.java
    - RegisterScreen.java
    - HomeScreen.java
    - PreferencesScreen.java

    - NewsScreen.java
    - ForumsScreen.java
- Utility:
    - TokenManager.java – JWT handling and storage.

13 Inspection Procedures

Inspection was performed informally but systematically:

1. **Code Reading**
    - Each group member reviewed specific files line by line.
2. **Checklist-Based Review**
    - Questions asked:
        - Are we using parameterized queries and avoiding string concatenation for SQL?
        - Are error paths handled (e.g., database down, API failures, missing token)?
        - Are responses consistent (status codes and JSON structure)?
        - Is the UI code separated from network logic and state management?
3. **Security Focus**
    - Confirm JWT verification is always applied to protected endpoints.
    - Confirm passwords are never stored in plain text.
4. **Performance Considerations**
    - Check indexes in the schema.
    - Check that frequently queried fields are indexed.

14 Inspection Results

Findings:

- **Strengths:**
    - Use of middleware for authentication is clear and reusable.
    - Parameterized queries reduce SQL injection risk.
    - API endpoints have consistent patterns (HTTP method, URL, request/response format).

    - Frontend navigation and HTTP requests are reasonably well-structured.

- **Areas for Improvement:**
    - Logging could be more structured (e.g., log levels, correlation IDs).
    - Password hashing is used but could be strengthened with more advanced algorithms and configurations.
    - Some error responses could be standardized further (e.g., error codes, error fields).

V Recommendations and Conclusions

CryptoAdvisor version 1.0 successfully achieves:

1. **Complete User Management**
    - Registration, authentication, profile update, account deletion.
2. **Personalized Recommendations**
    - Tailored to user preferences (asset type, strategy, industries, cryptos).
    - Includes current prices and explanatory reasoning.
3. **Real-time Data Integration**
    - External APIs integrated for stock and crypto prices, plus financial news.
4. **Community Features**
    - Forums for users to create posts and replies, fostering discussion.
5. **Intuitive UI**
    - JavaFX interface with clear navigation, color coding, cards, and feedback.

VI Project Issues

15 Open Issues

1. **API Rate Limits**
    - Alpha Vantage API rate limits (5 calls/minute) can still be hit if many stocks are requested in a short time.
2. **Crypto Price Representation**

    - Some cryptocurrencies have very low prices (10^-4) which are tricky to display nicely; better formatting and precision handling is needed.

3. **Error Handling Edge Cases**
    - Some rare network failures and timeouts are not fully covered and could produce generic error messages.

16 Waiting Room

Features not implemented in v1.0 but considered valuable for the future:

- Real-time trading execution through integrated broker APIs.
- Portfolio management and tracking of user holdings.
- Advanced charting and technical indicators built into the UI.
- Native mobile apps (Android and iOS).
- AI/ML Model to set up buy/sell orders
- Push notifications and price alerts.

17 Ideas for Solutions

Some ideas for addressing open issues and implementing future enhancements:

1. **Performance and Scaling**
    - Cache Popular API calls such as S&P 500 stocks

2. **Advanced Analytics**
    - Integrate a true ML/AI model for price prediction and risk scoring.
    - Track recommendation effectiveness(did the user invest in recommended stocks)
3. **Security**
    - Add two-factor authentication.
    - Implement rate limiting on login and certain API endpoints.
    - Enforce stronger password policies.
4. **UX Enhancements**
    - More detailed error dialogues and tooltips.
    - Better pagination and filtering for forums and news.

18 Project Retrospective

Lessons Learned:

1. **API Rate Limits**
    - External APIs impose real constraints; caching and batching strategies are essential.
2. **Error Handling**
    - Detailed error handling greatly improves the robustness and perceived quality of the system.
3. **User Experience**
    - Users benefit significantly from loading indicators, clear successes/failures, and intuitive navigation.
4. **Database Design**
    - Proper indexing and relational constraints (including cascade deletes) prevent dead data and performance problems.
5. **Testing**
    - More comprehensive automated testing would make future refactoring safer and easier.

VII Glossary

- **AI Predictor** – Logic that produces prediction messages and reasoning for recommendations.
- **Asset Type** – Whether an asset is a stock or cryptocurrency.
- **Day Trade** – Short-term trading strategy, focused on intraday moves.
- **Long-Term** – Strategy involving holding assets for longer periods.
- **JWT** – JSON Web Token, used for authentication in stateless APIs.
- **Portfolio** – Collection of assets a user holds (planned future feature).
- **Preference** – Stored settings describing a user's asset type, industries, investment type, and cryptos.
- **Recommendation** – Suggested stock or crypto with price, reasoning, and confidence.
- **Volatility** – Measure of how much an asset's price fluctuates; important for Day Trading.

VIII References / Bibliography

- Alpha Vantage: Stock API Documentation – https://www.alphavantage.co/documentation/
- CoinGecko: Crypto API Documentation – https://www.coingecko.com/en/api/documentation
- MarketAux: News API Documentation – https://marketaux.com/documentation
- Node.js – Official Documentation - https://nodejs.org/docs/latest/api/
- Express.js – Official Documentation - https://expressjs.com/en/5x/api.html
- Group 15 Final Development Report PDF
- SE Coding_Project_Report_Template v2.0 – CS 440, University of Illinois Chicago

---