Software Requirements Specification (SRS)

For “Coin Market System”.

Version 2.0

Prepared by Bui The Ky Cuong

Contents

Definition and Acronyms

Acronym	Definition
BR	Business Rule
ERD	Entiy Relationship Diagram
GUI	Graphical User Interface
UC	Use case
API	Application Program Interface
Patron	User who registered to the Coin Market System

I. Introduction

1 Purpose

This document outlines the software requirements for the "Crypto Market System". It will cover the system's capabilities for tracking and displaying market data such as prices, market capitalization, and trading volumes, as well as handling user-defined alerts and indicators. The system integrates data from Binance, CoinMarketCap (CMC), and CoinGecko (CG), and provides APIs for internal and external usage.

2 Document Conventions

- APIs referred to will be in REST format unless stated otherwise.
- Time intervals are presented in seconds (s), minutes (min), and hours (h).
- User classes are segmented into permission levels (VIP-0, VIP-1, VIP-2, VIP-3).

3 Project Scope

The system serves to track real-time cryptocurrency market data (e.g., Spot and Future prices), offering advanced monitoring, alerts, and statistical analysis features. Users can subscribe to price alerts and integrate additional technical indicators via customizable plugins. The scope includes:

- Market data retrieval from multiple APIs.
- Price alerts with custom trigger and snooze conditions.
- Role-based access to API data.

4 References

- Binance API Documentation: https://developers.binance.com/docs
- CoinMarketCap API Documentation: https://coinmarketcap.com/api
- CoinGecko API Documentation: https://www.coingecko.com/en/api

5 Product Perspective

The product is a self-contained system integrating data from Binance, CMC, and CG to track and provide detailed cryptocurrency market data. It includes both frontend and backend components, supported by a real-time API and admin management system.

6 Product Features

- Real-time Market Data: Spot price, Future price, Kline price, funding rate, and countdown updated every 1 second from Binance.
- Price Alerts: Customizable alerts based on specific triggers for price, funding rate changes, and indicator thresholds.
- User Roles: Role-based access control via API tokens (VIP-0, VIP-1, VIP-2, VIP-3).

7 User Classes and Characteristics

- VIP-0 Users: Access basic price data only (Spot, Future).
- VIP-1 Users: Access advanced data like Kline prices.
- VIP-2 Users: Set and manage price alerts.
- VIP-3 Users: Use advanced statistical indicators.

8 Operating Environment

Backend: Hosted on Go-based services.
API: Public-facing APIs for client consumption.
Telegram Bot: Integrated for user queries.
Admin Web Portal: Interface for managing users, permissions, and configurations.

II. Functional Requirements

User Requirements Overview

1 User Requirements

System Actors

#	Actor	Description
1	Guest	Users do not have logged in the Coin Price System web application.
2	Patron	User who have logged in to use the features in the Coin Price System web application.
3	Admin	User who have logged in to manage the Coin Price System web application.
4	System Handler	Coin Price System for real-time handling.

1.1 Guest

Users (have no account in Coin Price System) who use web application, can use these following functions:

- Sign Up (Register)

1.2 Registered User (Patron)

This system is a level-based, so for each VIP level of investor (registered user), they can use different functions:

Features	VIP-0	VIP-1	VIP-2	VIP-3
Logout	X	X	X	X
Login	X	X	X	X
Look up for Spot price	X	X	X	X
Look up for Future price	X	X	X	X
Look up for Kline Price		X	X	X
Look up for Funding Rate	X	X	X	X
Look up for Funding Rate Countdown	X	X	X	X
Look up for Funding Rate Interval	X	X	X	X
Configure Alert for Spot and Future			X	X
Configure Alert for Funding Rate			X	X
Configure Alert for Funding Rate Interval			X	X
Set up new or delisted Symbol alerts			X	X
Set up alerts based on price differences between Spot and Futures			X	X
Set up alerts based on technical indicators				X

Note:

[x] Capable Of.

[ ] Incapable Of.

1.3 Admin

Admin (login required) are capable of using these following functions:

- Get all users' information.
- Get user's payment history.
- Delete (Ban) a user or all users in the system.

Business Rules

Code	Business Definition
BR-01	User's login session lasts for 6 hours by default.
BR-02	Name length limit must be in range 1 – 50 characters.
BR-03	Name only contains alphabetical characters.
BR-04	Phone number only contains numeric characters.
BR-05	Phone number length limit must be 10 characters.
BR-06	Phone number and email must be unique.
BR-07	User can only update their own profile, includes their avatar, full name, phone number, email, birthday, and gender.
BR-08	User's avatar file must be image file (eg. jpg, jpeg, png) format.
BR-09	User can only view their own notification.
BR-10	Users must enter a password that is at least 8 characters long.
BR-11	If a user enters the wrong password 5 times in a row, the account will be temporarily locked for 30 minutes.
BR-12	Users can reset their password via email.
BR-13	Each user can create up to 10 alerts for the same coin.
BR-14	A price alert can only be triggered when the coin price reaches or crosses a set threshold (>= or <=).
BR-15	Alerts can be sent via email or Telegram if the user has configured the integration.
BR-16	The system only triggers the alert once per price condition.
BR-17	The alert will be automatically disabled if the user clears the alert or the coin price changes beyond the past threshold.
BR-18	Users can only change their password if they know the current password.
BR-19	Administrators can lock or delete user accounts that violate policies.
BR-20	If User wants to purchase VIP level, they must verify their phone number by OTP, then make payment through Momo.

Use Case Diagram

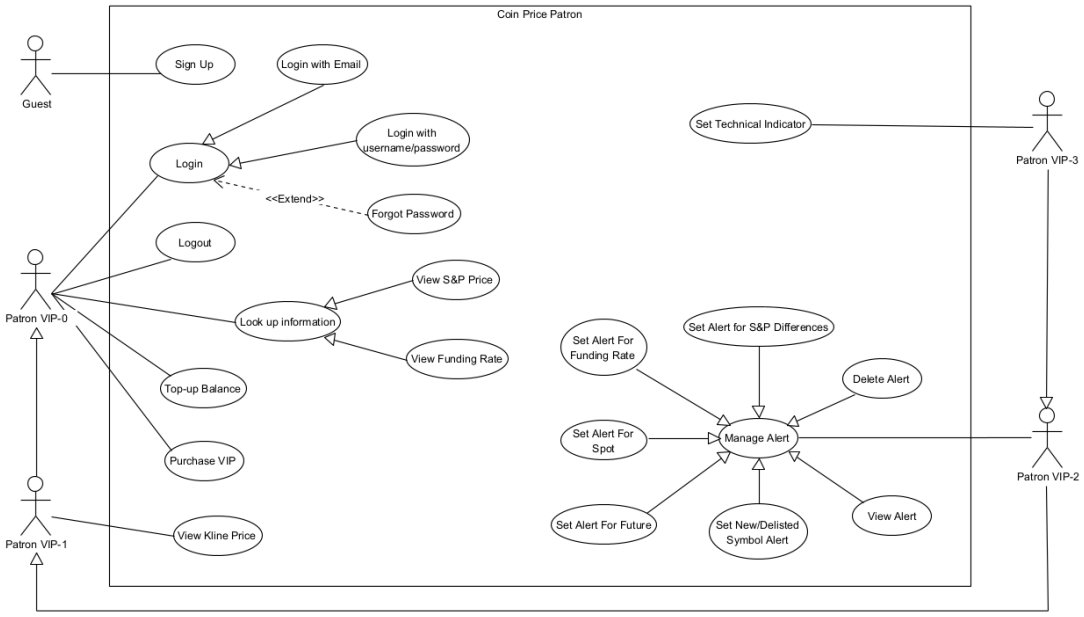


Figure 1: Patron Usecase Diagram

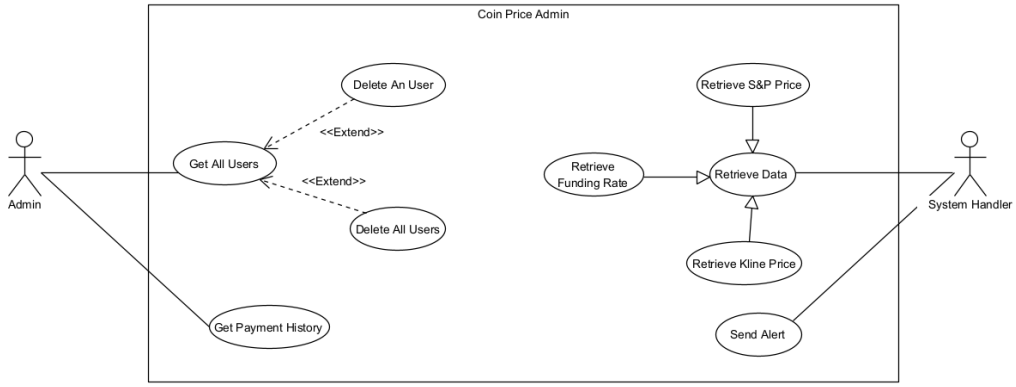


Figure 2: Admin & System Handler Usecase Diagram

2 Use Cases List

2.1 Patron

The table below lists which features a Patron in the system can experience.

Name	ID	Primary Actor
Login (with user/password)	UC-01	Guest
Login (with Email)	UC-02	Guest
Logout	UC-03	Patron
Look up information	UC-04	Patron VIP0, VIP1, VIP2, VIP3
Purchase VIP through Momo	UC-05	Patron VIP0, VIP1, VIP2
Manage Profile	UC-06	Patron VIP0, VIP1, VIP2, VIP3
View Kline Price	UC-07	Patron VIP1, VIP2, VIP3
Create Alert For Funding Rate	UC-08	Patron VIP2, VIP3
Create Alert For Spot Price	UC-09	Patron VIP2, VIP3
Create Alert For Future Price	UC-10	Patron VIP2, VIP3
Create New or Delisted Symbol Alert	UC-11	Patron VIP2, VIP3
Set Alert for Spot & Future price differences	UC-12	Patron VIP2, VIP3
View Alert	UC-13	Patron VIP2, VIP3
Delete Alert	UC-14	Patron VIP2, VIP3
Set Technical Indicator	UC-15	Patron VIP3

2.2 Admin and System Handler

The table below lists which features an Admin in the system can experience.

Name	ID	Primary Actor
Get All Users	UC-16	Admin
Get Payment History	UC-17	Admin
Delete An User	UC-18	Admin
Delete All Users	UC-19	Admin
Send Alert	UC-20	System Handler
Retrieve S&P Price	UC-21	System Handler
Retrieve Kline Price	UC-22	System Handler
Retrieve Funding Rate	UC-23	System Handler

3 Use Case Specification

Alternative Flows: All the alternative cases will return to the main success scenario once all conditions are met.

Use Case Specification			
Use Case No.:	UC-01		
Use Case Name:	Login with username/password		
Created By:	Bui The Ky Cuong		
Date:	15/10/2024	Priority	High
Actors:	Guest (Unauthenticated user)		
Summary:	The user enters login information (email and password) into the system via the Next.js page, the Go backend system checks the validity of the information, generates a JWT token and returns this token to the frontend to store in a cookie.		
Trigger:	The user wants to log in to the system.		
Preconditions:	User already has a valid account. The user has navigated to the login page.		
Post–conditions:	The user has successfully logged in and received the token stored in the cookie. The system stores session information and access rights (VIP-0, VIP-1, VIP-2, VIP-3).		

Main Success Scenario/Main Flow/Normal Flow/Main Path:

Step.	Actor Action	System Response
1	User accesses the login page.	The frontend system displays the login form (email, password).
2	User enters email and password.	The system checks that the data is entered in the correct format.
3	User clicks the "Log in" button.	The frontend sends an HTTP POST request with user information to the backend's /auth/login endpoint.
4	Backend receives and processes the request.	The system checks the login information from the MongoDB database (email, password).
5	Valid information.	The system generates a JWT token with the user's role and sends it back to the frontend with a 200 OK response.
6	Frontend receives the token.	Frontend saves the token to a cookie and redirects the user to the dashboard page.
7	User logs in successfully.	The system records the user's session and maintains the login status.

Alternative Flows:

2a. Missing information

Step	Actor Action	System Response
1	The user did not fill in all the information.	The frontend checks and displays an error asking the user to fill in all the information (email or password).

5a. Invalid information

Step	Actor Action	System Response
1	The user entered an incorrect email or password.	The backend system returned a 401 Unauthorized error message with the error information of incorrect email or password.

5b. Account is not registered

Step	Actor Action	System Response
1	The user has not registered an account.	The backend system returned a message that the account does not exist, asking the user to register.

Exceptions:

No.	Actor Action	System Response
1	The backend system cannot connect to the database.	Returns a 500 Internal Server Error message, please try again later.
2	Tokens cannot be created successfully (due to security errors or invalid secret key).	The system returns a 500 Internal Server Error message.

Business Rules:	- The JWT token generated by the backend needs to contain the user's role information (VIP-0, VIP-1, VIP-2, VIP-3) to control access after a successful login. - The cookie that stores the token is HttpOnly to prevent XSS attacks. - The login session needs to be maintained for a specified period of time (e.g. 24 hours), after which the token will expire and require the user to log in again. - Ensure that the JWT token is encrypted with a secure secret key and that the system checks the validity of the token at the endpoints that need authentication.

Implementation Notes:

Frontend (Next.js):

Login interface at /login with form containing input for email and password.

When user presses "Log in" button, information will be sent to /auth/login endpoint via POST method using axios or fetch.

Store JWT token in cookie using package like js-cookie.

Backend:

Handle authentication logic in handler of /auth/login endpoint.

After successful authentication, create JWT token with payload containing user_id, role and expiration time.

Use library like jwt-go to create and handle token.

Use Case Specification

Use Case No.:	UC-02		
Use Case Name:	Login with Email		
Created By:	Bui The Ky Cuong		
Date:	15/10/2024	Priority	High
Actors:	Guest (Unauthenticated user)		
Summary:	The user logs into the system using their Google account. The frontend sends an authentication request to Google, and the backend verifies the token, retrieves the user information, and issues a session token (JWT) if the login is successful.		
Trigger:	The user wants to log in to the system.		
Preconditions:	The user must have a Google account. The user has navigated to the login page.		
Post–conditions:	The user has successfully logged in and received the token stored in the cookie. The system stores session information and access rights (VIP-0, VIP-1, VIP-2, VIP-3).		

Main Success Scenario/Main Flow/Normal Flow/Main Path:

Step.	Actor Action	System Response
1	User accesses the login page.	The system displays a login button labeled "Login with Google".
2	User clicks the "Login with Google" button.	Frontend sends a request to Google's OAuth 2.0 authentication service.
3	User enters their Google credentials (email and password) in the Google login form	Google OAuth login page is displayed.
4	User successfully logs in with Google.	Google generates an ID token and returns it to the frontend.
5	User waits for the process.	Frontend sends the Google ID token to the backend server via HTTP POST to the /auth/google-login endpoint.
6	Backend verifies the ID token.	The backend checks the token's validity using Google's OAuth 2.0 token verification API.
7	Token is valid.	Backend creates a JWT session token with the user role and sends it back to the frontend.
8	Frontend receives the JWT.	Frontend stores the JWT token in an HttpOnly cookie and redirects the user to the dashboard.
9	User is successfully logged in.	The system tracks the user's session and maintains login state.

Alternative Flows:

3a. Cancel login process

Step	Actor Action	System Response

1	User cancels Google login process.	Frontend remains on the login page, and no login attempt is made.

4a. Login failed

Step	Actor Action	System Response
1	User login fails on Google.	Google returns an error message (e.g., invalid credentials or permission denied), which is displayed on the frontend.

7a. Invalid token

Step	Actor Action	System Response
1	Google returns an invalid token.	Backend returns a 401 Unauthorized error, indicating login failure.

Exceptions:

No.	Actor Action	System Response
1	Google OAuth service is down or unreachable.	System returns a 500 Internal Server Error and asks the user to try again later.
2	Backend cannot verify Google token.	The system responds with a 500 Internal Server Error, logging the failure.
3	JWT creation fails (due to security issues or invalid secret key).	System returns a 500 Internal Server Error, logging the issue.

Business Rules:	- The backend (Go, Java) server must verify the Google ID token using Google's OAuth 2.0 API to ensure that the token is valid and has not expired. - JWT session tokens created by the backend should include the user's role (VIP-0, VIP-1, VIP-2, VIP-3) for access control. - The JWT token is stored in an HttpOnly cookie to prevent XSS attacks. - Login sessions should have a predefined expiration time (e.g., 24 hours), after which the user will be required to log in again. - Google login should require user consent to access basic profile information and email.

Implementation Notes:

Frontend (Next.js):

- Use Google OAuth 2.0 client library for handling the login flow.
- On successful Google login, obtain the ID token and send it to the backend.
- Store the JWT session token in an HttpOnly cookie once received from the backend.

Backend:

- Create an endpoint /auth/google-login to handle the POST request with the Google ID token.
- Verify the Google ID token using the Google OAuth 2.0 token verification API (https://oauth2.googleapis.com/tokeninfo).
- Generate a JWT token upon successful verification, with the user's role, and send it back to the frontend.

Use Case Specification

Use Case No.:	UC-03		
Use Case Name:	Logout		
Created By:	Bui The Ky Cuong		
Date:	15/10/2024	Priority	Medium
Actors:	Guest (Unauthenticated user)		
Summary:	A logged-in user can log out of the system either from the "User Profile" page or from the dropdown menu by clicking on the user icon located in the top-right corner.		
Trigger:	The user clicks the "Logout" button in either the User Profile page or the dropdown menu under the user icon.		
Preconditions:	The user must be logged in to perform this action.		
Post–conditions:	The user is logged out of the system. The user's session and any relevant authentication tokens are invalidated. The user is redirected to the login page.		

Main Success Scenario/Main Flow/Normal Flow/Main Path:

Step.	Actor Action	System Response
1	User clicks the user icon located in the top-right corner of the screen or navigates to the "User Profile" page.	The system displays the dropdown menu or the "User Profile" page where the "Logout" button is available.
2	User clicks the "Logout" button.	System logs the user out and redirects them to the login page (or home page depending on the system's design). The system ensures that the user no longer has access to authenticated areas of the application.

Alternative Flows: No

Exceptions:

No.	Actor Action	System Response
1	Logout button is clicked, but the server does not respond.	The system notifies the user of the failure and suggests retrying.

Business Rules:	**Session Invalidation**: The system must securely invalidate the user's session and any authentication tokens stored in cookies. **Redirection**: After successful logout, the user should be redirected to the login page. **Security**: All cached data and session cookies must be cleared after logout to prevent unauthorized access.

Frontend (Next.js):

- The "Logout" button should be implemented as part of the user profile or dropdown menu under the user icon.
- Clicking "Logout" triggers an API call to the backend to invalidate the session and delete any tokens stored in the browser (cookies)
- *Optionally,* a confirmation modal can be added to prevent accidental logout.
- After logout, the user should be redirected to the login or home page.

Backend:

- Implement a /logout endpoint that handles session invalidation.
- Clear any session cookies or authentication tokens stored on the server.

- If using JWT for authentication, consider blacklisting the token until it expires, or ensuring that the user session is no longer valid.
- Handle logout confirmation messages and ensure a secure logout process, including removing any user-specific data from cache or sessions.
- Provide appropriate error handling in case the logout process fails.

Use Case Specification

Use Case No.:	UC-04		
Use Case Name:	Look up information (Spot Price, Future Price, Funding Rate)		
Created By:	Bui The Ky Cuong		
Date:	15/10/2024	Priority	High
Actors:	Patron VIP-0, VIP-1, VIP-2, VIP-3		
Summary:	A user on the dashboard can view standard market information such as spot price, future price, and funding rate. By clicking on these values, the user can access more detailed information, including charts, historical data, and any other relevant metrics.		
Trigger:	User wants to look up specific market information while on the dashboard.		
Preconditions:	The user must be logged in and have access to the dashboard. The system must be connected to real-time data sources for spot prices, future prices, and funding rates. The dashboard should display standard data as an overview.		
Post–conditions:	The user views detailed information about spot prices, future prices, or funding rates, including historical charts. The system may allow the user to switch between different time frames (e.g., 1h, 24h, 7d).		

Main Success Scenario/Main Flow/Normal Flow/Main Path:

Step.	Actor Action	System Response
1	User logs into the system and navigates to the dashboard.	The system displays an overview of market information, including spot prices, future prices, and funding rates.
2	User views the standard information (Preview) on the dashboard.	The system shows real-time spot price, future price, and funding rate updates.
3	User clicks on a specific piece of information (e.g., spot price).	The system opens a detailed view, showing historical charts and related data for the selected market information.
4	User interacts with the charts (e.g., changes the time frame).	The system updates the charts to reflect the new time frame and provides additional details (e.g., historical highs, lows, etc.).
5	User views funding rate details, including recent funding rate history and projected future rates.	The system displays detailed funding rate information, including past funding periods and rates forecast for the next cycle.
6	User closes the detail view.	The system returns the user to the main dashboard with the standard overview.

Alternative Flows:

2a. Data Unavailable

Step	Actor Action	System Response
1	User views the dashboard, but market data is unavailable (e.g., due to API failure).	The system displays an error message indicating that market data is temporarily unavailable, and asks the user to try again later.
2	User attempts to refresh the page or check back later.	System continues to check for data availability and refreshes automatically when data becomes available.

4a. Switching Time Frames:

Step	Actor Action	System Response
1	User selects a different time frame (e.g., 1h, 24h, 7d, etc.) on the chart view.	The system fetches and displays the corresponding historical data for the selected time frame.
2	User zooms in or out on the chart for a specific period.	System dynamically adjusts the chart based on the zoom level.

5a. Funding Rate Data Update:

Step	Actor Action	System Response
1	User views the funding rate section, and data for the next funding cycle is updated.	System highlights the updated information, including any changes in projected funding rates.

Exceptions:

No.	Actor Action	System Response
1	API providing spot price or future price data fails.	The system displays an error message on the dashboard, indicating "Data Unavailable".
2	Data for the selected time frame (e.g., historical data) is not available.	The system shows a message saying "No data available for the selected period".
3	System fails to retrieve funding rate data.	The system returns a "500 Internal Server Error" and logs the issue.

Business Rules:	- **Real-time Updates**: The system must continuously retrieve real-time spot prices, future prices, and funding rates from trusted data sources (such as financial market APIs) and update the dashboard in real-time. - **Chart View**: When viewing details, the system should allow users to switch between various time frames (e.g., 1h, 24h, 7d, 30d) and dynamically adjust the chart view based on user interactions. - **Data Consistency**: The system must ensure that all displayed data is accurate and synchronized with external data sources. - **User Permissions**: Only logged-in Patron with according VIP permission should be able to access detailed information and view full charts. Guest users can only see the standard overview. - **Caching and Performance**: Market data should be cached to optimize performance and reduce redundant calls to external APIs.

Implementation Notes:

Frontend (Next.js):

- Use chart libraries (e.g., Chart.js or D3.js) for rendering interactive historical charts on the detail view.
- Ensure real-time data updates are handled efficiently using WebSockets or Server-Sent Events (SSE) to minimize latency.
- Handle API failures gracefully by displaying error messages and offering retry options.

Backend (Go):

- Create endpoints such as /market/spot, /market/futures, /market/funding-rate to handle requests for different market data types.
- Integrate with financial market data providers' APIs to fetch real-time and historical data for spot prices, future prices, and funding rates.
- Implement caching mechanisms (e.g., Redis) to reduce load on external APIs and improve performance.
- Implement authentication and authorization to ensure only authenticated users can access detailed information.

Use Case Specification

Use Case No.:	UC-05		
Use Case Name:	Purchase VIP through Momo		
Created By:	Bui The Ky Cuong		
Date:	15/10/2024	Priority	High
Actors:	Patron VIP-0, VIP-1, VIP-2, Momo Payment Gateway		
Summary:	A user can purchase VIP status through the pricing page using the Momo payment gateway. The user can access the pricing page by clicking on the "Pricing" button in the navigation bar or after attempting to use a VIP-restricted feature.		
Trigger:	The user clicks on the "Pricing" button or attempts to access a VIP feature, prompting a pricing page or pop-up.		
Preconditions:	The user must be logged in to make a purchase. Momo payment gateway is integrated and available for transactions. The system should have an active pricing plan for VIP status.		
Post–conditions:	The user successfully purchases VIP status, and their account is upgraded. The system records the transaction details and updates the user's subscription status.		

Main Success Scenario/Main Flow/Normal Flow/Main Path:

Step.	Actor Action	System Response
1	User clicks the "Pricing" button from the navigation bar or attempts to use a VIP-restricted feature.	The system navigates to the "Pricing" page.
2	User selects the desired VIP subscription plan.	The system highlights the selected plan and provides a "Purchase" button.
3	User clicks the "Purchase" button.	The system redirects the user to the Momo payment gateway for processing the payment.
4	User completes the payment through Momo.	The Momo payment gateway processes the payment and returns a confirmation status (success/failure) to the system.

5	If payment is successful, the system upgrades the user's account to VIP status.	The system updates the user's account, sets their VIP status, and confirms the subscription. A success message is shown to the user.
6	User returns to the application and can now access VIP-restricted features.	The system grants access to VIP features and displays the user's current VIP status in their profile.

Alternative Flows:

1a. User is Not Logged In:

Step	Actor Action	System Response
1	User clicks the "Pricing" button or attempts to access VIP features without being logged in.	The system prompts the user to log in before proceeding to the pricing page.
2	User logs in successfully.	The system redirects the user to the pricing page after login.

3a. User Cancels Purchase:

Step	Actor Action	System Response
1	User selects a plan but cancels the purchase before completing the payment.	The system cancels the transaction and returns the user to the pricing page with no changes to their subscription.

4a. Payment Failure:

Step	Actor Action	System Response
1	User attempts payment, but the payment fails due to insufficient funds or other issues.	The system receives the failure response from Momo, displays an error message to the user, and offers retry options.

Exceptions:

No.	Actor Action	System Response
1	The Momo payment gateway is temporarily unavailable.	The system notifies the user of the issue, provides an error message, and allows the user to try again later.
2	User's account does not update after payment.	The system logs the issue, displays a temporary error message to the user, and prompts the user to contact support.

Business Rules:	- Momo Payment Integration: The system must be integrated with the Momo payment gateway and handle all responses from the gateway securely, including success, failure, and cancellation statuses. - Subscription Update: After successful payment, the system must immediately update the user's VIP status and grant access to VIP-only features. - Pricing Page Display: The pricing page should clearly show all available VIP plans, including duration and cost. Users can also view VIP benefits. - Access Control: Only VIP users should be able to access VIP-restricted features. If a non-VIP user attempts access, they are redirected to the pricing page.

Implementation Notes:

Frontend (Next.js):

- Use a modal pop-up for redirecting non-VIP users attempting to access VIP-restricted features.
- The pricing page should use React components to display real-time pricing and include a seamless payment flow with Momo.
- Integrate Momo's SDK for payment processing to handle user redirection, authentication, and transaction confirmation.
- Handle success and error states with clear messaging for the user.

Backend (Go):

- Create an endpoint (e.g., /payment/momo) to handle Momo payment initiation and webhook callbacks.
- Use secure methods to handle the user's payment token from Momo and ensure that transaction details are recorded.
- Update the user's subscription status in the database upon receiving confirmation of payment from Momo.
- Implement logging and error handling to manage failed transactions or communication issues with Momo.
- Send a confirmation email or notification to the user after successful payment and VIP upgrade.

Use Case Specification

Use Case No.:	UC-06		
Use Case Name:	Manage Profile		
Created By:	Bui The Ky Cuong		
Date:	16/10/2024	Priority	High
Actors:	Patron VIP-0, VIP-1, VIP-2, VIP-3		
Summary:	A patron can view and update their profile information from the user profile page. The user can see their personal information (name, email, etc.) and update it as needed.		
Trigger:	The user navigates to the "User Profile" page by selecting the “view profile” option from the dropdown menu when clicking on the Avatar Icon or a designated link within the web application.		
Preconditions:	The user must be logged in to access the profile management functionality. The system has a valid profile record for the user stored in the database.		
Post–conditions:	Patron can view their profile with latest update information. Patron’s profile information is updated in the system. The system confirms the changes have been successfully saved.		

Main Success Scenario/Main Flow/Normal Flow/Main Path:

Step.	Actor Action	System Response
1	Patron navigates to the "Profile" page via the user icon in the top-right corner or through a dedicated profile link.	The system retrieves the user's profile information (e.g., name, email, etc.) from the database and displays it.
2	User reviews the displayed profile information.	The system shows the user's current profile details in editable fields (name, email, password, etc.).

3	User updates their profile information (e.g., changing their name or email address).	The system validates the input fields and ensures that the provided data is correct (e.g., email format, password strength).
4	User clicks the "Save" button.	The system validates the updated information and checks for any conflicts (e.g., existing email address). If the input is valid, the system updates the user profile in the database. Then, the system displays a success message confirming the changes have been saved.
5	The user's updated information is reflected in the system and used in subsequent interactions.	The system continues to show the updated profile information in the user's account.

Alternative Flows:

3a. Invalid Email Format:

Step	Actor Action	System Response
1	User provides an email in an incorrect format.	The system shows an error message indicating the invalid email format.
2	User corrects the email address and resubmits.	The system accepts the valid email format and continues the flow.

4a. Email Already Exists:

Step	Actor Action	System Response
1	User tries to update their profile with an email that is already in use by another account.	The system shows an error message indicating the email address is already in use.
2	User provides a different email address.	The system checks for conflicts again and allows the process to continue if there are no issues.

4b. Weak Password:

Step	Actor Action	System Response
1	User provides a weak password (if updating password).	The system shows a warning or error message indicating that the password does not meet security criteria (e.g., length, character diversity).
2	User updates the password with a stronger one.	The system accepts the new password and continues the flow.

5a. System Error During Update:

Step	Actor Action	System Response
1	The system encounters an error while updating the user's profile (e.g., database failure).	The system shows an error message and suggests retrying the action later.
2	Patron retries the update or logs out.	

		The system retries the update or waits for the issue to be resolved.

Exceptions:

No.	Actor Action	System Response
1	User loses connection or closes the window before saving changes.	The system discards unsaved changes and does not update the profile.
2	System fails to retrieve the user's profile data.	The system shows an error message and does not allow updates until the issue is resolved.

Business Rules:	- **Profile Uniqueness:** The system must ensure that certain profile details (e.g., email address) remain unique across all users. - **Input Validation:** The system must validate all profile fields, ensuring that inputs meet required formats (e.g., email format, password criteria). - **Security:** Sensitive profile fields (e.g., password) must be updated securely. Any password changes should follow best security practices (e.g., hashing, encryption). - **Notification (Optional):** The system could notify users via email when a significant profile change (e.g., email or password) is made.

Implementation Notes:

Frontend (Next.js):

- The "User Profile" page should include editable fields for each profile attribute (e.g., name, email, password).
- Use form validation for fields such as email (regex for proper formatting) and password (strength requirements).
- Provide user feedback for successful updates or errors during the update process.
- Implement responsive design to ensure the profile page works well across different devices.
- After successful updates, the profile page should reflect the new data without requiring a page reload.

Backend:

- The /user/profile API endpoint should support both **GET** (to fetch current user data) and **PUT** (to update user data) operations.
- When updating, the system should validate the data (e.g., checking for duplicate emails or weak passwords).
- Securely update sensitive fields like the password, ensuring it is hashed before storage.
- Handle potential concurrency issues, especially if the user is trying to update their profile while another process is modifying their data.
- Use transaction management to ensure consistency when updating multiple fields.
- Log any significant profile changes (e.g., email or password updates) for audit and security purposes.

Use Case Specification

Use Case No.:	UC-07		
Use Case Name:	View Kline Price		
Created By:	Bui The Ky Cuong		
Date:	16/10/2024	Priority	Medium
Actors:	Patron VIP-1, VIP-2, VIP-3		

Summary:	A patron can look up the Kline price chart. This feature is unlocked for users who have purchased at least VIP-1 status. The user can see the Kline price chart on the dashboard or pricing details.
Trigger:	The patron clicks on the "Kline Price Preview" feature in the dashboard.
Preconditions:	The Patron must be logged in and have VIP-1 or higher status. The system has up-to-date Kline price information stored in the database or accessible from an external API.
Post–conditions:	The patron successfully views the Kline price chart.

Main Success Scenario/Main Flow/Normal Flow/Main Path:

Step.	Actor Action	System Response
1	Patron logs into their account.	The system verifies the user's credentials and VIP status.
2	Patron navigates to the "Dashboard"	The system displays available features, including the "Kline Price" option.
3	Patron clicks on the "Kline Price" option.	The system checks if the user has VIP-1 or higher access.
4	Patron waits for system responses.	The system retrieves the latest Kline price data (historical price data for the requested asset) from the database or via an external API.
5	The user interacts with the chart (zoom in/out, change time frames).	The system displays the Kline price chart, showing relevant price trends and charting options (e.g., time range selection, zoom features).
6	The patron views and interacts with the chart as needed (e.g., selecting different time intervals).	The system updates the chart dynamically based on user input.

Alternative Flows:

3a. User Does Not Have VIP-1 or Higher Access:

Step	Actor Action	System Response
1	User without VIP-1 tries to access the Kline Price chart.	The system displays an error message or pop-up suggesting the user purchase VIP-1 or higher to unlock this feature.
2	User clicks on the pop-up to navigate to the "Pricing" page.	The system redirects the user to the pricing page to review VIP plans.

4a. System Cannot Retrieve Kline Price Data:

Step	Actor Action	System Response
1	The system fails to retrieve the Kline price data due to an external API or internal database error.	The system displays a friendly error message indicating that the price data is temporarily unavailable and suggests trying again later.
2	Patron attempts to refresh the page.	The system retries fetching the data and, if successful, displays the Kline price chart.

Exceptions:

No.	Actor Action	System Response

1	Patron loses connection during the lookup.	The system retries the connection or shows an error message indicating a connection issue.
2	The system encounters unexpected issues (e.g., database crash) during Kline price retrieval.	The system logs the error and displays a fallback message suggesting that the feature is temporarily unavailable.

Business Rules:	**- VIP Access Control:** Only users with VIP-1 or higher access can look up the Kline price chart. Users with lower access levels should be prompted to upgrade. - **Real-Time Data:** The system should provide the most up-to-date Kline price information, either from its own database or from external sources (e.g., financial APIs). - **Interaction:** Users should be able to interact with the chart to customize their view, such as selecting different timeframes (1 minute, 5 minutes, 1 hour, 1 day) or zooming in and out.

Implementation Notes:

Frontend (Next.js):

- The frontend should have a "Kline Price" component or page that renders the price chart using a charting library such as **Chart.js** or **Highcharts**.
- Implement user interface components to allow interaction, such as time range selection (1m, 5m, 1h, 1d) and zooming options.
- Handle cases where the user does not have the required VIP access by displaying a pop-up message and redirecting to the pricing page.

Backend:

- The backend should validate the user's VIP level when accessing the Kline Price feature.
- Create an API endpoint (e.g., /klineprice) to handle requests for retrieving Kline price data. This can be done through a combination of in-house database storage and external financial data APIs (e.g., Binance, Alpha Vantage).
- Implement proper caching mechanisms to avoid repeatedly fetching the same data from external APIs.
- Log every access attempt to the Kline Price chart, including user ID and timestamp, for future audits and usage statistics.

Security Considerations:

- Ensure that only authorized users (VIP-1 or higher) can access the Kline Price chart by validating the user's session and VIP level on both the frontend and backend.
- All API calls to external financial services should be securely handled (use HTTPS and proper authentication mechanisms).

Performance Considerations:

- Use caching where appropriate to avoid unnecessary load on external APIs and to improve page load times when rendering the Kline chart.
- Ensure the chart is dynamically updated without requiring full page reloads for better user experience.

Use Case Specification

Use Case No.:	UC-08		
Use Case Name:	Create Alert for Funding Rate		
Created By:	Bui The Ky Cuong		
Date:	16/10/2024	Priority	High
Actors:	Patron VIP-2, VIP-3		

Summary:	A patron with VIP-2 or higher can set an alert for funding rates on the "Alert Management Page." The alert will notify the user when a specified threshold is met, such as when the funding rate rises above or falls below a certain value.
Trigger:	The patron navigates to the "Alert Management" page and selects the option to set an alert for the funding rate.
Preconditions:	The user must be logged in and have VIP-2 or higher status. The system has access to up-to-date funding rate information via an internal or external data source. The "Alert Management" page must be accessible, and the user should have previously set up the ability to receive notifications (email, app notification, etc.).
Post–conditions:	The system successfully records the alert parameters and monitors the funding rate for threshold violations. The patron receives a notification when the funding rate crosses the specified threshold.

Main Success Scenario/Main Flow/Normal Flow/Main Path:

Step.	Actor Action	System Response
1	Patron logs into their account.	The system verifies the user's credentials and VIP status.
2	Patron navigates to the "Alert Management" page from the dashboard or profile section.	The system displays the user's current alert settings and options to add a new alert.
3	Patron selects the option to set an alert for the funding rate.	The system shows a form for the patron to configure the alert parameters (e.g., threshold values for the funding rate, frequency of alerts).
4	Patron enters the desired parameters, such as: - The funding rate threshold (e.g., rate higher than 0.01% or lower than -0.01%). - How frequently they wish to be notified (once per day, immediately, etc.). - The method of notification (email, app notification, telegram.).	Wait for patron's input.
5	Patron submits the form.	The system validates the input data and confirms that the user is eligible (VIP-2 or higher). The system saves the alert settings in the database.
6	The system starts monitoring the funding rate based on the user's specified parameters.	The system will check the funding rate periodically to see if it meets the threshold condition set by the user.

Alternative Flows:

3a. User Does Not Have VIP-2 or Higher Access:

Step	Actor Action	System Response
1	A patron without VIP-2 access tries to set an alert for the funding rate.	The system displays an error message or pop-up suggesting that the user needs to upgrade to VIP-2 or higher to access this feature.
2	User clicks on the pop-up to navigate to the "Pricing" page.	The system redirects the user to the pricing page to review VIP plans.

5a. Invalid Input for Funding Rate Parameters:

Step	Actor Action	System Response

1	Patron enters invalid data (e.g., a non-numeric value for the funding rate threshold).	The system displays a validation error indicating that the input is invalid and prompts the patron to enter valid data.

Exceptions:

No.	Actor Action	System Response
1	Patron loses connection during the alert setup.	The system shows a connection error message and discard the progress.
2	The system encounters issues fetching funding rate data.	The system displays a fallback message indicating that the feature is temporarily unavailable and suggests trying again later.

Business Rules:	- **VIP Access Control:** Only users with VIP-2 or higher access can set an alert for the funding rate. Lower-tier users should be prompted to upgrade. - **Threshold Monitoring:** The system must continuously monitor funding rate changes and trigger notifications if the user's set threshold is crossed. - **Real-Time Data:** The system must fetch the most up-to-date funding rate data either from its own database or via an external financial API.

Implementation Notes:

Frontend (Next.js):

- The frontend should have a dedicated page or section (e.g., "Alert Management" page) where users can manage their alert settings. This will include a form where they can configure parameters for the funding rate alert.
- The form should include input validation (e.g., checking that the threshold values are numeric).
- The form should also allow the user to choose notification methods (e.g., email, app notification, SMS).

Backend:

- Implement an API endpoint (e.g., /alerts/funding-rate) to handle the creation, modification, and deletion of funding rate alerts.
- Upon submission of the alert setup form, validate the user's VIP level and the input data.
- Store the alert details in the database and associate them with the user's account.
- A background process should continuously monitor the funding rate against the user-defined thresholds and send notifications if necessary.

Notification System:

- Ensure the backend is capable of sending notifications via different channels (email, in-app, SMS) based on the user's preferences.
- Create a background service to regularly check the funding rate and compare it with user-defined thresholds. When an alert is triggered, send the appropriate notification to the user.

Security Considerations:

- Validate that only authorized users (VIP-2 or higher) can set and manage funding rate alerts.
- Ensure that sensitive information (e.g., user email or phone number for notifications) is securely handled and stored.

Performance Considerations:

- Use caching and efficient data retrieval methods to minimize the impact of frequent funding rate checks.
- Consider using a background job queue to handle alert triggers and notifications, ensuring that these tasks do not block other user operations.

Use Case Specification

Use Case No.:	UC-09		
Use Case Name:	Create Alert for Spot Price		
Created By:	Bui The Ky Cuong		
Date:	16/10/2024	Priority	High
Actors:	Patron VIP-2, VIP-3		
Summary:	A patron with VIP-2 or higher can set an alert for spot prices on the "Alert Management Page." The alert will notify the user when a specified spot price crosses a predefined threshold, either above or below a certain value.		
Trigger:	The patron navigates to the "Alert Management" page and selects the option to set an alert for spot price.		
Preconditions:	The user must be logged in and must have VIP-2 or higher status. The system must have access to real-time spot price data through an internal or external financial service API. The "Alert Management" page is accessible, and the user should have configured notification preferences (such as email, app notification, or Telegram).		
Post–conditions:	The system successfully stores the spot price alert parameters and monitors spot price changes. The patron is notified when the spot price crosses the predefined threshold set by the user.		

Main Success Scenario/Main Flow/Normal Flow/Main Path:

Step.	Actor Action	System Response
1	Patron logs into their account.	The system verifies the user's credentials and VIP status.
2	Patron navigates to the "Alert Management" page from the dashboard or profile section.	The system displays the user's current alert settings and options to add a new alert.
3	Patron selects the option to set an alert for the spot price.	The system shows a form for the user to configure the alert parameters (e.g., spot price threshold, notification frequency, etc.).
4	Patron enters the desired parameters, such as: - Spot price threshold (e.g., price above $50 or below $30). - Notification frequency (e.g., once per hour, once per day). - Notification method (e.g., email, app notification, Telegram).	Wait for patron’s input.
5	Patron submits the form.	The system validates the input and checks that the user has VIP-2 or higher status. The system saves the alert settings to the database.
6	The system monitors spot price data in real-time based on the user's alert parameters.	The system checks the current spot price periodically and triggers the alert if the threshold is reached.

Alternative Flows:

3a. User Does Not Have VIP-2 or Higher Access:

Step	Actor Action	System Response
1	A user without VIP-2 access tries to set an alert for spot prices.	The system displays an error message or pop-up suggesting that the user needs to upgrade to VIP-2 or higher to access this feature.

2	User clicks on the pop-up to navigate to the "Pricing" page.	The system redirects the user to the pricing page to review VIP plans.

5a. Invalid Input for Spot Price Parameters:

Step	Actor Action	System Response
1	Patron enters invalid data (e.g., a non-numeric value for the Spot price).	The system displays a validation error and prompts the user to enter valid data (e.g., numeric values for the spot price threshold).

Exceptions:

No.	Actor Action	System Response
1	Patron loses connection during the alert setup.	The system shows a connection error message and discard the progress.
2	The system cannot fetch real-time spot price data.	The system informs the user that the spot price data is temporarily unavailable and suggests trying again later.

Business Rules:	- **VIP Access Control:** Only users with VIP-2 or higher access can set and manage spot price alerts. - **Threshold Monitoring:** The system should continually monitor the spot price and send notifications when user-defined thresholds are reached. - **Real-Time Spot Price Data:** The system must fetch the latest spot price data either from its own database or through an external API in real-time or near real-time..

Implementation Notes:

Frontend (Next.js):

- The "Alert Management" page should include a form where users can define the parameters for spot price alerts. The form should allow for setting the price threshold and frequency of alerts, as well as choosing the notification method.
- The form should include input validation to ensure that spot price thresholds are numeric and within a reasonable range.
- The user interface should allow users to view, modify, or delete existing spot price alerts.

Backend:

- The backend must provide an API endpoint (e.g., /alerts/spot-price) for creating, updating, and deleting spot price alerts.
- The API should validate that the user has VIP-2 or higher access before allowing them to set or update alerts.
- Upon receiving a request to set an alert, the backend should store the alert configuration in the database and associate it with the user's account.

Spot Price Monitoring Service:

- A background service should monitor the spot price in real-time, comparing the current price against user-defined thresholds.
- When an alert condition is met, the system should trigger a notification to the user based on their preferences (e.g., email, app notification, Telegram).

Notification System:

- Ensure that the system can send notifications via multiple channels (email, app notifications, Telegram) depending on the user's settings.
- The notification system should trigger promptly when the spot price crosses the user-defined threshold.

Security Considerations:

- Only authenticated users with VIP-2 access should be allowed to set spot price alerts.
- The system must securely store user preferences and alert configurations to prevent unauthorized access.

Performance Considerations:

- The system should use caching and efficient polling techniques to minimize the performance impact of real-time spot price monitoring.
- A background job queue could be employed to handle multiple spot price alert triggers in parallel, ensuring timely notifications without blocking other processes.

Use Case Specification

Use Case No.:	UC-10		
Use Case Name:	Create Alert for Future Price		
Created By:	Bui The Ky Cuong		
Date:	16/10/2024	Priority	High
Actors:	Patron VIP-2, VIP-3		
Summary:	A patron with VIP-2 or higher can set an alert for future prices on the "Alert Management Page." The alert will notify the user when a specified future price crosses a predefined threshold, either above or below a certain value.		
Trigger:	The patron navigates to the "Alert Management" page and selects the option to set an alert for the future price.		
Preconditions:	The user must be logged in and must have VIP-2 or higher status. The system must have access to real-time spot price data through an internal or external financial service API. The "Alert Management" page is accessible, and the user should have configured notification preferences (such as email, app notification, or Telegram).		
Post–conditions:	The system successfully stores the future price alert parameters and monitors future price changes. The patron is notified when the future price crosses the predefined threshold set by the user.		

Main Success Scenario/Main Flow/Normal Flow/Main Path:

Step.	Actor Action	System Response
1	Patron logs into their account.	The system verifies the user's credentials and VIP status.
2	Patron navigates to the "Alert Management" page from the dashboard or profile section.	The system displays the user's current alert settings and options to add a new alert.
3	Patron selects the option to set an alert for the future price.	The system shows a form for the user to configure the alert parameters (e.g., spot price threshold, notification frequency, etc.).
4	Patron enters the desired parameters, such as: - Future price threshold (e.g., price above $500 or below $300). - Notification frequency (e.g., once per hour, once per day). - Notification method (e.g., email, app notification, SMS).	Wait for input.
5	Patron submits the form.	The system validates the input and checks that the user has VIP-2 or higher status. The system saves the alert settings to the database.

6	The system monitors future price data in real-time based on the user's alert parameters.	The system checks the current future price periodically and triggers the alert if the threshold is reached.

Alternative Flows:

3a. User Does Not Have VIP-2 or Higher Access:

Step	Actor Action	System Response
1	A user without VIP-2 access tries to set an alert for future prices.	The system displays an error message or pop-up suggesting that the user needs to upgrade to VIP-2 or higher to access this feature.
2	User clicks on the pop-up to navigate to the "Pricing" page.	The system redirects the user to the pricing page to review VIP plans.

5a. Invalid Input for Spot Price Parameters:

Step	Actor Action	System Response
1	Patron enters invalid data (e.g., a non-numeric value for the Future price).	The system displays a validation error and prompts the user to enter valid data (e.g., numeric values for the future price threshold).

Exceptions:

No.	Actor Action	System Response
1	Patron loses connection during the alert setup.	The system shows a connection error message and discard the progress.
2	The system cannot fetch real-time Future price data.	The system informs the user that the future price data is temporarily unavailable and suggests trying again later.

Business Rules:	- **VIP Access Control:** Only users with VIP-2 or higher access can set and manage spot price alerts. - **Threshold Monitoring:** The system should continually monitor the future price and send notifications when user-defined thresholds are reached. - **Real-Time Spot Price Data:** The system must fetch the latest spot future data either from its own database or through an external API in real-time or near real-time..

Frontend (Next.js):

- The "Alert Management" page should include a form where users can define the parameters for future price alerts. The form should allow for setting the price threshold and frequency of alerts, as well as choosing the notification method.
- The form should include input validation to ensure that future price thresholds are numeric and within a reasonable range.
- The user interface should allow users to view, modify, or delete existing future price alerts.

Backend (Go):

- The backend must provide an API endpoint (e.g., /alerts/future-price) for creating, updating, and deleting future price alerts.
- The API should validate that the user has VIP-2 or higher access before allowing them to set or update alerts.
- Upon receiving a request to set an alert, the backend should store the alert configuration in the database and associate it with the user's account.

Future Price Monitoring Service:

- A background service should monitor the future price in real-time, comparing the current price against user-defined thresholds.
- When an alert condition is met, the system should trigger a notification to the user based on their preferences (e.g., email, app notification, Telegram).

Use Case Specification

Use Case No.:	UC-11		
Use Case Name:	Create New or Delisted Symbol Alert		
Created By:	Bui The Ky Cuong		
Date:	16/10/2024	Priority	High
Actors:	Patron VIP-2, VIP-3		
Summary:	A patron with VIP-2 or higher can set an alert to be notified when new symbols (coins or tokens) are listed on an exchange or when a symbol is delisted. The system will notify the user when a new symbol is added or removed from the platform, based on user-defined criteria.		
Trigger:	The patron navigates to the "Alert Management" page and selects the option to set an alert for new or delisted symbols.		
Preconditions:	The user must be logged in and have VIP-2 or higher status. The system must be able to access exchange data for new and delisted symbols via API or other sources. The "Alert Management" page is accessible, and the user should have configured notification preferences (such as email, app notification, or Telegram).		
Post–conditions:	The system successfully stores the alert parameters and monitors for new or delisted symbols. The patron is notified when a new symbol is listed or an existing one is delisted		

Main Success Scenario/Main Flow/Normal Flow/Main Path:

Step.	Actor Action	System Response
1	Patron logs into their account.	The system verifies the user's credentials and VIP status.
2	Patron navigates to the "Alert Management" page from the dashboard or profile section.	The system displays the user's current alert settings and options to add a new alert.
3	Patron selects the option to set an alert for new or delisted symbols.	The system shows a form for the user to configure alert parameters (e.g., whether to notify about new listings, delistings, or both, and the method of notification).
4	Patron enters the desired parameters, such as: - Notification type (new listing, delisting, or both). - Frequency of alerts (e.g., once per day, immediately). - Notification method (email, app notification, Telegram).	

5	Patron submits the form.	The system validates the input and checks that the user has VIP-2 or higher status. The system saves the alert settings to the database.
6	The system monitors the exchange data for new or delisted symbols based on the user's alert parameters..	The system triggers a notification when a new symbol is listed or an existing symbol is delisted.

Alternative Flows:

3a. User Does Not Have VIP-2 or Higher Access:

Step	Actor Action	System Response
1	A user without VIP-2 access tries to set an alert for new or delisted symbols.	The system displays an error or pop-up suggesting that the user upgrade to VIP-2 or higher to use this feature.
2	User clicks on the pop-up to navigate to the "Pricing" page.	The system redirects the user to the pricing page to review VIP plans.

5a. Invalid Input for Spot Price Parameters:

Step	Actor Action	System Response
1	Patron enters invalid data (e.g., invalid alert frequency).	The system displays a validation error and prompts the user to enter valid data (e.g., a valid notification frequency or alert type).

Exceptions:

No.	Actor Action	System Response
1	Patron loses connection during the alert setup.	The system shows a connection error message and discard the progress.
2	The system cannot fetch exchange data for new or delisted symbols.	The system informs the user that the exchange data is temporarily unavailable and suggests trying again later.

Business Rules:	**VIP Access Control:** Only users with VIP-2 or higher access can set alerts for new or delisted symbols. **Real-Time Monitoring:** The system must continuously monitor exchange data for changes in listed symbols (new additions and delistings). **Notification Type:** The user can select whether they want to be notified about new symbol listings, delistings, or both. **Real-Time Data Access:** The system must fetch real-time or near real-time data about new and delisted symbols from external APIs (such as Binance or CoinMarketCap).

Implementation Notes:

Frontend (Next.js):

- The "Alert Management" page should include a form where users can set alerts for new or delisted symbols. The form should allow the user to select notification types (new listings, delistings, or both), set alert frequency, and choose the notification method (email, app notification, Telegram).
- The form should include input validation to ensure that the alert parameters are valid.
- The user interface should allow users to view, modify, or delete existing alerts for symbols.

Backend:

- The backend must provide an API endpoint (e.g., /alerts/symbol-alerts) for creating, updating, and deleting alerts for new or delisted symbols.

- The API should validate that the user has VIP-2 or higher access before allowing them to set or update symbol alerts.
- Upon receiving a request to set an alert, the backend should store the alert configuration in the database and associate it with the user's account.

Symbol Monitoring Service:

- A background service should monitor the exchange data (from sources like Binance, CoinMarketCap, or CoinGecko) for changes in listed symbols.
- When a new symbol is listed or an existing symbol is delisted, the service should compare the change against user-defined alerts and trigger notifications accordingly.

Notification System:

- The system must be able to send notifications via multiple channels (email, app notifications, Telegram) depending on the user's settings.
- The notification system should trigger promptly when a new symbol is listed or an existing symbol is delisted based on the user-defined alert parameters.

Use Case Specification

Use Case No.:	UC-12		
Use Case Name:	Create Alert for Spot & Future Price Differences		
Created By:	Bui The Ky Cuong		
Date:	16/10/2024	Priority	High
Actors:	Patron VIP-2, VIP-3		
Summary:	A patron with VIP-2 or higher can set an alert to be notified when the difference between the spot price and future price of a symbol exceeds a specified threshold. The system will notify the user when the difference meets or exceeds the user-defined threshold.		
Trigger:	The patron navigates to the "Alert Management" page and selects the option to set an alert for spot and future price differences.		
Preconditions:	The user must be logged in and have VIP-2 or higher status. The system must have access to real-time spot and future price data from a financial API or other internal/external sources. The "Alert Management" page is accessible, and the user should have configured notification preferences (such as email, app notification, or Telegram).		
Post–conditions:	The system successfully stores the alert parameters and monitors the spot and future price difference. The patron is notified when the spot-future price difference meets or exceeds the predefined threshold set by the user.		

Main Success Scenario/Main Flow/Normal Flow/Main Path:

Step.	Actor Action	System Response
1	Patron logs into their account.	The system verifies the user's credentials and VIP status.
2	Patron navigates to the "Alert Management" page from the dashboard or profile section.	The system displays the user's current alert settings and options to add a new alert.
3	Patron selects the option to set an alert for spot and future price differences.	The system shows a form for the user to configure the alert parameters (e.g., threshold for price difference, notification frequency, etc.).

4	Patron enters the desired parameters, such as: - Price difference threshold (e.g., spot price is 5% higher than future price). - Notification frequency (e.g., every hour, every day). - Notification method (e.g., email, app notification, Telegram).	Wait for patron's input.
5	Patron submits the form.	The system validates the input and checks that the user has VIP-2 or higher status. The system saves the alert settings to the database.
6	The system monitors the spot and future price data in real-time based on the user's alert parameters.	The system calculates the price difference and triggers the alert if the threshold is exceeded.

Alternative Flows:

3a. User Does Not Have VIP-2 or Higher Access:

Step	Actor Action	System Response
1	A user without VIP-2 access tries to set an alert for spot and future price differences.	The system displays an error or pop-up suggesting that the user upgrade to VIP-2 or higher to use this feature.
2	User clicks on the pop-up to navigate to the "Pricing" page.	The system redirects the user to the pricing page to review VIP plans.

5a. Invalid Input for Spot Price Parameters:

Step	Actor Action	System Response
1	Patron enters invalid data (e.g., non-numeric threshold value).	The system displays a validation error and prompts the user to enter valid data (e.g., numeric values for the price difference threshold).

Exceptions:

No.	Actor Action	System Response
1	Patron loses connection during the alert setup.	The system shows a connection error message and discard the progress.
2	The system cannot fetch real-time spot or future price data.	The system informs the user that the spot or future price data is temporarily unavailable and suggests trying again later.

Business Rules:	**VIP Access Control:** Only Patron with VIP-2 or higher access can set and manage alerts for spot and future price differences. **Threshold Monitoring:** The system must monitor the difference between the spot price and future price in real-time and send notifications when the user-defined threshold is exceeded. **Real-Time Data Access:** The system must fetch real-time or near real-time data for both spot and future prices from external financial APIs

Implementation Notes:

Frontend (Next.js):

- The "Alert Management" page should include a form where users can set alerts for spot and future price differences. The form should allow the user to set the threshold value, select alert frequency, and choose a notification method (e.g., email, app notification, SMS).
- The form should include input validation to ensure that threshold values are numeric and within a valid range.

- The user interface should allow users to view, modify, or delete existing price difference alerts.

Backend:

- The backend must provide an API endpoint (e.g., /alerts/spot-future-difference) for creating, updating, and deleting alerts for price differences between spot and future prices.
- The API should validate that the user has VIP-2 or higher access before allowing them to set or update price difference alerts.
- Upon receiving a request to set an alert, the backend should store the alert configuration in the database and associate it with the user's account.

Price Difference Monitoring Service:

- A background service should monitor spot and future prices in real-time, calculating the difference based on user-defined thresholds.
- When the price difference exceeds the threshold, the service should trigger a notification to the user based on their preferences.

Notification System:

- The system should send notifications via multiple channels (email, app notifications, Telegram) based on the user's settings.
- The notification system should trigger promptly when the spot-future price difference meets or exceeds the user-defined threshold.

Use Case Specification

Use Case No.:	UC-13		
Use Case Name:	View Alert		
Created By:	Bui The Ky Cuong		
Date:	06/12/2024	Priority	High
Actors:	Patron VIP-2, VIP-3		
Summary:	Patrons with VIP-2 or higher can view their configured alerts via Telegram by issuing a specific command to the Telegram bot. The bot responds with a list of active alerts, including details like type, conditions, and notification methods.		
Trigger:	The patron issues a command in Telegram (e.g., /view_alerts) to retrieve their current alert configurations.		
Preconditions:	The patron must be logged in and have VIP-2 or higher status. The patron must have connected their Telegram account to the system. The system must have active alerts configured for the patron. The Telegram bot must be operational and connected to the system backend.		
Post–conditions:	The patron receives a response from the Telegram bot displaying their active alerts. The system logs the request for viewing alerts for auditing purposes.		

Main Success Scenario/Main Flow/Normal Flow/Main Path:

Step.	Actor Action	System Response
1	Patron opens the Telegram bot and issues the command /view_alerts.	Telegram bot forwards the request to the system backend.
2	System validates the patron’s authentication and VIP level.	System retrieves the list of active alerts associated with the patron.
3	System formats the alert details into a readable message	Telegram bot sends the message back to the patron.

4	Patron views the list of alerts directly in the Telegram chat.	System logs the delivery and ensures the user can view it on the selected platform (e.g. email).

Alternative Flows:

1a. Patron Is Not Connected to Telegram Bot:

Step	Actor Action	System Response
1	Patron issues the /view_alerts command but has not connected their Telegram account.	Telegram bot responds with a message: "Please connect your Telegram account to view alerts."

2a. Patron Has No Active Alerts:

Step	Actor Action	System Response
1	Patron issues the /view_alerts command.	Telegram bot responds: "You currently have no active alerts configured."

2b. Patron Does Not Have VIP-2 or Higher Access:

Step	Actor Action	System Response
1	Patron issues the /view_alerts command but is a VIP-1 or lower user.	Telegram bot responds with an error: "This feature is available for VIP-2 or higher users only."

Exceptions:

No.	Actor Action	System Response
1	Telegram bot fails to communicate with the system backend.	Telegram bot sends a message: "Unable to retrieve alerts at this time. Please try again later."
2	Patron enters an invalid command in Telegram.	Telegram bot sends a message: "Invalid command. Use /view_alerts to view your active alerts."

Business Rules:	**Access Control**: Only VIP-2 or higher users can configure and receive notifications. **Real-Time Data**: The system must provide up-to-date details on the user's active alerts. **Telegram Integration**: Patrons must have connected their Telegram account to the system to use this feature. **Audit Log**: The system must log all requests to view alerts for security and debugging purposes.

Telegram Command Example

Command: /view_alerts

Response:

[Crypto Price Alert System]
Here are your active alerts:

1. **BTCUSDT Spot Price**
   - Condition: >= $50,000
   - Notification Method: Email

2. **ETHUSDT Funding Rate**
   - Condition: <= 0.02%
   - Notification Method: Telegram

Use Case Specification

Use Case No.:	UC-14		
Use Case Name:	Delete Alert		
Created By:	Bui The Ky Cuong		
Date:	19/12/2024	Priority	Medium
Actors:	Patron VIP-2, VIP-3		
Summary:	Patrons with VIP-2 or higher can delete their existing alerts using the **Alert Management** page. The system removes the selected alert from the user's account and stops monitoring for the specified condition.		
Trigger:	The patron navigates to the **Alert Management** page and selects an alert to delete.		
Preconditions:	The patron must be logged in and have VIP-2 or higher status. The patron must have at least one active alert configured. The system must be able to identify and delete alerts based on the user's selection.		
Post–conditions:	The specified alert is removed from the user's account. The system stops monitoring the condition associated with the deleted alert. The deletion is logged in the system for auditing purposes.		

Main Success Scenario/Main Flow/Normal Flow/Main Path:

Step.	Actor Action	System Response
1	Patron logs in and navigates to the **Alert Management** page.	System displays the list of active alerts for the patron.
2	Patron selects an alert to delete and confirms the action.	System validates the request and identifies the alert to delete.
3	System deletes the specified alert from the database.	System stops monitoring the condition associated with the alert.
4	Patron sees a confirmation message.	System logs the deletion for auditing purposes.

Alternative Flows:

2a. Patron Cancels Deletion:

Step	Actor Action	System Response
1	Patron selects an alert but cancels the deletion confirmation.	System retains the alert and displays the current list of active alerts.

2b. Alert Does Not Exist:

Step	Actor Action	System Response
1	Patron selects an alert that has already been deleted or does not exist.	System displays an error message: "This alert no longer exists."

Exceptions:

No.	Actor Action	System Response
1	System fails to delete the alert due to a database error.	System logs the error and displays a message: "Failed to delete alert. Please try again later."
2	Patron tries to delete an alert but loses connection mid-process.	System retries the deletion or prompts the patron to reconnect and retry.

Business Rules:	**Access Control**: Only VIP-2 or higher users can delete alerts. **Data Validation**: Ensure the alert ID belongs to the authenticated user. **Audit Log**: Log all deletion requests, including the user ID, alert ID, timestamp, and status (success/failure). **Immediate Effect**: The deletion must immediately stop the system from monitoring the condition associated with the alert.

Use Case Specification

Use Case No.:	UC-15		
Use Case Name:	Set Technical Indicator		
Created By:	Bui The Ky Cuong		
Date:	19/12/2024	Priority	Medium
Actors:	Patron VIP-3		
Summary:	Patrons with VIP-3 status can set up advanced technical indicators for specific cryptocurrencies in the **Alert Management** section. Indicators include tools like Moving Average (MA), Exponential Moving Average (EMA), Bollinger Bands, and custom indicators. These settings will trigger notifications when indicator-specific conditions are met.		
Trigger:	The patron navigates to the **Alert Management** section and selects the option to configure a technical indicator.		
Preconditions:	The patron must be logged in and have VIP-3 status. The patron must have an active subscription that supports advanced indicators. The system must support technical indicator calculations and monitoring. The cryptocurrency symbol and desired parameters must be valid and available.		
Post–conditions:	The specified technical indicator is saved to the user's account. The system begins monitoring the indicator based on the user-defined parameters. The configuration is logged in the system for auditing purposes.		

Main Success Scenario/Main Flow/Normal Flow/Main Path:

Step.	Actor Action	System Response
1	Patron logs in and navigates to the **Alert Management** section.	System displays a list of configured alerts and an option to configure indicators.
2	Patron selects the option to **Set Technical Indicator**.	System displays a form for entering the indicator parameters.
3	Patron fills in the form, including the symbol, indicator type, and parameters (e.g., period for MA).	System validates the input parameters and saves the configuration.

4	System begins monitoring the specified indicator and conditions.	Patron receives a confirmation message that the technical indicator has been set successfully.

Alternative Flows:

2a. Invalid VIP Level:

Step	Actor Action	System Response
1	A user without VIP-3 status attempts to set a technical indicator.	System denies access and displays an error message: "This feature is available for VIP-3 users only."

3a. Invalid Parameters:

Step	Actor Action	System Response
1	Patron enters invalid parameters for the technical indicator (e.g., non-numeric values).	System displays an error message: "Invalid input. Please check the indicator parameters."

3b. Unsupported Cryptocurrency Symbol

Step	Actor Action	System Response
1	Patron selects a cryptocurrency that is not supported for technical indicators.	System displays an error message: "This symbol is not supported for technical indicators."

Exceptions:

No.	Actor Action	System Response
1	System fails to save the configuration due to a database error.	System displays a message: "Unable to save the indicator. Please try again later."
2	System encounters an error while monitoring the indicator.	System logs the error and notifies the patron to review their configuration.

Business Rules:	**Access Control**: Only VIP-3 users can set technical indicators. **Data Validation**: Ensure the input parameters for the indicator are valid. **Symbol Support**: The system should only allow symbols that support the selected indicator. **Immediate Monitoring**: The system should begin monitoring the indicator as soon as the configuration is saved. **Audit Log**: Log all configuration changes for technical indicators, including user ID, timestamp, and parameters.

Technical Indicators Supported

1. Moving Average (MA):
   Parameter: Period (e.g., 14 days).
   Trigger Condition: Price crosses above or below the MA.
2. Exponential Moving Average (EMA):
   Parameter: Period (e.g., 14 days).
   Trigger Condition: Price crosses above or below the EMA.
3. Bollinger Bands:
   Parameters: Period (e.g., 20 days), Deviation (e.g., 2).
   Trigger Condition: Price moves above or below the bands.
4. Custom Indicators:
   Parameter: User-defined script or plugin.

Trigger Condition: Based on user-defined logic.

Use Case Specification

Use Case No.:	UC-16		
Use Case Name:	Get All Users		
Created By:	Bui The Ky Cuong		
Date:	19/12/2024	Priority	Medium
Actors:	Admin		
Summary:	The admin can view a list of all users, including details such as user ID, email, registration date, and VIP level. This feature is available only on the Admin Dashboard, which is locally hosted.		
Trigger:	The admin accesses the **Admin Dashboard** on the local host and selects the option to view all users.		
Preconditions:	The admin must be authenticated and logged in on the local Admin Dashboard. The Admin Dashboard must be accessible only through a local host (e.g., http://localhost:8080/admin). The system must have access to the user database to retrieve the required data.		
Post–conditions:	The admin is presented with a list of all registered users in a tabular format. The data is paginated for better usability. The action is logged for auditing purposes.		

Main Success Scenario/Main Flow/Normal Flow/Main Path:

Step.	Actor Action	System Response
1	Admin logs into the local Admin Dashboard.	System validates the admin's credentials and displays the dashboard interface.
2	Admin selects the option to view all users.	System retrieves user data from the database.
3	Admin wait for retrieved data.	System displays the user data in a table, including: User ID, email, registration date, VIP level, and status (active/inactive).
4	Admin navigates through the paginated table to review all users.	System supports pagination and filtering for large datasets.

Alternative Flows:

1a. Unauthorized Access to Admin Dashboard:

Step	Actor Action	System Response
1	Non-admin user attempts to access the Admin Dashboard.	System denies access and displays a 403 Forbidden page.

3a. No Users in the Database:

Step	Actor Action	System Response
1	Admin selects the option to view all users, but the database is empty.	System displays a message: "No users found."

3b. Database Retrieval Error:

Step	Actor Action	System Response
1	Admin selects the option to view all users	System displays an error message: "Unable to retrieve user data. Try again later."

Exceptions:

No.	Actor Action	System Response
1	Database connection fails.	System logs the error and displays: "Failed to connect to the database."
2	Admin loses connection to the local server.	System prompts the admin to reconnect and refresh the page.
3	Unauthorized API call.	System logs the attempt and responds with 403 Forbidden.

Business Rules:	**Access Restriction**: The Admin Dashboard can only be accessed via a local host environment. Only authenticated admins can access the "Get All Users" feature. **Data Privacy**: Ensure sensitive user information (e.g., passwords) is excluded from the response. **Audit Logging**: Log all requests to retrieve user data, including admin ID and timestamp. **Pagination and Filtering**: For large datasets, implement pagination to optimize performance. Provide search and filter options (e.g., by email, VIP level, or registration date).

Use Case Specification

Use Case No.:	UC-17		
Use Case Name:	Get Payment History		
Created By:	Bui The Ky Cuong		
Date:	19/12/2024	Priority	Medium
Actors:	Admin		
Summary:	Admins can view a comprehensive list of all payment transactions made by users. This feature is accessible exclusively through the locally hosted Admin Dashboard and includes details such as transaction ID, user ID, payment amount, date, and status.		
Trigger:	The admin accesses the **Admin Dashboard** on the local host and selects the option to view the payment history.		
Preconditions:	The admin must be authenticated and logged in on the local Admin Dashboard. The Admin Dashboard must be accessible only through a local host (e.g., http://localhost:8080/admin). The system must have access to the payment history database to retrieve the required data.		
Post–conditions:	The admin is presented with a paginated list of payment transactions. The system supports filtering and sorting options for easier review. The action is logged for auditing purposes.		

Main Success Scenario/Main Flow/Normal Flow/Main Path:

Step.	Actor Action	System Response
1	Admin logs in and navigates to the **Admin Dashboard**.	System validates the admin's credentials and displays the dashboard interface.
2	Admin selects the option to view payment history.	System retrieves payment data from the database.
3	Admin waits for retrieved data.	System displays the payment history in a table, including: Transaction ID, user ID, payment amount, payment method, date, and status.
4	Admin navigates through the paginated table to review the payment history.	System supports pagination, sorting, and filtering for large datasets.

Alternative Flows:

1a. Unauthorized Access to Admin Dashboard:

Step	Actor Action	System Response
1	Non-admin user attempts to access the Admin Dashboard.	System denies access and displays a 403 Forbidden page.

3a. No Payment History Found:

Step	Actor Action	System Response
1	Admin selects the option to view payment history, but no transactions exist in the database.	System displays a message: "No payment history found."

3b. Database Retrieval Error:

Step	Actor Action	System Response
1	Admin selects the option to view payment history.	System displays an error message: "Unable to retrieve payment history. Try again later."

Exceptions:

No.	Actor Action	System Response
1	Database connection fails.	System logs the error and displays: "Failed to connect to the database."
2	Admin loses connection to the local server.	System prompts the admin to reconnect and refresh the page.
3	Unauthorized API call.	System logs the attempt and responds with 403 Forbidden.

Business Rules:	**Access Restriction**: The Admin Dashboard can only be accessed via a local host environment. Only authenticated admins can access the "Get Payment History" feature. **Data Privacy**: Ensure sensitive payment information (e.g., card details) is excluded from the response. **Audit Logging**: Log all requests to retrieve payment history, including admin ID and timestamp.

	Pagination and Filtering: For large datasets, implement pagination and allow filtering by user ID, date, and status.

Use Case Specification

Use Case No.:	UC-18		
Use Case Name:	Delete An User		
Created By:	Bui The Ky Cuong		
Date:	19/12/2024	Priority	Medium
Actors:	Admin		
Summary:	Admins can delete a user from the system. This function is accessible only through the locally hosted Admin Dashboard and is available after the admin has retrieved the list of all users using the **Get All Users** use case.		
Trigger:	The admin navigates to the **Admin Dashboard**, retrieves the list of users, and selects a user to delete.		
Preconditions:	The admin must be authenticated and logged in on the local Admin Dashboard. The Admin Dashboard must be accessible only through a local host (e.g., http://localhost:8080/admin). The **Get All Users** use case must have been activated to display the list of users. The user to be deleted must exist in the system.		
Post–conditions:	The selected user is deleted from the system's database. All associated data (e.g., alerts, payment history) is either deleted or marked as inactive, depending on the system rules.		

Main Success Scenario/Main Flow/Normal Flow/Main Path:

Step.	Actor Action	System Response
1	Admin logs in and navigates to the **Admin Dashboard**.	System validates the admin's credentials and displays the dashboard interface.
2	Admin retrieves the list of all users using the **Get All Users** use case.	System displays the list of users in a table.
3	Admin selects a user to delete and clicks the "Delete" button.	System displays a confirmation prompt asking for admin approval.
4	Admin confirms the deletion.	System deletes the selected user and displays a success message.

Alternative Flows:

3a. Admin Cancels Deletion:

Step	Actor Action	System Response
1	Admin selects a user to delete but cancels the confirmation prompt.	System retains the user and returns to the user list without changes.

3b. Selected User Does Not Exist:

Step	Actor Action	System Response

1	Admin selects a user to delete, but the user has already been deleted by another process.	System displays an error message: "This user no longer exists."

Exceptions:

No.	Actor Action	System Response
1	Database connection fails during deletion.	System logs the error and displays: "Failed to delete user. Please try again later."
2	Admin loses connection to the local server.	System prompts the admin to reconnect and refresh the page.
3	Unauthorized API call.	System logs the attempt and responds with 403 Forbidden.

Business Rules:	**Access Restriction**: The Admin Dashboard can only be accessed via a local host environment. Only authenticated admins can delete users. **Data Deletion**: Associated data (e.g., alerts, payment history) must either be deleted or flagged as inactive, based on system requirements.

Use Case Specification

Use Case No.:	UC-19		
Use Case Name:	Delete All Users		
Created By:	Bui The Ky Cuong		
Date:	19/12/2024	Priority	Medium
Actors:	Admin		
Summary:	Admins can delete all users from the system at once. This function is accessible only through the locally hosted Admin Dashboard and is available after the admin has retrieved the list of all users using the **Get All Users** use case.		
Trigger:	The admin navigates to the **Admin Dashboard**, retrieves the list of users, and selects the option to delete all users.		
Preconditions:	The admin must be authenticated and logged in on the local Admin Dashboard. The Admin Dashboard must be accessible only through a local host (e.g., http://localhost:8080/admin). The **Get All Users** use case must have been activated to display the list of users. The system must confirm admin approval before proceeding with the deletion.		
Post–conditions:	All user records are deleted from the system's database. All associated data (e.g., alerts, payment history) is either deleted or marked as inactive, depending on the system rules.		

Main Success Scenario/Main Flow/Normal Flow/Main Path:

Step.	Actor Action	System Response
1	Admin logs in and navigates to the **Admin Dashboard**.	System validates the admin's credentials and displays the dashboard interface.

2	Admin retrieves the list of all users using the **Get All Users** use case.	System displays the list of users in a table.
3	Admin selects the "Delete All Users" option.	System displays a confirmation prompt asking for admin approval.
4	Admin confirms the deletion.	System deletes all users and displays a success message.

Alternative Flows:

3a. Admin Cancels Deletion:

Step	Actor Action	System Response
1	Admin selects the "Delete All Users" option but cancels the confirmation prompt.	System retains all users and returns to the user list without changes.

3b. No Users to Delete:

Step	Actor Action	System Response
1	Admin selects the "Delete All Users" option, but the database is already empty.	System displays a message: "No users found to delete."

Exceptions:

No.	Actor Action	System Response
1	Database connection fails during deletion.	System logs the error and displays: "Failed to delete users. Please try again later."
2	Admin loses connection to the local server.	System prompts the admin to reconnect and refresh the page.
3	Unauthorized API call.	System logs the attempt and responds with 403 Forbidden.

Business Rules:	**Access Restriction**: The Admin Dashboard can only be accessed via a local host environment. Only authenticated admins can delete all users. **Data Deletion**: Associated data (e.g., alerts, payment history) must either be deleted or flagged as inactive, based on system requirements. **Confirmation Prompt**: Deleting all users requires explicit confirmation from the admin to prevent accidental deletions.

The following use cases represent automated actions performed by the system as a "System Handler" actor. These processes are triggered automatically, based on pre-configured settings or periodic updates, and do not require direct user interaction.

UC-20: Send Alert

Actor: System Handler
Description:
The system automatically monitors the configured conditions for user-defined alerts. When a condition is met (e.g., spot price exceeds a threshold, funding rate changes, etc.), the system triggers an alert and sends notifications via the user's preferred method (e.g., email, Telegram, or web interface).

Key Functions:

- Monitor user-configured alert conditions in real-time.
- Retrieve relevant data to validate the condition.
- Send notifications to users when conditions are met.
- Log the alert for auditing and user review.

UC-21: Retrieve S&P Price

Actor: System Handler
Description:
The system periodically fetches real-time **Spot & Future Prices (S&P)** for all tracked cryptocurrency pairs from external APIs such as Binance. The data is stored in the system database for user queries, alerts, and analytics.

Key Functions:

- Execute API calls to external providers (e.g., Binance).
- Parse and validate the response data.
- Update the database with the latest S&P prices.
- Handle API errors and retry failed requests.

UC-22: Retrieve Kline Price

Actor: System Handler
Description:
The system fetches historical candlestick data (Kline) for selected cryptocurrency pairs. This data is used for technical analysis, generating charts, and validating user-defined conditions for advanced indicators.

Key Functions:

- Execute API calls to fetch Kline data for specified time intervals.
- Parse, validate, and store the response data in the database.
- Ensure data consistency for charting and analysis tools.

UC-23: Retrieve Funding Rate

Actor: System Handler
Description:
The system retrieves real-time **Funding Rate** and associated metadata (e.g., countdown to next funding rate update) from external APIs. The data is stored and used for user queries, alerts, and analytics.

Key Functions:

- Periodically call external APIs to fetch the current funding rate.
- Parse and validate the data before updating the database.
- Monitor and log the countdown for the next funding rate update.
- Handle API errors and retry failed requests.

III. Non-functional Requirements

1 Performance Requirements
- Real-time data updates should occur with an interval of approximately 1 second for Spot and Future prices.
- MarketCap and Trading Volume data updates occur every 15 minutes.

2 Security Requirements
- API tokens are required for all system access. Tokens are role-based with distinct privileges.
- Admin access is protected via two-factor authentication (2FA).

- All input data are validated before saving to database.

- Each role only access to a group of functions.

3 Communication Interfaces
- Authencatio Service: The System can be integrated with Firebase Authentication framework.

- The system can call API through data transferred by HTTP.

4 Usability
- All text, label and message should be uniformly written in Vietnamese.

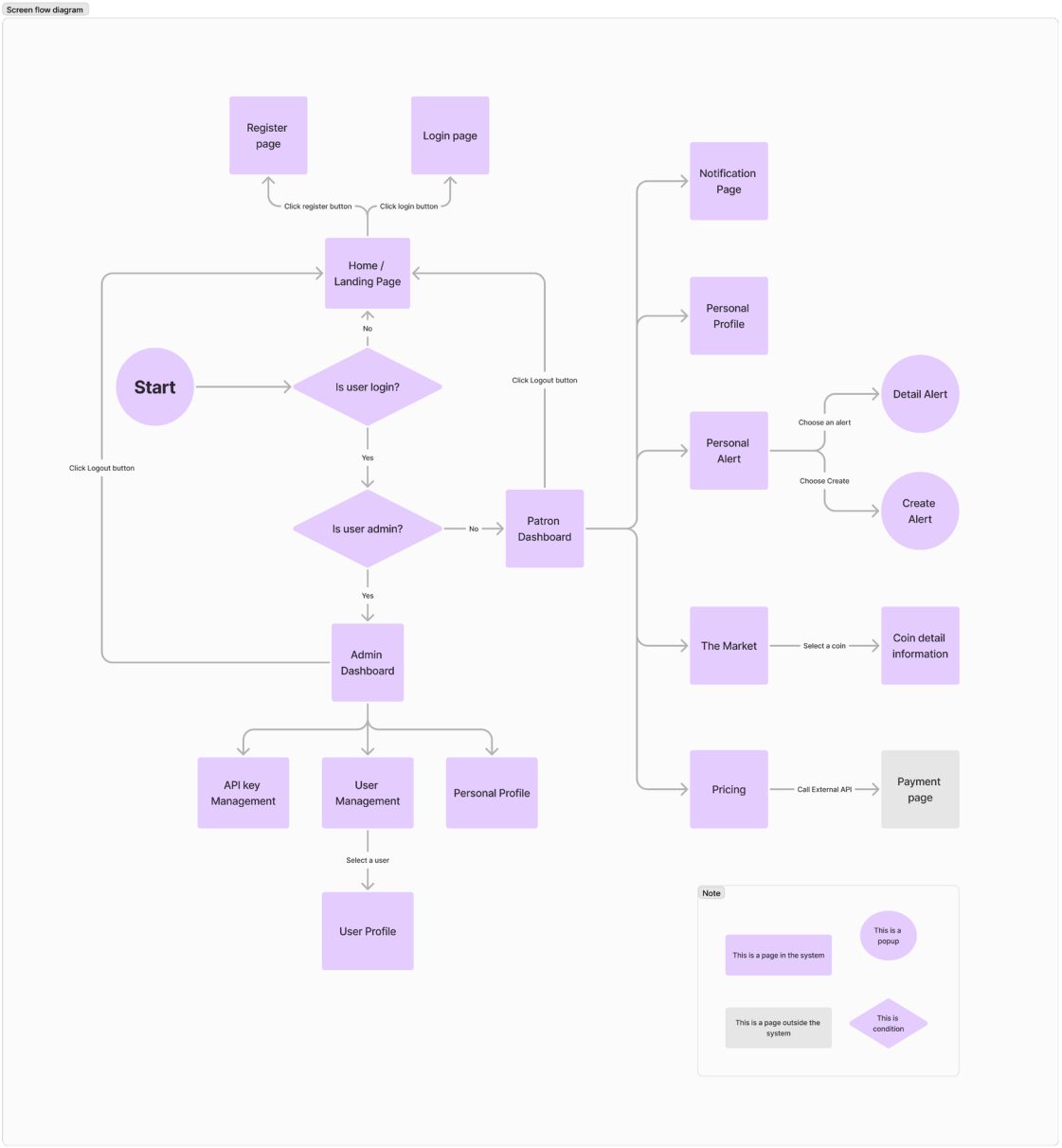
- The web application should be friendly and easy for users to use without training.

- The web application for admin should require no more than 2 days of training of use.

IV. System Modelling

System Functional Overview

1 Screen Flow Diagram



2 Figma Design