

# Software Requirements Specification (SRS)

---

For “Coin Market System”.

Version 1.2

Prepared by Bui The Ky Cuong

## Contents

Definition and Acronyms.....	3
I. Introduction .....	3
1 Purpose .....	3
2 Document Conventions .....	3
3 Project Scope.....	3
4 References.....	3
5 Product Perspective.....	3
6 Product Features .....	4
7 User Classes and Characteristics .....	4
8 Operating Environment.....	4
II. Overall Description.....	4
1 Guest.....	4
2 Registered User (Patron) .....	4
3 Admin.....	5
4 User Requirements.....	6
5 Use Cases List.....	8
5.1 Patron .....	8
5.2 Admin and System Handler .....	8
6 Use Case Specification.....	9
III. Functional Requirements .....	12
1 Screen Flow .....	33
1.1 Coin Market System - Patron Web Application.....	33
1.2 Coin Market System - Admin Web Application .....	33

2 Screen Description.....	33
2.1 Patron Web Application.....	33
2.2 Admin Web Application .....	33
3 Non-Screen Functions.....	33
3.1 Entity Relationship Diagram.....	33
3.1.x: #of Entity and description.....	33
IV. Nonfunctional Requirements.....	33
1 Performance Requirements.....	33
2 Security Requirements .....	33
3 Communication Interfaces .....	34
4 Usability .....	34

# 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. Overall Description

## User Requirements Overview

---

### 1 Guest

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

- Sign Up (Register)

### 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				
Login				
Look up for Spot price				

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

Note:



Capable Of.



Incapable Of.

### 3 Admin

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

- Manage Users' Account.
- Authenticate User using API-TOKEN.
- Update permission for user.
- Delete (Ban) a user or move to blacklist.
- View statistics on VIP-purchase activities.

# 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 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 ( $\geq$ or $\leq$ ).
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.

## 4 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.

# Use Case Diagram

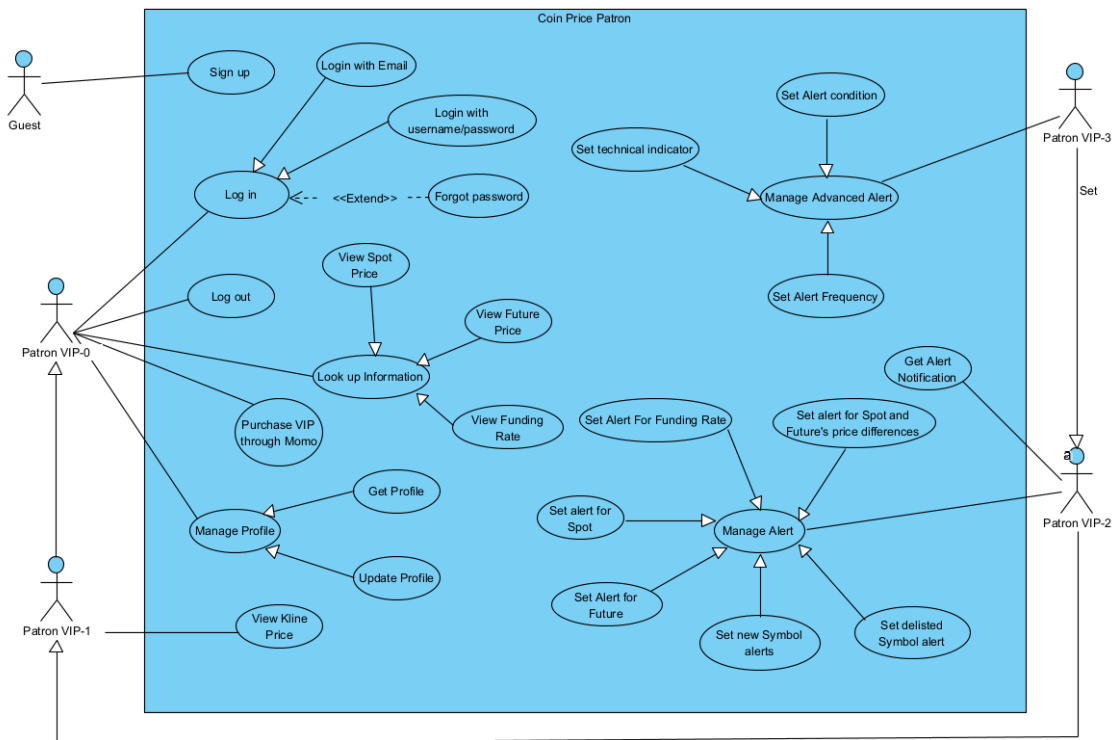


Figure 1: Patron Usecase Diagram

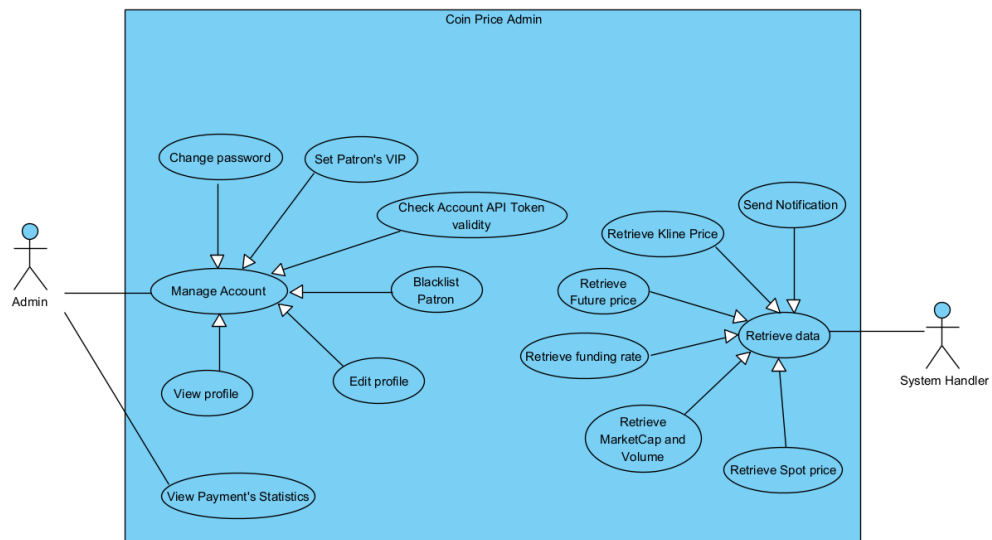


Figure 2: Admin & System Handler Usecase Diagram

## 5 Use Cases List

## 5.1 Patron

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
Set Alert For Funding Rate	UC-08	Patron VIP2, VIP3
Set Alert For Spot Price	UC-09	Patron VIP2, VIP3
Set Alert For Future Price	UC-10	Patron VIP2, VIP3
Set New or Delisted Symbol Alert	UC-11	Patron VIP2, VIP3
Set Alert for Spot & Future price differences	UC-12	Patron VIP2, VIP3

## 5.2 Admin and System Handler

[illegible]



## 6 Use Case Specification

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 Go 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 (Go):

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	Frontend sends the Google ID token to the backend Go server via HTTP POST to the /auth/google-login endpoint.		
6	Backend Go 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.
<b>Business Rules:</b> <ul style="list-style-type: none"><li>- 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.</li><li>- JWT session tokens created by the backend should include the user's role (VIP-0, VIP-1, VIP-2, VIP-3) for access control.</li><li>- The JWT token is stored in an HttpOnly cookie to prevent XSS attacks.</li><li>- Login sessions should have a predefined expiration time (e.g., 24 hours), after which the user will be required to log in again.</li><li>- Google login should require user consent to access basic profile information and email.</li></ul>		
<b>Implementation Notes:</b>		
<b>Frontend (Next.js):</b> <ul style="list-style-type: none"><li>• Use Google OAuth 2.0 client library for handling the login flow.</li><li>• On successful Google login, obtain the ID token and send it to the backend.</li><li>• Store the JWT session token in an HttpOnly cookie once received from the backend.</li></ul>		
<b>Backend (Go):</b> <ul style="list-style-type: none"><li>• Create an endpoint /auth/google-login to handle the POST request with the Google ID token.</li><li>• Verify the Google ID token using the Google OAuth 2.0 token verification API (<a href="https://oauth2.googleapis.com/tokeninfo">https://oauth2.googleapis.com/tokeninfo</a>).</li><li>• Generate a JWT token upon successful verification, with the user's role, and send it back to the frontend.</li></ul>		

## 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): <ul style="list-style-type: none"><li>The "Logout" button should be implemented as part of the user profile or dropdown menu under the user icon.</li><li>Clicking "Logout" triggers an API call to the backend to invalidate the session and delete any tokens stored in the browser (cookies)</li><li>Optionally, a confirmation modal can be added to prevent accidental logout.</li><li>After logout, the user should be redirected to the login or home page.</li></ul>			
Backend: <ul style="list-style-type: none"><li>Implement a /logout endpoint that handles session invalidation.</li><li>Clear any session cookies or authentication tokens stored on the server.</li></ul>			

- 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.

<b>Business Rules:</b>	<ul style="list-style-type: none"><li>- <b>Real-time Updates:</b> 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.</li><li>- <b>Chart View:</b> 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.</li><li>- <b>Data Consistency:</b> The system must ensure that all displayed data is accurate and synchronized with external data sources.</li><li>- <b>User Permissions:</b> 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.</li><li>- <b>Caching and Performance:</b> Market data should be cached to optimize performance and reduce redundant calls to external APIs.</li></ul>
------------------------	--

**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.

<b>Business Rules:</b>	<ul style="list-style-type: none"> <li>- 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.</li> <li>- Subscription Update: After successful payment, the system must immediately update the user's VIP status and grant access to VIP-only features.</li> <li>- Pricing Page Display: The pricing page should clearly show all available VIP plans, including duration and cost. Users can also view VIP benefits.</li> <li>- 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.</li> </ul>
------------------------	--

#### 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).
5		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.
6	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.
<b>Business Rules:</b> <ul style="list-style-type: none"><li>- <b>Profile Uniqueness:</b> The system must ensure that certain profile details (e.g., email address) remain unique across all users.</li><li>- <b>Input Validation:</b> The system must validate all profile fields, ensuring that inputs meet required formats (e.g., email format, password criteria).</li><li>- <b>Security:</b> Sensitive profile fields (e.g., password) must be updated securely. Any password changes should follow best security practices (e.g., hashing, encryption).</li><li>- <b>Notification (Optional):</b> The system could notify users via email when a significant profile change (e.g., email or password) is made.</li></ul>		
<b>Implementation Notes:</b> <b>Frontend (Next.js):</b> <ul style="list-style-type: none"><li>• The "User Profile" page should include editable fields for each profile attribute (e.g., name, email, password).</li><li>• Use form validation for fields such as email (regex for proper formatting) and password (strength requirements).</li><li>• Provide user feedback for successful updates or errors during the update process.</li><li>• Implement responsive design to ensure the profile page works well across different devices.</li><li>• After successful updates, the profile page should reflect the new data without requiring a page reload.</li></ul> <b>Backend:</b> <ul style="list-style-type: none"><li>• The /user/profile API endpoint should support both <b>GET</b> (to fetch current user data) and <b>PUT</b> (to update user data) operations.</li><li>• When updating, the system should validate the data (e.g., checking for duplicate emails or weak passwords).</li><li>• Securely update sensitive fields like the password, ensuring it is hashed before storage.</li><li>• Handle potential concurrency issues, especially if the user is trying to update their profile while another process is modifying their data.</li><li>• Use transaction management to ensure consistency when updating multiple fields.</li><li>• Log any significant profile changes (e.g., email or password updates) for audit and security purposes.</li></ul>		

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	The system confirms the user has the required access level (VIP-1 or higher).	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.

<b>Business Rules:</b>	<ul style="list-style-type: none"><li>- <b>VIP Access Control:</b> 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.</li><li>- <b>Real-Time Data:</b> 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).</li><li>- <b>Interaction:</b> 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.</li></ul>
<b>Implementation Notes:</b> <b>Frontend (Next.js):</b> <ul style="list-style-type: none"><li>• The frontend should have a "Kline Price" component or page that renders the price chart using a charting library such as <b>Chart.js</b> or <b>Highcharts</b>.</li><li>• Implement user interface components to allow interaction, such as time range selection (1m, 5m, 1h, 1d) and zooming options.</li><li>• Handle cases where the user does not have the required VIP access by displaying a pop-up message and redirecting to the pricing page.</li></ul> <b>Backend:</b> <ul style="list-style-type: none"><li>• The backend should validate the user's VIP level when accessing the Kline Price feature.</li><li>• 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).</li><li>• Implement proper caching mechanisms to avoid repeatedly fetching the same data from external APIs.</li><li>• Log every access attempt to the Kline Price chart, including user ID and timestamp, for future audits and usage statistics.</li></ul> <b>Security Considerations:</b> <ul style="list-style-type: none"><li>• 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.</li><li>• All API calls to external financial services should be securely handled (use HTTPS and proper authentication mechanisms).</li></ul> <b>Performance Considerations:</b> <ul style="list-style-type: none"><li>• Use caching where appropriate to avoid unnecessary load on external APIs and to improve page load times when rendering the Kline chart.</li><li>• Ensure the chart is dynamically updated without requiring full page reloads for better user experience.</li></ul>	

Use Case Specification			
Use Case No.:	UC-08		
Use Case Name:	Set 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 "Personal Alert 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 "Personal Alert" 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 "Personal Alert" page must be accessible, and the user should have previously set up the ability to receive notifications (email, app notification, etc.).		

<b>Post-conditions:</b>	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.	
<b>Main Success Scenario/Main Flow/Normal Flow/Main Path:</b>		
<b>Step.</b>	<b>Actor Action</b>	<b>System Response</b>
1	Patron logs into their account.	The system verifies the user's credentials and VIP status.
2	Patron navigates to the "Personal Alert" 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.).	
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.
<b>Alternative Flows:</b>		
3a. User Does Not Have VIP-2 or Higher Access:		
<b>Step</b>	<b>Actor Action</b>	<b>System Response</b>
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:		
<b>Step</b>	<b>Actor Action</b>	<b>System Response</b>
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.
<b>Exceptions:</b>		
<b>No.</b>	<b>Actor Action</b>	<b>System Response</b>
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:	<ul style="list-style-type: none"><li>- <b>VIP Access Control:</b> Only users with VIP-2 or higher access can set an alert for the funding rate. Lower-tier users should be prompted to upgrade.</li><li>- <b>Threshold Monitoring:</b> The system must continuously monitor funding rate changes and trigger notifications if the user's set threshold is crossed.</li><li>- <b>Real-Time Data:</b> The system must fetch the most up-to-date funding rate data either from its own database or via an external financial API.</li></ul>	
<p><b>Implementation Notes:</b></p> <p><b>Frontend (Next.js):</b></p> <ul style="list-style-type: none"><li>• The frontend should have a dedicated page or section (e.g., "Personal Alert" page) where users can manage their alert settings. This will include a form where they can configure parameters for the funding rate alert.</li><li>• The form should include input validation (e.g., checking that the threshold values are numeric).</li><li>• The form should also allow the user to choose notification methods (e.g., email, app notification, SMS).</li></ul> <p><b>Backend:</b></p> <ul style="list-style-type: none"><li>• Implement an API endpoint (e.g., /alerts/funding-rate) to handle the creation, modification, and deletion of funding rate alerts.</li><li>• Upon submission of the alert setup form, validate the user's VIP level and the input data.</li><li>• Store the alert details in the database and associate them with the user's account.</li><li>• A background process should continuously monitor the funding rate against the user-defined thresholds and send notifications if necessary.</li></ul> <p><b>Notification System:</b></p> <ul style="list-style-type: none"><li>• Ensure the backend is capable of sending notifications via different channels (email, in-app, SMS) based on the user's preferences.</li><li>• 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.</li></ul> <p><b>Security Considerations:</b></p> <ul style="list-style-type: none"><li>• Validate that only authorized users (VIP-2 or higher) can set and manage funding rate alerts.</li><li>• Ensure that sensitive information (e.g., user email or phone number for notifications) is securely handled and stored.</li></ul> <p><b>Performance Considerations:</b></p> <ul style="list-style-type: none"><li>• Use caching and efficient data retrieval methods to minimize the impact of frequent funding rate checks.</li><li>• Consider using a background job queue to handle alert triggers and notifications, ensuring that these tasks do not block other user operations.</li></ul>		

Use Case Specification			
<b>Use Case No.:</b>	UC-09		
<b>Use Case Name:</b>	Set Alert for Spot Price		
<b>Created By:</b>	Bui The Ky Cuong		
<b>Date:</b>	16/10/2024	<b>Priority</b>	High
<b>Actors:</b>	Patron VIP-2, VIP-3		



<b>Summary:</b>	A patron with VIP-2 or higher can set an alert for spot prices on the "Personal Alert Page." The alert will notify the user when a specified spot price crosses a predefined threshold, either above or below a certain value.
<b>Trigger:</b>	The patron navigates to the "Personal Alert" page and selects the option to set an alert for spot price.
<b>Preconditions:</b>	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 "Personal Alert" page is accessible, and the user should have configured notification preferences (such as email, app notification, or Telegram).
<b>Post-conditions:</b>	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 "Personal Alert" 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).	
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 "Personal Alert" 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:	Set 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 "Personal Alert 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 "Personal Alert" 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 "Personal Alert" 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 "Personal Alert" 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).		
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.

<b>Business Rules:</b>	<ul style="list-style-type: none"> <li>- <b>VIP Access Control:</b> Only users with VIP-2 or higher access can set and manage spot price alerts.</li> <li>- <b>Threshold Monitoring:</b> The system should continually monitor the future price and send notifications when user-defined thresholds are reached.</li> <li>- <b>Real-Time Spot Price Data:</b> 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..</li> </ul>
------------------------	---

<p><b>Frontend (Next.js):</b></p> <ul style="list-style-type: none"> <li>The "Personal Alert" 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.</li> <li>The form should include input validation to ensure that future price thresholds are numeric and within a reasonable range.</li> <li>The user interface should allow users to view, modify, or delete existing future price alerts.</li> </ul> <p><b>Backend (Go):</b></p> <ul style="list-style-type: none"> <li>The backend must provide an API endpoint (e.g., /alerts/future-price) for creating, updating, and deleting future price alerts.</li> <li>The API should validate that the user has VIP-2 or higher access before allowing them to set or update alerts.</li> <li>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.</li> </ul> <p><b>Future Price Monitoring Service:</b></p> <ul style="list-style-type: none"> <li>A background service should monitor the future price in real-time, comparing the current price against user-defined thresholds.</li> <li>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).</li> </ul>
---

Use Case Specification			
Use Case No.:	UC-11		
Use Case Name:	Set 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 "Personal Alert" 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 "Personal Alert" 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 "Personal Alert" 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.

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

#### Implementation Notes:

##### Frontend (Next.js):

- The "Personal Alert" 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:	Set 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 "Personal Alert" 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 "Personal Alert" 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 "Personal Alert" 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).		
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.

<b>Business Rules:</b>	<p><b>VIP Access Control:</b> Only Patron with VIP-2 or higher access can set and manage alerts for spot and future price differences.</p> <p><b>Threshold Monitoring:</b> 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.</p> <p><b>Real-Time Data Access:</b> The system must fetch real-time or near real-time data for both spot and future prices from external financial APIs</p>
------------------------	---

**Implementation Notes:****Frontend (Next.js):**

- The "Personal Alert" 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.

## **III. Functional Requirements**

### **System Functional Overview**

#### **1 Screen Flow**

1.1 Coin Market System - Patron Web Application

1.2 Coin Market System - Admin Web Application

#### **2 Screen Description**

2.1 Patron Web Application

2.2 Admin Web Application

#### **3 Non-Screen Functions**

3.1 Entity Relationship Diagram

3.1.x: #of Entity and description

## **IV. Nonfunctional 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**

- Authentcatio 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. Software Design Description (Updating)**