Group symbol: **C**

Team: **3**

Project title: **<Trading Bot>**

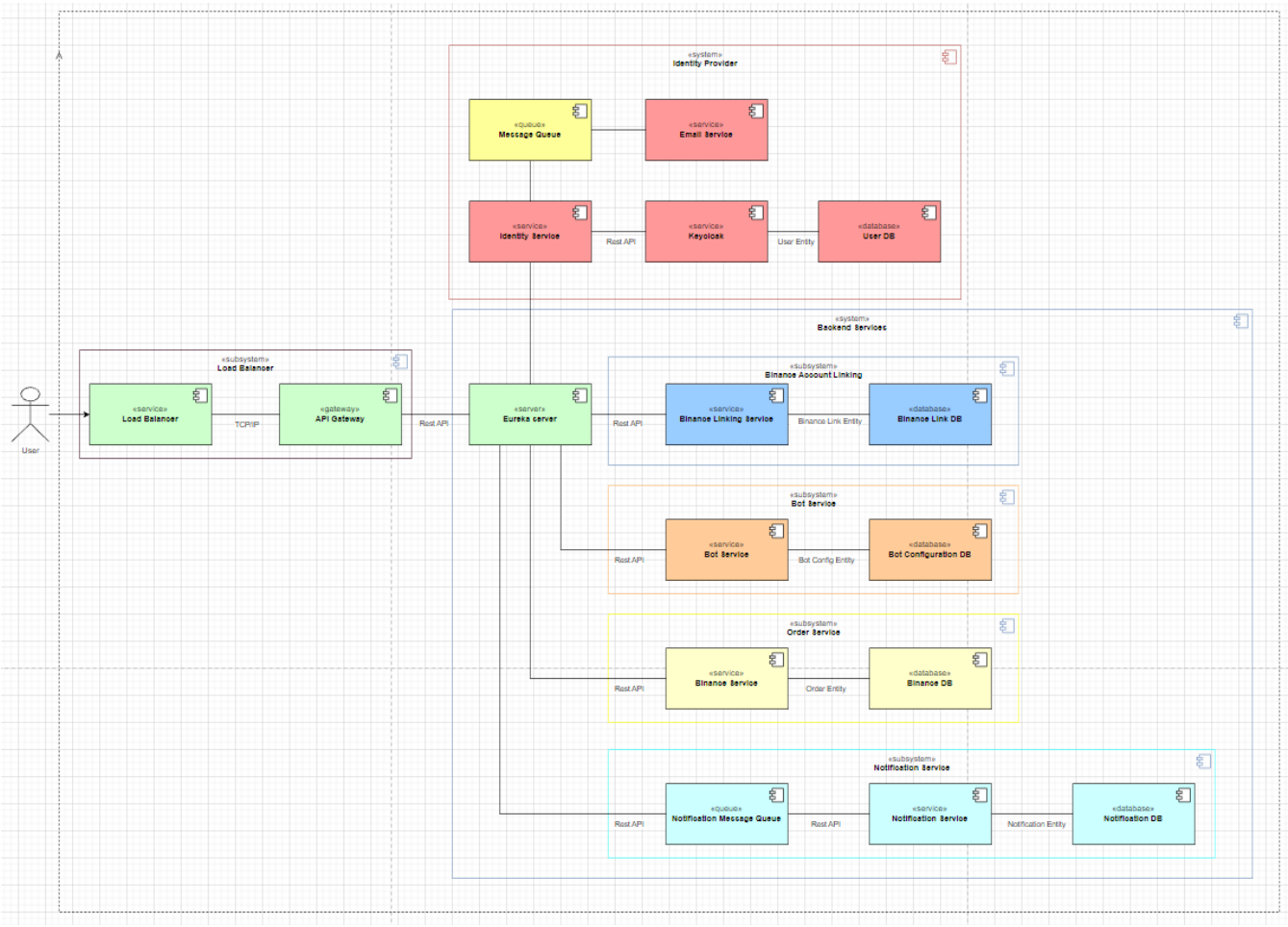
Team members (*filled by PM, Team Leader*):

No	Name	Surname	Student ID	Role
1	Bohdan	Kyryliuk	267855	*PM, Team Leader*
2	Serhii	Ohurtsov	251530	*Team member*
3	Sergiy	Vergun	251203	*Team member*
4	Ilgin	Sogut	282416	*Team member*

3. Design **(F3)**

3.1. Logical Software Architecture

The logical architecture aspect of the system should be present as a Component Diagram expressed in UML 2.5.



Load Balancer (Subsystem):

- Acts as the entry point for user requests, distributing incoming network traffic across multiple servers to ensure no single server bears too much demand.

API Gateway (Subsystem):

- Provides a single entry point for all client requests, directing them to the appropriate microservice and may also handle cross-cutting concerns such as authentication, SSL termination, and rate limiting.

Eureka Server (Subsystem):

- Functions as a service registry that enables service instances to find each other in a dynamic landscape, facilitating load balancing and failover.

Identity Provider (Subsystem):

Comprises services and databases related to identity and access management. Includes:

- Identity Service: Manages user authentication and authorization.
- Keycloak Service: An open-source identity and access management service that handles user federation, and token issuance.
- Email Service: Responsible for sending emails, likely for purposes such as verification, notifications, or password resets.
- User DB: The database that stores user-related information, probably including credentials and roles.

Backend Services (Subsystem):

A collection of core business logic services. Includes:

- Binance Account Linking Service: Manages the linkage between user accounts and their respective Binance exchange accounts.
- Bot Service: Handles the logic for automated trading strategies.
- Binance Service: Manages interactions with Binance API.
- Notification Service: Responsible for sending out notifications to users, possibly related to trade alerts, system updates, or account actions.

Databases:

Each service has its dedicated database, which is a typical pattern in microservices to ensure loose coupling and service autonomy.

- Binance Link DB: Stores information relevant to the linkage between the application's user accounts and Binance accounts.
- Bot Configuration DB: Contains the configuration settings for each user's trading bot(s).
- Binance DB: Holds the data related to binance account (user account data, such as trading history, wallet etc).
- Notification DB: Keeps records of notifications to be dispatched to users.

Message Queues:

Utilized for asynchronous communication between services.

- Notification Message Queue: Specifically for queuing notification messages, ensuring that notification delivery can handle high loads and is resilient to spikes in activity.

3.2. Business Logic Model

3.2.1. Behavioural Model

The Flow Chart

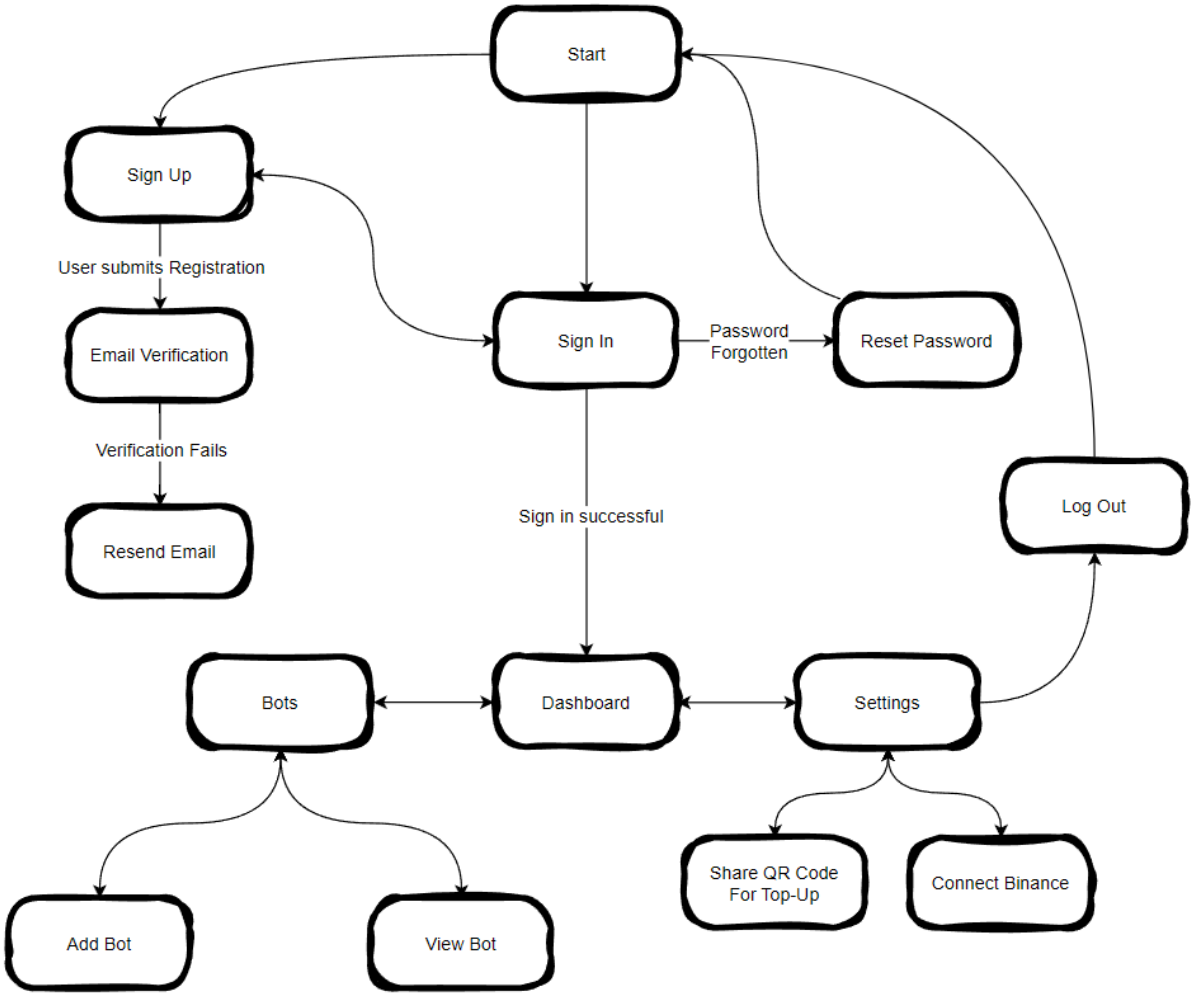


Figure 1: Flow Chart

The flowchart (Figure 1) delineates the essential navigational pathways and decision-making processes within our trading bot application. Beginning with the 'Start' node, it guides the user through various functional milestones such as signing up, verifying email, and handling password resets. The diagram culminates in the access of the application's main features: Bots, Dashboard, and Settings, detailing the interconnectivity of these components. This visual guide aids in apprehending the app's architecture, showcasing how users transition from initial access to engaging with the core functionalities of the platform.

The State Diagram:

The state diagram (Figure 2) encapsulates the various modes of operation within our application, providing a high-level overview of its stateful behavior. Each circle represents a distinct state, such as 'Not Logged In / Not Registered,' 'Idle,' or 'Bot Active.' The transitions between these states, represented by arrows, are triggered by user actions like logging in or engaging in manual trading. This diagram is essential for understanding the life cycle of user interactions and the system's responses, illustrating the sequence and conditions under which the application transitions from one state to another.

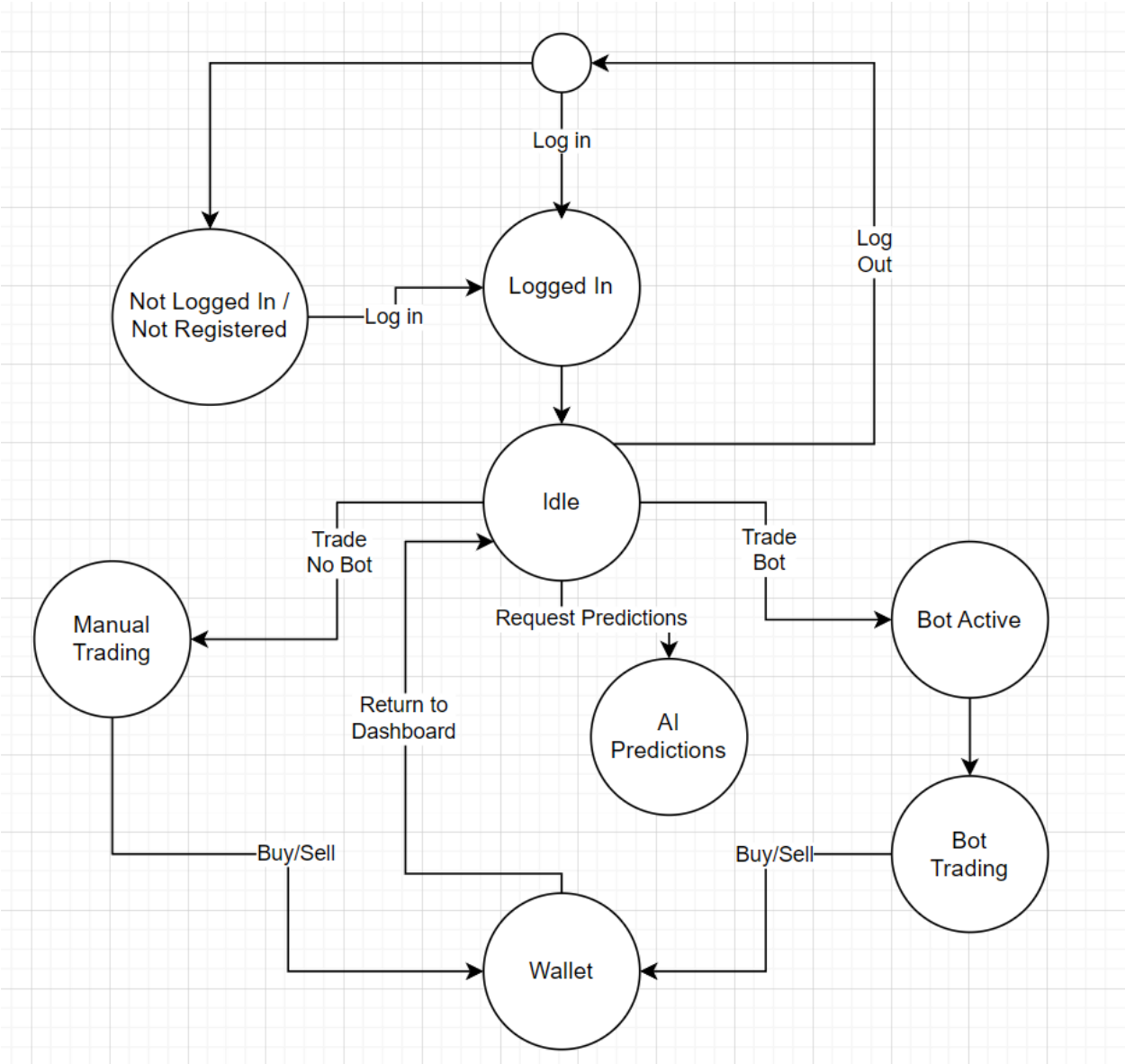


Figure 2: State Diagram

The Sequence Diagram

Our sequence diagram (Figure 3) offers an in-depth look at the interactions between system components during the 'Activate Bot' use case. It traces the chronological exchange of messages starting from the user's command to the system's execution of the bot activation. The user initiates the process via the UI, which communicates with the BotManager to retrieve and confirm the bot settings from the Database before instructing the Bot to commence trading. The diagram exemplifies the sequential logic and the collaborative nature of our system's architecture in response to user-initiated actions.

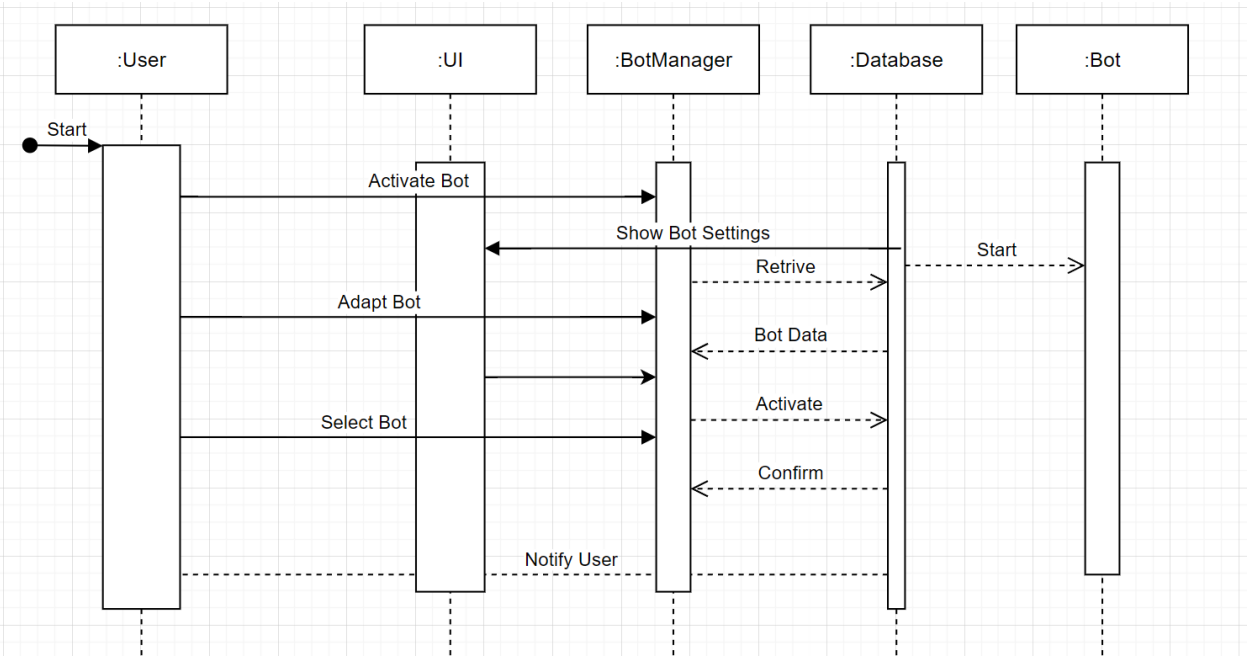


Figure 3: A Sample Sequence Diagram

The Activity Diagram

The activity diagram (Figure 4) provides a step-by-step breakdown of the 'Activate Bot' use case. It starts with the user's decision to activate a bot and branches into different paths based on whether the bot settings are pre-configured. The user can directly start a bot with existing settings or proceed to configure the bot before activation. The flow of activities progresses from decision nodes to action nodes, culminating in the 'Bot Running' state, thereby illustrating the procedural logic behind the user setting up a bot for trading within our application.

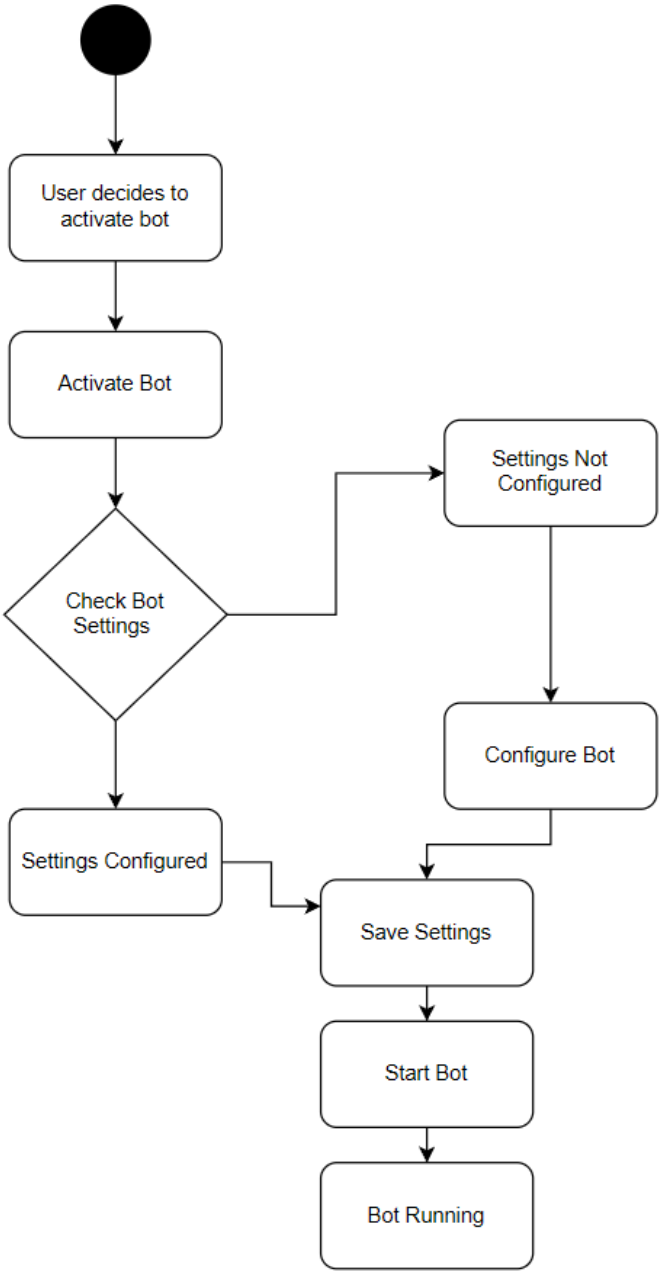


Figure 4: A Sample Activity Diagram

3.2.2. Structural Model

Prepare the model as class diagram expressed in UML 2.5. Use natural language to describe the semantics of models' elements. Besides, use the object diagram to present an instance of the model at the time of system execution.

Class Diagram:

The class diagram (Figure 5) systematically organizes the key structural components of our trading bot application into a coherent schema. It portrays the classes such as 'User', 'Wallet', 'Bot', and 'Order' along with their attributes and methods, establishing a blueprint of our system's architecture. Relationships between classes are clearly indicated, for instance, a 'User' is linked to a single 'Wallet' and potentially multiple 'Bots'. This diagram is instrumental for developers to comprehend the system's design and for stakeholders to visualize the interrelations of its various parts.

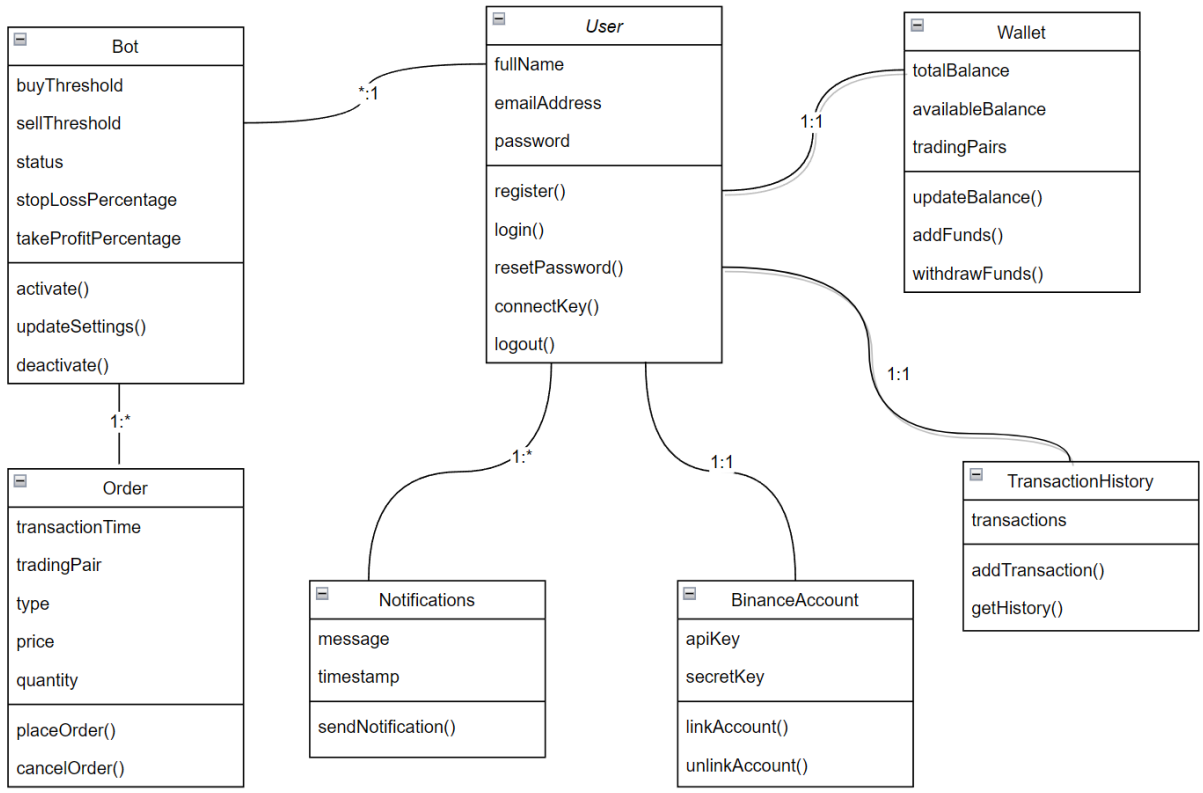


Figure 1: Class Diagram

Object Diagram

The object diagram (Figure 6) provides a snapshot of the system at a specific instance, illustrating the real-time state of various objects within the application. It presents an example scenario where a user, represented by the 'User' object, has an active 'Wallet' and has configured a 'Bot' for trading. The attributes of each object are populated with sample data to showcase their current values, such as the 'Wallet's' balances and the 'Bot's' thresholds. This diagram conveys the tangible state of the system's components during operation, offering an empirical view that complements the theoretical constructs outlined in the class diagram.

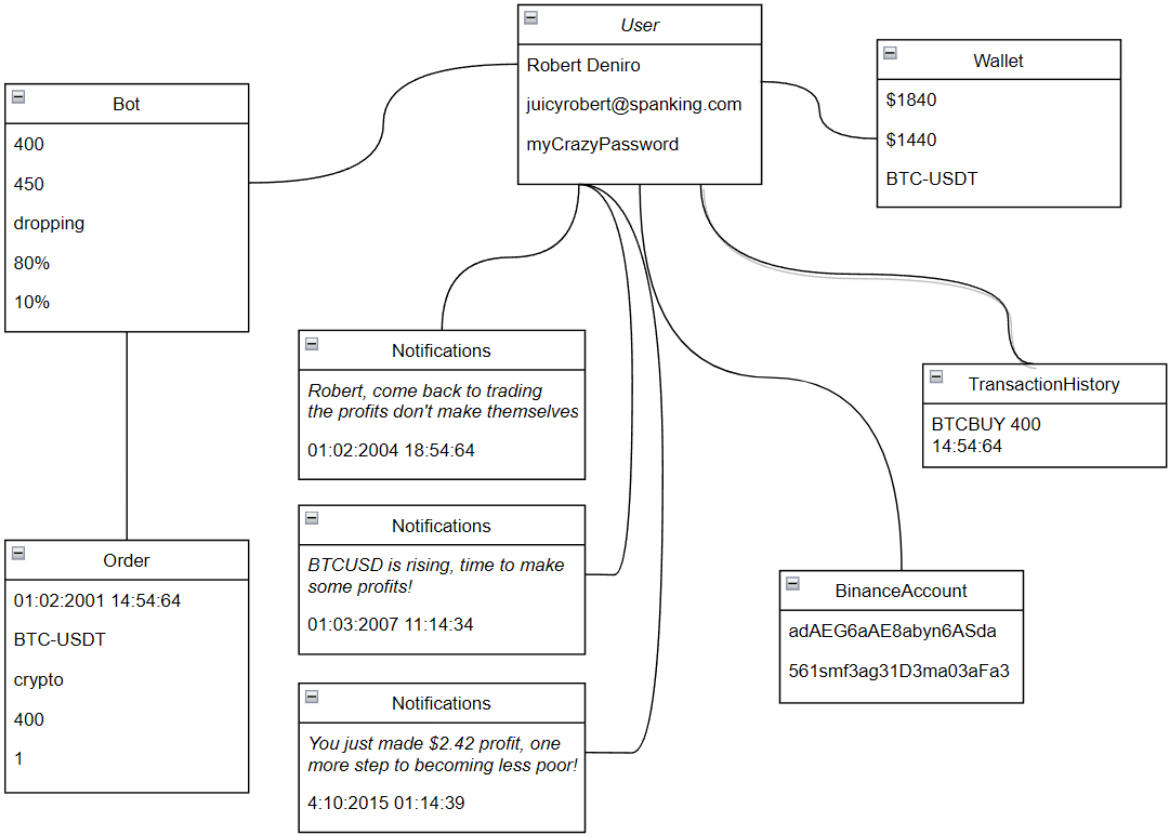
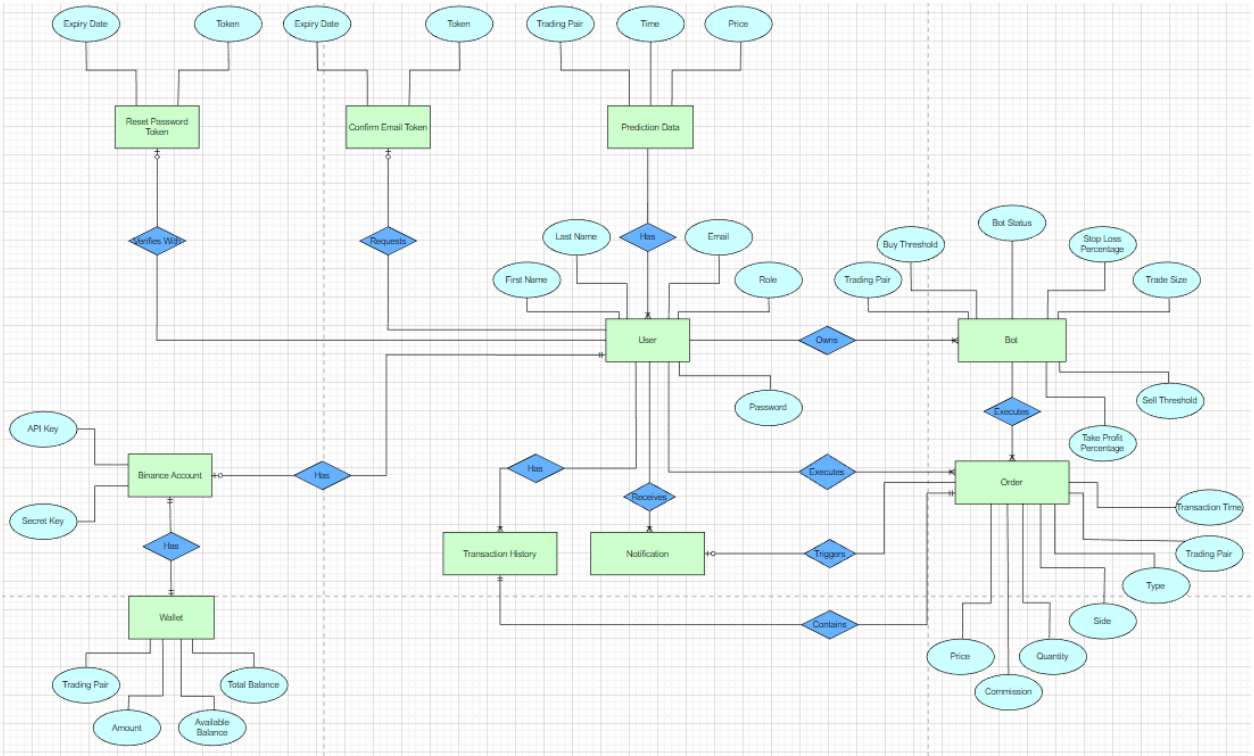


Figure 2: Object Diagram

3.3. Database Model

3.3.1. Conceptual Model

The model contains the definition of entities and relationships among them in terms of the database (persistence). You should present it as a diagram and description of key elements in natural language.



The diagram centers around the **User** entity, which has attributes like **First Name**, **Last Name**, **Email**, **Role**, and **Password**. The **User** can receive **Notifications** and has **Transaction History**.

There is a **Bot** entity that is connected to the User, suggesting that users can configure or own bots on the platform. The Bot has attributes like **Buy Threshold, Sell Threshold**, **Bot Status**, **Stop Loss Percentage**, and **Take Profit Percentage**.

Orders are executed by the Bot, as shown by the Executes relationship. An Order has attributes like **Transaction Time**, **Trading Pair**, **Type**, **Price**, **Quantity**, and **Commission**.

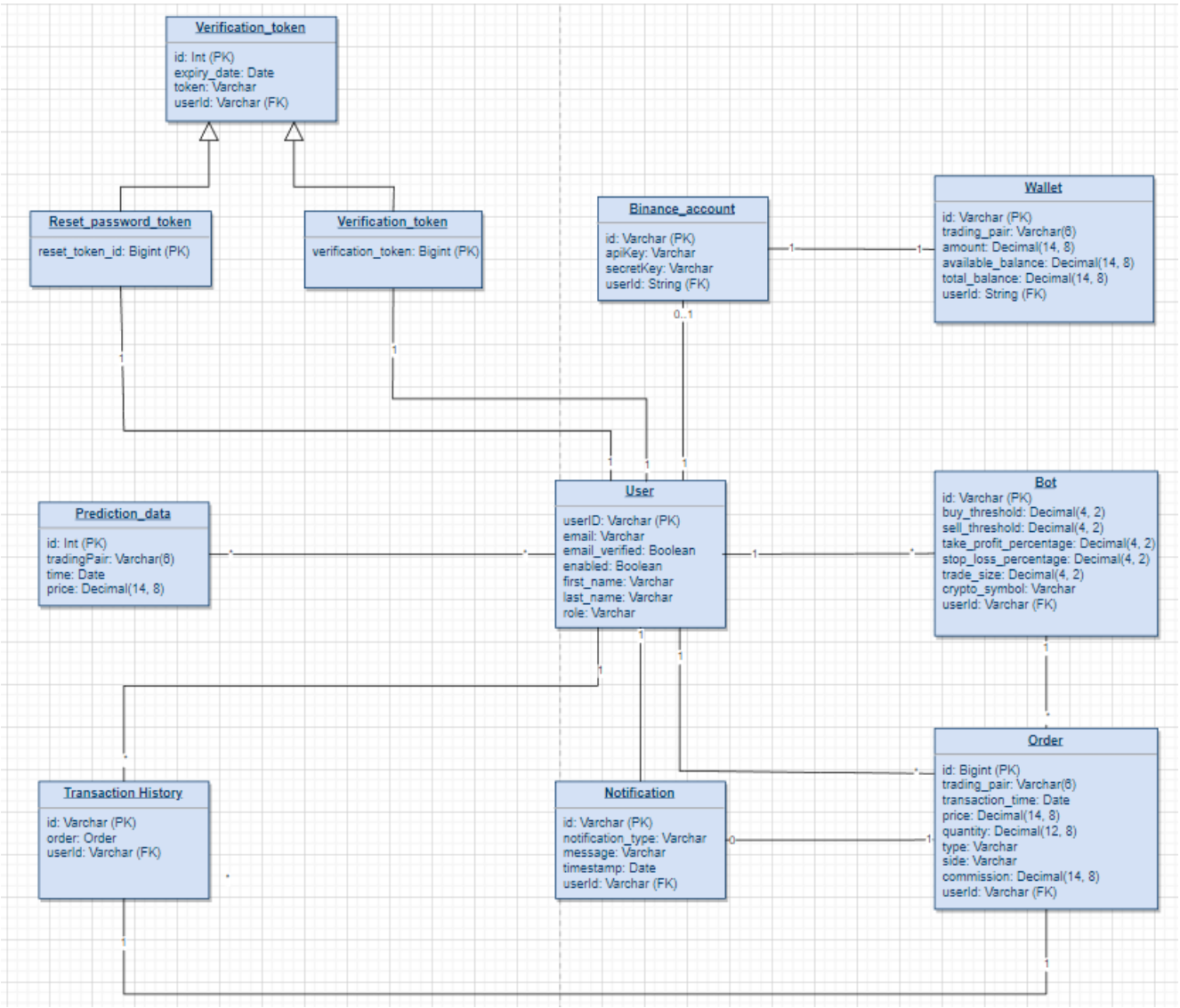
The User has a **Wallet**, which is related to a **Binance Account** through an **API Key** and **Secret Key**. The Wallet contains a **Total Balance** and **Available Balance**, and is associated with **Trading Pairs** and **Amounts**.

There are tokens involved for various purposes, such as a **Reset Password Token** and **Confirm Email Token**, which are tied to **Expiry Date**.

Prediction Data is another entity that includes **Trading Pair**, **Time** and **Price**, which might suggest the platform provides market predictions for trading pairs.

3.3.2. Physical Model

The model presented as a diagram of the database that constitutes the implementation of the conceptual model in the selected database management system.



1. **User Entity**:
    - This is the core entity representing the users of the application.
    - Attributes include **userID**, **email**, **email_verified**, **enabled**, **first_name**, **last_name**, and **role**.
    - The **User** entity has one to zero or one relationships with **Binance_account** and one-to-many relationships with **Bot**, **Notification**, and **Transaction History**, indicating each user

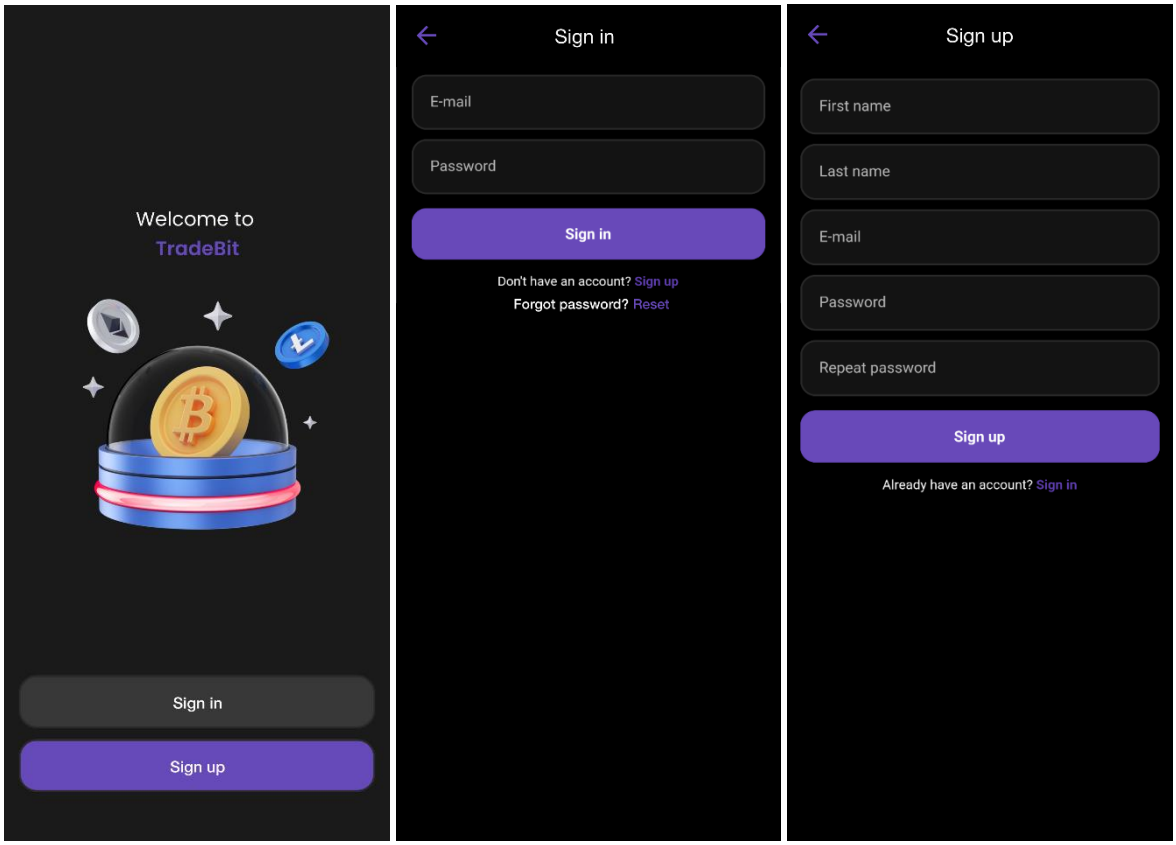
can have multiple bots, notifications, and transactions but only one Binance account and wallet.

2. **Binance_account Entity**:
    - Holds the API key and secret for integrating with a user's Binance account.
    - Has a one-to-one association with **User**, which is indicated by the **userid** foreign key.
3. **Wallet Entity**:
    - Stores balance information for a user's wallet.
    - Has a one-to-many relationship with **User** indicated by the **userid** foreign key.
4. **Bot Entity**:
    - Represents the configuration settings for the user's trading bot.
    - Has a one-to-many relationship with **User**, allowing users to have multiple bots.
5. **Order Entity**:
    - Captures details of trading orders placed by bots or a user.
    - Includes fields such as **trading_pair**, **transaction_time**, **price**, **quantity**, **type**, **side**, and **commission**.
    - Associated with **User** through the **userid** foreign key.
6. **Transaction History Entity**:
    - Logs the transaction history for each order.
    - Includes a reference to the **Order** entity and **userid**, suggesting that each transaction is tied to a specific order and user.
7. **Notification Entity**:
    - Manages notifications sent to users.
    - Has a one-to-many relationship with **User**, as indicated by the **userid** foreign key.
8. **Prediction_data Entity**:
    - Stores prediction data for trading pairs, including the time and price.
    - Lacks a direct association with **User**, suggesting that prediction data may be a shared resource rather than user-specific.
9. **Verification_token and Reset_password_token Entities**:
    - These entities are specialized versions of a conceptual "**Token**" entity
    - These entities manage tokens for password resetting and account verification.
    - Each contains a token value and an expiry date, along with a userid foreign key to link to the respective user.

It is required to include an elaboration on the transformation of the conceptual model to the physical model. Description of transformation should include how elements like, e.g. inheritance and associations or other sophisticated modelling concepts have been handled.

3.4. User Interface Design

In this section, you should present the distribution of user interface components with events and their assignments to the behaviour specifications, which these elements initiate or take part.



1)Landing Page:

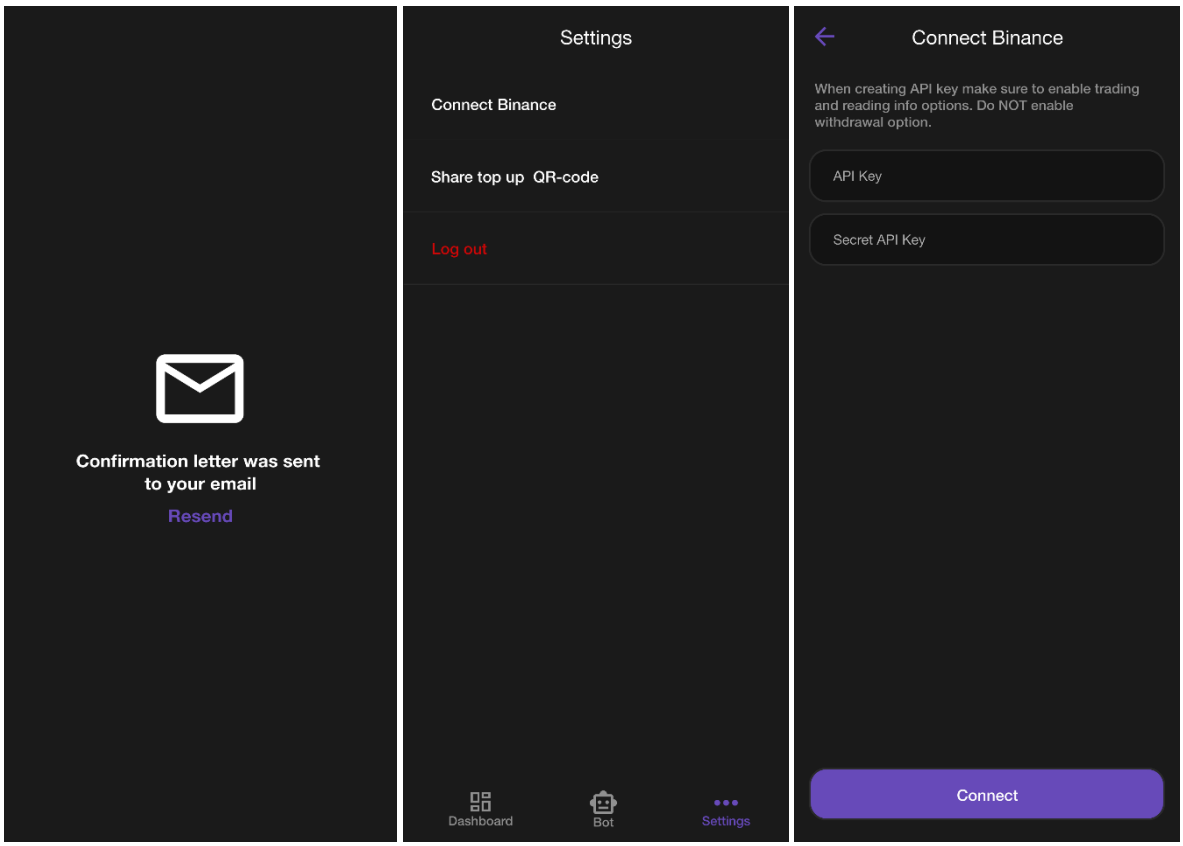
- This is the welcome screen for the TradeBit app, presenting a sleek design with a space-themed background. Below the graphic, users are greeted with "Welcome to TradeBit" and given two options: 'Sign in' and 'Sign up', indicating the entry points for returning users and new users respectively.

2)Sign-In Screen:

- This is a straightforward login interface where users can enter their credentials, including email and password. Below the 'Sign in' button, there are options for users to navigate to the sign-up screen or to reset a forgotten password.

3)Sign-Up Screen:

- This screen is for new users to create an account. It asks for the first name, last name, email, password, and a repetition of the password for confirmation. Below the 'Sign up' button, there's a link for users who already have an account to switch to the sign-in page. The design is intuitive, making it easy for new users to register and start using the application.



4)Email Confirmation Screen:

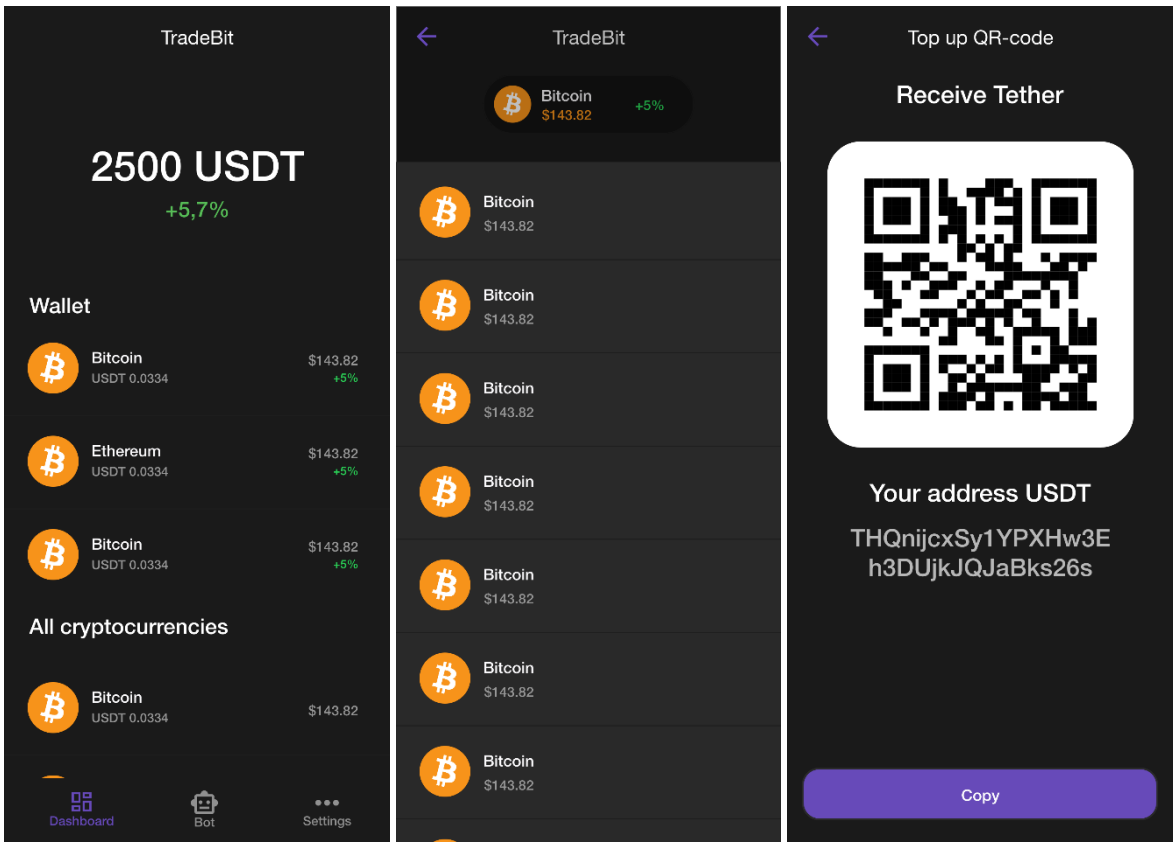
- After signing up, users are directed to this screen, which notifies them that a confirmation letter has been sent to their email. There is also an option to resend the email, suggesting a focus on a seamless user experience and ensuring that users can verify their accounts without hassle.

5)Settings Screen:

- This screen allows users to manage their account settings. It includes options to connect to Binance, share a top-up QR-code, and log out. The design is clean and straightforward, following the dark theme for better visibility and reduced eye strain.

6)Connect Exchange Screen:

- This screen is for connecting to the Binance exchange. It includes input fields for the API Key and Secret API Key. Instructions caution the user to enable trading and reading info options but not to enable the withdrawal option. It follows the same design language with a dark theme and a 'Connect' button at the bottom.



7)Dashboard Screen:

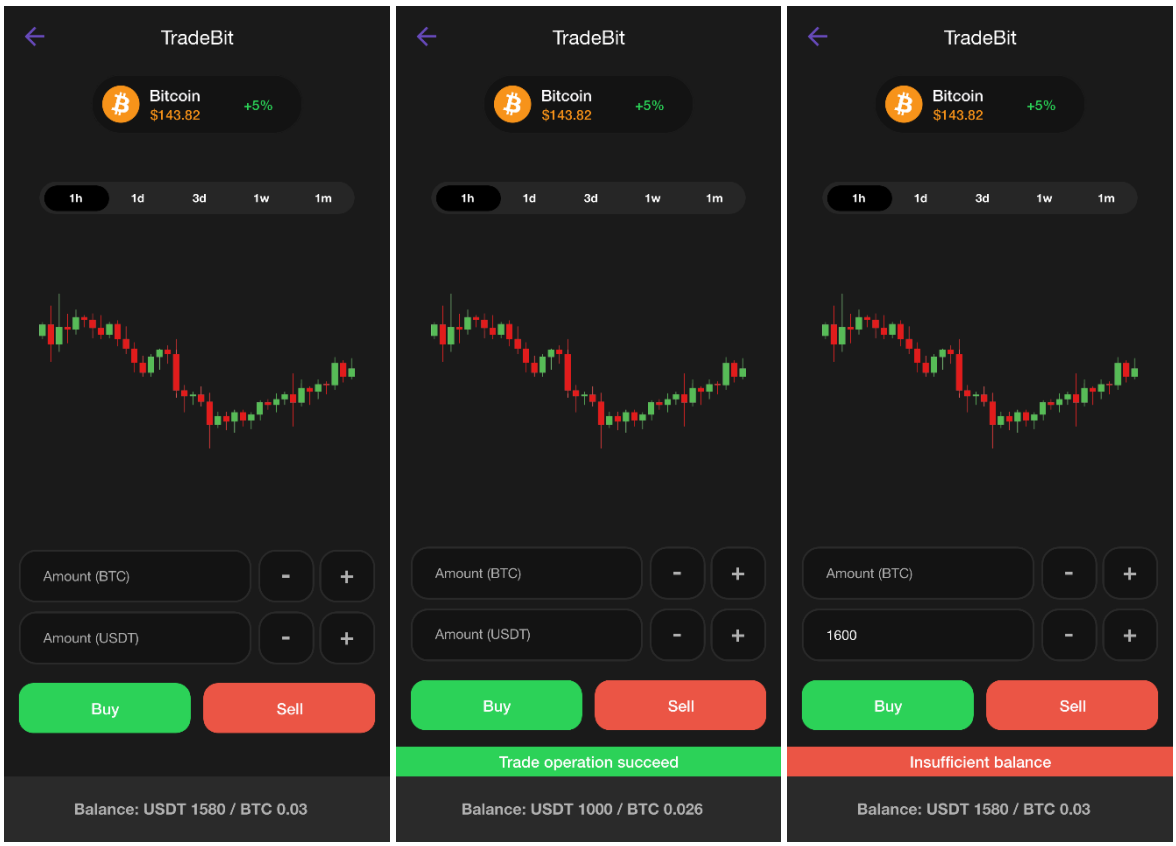
- The dashboard screen displays the user's wallet balance and the performance of various cryptocurrencies. It shows the total balance in USDT and the percentage change. Below, there are individual entries for Bitcoin and Ethereum, each showing the amount held and the current value in USDT. There's also a section for all cryptocurrencies, suggesting a summary view of the user's portfolio.

8)Select Cryptocurrency Screen:

- This screen is a scrollable list of cryptocurrencies available for trading on the app. Each cryptocurrency is represented by its logo (Bitcoin in this case) and its current price, alongside a percentage that may indicate the recent price change. The uniformity of the currency shown suggests this might be a placeholder or a template for when more currencies are added.

9)Top-up QR-code Screen:

- This screen provides a QR code for topping up the wallet with Tether (USDT). It displays the QR code along with the user's address for USDT. There's also a 'Copy' button for easily sharing or saving the wallet address. The use of QR codes indicates a user-friendly approach to transferring funds.



10)Trade Screen:

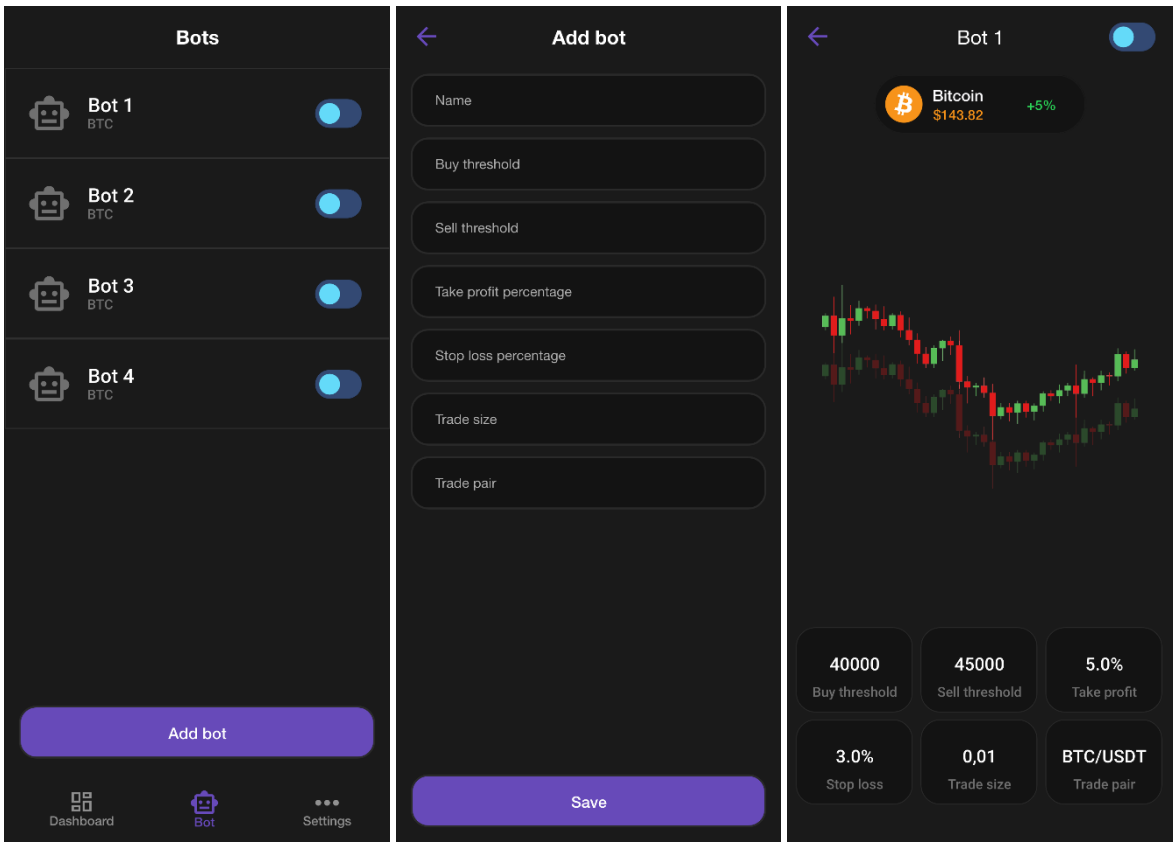
- This one allows the user to buy or sell Bitcoin. It shows the current price, the percentage change, and the candlestick chart. Input fields for specifying the amount in BTC or USDT are provided, along with 'Buy' and 'Sell' buttons. The user's balance is displayed at the bottom, and unlike the previous screen with an 'Insufficient balance' alert, this one does not show any warning, indicating that the user has sufficient funds for trading.

11)Trade Screen (Successful Operation):

- This screen is similar to the priveous screen but indicates a 'Trade operation succeed' message, showing that a transaction has been completed. The balance has been updated to reflect the new amounts of USDT and Bitcoin.

12)Trade Screen (Insufficient Balance):

- This screen is designed to inform the user of an unsuccessful trade operation due to insufficient funds.



13)Bots List Screen:

- This screen displays a list of trading bots that the user has configured. Each bot has an icon, a name, and the traded currency pair indicated next to it. There's a toggle to activate or deactivate each bot. At the bottom of the screen, there's a button to add a new bot. The interface follows the same dark theme, ensuring visual consistency.

14)Add Bot Screen:

- This screen allows the user to configure a new trading bot. It has input fields for the bot's name, buy threshold, sell threshold, take profit percentage, stop loss percentage, trade size, and trade pair. The screen has a dark theme with rounded input fields and a 'Save' button at the bottom.

15)Individual Bot Screen:

- This screen details an individual trading bot's settings. It shows the bot's name, the currency it trades (Bitcoin), and a candlestick chart indicating the performance of the bot. Below the chart, there are parameters such as buy threshold, sell threshold, stop loss percentage, trade size, and trade pair (BTC/USDT), which the user can presumably adjust. A toggle button at the top right corner likely enables the user to activate or deactivate the bot.