AQA A-LEVEL COMPUTER SCIENCE

NEA

**CRYPTO**

**BOT**

PAUL SHYAMALA (13-J)

# Table of Contents

[Table of Contents 2](#_Toc197891420)

[1 Analysis 7](#_Toc197891421)

[1.1 Introduction 7](#_Toc197891422)

[1.2 Client Background and Needs 7](#_Toc197891423)

[1.2.1 Interview with Beginner Trader 7](#_Toc197891424)

[1.2.2 Interview with Advanced Trader 9](#_Toc197891425)

[1.2.3 Common Needs 11](#_Toc197891426)

[1.2.4 Skills of Different Traders 12](#_Toc197891427)

[1.2.5 Trading Statistics 13](#_Toc197891428)

[1.3 Current Systems 14](#_Toc197891429)

[1.3.1 Observations 14](#_Toc197891430)

[1.3.2 Current Platform in Use 15](#_Toc197891431)

[1.4 Platform Reviews 18](#_Toc197891432)

[1.4.1 Investopedia’s Methodology 18](#_Toc197891433)

[1.5 Objectives 20](#_Toc197891434)

[1.5.1 Develop a UI with Relevant Features 20](#_Toc197891435)

[1.5.2 Accessbility 20](#_Toc197891436)

[1.5.3 Get Bitcoin Price Data from Binance API 20](#_Toc197891437)

[1.5.4 Make Algorithms for Technical Indicators 20](#_Toc197891438)

[1.5.5 Sustainability of Code and Readability 21](#_Toc197891439)

[1.5.6 Use of SQL Databases 21](#_Toc197891440)

[1.5.7 Applying a Risk Level System 21](#_Toc197891441)

[1.5.8 Choosing Between Live and Historical Trading 21](#_Toc197891442)

[1.5.9 Trade Reasoning (Educational Feedback) 22](#_Toc197891443)

[1.6 Skills in Use 22](#_Toc197891444)

[1.6.1 API Integration 22](#_Toc197891445)

[1.6.2 Technical Analysis 22](#_Toc197891446)

[1.6.3 Beginner Friendly UI 23](#_Toc197891447)

[1.6.4 Testing 23](#_Toc197891448)

[1.7 Limitations 24](#_Toc197891449)

[1.7.1 API Constraints 24](#_Toc197891450)

[1.7.2 Challenges in Technical Analysis 24](#_Toc197891451)

[1.7.3 Data Integrity 24](#_Toc197891452)

[1.7.4 UI 25](#_Toc197891453)

[1.7.5 Security 25](#_Toc197891454)

[1.8 IPSO Diagrams 26](#_Toc197891455)

[1.8.1 Input 26](#_Toc197891456)

[1.8.2 Process 26](#_Toc197891457)

[1.8.3 Storage 26](#_Toc197891458)

[1.8.4 Output 26](#_Toc197891459)

[1.8.5 Data Flow Diagram 27](#_Toc197891460)

[2 Documented Design 28](#_Toc197891461)

[2.1 Overview 28](#_Toc197891462)

[2.1.1 User Interface (UI) 28](#_Toc197891463)

[2.1.2 Data Retrieval and Processing 28](#_Toc197891464)

[2.1.3 Trade Feedback 28](#_Toc197891465)

[2.1.4 Database Integration (SQL) 28](#_Toc197891466)

[2.2 API Documentation 29](#_Toc197891467)

[2.2.1 Fetching Market Data 29](#_Toc197891468)

[2.2.2 Placing Trades 29](#_Toc197891469)

[2.2.3 Checking Trade History 30](#_Toc197891470)

[2.3 Library Documentation 31](#_Toc197891471)

[2.3.1 Tkinter 31](#_Toc197891472)

[2.3.2 Matplotlib 31](#_Toc197891473)

[2.3.3 SQLite3 31](#_Toc197891474)

[2.3.4 Pandas 31](#_Toc197891475)

[2.3.5 Requests 32](#_Toc197891476)

[2.3.6 Time 32](#_Toc197891477)

[2.4 Flowchart 33](#_Toc197891478)

[2.4.1 Summary 34](#_Toc197891479)

[2.5 Top-Down View 35](#_Toc197891480)

[2.6 GUI Interface 37](#_Toc197891481)

[2.7 Data Structures 42](#_Toc197891482)

[2.7.1 User Profile 43](#_Toc197891483)

[2.7.2 Trade History 43](#_Toc197891484)

[2.7.3 Application Logs 45](#_Toc197891485)

[2.7.4 Risk Thresholds 45](#_Toc197891486)

[2.7.5 Configuration Table 45](#_Toc197891487)

[2.7.6 Account History Table 46](#_Toc197891488)

[2.8 Pseudocode Algorithms 47](#_Toc197891489)

[2.8.1 RSI Pseudocode 47](#_Toc197891490)

[2.8.2 MACD Pseudocode 49](#_Toc197891491)

[2.8.3 Supertrend Pseudocode 51](#_Toc197891492)

[2.8.4 Password Hashing Pseudocode 53](#_Toc197891493)

[2.8.5 Buying and Selling Pseudocode 54](#_Toc197891494)

[2.9 Testing Plan 55](#_Toc197891495)

[2.9.1 Unit Testing and Parameter Optimization 55](#_Toc197891496)

[2.9.2 Integrated Testing 55](#_Toc197891497)

[2.9.3 Error Handling Tests 55](#_Toc197891498)

[2.9.4 Trading Tests 55](#_Toc197891499)

[3 Technical Solution 56](#_Toc197891500)

[3.1 Python Files 56](#_Toc197891501)

[3.1.1 main.py 57](#_Toc197891502)

[3.1.2 ui.py 57](#_Toc197891503)

[3.1.3 activate\_bot.py 86](#_Toc197891504)

[3.1.4 bot\_indicators.py 90](#_Toc197891505)

[3.1.5 btccschart.py 98](#_Toc197891506)

[3.1.6 auth.py 103](#_Toc197891507)

[3.1.7 api.py 105](#_Toc197891508)

[3.1.8 database.py 110](#_Toc197891509)

[3.1.9 utils.py 120](#_Toc197891510)

[3.2 Database 122](#_Toc197891511)

[3.2.1 user\_profile 122](#_Toc197891512)

[3.2.2 trade\_history 122](#_Toc197891513)

[3.2.3 application\_logs 124](#_Toc197891514)

[3.2.4 risk\_thresholds 124](#_Toc197891515)

[3.2.5 bitcoin\_config 124](#_Toc197891516)

[3.2.6 account\_history 125](#_Toc197891517)

[3.3 Image Files 126](#_Toc197891518)

[3.3.1 btblogo.png 126](#_Toc197891519)

[3.3.2 profile.png 126](#_Toc197891520)

[3.3.3 true.png 126](#_Toc197891521)

[3.3.4 false.png 126](#_Toc197891522)

[3.4 UI Implementation 127](#_Toc197891523)

[3.4.1 Login/Register Page 127](#_Toc197891524)

[3.4.2 Preference Page 127](#_Toc197891525)

[3.4.3 Home Page 128](#_Toc197891526)

[3.4.4 Profile Page 128](#_Toc197891527)

[3.4.5 Balance Page 129](#_Toc197891528)

[3.4.6 Trade History Page 129](#_Toc197891529)

[3.4.7 Date Page 130](#_Toc197891530)

[3.4.8 Risk Page 130](#_Toc197891531)

[3.4.9 Bot Indicators Page 131](#_Toc197891532)

[3.4.10 Workstation Page 131](#_Toc197891533)

[3.5 GitHub Repository 132](#_Toc197891534)

[4 Testing 133](#_Toc197891535)

[4.1 Input Validation Tests 133](#_Toc197891536)

[4.1.1 Username 133](#_Toc197891537)

[4.1.2 Password 135](#_Toc197891538)

[4.1.3 Add Money 138](#_Toc197891539)

[4.1.4 Trading Timeframe 139](#_Toc197891540)

[4.2 Exception Handling Tests 143](#_Toc197891541)

[4.2.1 Valid User 143](#_Toc197891542)

[4.2.2 Valid Password 144](#_Toc197891543)

[4.2.3 Database Exception 144](#_Toc197891544)

[4.2.4 API Exceptions 145](#_Toc197891545)

[4.3 Functionality Tests 147](#_Toc197891546)

[4.3.1 Register User 147](#_Toc197891547)

[4.3.2 Trading Preference 148](#_Toc197891548)

[4.3.3 Display Profile 148](#_Toc197891549)

[4.3.4 Display Trade History 149](#_Toc197891550)

[4.3.5 Balance Update 150](#_Toc197891551)

[4.3.6 Indicator Graphs – Live 151](#_Toc197891552)

[4.3.7 Indicator Graphs – Historical 152](#_Toc197891553)

[4.3.8 Activate Bot – Live 152](#_Toc197891554)

[4.3.9 Activate Bot – Historical 153](#_Toc197891555)

[4.3.10 Application Log 154](#_Toc197891556)

[4.4 Trading Tests 154](#_Toc197891557)

[4.4.1 Bullish Trend Test 1 154](#_Toc197891558)

[4.4.2 Bullish Trend Test 2 155](#_Toc197891559)

[4.4.3 Bearish Trend Test 1 156](#_Toc197891560)

[4.4.4 Bearish Trend Test 2 156](#_Toc197891561)

[5 Evaluation 158](#_Toc197891562)

[5.1 Objectives 158](#_Toc197891563)

[5.2 Unmet Objectives 159](#_Toc197891564)

[5.3 Client Review and Feedback 159](#_Toc197891565)

[5.4 Improvements 160](#_Toc197891566)

# Analysis

## Introduction

Cryptocurrencies are very volatile, which can cause beginners to experience substantial losses when starting out. This is why trading bots have become increasingly useful as they eliminate human vulnerabilites, allowing users to focus on the learning part without the risk of losing money. However, most platforms do not explain to the user why particular trades occur. This beginner trading bot aims to solve this issue. Making use of well-known indicators, users can understand why trades are executed which will equip them with the knowledge to succeed in their trading journey. The main purpose of this project is to assist beginner traders with their learning process by exposing the inner workings of each trade that is executed. Through this, users can have an enhanced overview and thought process when it comes to trading which will as a result amplify their gains and minimize losses.

## Client Background and Needs

### Interview with Beginner Trader

#### Interview

**Client Details:**

Prasad Shyamala, a beginner crypto trader. He has previously traded gold and oil, but often without rational thought. Now trying to hop on the crypto train, he is looking to solidify his understanding of the markets

Me: I remember you’ve told me a while back that you started trading Bitcoin. Could you tell me a bit about how it has been?

Client: Yes, I’ve recently started trading Bitcoin due to its increased volatility as people say. I have made some decent profits, but at times it has been difficult. It’s difficult to time my trades as Bitcoin changes price very quickly a lot; this makes it mandatory to never take your eyes off the charts. However, I don’t have the time to watch it constantly. This has caused me to take some losses

Me: Oh, that’s unfortunate. It sounds to me like predicting the market is a main challenge for you. Does the idea of having an automated program that trades on its own help you in this situation?

Client: That sounds perfect! It sounds like the bot makes faster and more logical decisions that I do. And I also think it would solve my problem of watching the markets 24/7 as the bot can watch it for me! Besides this, I get quite emotional when something extreme happens, which has led me to lose money. So, I’m hoping this bot can solve this

Me: Yes, this is a problem that lot of beginner traders experience. The stress is overwhelming at first, but it will get better over time. Other than what you said, what else are you looking to improve on?

Client: I’ve realized that when I sit down to trade, I just read the markets blindly and guess most of the time. I know that this is very risky and not sustainable at all. So, I guess I would like to learn a bit more about the trades that happen.

Me: It’s great that you’ve identified that. One feature of the bot that I’m making is that it gives you details on each trade that happens. It not only makes decisions for you but also explains why it made those choices, whether it’s based on chart patterns or market signals. Do you think this will benefit you?

Client: That’s great! Once I’ve learnt the “behind the scenes” of trading, I can apply these lessons to when I trade on my own.

Me: Looking ahead, how do you think that would impact your long-term goals as a crypto trader?

Client: It would really help me grow as a trader. Right now, I’m just hoping to make some extra income, but if I could learn and gain confidence with each trade, I’d start thinking about bigger, more long-term investments. I’d feel more comfortable taking calculated risks. In the long run, it would give me the tools to build a stronger portfolio and make informed decisions on my own.

Me: That’s great. Looks like this bot would be perfect for your situation.

Client: Thank you so much. Goodbye!

#### Summary

**Client Background:**

The client is a beginner crypto trader with limited experience in the cryptocurrency market. He has been trading for a few months but feels overwhelmed by the volatility of the crypto market.

**Key Challenges:**

* The client often makes trades with no reasoning
* They struggle to keep up with the 24/7 nature of crypto markets and miss opportunities when they’re unable to monitor the market
* The client finds it difficult to understand the technical side and feels like they are guessing most of the time.

**Client Needs:**

* The client needs a bot that can handle trades automatically
* The client is looking for more than just a passive income tool—they want a bot that teaches them why trades are made.
* They need a simple, intuitive platform

### Interview with Advanced Trader

#### Interview

**Client Details:**

Ashish Yeruva, an experienced crypto trader, has worked with cryptocurrencies for about 2 years. He has created profits with not just Bitcoin but also Ethereum and is looking to expand his portfolio.

Me: Hey Ashish, thanks for hopping on this call with me. I just wanted to ask you a few questions about trading cryptocurrencies. As an experienced crypto trader, I’m sure you’ve built effective strategies for generating profit. But how do you handle making money when you’re not actively trading?

Ashish: That’s the part where you miss the most opportunities. Even with all my strategies, I can’t be in front of my computer 24/7. Even though I’ve made profits when I’m actively trading, I still think about what happens when I’m not and the amount of money I could have made. I know it sounds arrogant, but I can’t help it.

Me: How do you feel about having a tool that can work in the background that’s supplementing your regular trades? Even though it generates small profits, it makes money!

Ashish: Well even if the profits are small, if they accumulate over time, it’s worth it. I already have my active trading strategies, so having a bot that can work in the background when I’m not around would be a good way to make my money work for me. Yes, I’ve read Rich Dad Poor Dad (Laughing)

Me: Me too! Would you say that this kind of system could be helpful for other traders, especially beginners who might not be ready for advanced strategies yet?

Ashish: Definitely. For beginners, the main issue is learning without losing too much money. This was a huge problem for me when I was first starting out. A bot that can generate small profits would help them build confidence and understand the market without having to crash and burn.

Me: I completely agree. The bot also explains each trade it makes, so beginners can learn why certain decisions were made. Do you think having that educational aspect would help them get better faster?

Ashish: Absolutely. It’s one thing to make a trade, but understanding why that trade was made is key to improving as a trader. For example, when I first started trading Bitcoin in 2022, I just read the news and followed the normal trends and bought and sold according to that. This is not how it should be done. Later, I focused on chart analysis, and this increased my profits a lot. For beginners, having the bot explain the decisions behind each trade will speed up the process.

Me: Do you think this bot would be useful for someone like you as well?

Ashish: For sure. While I focus on my bigger trades, this bot could generate small profits on the side. It doesn’t have to be huge gains—small, consistent profits add up over time, especially if you’re not actively managing it.

#### Summary

**Client Needs:**

* Ashish cannot monitor the market 24/7, and he often miss opportunities during market movements when he is not actively trading.
* He wants to make the most of time he is not trading,
* He also acknowledges that beginners struggle with both learning the market and avoiding huge losses and sees value in platforms that help them step-by-step.

**Reasoning Behind Trades:**

For beginners, understanding why trades are made is critical to long-term success in the crypto market. Ashish recognizes this and appreciates that the bot helps the user learn why trades are made.

### Common Needs

Automated Trading – Executes trades without supervision.  
Emotion-Free Decision Making – Uses logic-based strategies to remove human emotions.  
Passive Profit Accumulation – Generates small gains over time.

Both Prasad and Ashish see value in a trading bot, though their needs differ slightly. Beginners like Prasad need automation and learning features, while experienced traders like Ashish want passive income.

### Skills of Different Traders

|  |  |  |
| --- | --- | --- |
| Domain | Beginners | Experienced |
| Market Understanding | Very basic, only know about Bitcoin and maybe Ethereum, gets most information from news and social media | Sophisticated knowledge, with concrete understanding of trends, correlations, consistently follows up on latest updates, doesn’t fall for news or social media hype and relies on logic |
| Technical Analysis | Often no attempt to analyze market conditions, only trades with the crowd | Extensive use of advanced indicators, patterns and analysis, spending a lot of time before commiting to a trade |
| Risk Management | Concept of risk held in low regard, has no plan, inconsistent trading | Uses stop losses, risk to reward ratios, position sizing, and has a tried and tested risk management system |
| Emotional Control | Has low tolerance for bad conditions, panics very easily, has FOMO, no decision based on logic | Very disciplined, and is aware of all potential psychologial traps, and always relies on logic |
| Analyzing Data | Can’t interpret a wide range of data, sticks to price charts and maybe a few indicators | Comfortable analyzing multiple types of data such as bid/ask price, bands, etc. |

### Trading Statistics

Cointelegraph -   
Cointelegraph - This article shows how 95% of traders lose money, primarily due to beginners not following a structured plan, making desicions based on emotions, and thinking it's easy to make money with crypto. Showcasing the fact that most beginners mistake luck for skill, it explains that beginners experience something called "random reinforcement", that later leads to trades without any meaning. The article also notes that 80% of day traders quit within two years, and nearly 40% stop within one month, once again showing the high failure rate among beginner traders.  
  
Volatilty is another huge factor that results in substantial losses for beginner traders. They blindly follow market hype, only to sell at a loss during bearish trends. Studies indicate that 73% to 81% of Bitcoin investors lost money between 2015 and 2022, with a median loss of $431 on a $900 investment, highlighting the financial risks for traders who fail to deal with volatility properly.

## Current Systems

Binance is the most popular platform for trading cryptocurrencies. So naturally, a beginner is drawn to this. They decide to try out Binance’s crypto bots, expecting an easy solution to managing their trades while they focus on learning. They think that the bot will do all the work for them and that it’s “easy money”. However, once they start exploring the platform, they realize it’s not as beginner friendly as expected.

### Observations

#### Setup Process

Account Setup and Bot Activation:  
After creating an account, the user needs to complete verificatoin and add money to their account. To start using the bot, the user needs to enable API access  
  
Choosing a Bot Strategy:  
Then the user is shown a variety of strategies that the bots use such as grid trading, spot etc.   
These are completely unknown to the beginner and could possibly overwhelm the user, which might cause them to make unfavorable desicions  
  
Monitoring and Adjusting the Bot:   
Binance also offers advanced features such as stop-loss configurations and sensitivity adjustments, but the beginner does not know how any of these work. The platform does not educate the user of these features. Therefore they are left to figure things out by themselves resulting in losses.  
  
Note: This is what Prasad Shyamala (The beginner trader) has informed me, when he used Binance. I have condensed the issues into these bullet points.

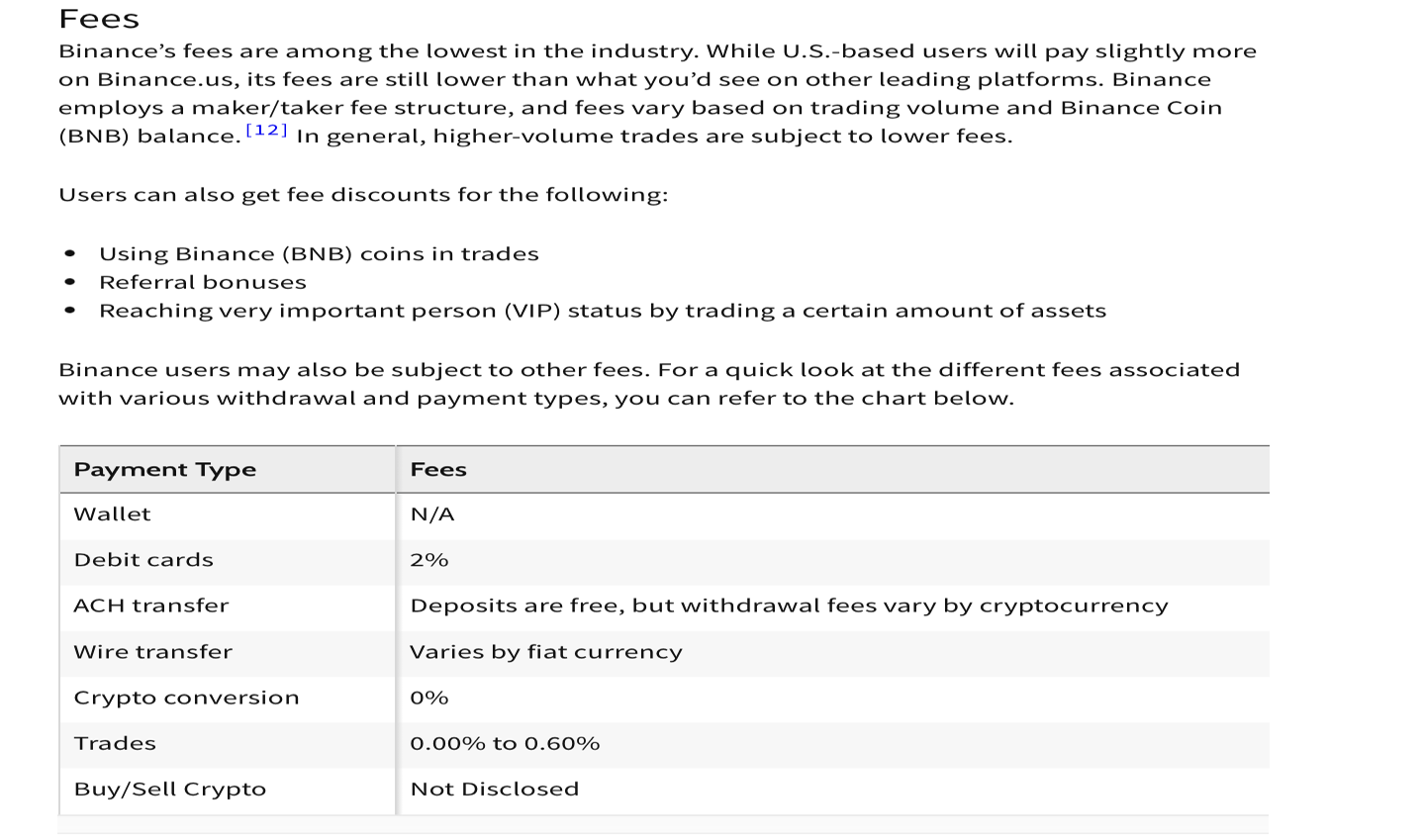
While Binance is a well-known platform, using it as the primary example of crypto bot functionality doesn’t provide a full understanding of the crypto market. Other platforms like Coinbase and Kraken have different interfaces, bot functionalities, and user experiences. To fully assess the issues and features of crypto bots, a thorough analysis across multiple exchanges and brokers is essential.

### Current Platform in Use

#### Opening an Account

When you open a Binance account, there is a long process to start tradin

#### Fees

After getting through the starting steps, you must think about fees. This would be quite overwhelming for beginners. And many of them stop at or after this point

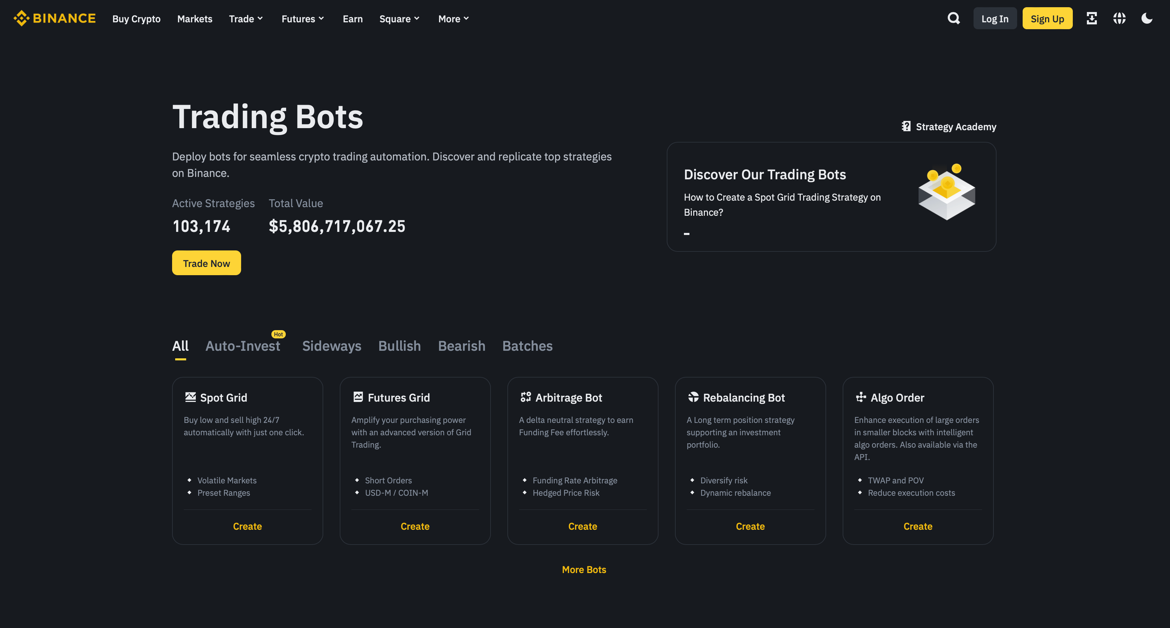
#### Login Page

A screenshot of a computer

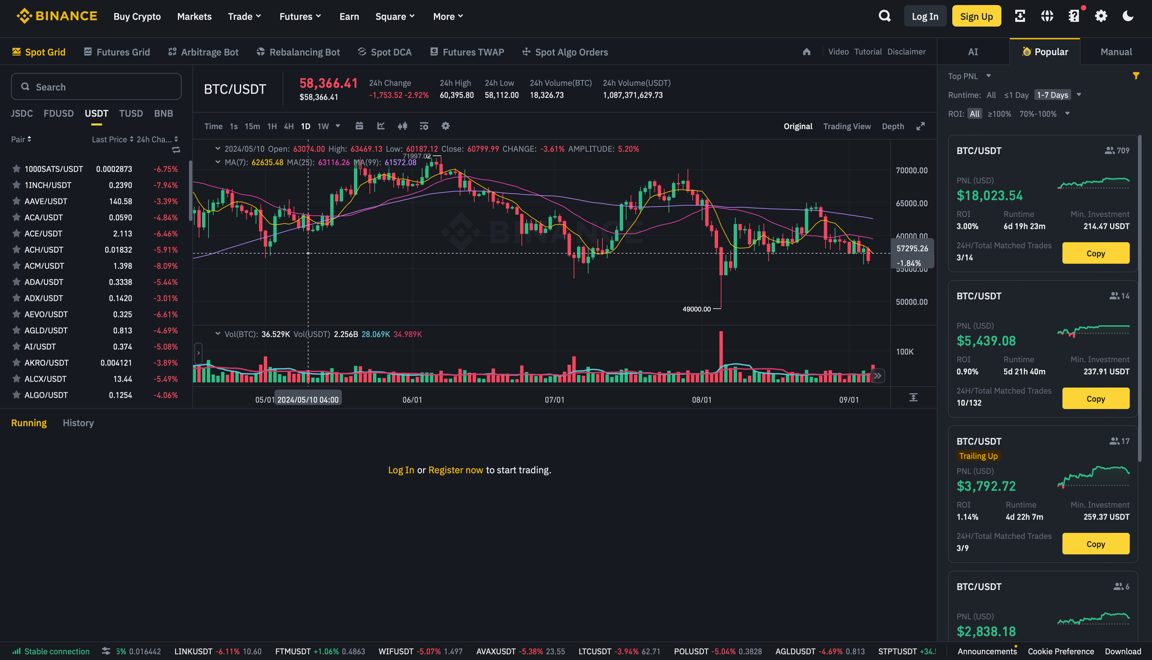
Description automatically generatedThe first page is the login page, where they must either create an account or sign in with an existing account

#### Variety of Bots

When you enter the trading bot section, the user is presented with a variety of bots leaving them puzzled as they have no idea which one to choose or how any of them works

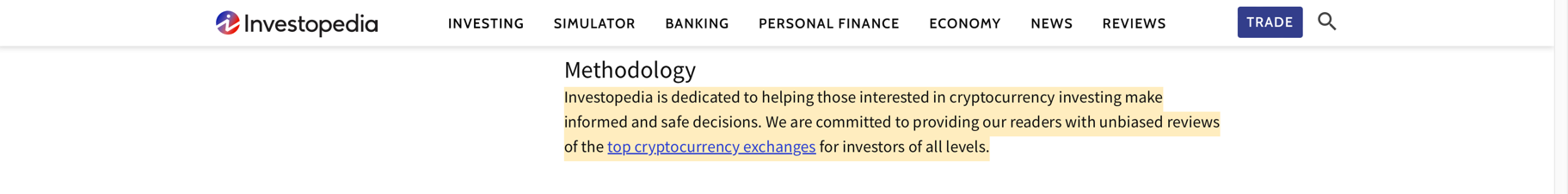


#### Complex Navigation

Once you choose a bot, a complex screen is presented. And this is hard to navigate.

## A screenshot of a computer screen Description automatically generatedA screenshot of a computer Description automatically generatedPlatform Reviews

### Investopedia’s Methodology

Based on this methodology, we can trust the assessments of Investopedia on crypto exchanges such as Binance.

**Pros:**

More than 150 tradable cryptocurrencies for U.S. customers: Binance offers more than 150 tradable coins for U.S. customers. This gives crypto traders and investors many options. If you're outside of the United States, you have even more options.

Low fees: The platform offers low fees, meaning traders get to keep more of their investments and earnings.

Wide selection of trading options and order types: Binance offers several trading options including peer-to-peer trading, spot trading, and margin trading. It also offers lots of order types, including limit order, market order, stop-limit order, stop market order, trailing stop order, post only order, and one-cancels-the-other order. Some of these options including margin trading are not available for U.S. customers.

**Cons:**

The platform is complex and may be confusing: While the broad range of features and trading options on Binance may be exciting, it can also be intimidating. Even experienced traders may feel overwhelmed by all the options available.

No built-in digital wallet available: While some popular exchanges have built-in digital wallets, Binance does not. It recommends Trust Wallet, which Binance owns. Trust Wallet has a good reputation but may only offer limited support if there are issues with Binance transfers.

Binance has run into regulatory trouble in several countries: Binance has faced several regulatory and legal issues in multiple countries, so crypto investors may want to consider other exchanges.

## Objectives

### Develop a UI with Relevant Features

* Make a login systen
* Allow user to select between Live Trading and Historical Trading (Backtesting)
* Display account metrics: Account Balance, Equity Curve, and Trade History.
* Allow users to add money
* Include Technical Analysis, showing indicator graphs and BTC price charts.
* Execute trades based on technical analysis

### Accessbility

* Wide rane of beginner users must be able to access and navigate the platform without any stress
* No complex features that could make the user think twice about what to do
* Can be used by beginners with varying levels of technical ability

### Get Bitcoin Price Data from Binance API

* Connect to the Binance API and fetch OHLC data for BTC/USDT.
* Error handling to deal with API request limits, timeouts, and network issues.
* Get required data from JSON responses for technical analysis and plotting charts.

### Make Algorithms for Technical Indicators

* Make code for:
  + RSI (Relative Strength Index)
  + MACD (Moving Average Convergence Divergence)
  + Supertrend
* Plot these indicators alongside the Bitcoin price chart.

### Sustainability of Code and Readability

* Write modular and efficient code, separating logic into many components, and reusing code where possible to reduce amount of code written.
* Apply OOP principles for maintainability and scalability.
* Make sure that the code is well-structured and commented so that it can be understood and debugged easily

### Use of SQL Databases

* Create a SQL database to store:
  + User Profiles
  + Trade History
  + Application Logs
  + Configuration
  + Risk Thresholds and Preferences

### Applying a Risk Level System

* Make a risk level system (e.g., 1 – Low, 2 – Medium, 3 – High) for users to select.
* These risk levels can adjust:
  + Investment amount per trade
  + Parameters of indicators
* Enable the bot to make different trading decisions based on the selected risk level.

### Choosing Between Live and Historical Trading

* Allow users to choose between:
  + Live Trading: Real-time trading with current market data.
  + Historical Trading (Back testing): Simulate past trades using historical data to show the user how the bot performs.
* This will let users gain trust in the bot before live trading.

### Trade Reasoning (Educational Feedback)

* Display information about trades such as:
  + Timestamp of trade
  + Price at entry and exit
  + Indicators that affected the trade
* Help users understand why a trade was executed based on the flags from technical indicators.

## Skills in Use

### API Integration

Firstly, we have to choose an API to get all our Bitcoin data from. This will most probably be done using the Binance API.

* **Fetching Data**: The bot will connect to the Binance API to get the required data.
* **Handling Data**: Using requests, the bot can connect to the API and the returned JSON data will be filtered to get required information, such as OHLC.

### Technical Analysis

Once the data is filtered, the next step is doing technical analysis. This analysis will be based on famous indicators that are commonly used.

Here are a few indicators that might be used.

**RSI** -

**Moving Average Convergence Divergence** **(MACD)**–

**Supertrend**

### Beginner Friendly UI

**User-Friendly Design:** The frontend of the platform will be minimalistic, containing only the relevant features so as to not overwhelm the user.

**Educational Component:** This could be done by displaying when the trade was bought and sold. Or what indicators were used when analyzing the trade. The users can then understand the reasoning behind each trade and can therefore enhance their understanding.

### Testing

Once the bot’s core functionality and user interface are built, testing will be done to ensure it operates smoothly and without error in a live trading environment. (Even when the bot is given to customers, it will act as a demo account for practice)

Back testing: The bot will also have testing with historical data to verify the accuracy of its trading decisions and technical analysis.

## Limitations

### API Constraints

API Rate Limit: Most API’s have limits on how many requests can be made at a particular time. This could affect how the data is being received and sometimes may cause the platform to work with no data.

Downtime: API’s sometimes experience downtime or disruptions, making it difficult to get data and mainting functionality of the platform. The bot will error handling to handle such issues, which can happen at any time.

Limited Historical Data: Some APIs may not have extensive historical data, which could reduce the bot’s capabilites for historical trading and effective backtesting

Latency: API responses can sometimes be delayed due to network issues, which means the data could arrive at a later time, which is not favourable in the fast moving market of crypto.

### Challenges in Technical Analysis

Lagging Indicators: Many beginner-friendly indicators, such as RSI, are lagging indicators. This means that they depend on historical data and may not respond quickly to volatile changes. This could lead to delayed trade execution or missed opportunities.

Over-Simplified Strategies: While simple strategies are beginner-friendly, they may not always work in the fast moving crypto market. This could lead to missed oppurtunities or losses if the bot isn't created with a better risk management system or adaptive strategies.

### Data Integrity

Inconsistent Data: Different exchanges might provide slightly different data, which could impact technical analysis and trade execution. Inconsistent data from APIs could lead to innacurate decisions.

### UI

Limited Real-Time Updates: Handling market updates in a desktop app can be difficult. Making sure that the platform remains responsive while performing background tasks (data fetching, trade execution) is vital.

### Security

Handling API Keys: Storing and securing API keys within a desktop platorm is very risky. If they are not properly encrypted, they could be leaked to unauthorized users, which could then compromise the data of users.

User Trust: Beginners may not fully understand the risks in algorithmic trading. To build trust in beginners, clear explanations on how the bot works is needed. However, if the explanations are over simplified, beginners may mistake information for something else, building false trust.

## IPSO Diagrams

### Input

* Username and Password
* Account Balance
* Risk Level
* Trading Preference

### Process

* Fetching data from Binance API
* Applying technical indicators
* Creating buy/sell signals based on technical analysis
* Managing API errors, and other issues such as network problems, etc. to ensure the bot doesn’t crash

### Storage

* Storing all information about executed trades, in a local database (SQLite3).
* Logs of the bot’s performance (profit/loss).
* Saving user settings, such as trading preferences and risk levels.

### Output

* Displaying information on the interface, such as account balance, equity curves, trade history, and bot progress
* Displaying candlestick charts and technical indicators using Matplotlib.
* Displaying all features of the platform

### A diagram of a system AI-generated content may be incorrect.Data Flow Diagram

Risk level

# Documented Design

## Overview

This beginner crypto trading bot is made to trade Bitcoin either on a live or historical setting, helping users learn how and why trades are made without the risk of losing their actual money. The bot operates with the use of three technical indicators which are Supertrend, RSI, and MACD. These indicators are used to make the final decision. The basic logic is that when 2 out of 3 indicate an uptrend, the bot buys, and when 2 out of 3 suggest a downtrend, it sells. Before all this, users can select a risk level from 1-3 that affects how much money is invested per trade, and how the indicators work. Higher risk means the probability of a trade happening is much more likely as the indicators are much more sensitive. After a trade is finished, the platform shows the details of the trade from which users can learn. An equity curve is displayed to showcase how their portfolio has grown.

### User Interface (UI)

The UI is designed to be intuitive and simple, which is essential beginners in crypto trading. It’s a desktop application with a clean layout that doesn’t overwhelm the user.

### Data Retrieval and Processing

The bot uses APIs to access real-time and historical Bitcoin price data.

* **API**: Connects to Binance API for fetching Bitcoin price data
* **Data Preprocessing**: Cleans and formats the data to make it compatible with the bot’s trading algorithm.
* **Technical Indicators**: Makes a decsion on wether ti buy or sell based on the indicator calculations

### Trade Feedback

After each trade, the bot provides details on it, helping users understand the trade's outcome and logic.

### Database Integration (SQL)

A SQL-based database stores historical trades, allowing the user to see past trades and analyze them. Also stores used profile, application logs, risk thresholds, etc.

## API Documentation

In this project, the Binance API will be used to get Bitcoin price data. It could also be used to place buy and sell orders.

### Fetching Market Data

Get Current Bitcoin Price

* **Function**: get\_symbol\_ticker(symbol='BTCUSDT')

This function gets the latest price of BTC/USDT and will be used for executing trades.

Get Historical Price Data

* **Function** client.get\_historical\_klines('BTCUSDT', '1d', timestamp, limit=1000)

This functions gets the historical data in OHLC format, which can be used for historical trading or backtesting

### Placing Trades

Execute Market Buy Order

* **Function**: order\_market\_buy(symbol='BTCUSDT', quantity 0.001)

This function places a buy order at the current market price. This will be used when the bot creates a buy signal

**Ex**ecute Market Sell Order

* **Function**: order\_market\_sell(symbol='BTCUSDT', quantity=0.001)

This function places a sell order at the current market price. This will be used when the bot creates a sell signal

### Checking Trade History

Get Past Trades

* **Function**: get\_my\_trades(symbol='BTCUSDT', limit=50)

**This function gets trade history of an particular account and returns data such as** trade price, quantity, timestamp, etc.

Note: these function may be built locally instead of usign the API’s functions as the API might not allow so many calls. Therfore the management of balance and trade execution will be done locally.

## Library Documentation

### Tkinter

Tkinter will probably be used to create the UI for this platform. It allows users to interact with the platform, and can let them start the bot, adjust settings, view information, etc.

This library allows to create windows which will contain the main pages. Buttons will be made to take selections. Text boxes will be created to take in inputs from the user such as the account details, balance, etc. Tkinter also allows for sizing and placement of all these features in the window.

### Matplotlib

Matplotlib will be used to plot graphs of the different indicators and the bitcoin price chart. These graphs are essential in any trading platform.

Maplotlib has a wide variety of graphs that can be plotted such as line graphs, histograms, etc. We can also change its appearance by using different lables, colours, titles, etc.

### SQLite3

SQLite3 is the database platform beign chosen. It will store all data related to the user such as user profiles, trade histroy and application logs. It will also store standalone tables such as risk thresholds and configurations that are vital in this project. This will allow easy integration of the database into the bot, and will enhance the debugging experience if needed.

* Executing SQL Queries: SELECT, INSERT, UPDATE, and DELETE.

### Pandas

This library is used to process and extract data from SQL queries. Also will be used to organize data in a structured way, which will then be used by the indicators for technical analysis.

* DataFrames: A DataFrame is a 2D data structure for handling data in table form. It allows for data indexing, slicing, and easy data manipulation.
* Operations: Pandas allows for operations like filtering, grouping, and aggregation.

### Requests

This library makes client-server calls to the Binance API.

* Fetch real-time and historical market data from the Binance API.
* Enable communication between the bot and external services for data retrieval.

Error Handling: It is important to handle errors such as timeouts or failed requests.

### Time

This library controls the timing of the operations of the trading bot. This is to make sure that everything is going at a structured pace, and not rushing operations. This could include pausing a bit before buying and selling, timing API calls, etc.

* Control timing for API requests to stay within rate limits.
* Help manage the intervals for executing trades.

## A diagram of a diagram AI-generated content may be incorrect.Flowchart

### Summary

**Login/Register and Select Trading Preference**

The user logs into an existing account or registers a new one. Before going to the home page, the user first has to select the trading preference. Then the user can add balance which determines how much capital the bot can use for trading

**Risk Level**

Before starting the trading process, the user is asked to select a risk level, in which the amount of money taken to trade with, and the indicator settings are changed according to the level chosen.

**Indicators Are Applied to Live Data**

The bot fetches Bitcoin price data using the Binance API. It applies technical indicators, which are formulas that analyze price trends and market conditions.

**Program Checks If Majority of Indicators Come True (Decision Point)**

* The bot decides whether a **majority** of indicators create a buy or sell signal.
* Two possible outcomes:
  + **True (Buy Signal):** If 2/3 indicators become true, the bot will buy Bitcoin.
  + **False (Sell Signal):** If 2/3 indictors become false, the bot will sell Bitcoin.

**Buy Amount of Bitcoin (If Indicators Suggest Buying)**

* If the bot detects a **buy signal**, it will purchase Bitcoin using the available account balance.
* The bot determines **how much Bitcoin to buy**, often based on the risk level:
  + **Percentage of Total Balance:** Example: Invest 50% of the total balance.

**Sell the Amount of Bitcoin Being Held (If Indicators Suggest Selling)**

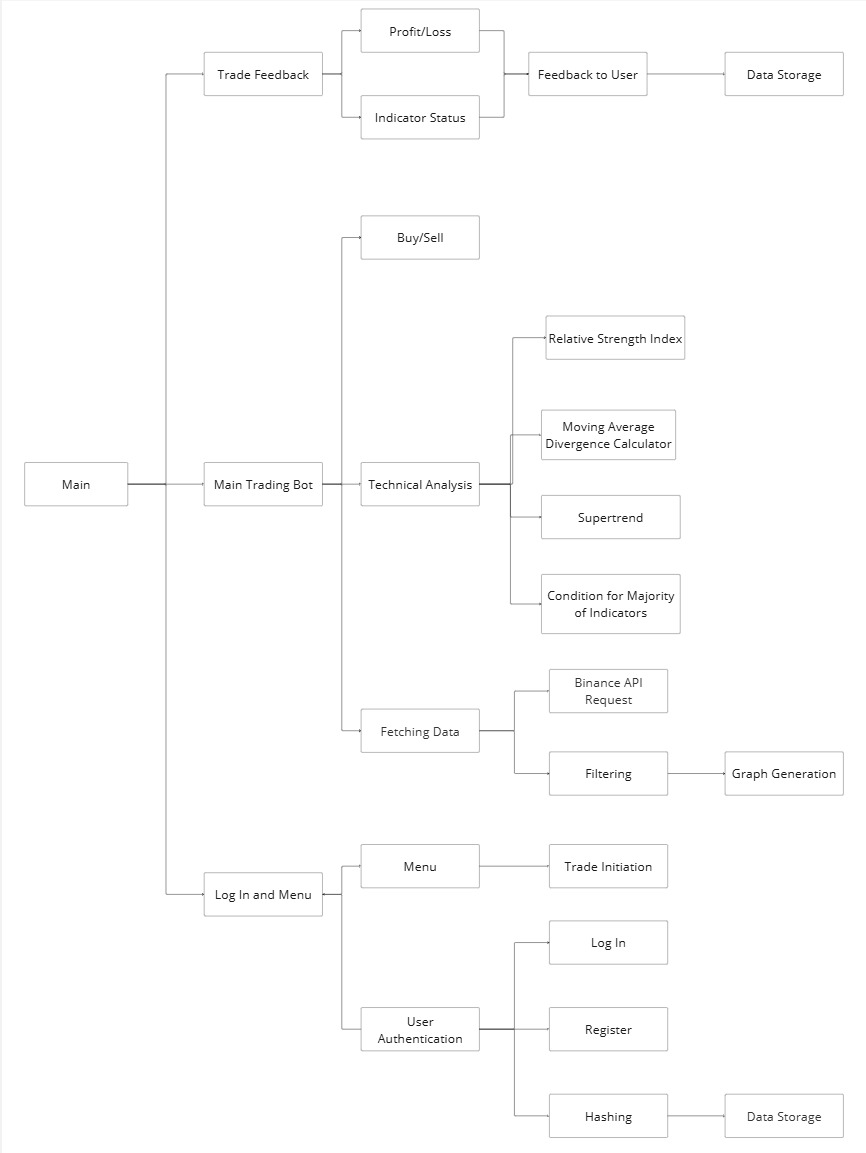
* If a **signal** is detected, the bot will sell the Bitcoin held.

**Output Profit/Loss and Explain Trade Decision**

The bot calculates all the information about each trade and displays all this in the trade history page of the platform

## Top-Down View

This top-down view shows the architecture of the trading platform, that’s divided into three main sections: Main Trading Bot, Log In and Menu, and Trade Feedback. The main trading bot is the part where the actual trading happens that is caused by the analysis of technical indicators (RSI, MACD, Supertrend). The data for this analysis is provided by the Binance API. The Log In and Menu section deals with authenticating user details entered form login or registration. It also handles the main UI of the main pages such as the home page, profile page, etc. The Trade History section (Trade Feedback) deals with showing to the user the information about each trade from which the user can then learn why and how the trade happens. Then all this is stored in the SQL database.



## A screenshot of a login form AI-generated content may be incorrect.GUI Interface

A screen shot of a chart

AI-generated content may be incorrect.This is the page where users can login/register

Here users can select their preference between live and historical trading

A screenshot of a computer

AI-generated content may be incorrect.

This is the home page, and users can view their profile stats, update their account, and activate the bot

A screenshot of a computer

AI-generated content may be incorrect.

In the account page, users can update their balance

A graph with a line going up

AI-generated content may be incorrect.In the profile page, user can view their equity curve and their account balance

A white rectangular grid with black text

AI-generated content may be incorrect.In trade history, information about each trade is displayed

A screenshot of a graph

AI-generated content may be incorrect.A screenshot of a screen

AI-generated content may be incorrect.In the risk page, users can select what risk level they want to trade at. As the risk increases, the indicators become more sensitive, which means that trades potentially could be executed faster (at the nearest buy and sell signal), and amount of money invested would also increase

In the workstation page, users can see what the bot is doing currently, and view the progress

Clicking the BTB logo will take you to the home page

* Under profile you can check your account details, balance, and the equity curve.
* Under trade history, you can check past trades that include information such as balance before and after, reasons, and buy/sell price.
* Clicking the Activate Bot button will take the user to the Risk page, where they can select the risk tolerance level they want
* Then the program will take the user to the Workstation page, where they can monitor what the bot is doing.
* Once a trade is completed (bought and sold), the user will be returned to the home screen.

## Data Structures

Note: risk\_thresholds and bitcoin\_config are standalone entities and are not related to the other table.

**Database Normalization**

The database schema for the trading bot has been normazlied up to the Second Normal Form (2NF).

* First Normal Form (1NF) - All fields in each table store atomic values, with no repeating columns or multi-valued fields in any table.
* Second Normal Form (2NF) – All linked tables have a single column primary key and non-key attributes depend on that primary key

The schema is not normalized up to the Third Normal Form (3NF) as it has some transitive dependencies. Examples include (in trade\_history) profit\_loss depending on sell\_price and buy\_price, and balance\_after depending on balance\_before and profit\_loss.

To full achieve 3NF, the trade\_history table should have non\_key attributes that don’t depend on other non-key attributes. So all non-key attributes that do depend on other non-key attributes should be moved to a different table. However, all these values were put in the same table for ease of displaying the trade history to the user one time at a single place instead of displaying multiple tables.

If this project were to be visited again in the future, the databases should be normalized upto 3NF to avoid data redundancy, maintain data integrity, and allow for proper scalabiltiy and organization.

### User Profile

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| user\_id | Integer | Primary Key |
| username | String | Name of user |
| hashed\_password | String | Hashed password |
| account\_balance | float | User’s balance in the system |

### Trade History

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| trade\_id | Integer | Primary Key |
| user\_id | Integer | Foreign Key |
| risk\_level | Float | The amount money being traded |
| buying\_time | Datetime | Time of buying |
| selling\_time | Datetime | Time of selling |
| buy\_price | Float | The price of Bitcoin at which a buy order was executed |
| sell\_price | Float | The price of Bitcoin at which a sell order was executed |
| profit\_loss | Float | The profit or loss made from the trade |
| balance\_before | Float | The account balance before the trade |
| balance\_after | Float | The account balance after the trade |
| rsi\_buy | Float | RSI (Relative Strength Index) value at the time of buying |
| macd\_buy | Float | MACD (Moving Average Convergence Divergence) value at buying |
| supertrend\_buy | Float | Supertrend indicator value at buying |
| rsi\_sell | Float | RSI value at the time of selling |
| macd\_sell | Float | MACD value at selling |
| supertrend\_sell | Float | Supertrend indicator value at selling |
| rsi\_flag\_buy | Integer | RSI buy signal flag |
| macd\_flag\_buy | Integer | MACD buy signal flag |
| supertrend\_flag\_buy | Integer | Supertrend buy signal flag |
| rsi\_flag\_sell | Integer | RSI sell signal flag |
| macd\_flag\_sell | Integer | MACD sell signal flag |
| supertrend\_flag\_sell | Integer | Supertrend sell signal flag |

### Application Logs

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| user\_id | Integer | Foreign Key |
| event\_type | String | Type of action  LOGIN  LOGOUT  CHANGE\_PASSWORD |
| event\_time | Datetime | Time of the event |
| comments | String | Extra information will be stored here |

### Risk Thresholds

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| risk\_level | Integer | Primary Key |
| invest\_thres | Integer | Percentage of User’s Balance to be used for investment |
| rsi\_oversold | Integer | RSI Oversold threshold |
| rsi\_overbought | Integer | RSI Overbought threshold |
| macd\_fast\_ema | Integer | MACD Fast EMA (short-term EMA) |
| macd\_slow\_ema | Integer | MACD Slow EMA (long-term EMA) |
| macd\_signal\_ema | Integer | MACD Signal EMA |
| supertrend\_atr\_period | Integer | ATR Period for Supertrend |
| supertrend\_multiplier | Float | Supertrend Multiplier |

### Configuration Table

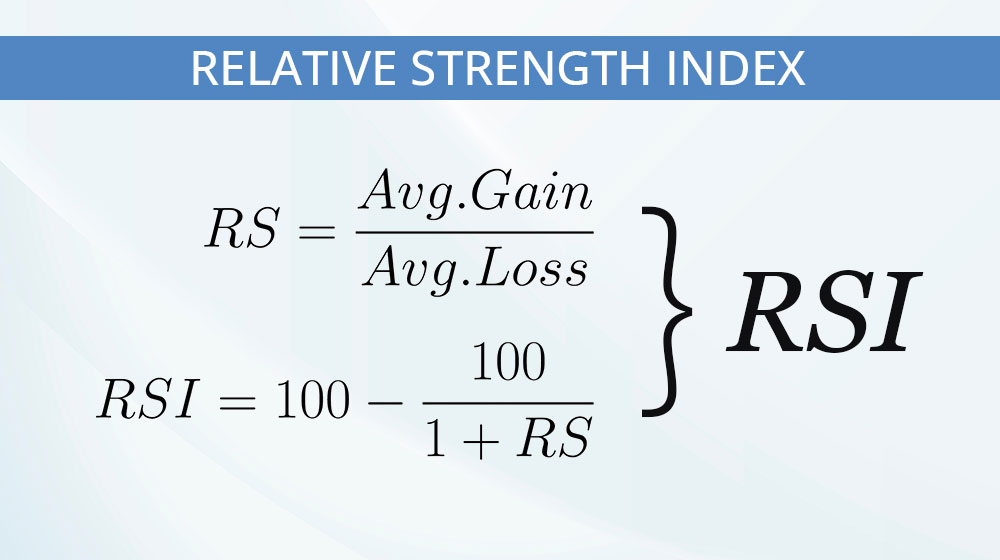
|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| config\_code | String | The values are: API\_KEY  API\_SECRET  SALT\_KEY |
| config\_value | String | Respective values |

### Account History Table

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| user\_id | Integer | Foreign Key |
| account\_date | Datetime | Date when the trade was made |
| account\_balance | Float | Account balance after the trade is finished |

## Pseudocode Algorithms

### RSI Pseudocode

* **Initialize lists for gains & losses**
* **Calculate price changes and classify them as gains or losses**
* **Compute initial average gain/loss**
* **Iterate through the rest of the data to calculate RSI using smoothing**
* **Classify RSI as Overbought (>70), Oversold (<30), or Neutral**
* **Return RSI values along with their status**

FUNCTION calculate\_rsi(prices, period):

SET gains = []

SET losses = []

// Calculate initial gains and losses

FOR i FROM 1 TO period:

SET change = prices[i] - prices[i - 1]

IF change > 0 THEN

APPEND change TO gains

APPEND 0 TO losses

ELSE:

APPEND absolute(change) TO losses

APPEND 0 TO gains

// Compute initial average gain and loss

SET avg\_gain = SUM(gains) / period

SET avg\_loss = SUM(losses) / period

// Step 3: Compute RSI for yesterday’s price

SET change = prices[i + 1] - prices[i]

SET gain = MAX(change, 0)

SET loss = ABSOLUTE(MIN(change, 0))

// Calculate smoothed averages

avg\_gain = ((avg\_gain \* (period - 1)) + gain) / period

avg\_loss = ((avg\_loss \* (period - 1)) + loss) / period

// Step 5: Compute RSI

IF avg\_loss == 0 THEN

SET rs = 100 // Prevent division by zero

ELSE:

SET rs = avg\_gain / avg\_loss

SET rsi = 100 - (100 / (1 + rs))

// Determine RSI status

SET status = "Neutral"

IF rsi > 70 THEN

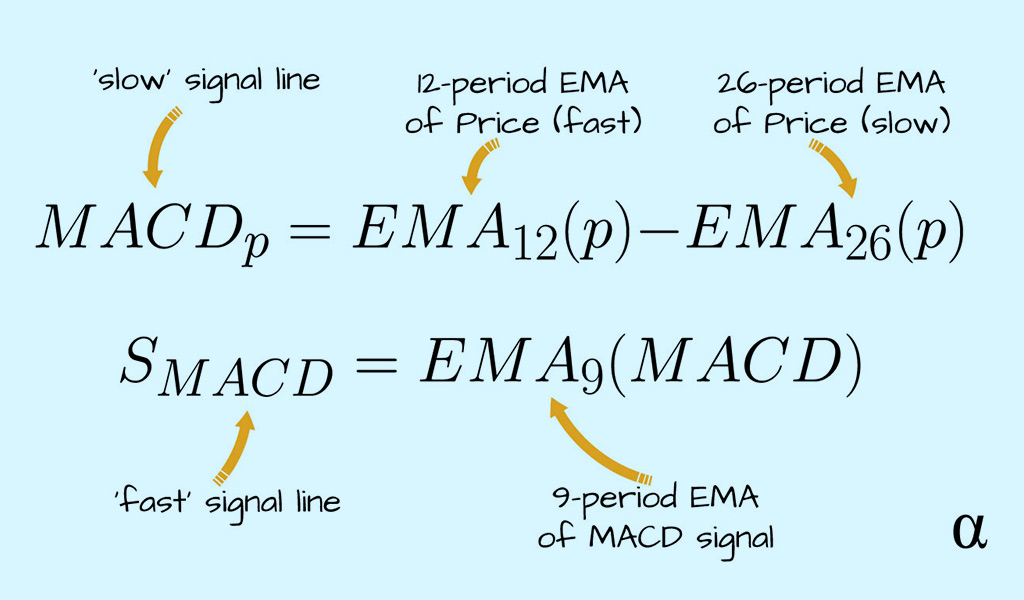
status = "Overbought"

ELSE IF rsi < 30 THEN

status = "Oversold

RETURN rsi

### MACD Pseudocode

* **Computes the Exponential Moving Averages (EMA) for 12 and 26 periods.**
* **Calculates the MACD line by subtracting EMA-26 from EMA-12.**
* **Generates the Signal Line using a 9-period EMA of the MACD line.**
* **Identifies Buy/Sell signals based on MACD crossover with the Signal Line.**
* **Finds the signal on the last day and uses that signal.**

// Example Bitcoin closing prices for 30 days

SET closing\_prices = [43150, 43200, 43300, ..., 44900]

// Function to calculate the Exponential Moving Average (EMA)

FUNCTION calculate\_ema(prices, period):

SET multiplier = 2 / (period + 1)

SET ema = []

SET sma = SUM(prices[0:period]) / period // Compute Simple Moving Average for the first EMA

APPEND sma TO ema

FOR i FROM period TO LENGTH(prices) - 1:

SET new\_ema = (prices[i] - ema[LAST]) \* multiplier + ema[LAST]

APPEND new\_ema TO ema

RETURN ema

// Function to calculate MACD Line and Signal Line

FUNCTION calculate\_macd\_and\_signal(prices):

SET ema\_12 = calculate\_ema(prices, 12)

SET ema\_26 = calculate\_ema(prices, 26)

// Align EMAs with original data

SET ema\_12 = [NULL] \* (12 - 1) + ema\_12

SET ema\_26 = [NULL] \* (26 - 1) + ema\_26

SET macd\_line = []

FOR i FROM 0 TO LENGTH(prices) - 1:

IF ema\_12[i] IS NOT NULL AND ema\_26[i] IS NOT NULL THEN

APPEND (ema\_12[i] - ema\_26[i]) TO macd\_line

ELSE

APPEND NULL TO macd\_line

// Remove NULL values for EMA calculation

SET valid\_macd = FILTER macd\_line TO REMOVE NULL

// Calculate Signal Line using EMA (9-period)

SET signal\_line = calculate\_ema(valid\_macd, 9)

SET signal\_line = [NULL] \* (26 + 9 - 2) + signal\_line

RETURN macd\_line, signal\_line

// Function to generate Buy/Sell signals based on MACD crossovers

FUNCTION generate\_signals(macd\_line, signal\_line):

SET signals = []

FOR i FROM 1 TO LENGTH(macd\_line) - 1:

IF macd\_line[i] > signal\_line[i] AND macd\_line[i-1] <= signal\_line[i-1] THEN

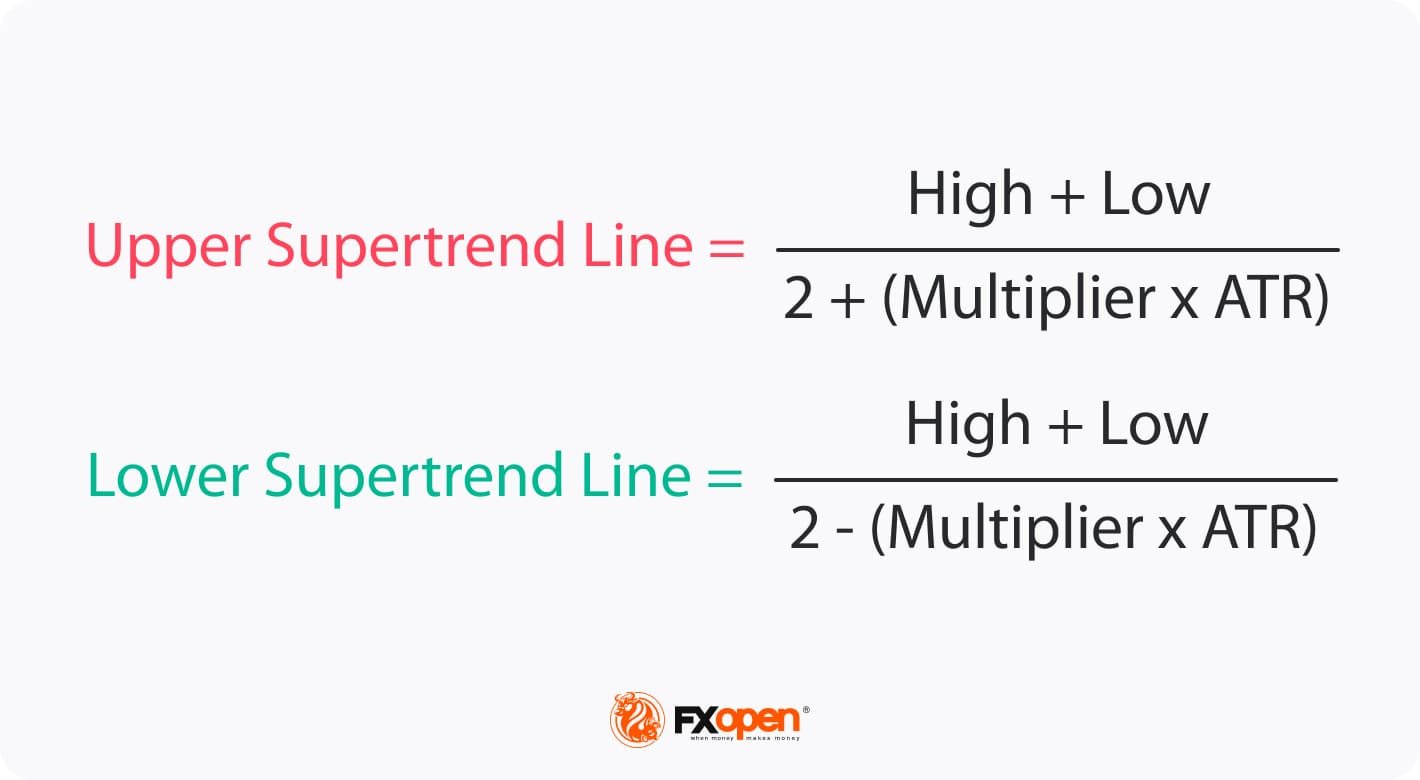
APPEND "Buy" TO signals

ELSE IF macd\_line[i] < signal\_line[i] AND macd\_line[i-1] >= signal\_line[i-1] THEN

APPEND "Sell" TO signals

RETURN signals[LAST]

### Supertrend Pseudocode

* Calculate ATR
* Find the True Range (TR) values.
* Compute ATR as the Simple Moving Average of TR over period days.
* Determine Upper & Lower Bands.
* Identify if the market is in an Uptrend or Downtrend.
* Store the Supertrend value based on the trend direction.
* Buy when price crosses above Supertrend.
* Sell when price crosses below Supertrend.

FUNCTION CalculateATR(highs, lows, closes, period):

TR = [] // List to store True Range values

ATR = [] // List to store ATR values

FOR i FROM 0 TO LENGTH(highs) - 1 DO:

IF i == 0 THEN:

TR[i] = highs[i] - lows[i] // First TR is simply High - Low

ELSE:

TR[i] = MAX(

highs[i] - lows[i],

ABS(highs[i] - closes[i-1]),

ABS(lows[i] - closes[i-1])

)

// Compute ATR as the Simple Moving Average of TR

FOR i FROM 0 TO LENGTH(TR) - 1 DO:

IF i < period THEN:

ATR[i] = NULL // Not enough data for ATR

ELSE:

ATR[i] = SUM(TR[i-period+1 TO i]) / period

RETURN ATR

END FUNCTION

FUNCTION CalculateSupertrend(highs, lows, closes, period, multiplier):

ATR = CalculateATR(highs, lows, closes, period)

Supertrend = LIST OF NULL VALUES WITH LENGTH(highs)

Signals = LIST OF NULL VALUES WITH LENGTH(highs)

FOR i FROM 0 TO LENGTH(highs) - 1 DO:

IF ATR[i] IS NULL THEN:

CONTINUE // Skip until ATR is available

// Calculate Upper and Lower Bands

UpperBand = (highs[i] + lows[i]) / 2 + (multiplier \* ATR[i])

LowerBand = (highs[i] + lows[i]) / 2 - (multiplier \* ATR[i])

// Determine Trend Direction

IF i == 0 THEN:

Supertrend[i] = LowerBand // Assume initial trend is Uptrend

TrendUp = TRUE

ELSE:

IF closes[i] > Supertrend[i-1] THEN:

TrendUp = TRUE

ELSE IF closes[i] < Supertrend[i-1] THEN:

TrendUp = FALSE

IF TrendUp THEN:

Supertrend[i] = MAX(LowerBand, Supertrend[i-1])

ELSE:

Supertrend[i] = MIN(UpperBand, Supertrend[i-1])

// Generate Buy/Sell Signals

IF i > 0 THEN:

IF closes[i] > Supertrend[i] AND closes[i-1] <= Supertrend[i-1] THEN:

Signals[i] = "Buy"

ELSE IF closes[i] < Supertrend[i] AND closes[i-1] >= Supertrend[i-1] THEN:

Signals[i] = "Sell"

RETURN Supertrend, Signals

END FUNCTION

### Password Hashing Pseudocode

This hashing is required in order to keep the passwords secure from unauthorised access. No one else other than the admin and the user can access these passwords. This algorithm will ensure safety and security.

Explanation:

* Takes two inputs that is the entered password and the salt key and concatenates them
* Looping through each character it converts it left shifts and multiplies by 8, right shifts and divides by 4, and adds the ASCII value of the current character
* Then limits to 32 bits

SET salt = "c6c1af8276b2adc81"

FUNCTION custom\_hash(password,salt)

data = password+salt

SET hash\_value = 0

FOR char in data DO:

SET hash\_value = (hash\_value LEFTSHIFT 3)

+ (hash\_value RIGHTSHIFT 2)

+ ASCII(char)

Hash\_value = hash\_value MOD 2^32

return hex(hash\_value)

Limitations: This algorithm is a simple hashing program that has a hardcoded salt key instead of generating a random key for each uesr and relies on simple bit-shifting operations to produce the hashed password. Therfore this lacks the security that could resist brute force attacks due its predictable mathematical operations that could easily be reverse engineered.

### Buying and Selling Pseudocode

FUNCTION buy\_btc()

// Execute buy if at least 2 indicators are positive

IF total\_flag >= 2 THEN

SET balance = get\_account\_balance(username)

// Calculate trading amount based on investment threshold

percentage

SET trading\_amount = balance \* (invest\_threshold / 100)

// Get Bitcoin price for the given date

SET price = get\_btc\_price()

IF price is a valid number THEN

SET buy\_btc\_price = price

END IF

END IF

END FUNCTION

FUNCTION sell\_btc()

// Check if there was a previous buy transaction

IF buy\_btc\_price <= 0 THEN

PRINT "No buy transaction found"

RETURN

END IF

// Execute sell if at least 2 indicators are negative

IF combined\_flag >= 2 THEN

SET price = get\_btc\_price()

IF price is a valid number THEN

SET sell\_btc\_price = price

// Calculate profit or loss

SET percentage\_difference = (sell\_btc\_price - buy\_btc\_price) / buy\_btc\_price

SET multiplier\_value = 1 + percentage\_difference

SET new\_trading\_amount = multiplier\_value \* trading\_amount

SET profit\_loss = new\_trading\_amount - trading\_amount

END IF

END IF

END FUNCTION

Explanation: After getting the flags from all 3 indicators, it checks if majority come true. Then it buys at the market price at that time. Then if majority come false it sells. Then it calculates the profit loss using the percentage differences between the sell and buy price.

## Testing Plan

### Unit Testing and Parameter Optimization

* Each indicator will be tested individually, and will be experimented with different parameters. This could include changing the risk levels, investment thresholds, multiplier values, etc.
* Check that the indicator graphs produced match with real indicator graphs on other trading platforms.
* Should ensure that the buy/sell logic works as intended.

### Integrated Testing

* Ensure that all pages work as one system, and check for seamless navigation between each page.
* Check that data is transferred appropriately between the pages and required data is updated in the databases.
* Check that each action is logged and stored in the correct format

### Error Handling Tests

* Check that all errors are caught and dealt with so that the system still functions upon any issues
* Ensure solutions for network issues, API errors, input validation, database errors, graph exceptions, etc.
* Check that the system logs and displays appropriate error messgaes and does not crash

### Trading Tests

* Test the bot under various conditions
* Ensure that bot makes profits under bullish trends
* Ensure that bot does does not interact with the market under bearish trends

# Technical Solution

## Python Files

Skills Table for Technical Solution

|  |  |  |
| --- | --- | --- |
| **Skill** | **Python File** | **Page Numbers** |
| **Single-dimensional arrays** | ui.py | 94 |
| **Single table database** | database.py | 117 |
| **Simple mathematical calculations** | bot\_indicators.py | 97 |
| **Dictionaries** | ui.py, api.py | 57 |
| **File(s) organised for sequential access** | All |  |
| **Simple scientific/mathematical/business model** | bot\_indicators.py | 70 |
| **Simple OOP model** | All |  |
| **Simple client-server model** | api.py | 111 |
| **Simple user-defined algorithms** | bot\_indicators.py | 70 |
| **Generation of objects based on simple OOP model** | All |  |
| **Calling Web service APIs and parsing JSON/XML** | api.py | 111 |
| **Complex data model in database** |  | Check 2.7 data structures and 3.2 Databases |
| **Hashing** | auth.py | 108 |
| **Complex scientific/mathematical/business model** | bot\_indicators.py | 97, but whole of bot\_indicators |
| **Complex user-defined use of OOP model** | ui.py | 57 |
| **Cross-table parameterised SQL** | database.py | Check 2.7 data structures and 3.2 Databases |
| **Aggregate SQL functions** | database.py | 119 |
| **User/CASE-generated DDL script** | database.py | 117 |
| **List operations** | bot\_indicators.py | 94 |
| **Complex user-defined algorithms** | bot\_indicators.py | 94, (risks applied) |
| **Dynamic generation of objects based on complex OOP model** | ui.py | 57 |
| **Calling parameterised Web service APIs** | api.py | 111 |

* **Complexity**: Skills like Complex user-defined use of OOP model are seen from inheritance and composition, while simpler skills like single-dimensional arrays are present.

### main.py

from ui import LoginPage

if \_\_name\_\_ == "\_\_main\_\_":

# Create an instance of the LoginPage class and start its main UI event loop

LoginPage().mainloop()

**Purpose**:

This code initializes and displays the login interface when the script is executed directly. It imports the LoginPage class from the ui module and creates an instance of it, launching the login window for user interaction.

### ui.py

import tkinter as tk

from tkinter import ttk

from tkinter import messagebox

from PIL import Image, ImageTk

from auth import AuthManager

from database import DatabaseManager

import re

from btccschart import BTCCandlestickChart

from bot\_indicators import BotIndicators

from activate\_bot import ActivateBot

from matplotlib.figure import Figure

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

import pandas as pd

import matplotlib.dates as mdates

from api import BinanceAPI

from datetime import datetime, timedelta

from utils import Utils

class DisplayPictures:

@staticmethod

def load\_image(page\_obj, username, page\_heading):

"""Function to load and resize images for branding and user profile."""

img = Image.open("assets/btblogo.png") # Load logo image

img = img.resize((125, 150), Image.LANCZOS) # Resize image

page\_obj.btblogo = ImageTk.PhotoImage(img)

# Create button with image that navigates home

tk.Button(page\_obj, image=page\_obj.btblogo, command=page\_obj.go\_homepage, borderwidth=0, highlightthickness=0, highlightbackground="black", activebackground="black").place(anchor="nw")

img = Image.open("assets/profile.png") # Load profile picture

img = img.resize((40, 40), Image.LANCZOS)

page\_obj.profilepic = ImageTk.PhotoImage(img)

# Show username with profile image beside it

tk.Label(page\_obj, text=username, image=page\_obj.profilepic, compound="left", font=("Arial", 30)).place(relx=1, rely=0, anchor="ne")

# Display page heading

tk.Label(page\_obj, text=page\_heading, font=("Arial", 60)).place(relx=0.5, rely=0.1, anchor="center")

class BasePage(tk.Tk):

def \_\_init\_\_(self, title):

super().\_\_init\_\_()

self.title(title) # Set window title

self.attributes('-fullscreen', True) # Enable fullscreen mode

def navigate\_to(self, page\_class, \*args):

self.destroy() # Close current page

page\_class(\*args).mainloop() # Load new page

class LoginPage(BasePage):

def \_\_init\_\_(self):

super().\_\_init\_\_("Login Page")

self.auth = AuthManager() # Handles authentication logic

btn\_style = {"font": ("Arial", 30), "width": 12, "height": 2} # Reusable style dictionary

# Username input field

tk.Label(self, text="Username:", \*\*btn\_style).pack(pady=5)

self.username\_entry = tk.Entry(self, bd=5, font=("Arial", 30), width=20)

self.username\_entry.pack(pady=5)

#self.username\_entry.insert(0, "charlie1") # Pre-fill example username

tk.Button(self, text="?", font=("Arial", 15), command=self.show\_username\_rules).place(relx=0.35, rely=0.09)

# Password input field

tk.Label(self, text="Password:", \*\*btn\_style).pack(pady=5)

self.password\_entry = tk.Entry(self, show = "\*", bd=5, font=("Arial", 30), width=20)

self.password\_entry.pack(pady=5)

#self.password\_entry.insert(0, "Harper1#") # Pre-fill example password

tk.Button(self, text="?", font=("Arial", 15), command=self.show\_password\_rules).place(relx=0.35, rely=0.23)

# Login and Register buttons

tk.Button(self, text="Login", \*\*btn\_style, command=self.handle\_login).place(relx=0.5, rely=0.5, anchor="center")

tk.Button(self, text="Register", \*\*btn\_style, command=self.handle\_register).place(relx=0.5, rely=0.6, anchor="center")

self.status\_label = tk.Label(self, text="") # To display success/error messages

self.status\_label.pack(pady=5)

def show\_username\_rules(self):

# Show popup with username requirements

self.\_create\_popup("Username Rules", [

"• The first letter should be an alphabet",

"• Username should be 8-12 characters long",

"• No special characters allowed"

])

def show\_password\_rules(self):

# Show popup with password requirements

self.\_create\_popup("Password Rules", [

"• At least 1 capital letter",

"• At least 1 number",

"• At least 1 special character",

"• Minimum 8 characters and maximum 15 characters"

])

def \_create\_popup(self, title, rules):

# Creates a reusable popup window

popup = tk.Toplevel(self)

popup.title(title)

popup.geometry("400x200")

popup.transient(self)

popup.resizable(False, False)

tk.Label(popup, text=title, font=("Arial", 18, "bold")).pack(pady=10)

for rule in rules:

tk.Label(popup, text=rule, font=("Arial", 12)).pack(anchor="w", padx=20)

tk.Button(popup, text="Close", command=popup.destroy).pack(pady=10)

def handle\_login(self):

# Get values from entries and validate them

username = self.username\_entry.get()

password = self.password\_entry.get()

message = self.auth.validate\_login(username, password) # Call auth method

self.status\_label.config(text=message) # Show login result

# If successful, navigate to next page

if message == "Login Successful":

self.auth.close()

self.navigate\_to(PreferencePage, username)

def validate\_user(self, username: str, password: str) -> bool:

# Username validation logic

if not re.search(r"^[A-Za-z]", username):

messagebox.showerror("Error", "The first letter of username should be an alphabet")

return False

if len(username) < 8 or len(username) > 12:

messagebox.showerror("Error", "Username should be between 8 and 12 characters")

return False

if re.search(r"[!@#$%^&\*(),.?\"\':{}|<>]", username):

messagebox.showerror("Error", "Special characters are not allowed in the username")

return False

# Password validation logic

if not re.search(r"[A-Z]", password):

messagebox.showerror("Error", "The password should contain atleast 1 capital letter")

return False

if not re.search(r"[0-9]", password):

messagebox.showerror("Error", "The password should contain atleast 1 number")

return False

if not re.search(r"[!@#$%^&\*(),.?\"\':{}|<>]", password):

messagebox.showerror("Error", "The password should contain atleast 1 special character")

return False

if len(password) < 8 or len(password) > 15:

messagebox.showerror("Error", "The password should contain between 8 and 15 characters")

return False

return True

def handle\_register(self):

# Handles registration after validation

username = self.username\_entry.get()

password = self.password\_entry.get()

if self.validate\_user(username, password):

if self.auth.register\_user(username, password):

self.status\_label.config(text="Register Successful, Press login to continue")

else:

self.status\_label.config(text="This username already exists, please choose a different username")

else:

print("Invalid user")

def on\_close(self):

# Gracefully close DB connection and app window

self.auth.close()

self.destroy()

class PreferencePage(BasePage):

def \_\_init\_\_(self, username):

super().\_\_init\_\_("Account Page")

self.username = username

# Load header images and heading text

DisplayPictures.load\_image(self, username, "Select Trading Preference")

# Style config for buttons

self.btn\_style = {"font": ("Arial", 30), "width": 12, "height": 2, "fg": "black", "bg": "white", "borderwidth": 2, "relief": "ridge"}

self.selected\_button = None

# Preference buttons

self.btn1 = tk.Button(self, text="Live Trading", \*\*self.btn\_style, command=lambda: self.level\_clicked(self.btn1))

self.btn2 = tk.Button(self, text="Historical Trading", \*\*self.btn\_style, command=lambda: self.level\_clicked(self.btn2))

# Position buttons on screen

self.btn1.place(relx=0.40, rely=0.5, anchor="center")

self.btn2.place(relx=0.6, rely=0.5, anchor="center")

def level\_clicked(self, clicked\_button):

button\_text = clicked\_button.cget("text")

if self.selected\_button:

# Reset previously selected button

self.selected\_button.config(fg="black", relief="ridge")

# Highlight currently clicked button

clicked\_button.config(fg="green", relief="solid")

self.selected\_button = clicked\_button

# Set preference based on button text

self.trading\_preference = 1 if button\_text == "Live Trading" else 0

# Create confirm button dynamically once an option is selected

confirm\_button = tk.Button(

self, text="Confirm", font=("Arial", 30),

command=lambda: self.go\_homepage(),

width=12, height=2, highlightbackground="green", bd=5, relief="solid"

)

confirm\_button.place(relx=0.50, rely=0.8, anchor="center")

def go\_homepage(self):

self.db = DatabaseManager(self.trading\_preference) # Connect to DB with preference

self.navigate\_to(HomePage, self) # Navigate to home with full user\_obj

class AccountPage(BasePage):

def \_\_init\_\_(self, user\_obj):

super().\_\_init\_\_("Account Page")

self.user\_obj = user\_obj

# Load header images and heading

DisplayPictures.load\_image(self, user\_obj.username, "Balance Update")

# Input prompt for balance

tk.Label(self, text="How much money do you want to put in?", font=("Arial", 30)).place(relx=0.5, rely=0.4, anchor="center")

# Balance entry field

self.accountbalance\_entry = tk.Entry(self, bd=5, font=("Arial", 30), width=20)

self.accountbalance\_entry.place(relx=0.6, rely=0.5, anchor="center")

btn\_style = {"font": ("Arial", 30), "width": 12, "height": 2}

# Label beside the entry

tk.Label(self, text="Balance ($)", font=("Arial", 30), bd=5, relief="solid").place(relx=0.35, rely=0.5, anchor="center")

# Confirm button to update balance

tk.Button(self, text="Confirm", \*\*btn\_style, command=self.add\_balance).place(relx=0.5, rely=0.6, anchor="center")

def go\_homepage(self):

self.navigate\_to(HomePage, self.user\_obj)

self.destroy()

def add\_balance(self):

"""Calls DatabaseManager to update the database with balance."""

try:

balance = float(self.accountbalance\_entry.get())

success, new\_balance = self.user\_obj.db.update\_account\_balance(self.user\_obj.username, balance)

if success:

# Show confirmation

messagebox.showinfo("Success", f"Balance updated: ${new\_balance:.2f}")

self.navigate\_to(HomePage, self.user\_obj)

self.destroy()

else:

messagebox.showerror("Error", "Failed to update balance. Account Balance should be between 500$ and 30000$")

except ValueError:

messagebox.showerror("Error", "Invalid input! Please enter a valid number.")

def on\_close(self):

self.destroy()

class HomePage(BasePage):

def \_\_init\_\_(self, user\_obj):

super().\_\_init\_\_("Home Page")

self.user\_obj = user\_obj

# Load header assets and heading

DisplayPictures.load\_image(self, user\_obj.username, "Home")

btn\_style = {"font": ("Arial", 30), "width": 12, "height": 2}

# Home menu buttons

tk.Button(self, text="Profile", \*\*btn\_style, command=self.profile).place(relx=0.35, rely=0.5, anchor="center")

tk.Button(self, text="Add Money", \*\*btn\_style, command=self.add\_money).place(relx=0.5, rely=0.5, anchor="center")

tk.Button(self, text="Trade History", \*\*btn\_style, command=self.trade\_history).place(relx=0.65, rely=0.5, anchor="center")

# Emphasized Start Trading button

tk.Button(self, text="Start Trading", font=("Arial", 30), command=self.start\_trading,

width=12, height=2, highlightbackground="green", bd=5, relief="solid").place(relx=0.5, rely=0.8, anchor="center")

def go\_homepage(self):

self.navigate\_to(HomePage, self.user\_obj)

self.destroy()

def profile(self):

self.navigate\_to(ProfilePage, self.user\_obj)

self.destroy()

def add\_money(self):

self.navigate\_to(AccountPage, self.user\_obj)

self.destroy()

def trade\_history(self):

self.navigate\_to(TradeHistoryPage, self.user\_obj)

self.destroy()

def start\_trading(self):

# Navigate to correct page depending on preference

if self.user\_obj.trading\_preference == 0:

self.navigate\_to(DatePage, self.user\_obj)

else:

self.user\_obj.from\_date = Utils.get\_date()

self.user\_obj.to\_date = Utils.get\_date()

self.navigate\_to(RiskPage, self.user\_obj)

self.destroy()

class DatePage(BasePage):

def \_\_init\_\_(self, user\_obj):

super().\_\_init\_\_("Account Page")

self.user\_obj = user\_obj

# Load header section

DisplayPictures.load\_image(self, user\_obj.username, "Enter Date Range")

btn\_style = {"font": ("Arial", 30), "width": 12, "height": 2}

# Date input fields

tk.Label(self, text="From Date:", \*\*btn\_style).place(relx=0.3, rely=0.3)

self.from\_date\_entry = tk.Entry(self, bd=5, font=("Arial", 30), width=20)

self.from\_date\_entry.place(relx=0.3, rely=0.4)

self.from\_date\_entry.insert(0, "YYYY-MM-DD")

tk.Label(self, text="To Date:", \*\*btn\_style).place(relx=0.5, rely=0.3)

self.to\_date\_entry = tk.Entry(self, bd=5, font=("Arial", 30), width=20)

self.to\_date\_entry.place(relx=0.5, rely=0.4)

self.to\_date\_entry.insert(0, "YYYY-MM-DD")

# Confirm button

tk.Button(self, text="Confirm", \*\*btn\_style, command=self.handle\_date\_entry).place(relx=0.5, rely=0.8, anchor="center")

def handle\_date\_entry(self):

from\_date = self.from\_date\_entry.get()

to\_date = self.to\_date\_entry.get()

try:

from\_ts = datetime.strptime(from\_date, "%Y-%m-%d")

to\_ts = datetime.strptime(to\_date, "%Y-%m-%d")

print("diff", (to\_ts - from\_ts).days)

if from\_ts > to\_ts:

messagebox.showerror("Error","From date cannot be greater than To date")

elif from\_ts >= datetime.now() or to\_ts >= datetime.now():

messagebox.showerror("Error","From date cannot be greater than current date")

elif (to\_ts - from\_ts).days > 60:

messagebox.showerror("Error","Date range cannot be greater than 60 days")

elif from\_ts<datetime.strptime("2018-01-01", "%Y-%m-%d"):

messagebox.showerror("Error","From date cannot be less than 2018-01-01")

else:

# Assign valid inputs to user object

self.user\_obj.from\_date = from\_ts

self.user\_obj.to\_date = to\_ts

self.go\_riskpage()

except:

# Handle incorrect date format

messagebox.showerror("Error", "Invalid Input. Please enter in the required format")

def go\_homepage(self):

self.navigate\_to(HomePage, self.user\_obj)

self.destroy()

def go\_riskpage(self):

self.navigate\_to(RiskPage, self.user\_obj)

self.destroy()

class TradeHistoryPage(BasePage):

# Maps internal database column names to human-readable labels

title\_mapping = {

"trade\_id": "Trade ID",

"user\_id": "User ID",

"money\_in": "Investment Amount",

"risk\_level": "Risk Level",

"buying\_time": "Buy Time",

"selling\_time": "Sell Time",

"buy\_price": "Buy Price",

"sell\_price": "Sell Price",

"profit\_loss": "Profit/Loss",

"balance\_before": "Balance Before",

"balance\_after": "Balance After",

"rsi\_buy": "RSI ",

"macd\_buy": "MACD ",

"supertrend\_buy": "Supertrend ",

"rsi\_sell": "RSI ",

"macd\_sell": "MACD ",

"supertrend\_sell": "Supertrend ",

"rsi\_flag\_buy": "RSI Signal",

"macd\_flag\_buy": "MACD Signal",

"supertrend\_flag\_buy": "Supertrend Signal",

"rsi\_flag\_sell": "RSI Signal",

"macd\_flag\_sell": "MACD Signal",

"supertrend\_flag\_sell": "Supertrend Signal"

}

def \_\_init\_\_(self, user\_obj):

super().\_\_init\_\_("Trade History Page")

self.user\_obj = user\_obj

# Display user profile image

DisplayPictures.load\_image(self, user\_obj.username, "Trade History")

# Fetch user's trade history

trade\_df = self.user\_obj.db.get\_trade\_history(self.user\_obj.username)

if trade\_df is None:

print("No trade history available")

else:

self.display\_tradehistory(trade\_df)

def create\_table(self, parent, df, title):

# Create a table to display a DataFrame

frame = ttk.LabelFrame(parent, text=title)

frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=5)

tree = ttk.Treeview(frame)

tree["columns"] = list(df.columns)

tree.column("#0", width=0, stretch=tk.NO) # Hide index column

for col in df.columns:

width = len(col)

tree.column(col, anchor=tk.W, width=width)

tree.heading(col, text=col, anchor=tk.W)

for \_, row in df.iterrows():

tree.insert("", "end", values=list(row))

tree.pack(fill=tk.BOTH, expand=True)

def display\_tradehistory(self, df):

df = df.round(2)

# Create separate DataFrames for buy and sell transactions

df\_buy = df[['trade\_id', 'money\_in', 'risk\_level', 'buying\_time','buy\_price',

'balance\_before', 'rsi\_buy', 'macd\_buy','supertrend\_buy',

'rsi\_flag\_buy', 'macd\_flag\_buy', 'supertrend\_flag\_buy']].copy()

df\_sell = df[['trade\_id','money\_in', 'risk\_level', 'selling\_time', 'sell\_price',

'profit\_loss','balance\_after', 'rsi\_sell', 'macd\_sell', 'supertrend\_sell',

'rsi\_flag\_sell','macd\_flag\_sell', 'supertrend\_flag\_sell']].copy()

# Rename columns for better readability

df\_buy.rename(columns=self.title\_mapping, inplace=True)

df\_sell.rename(columns=self.title\_mapping, inplace=True)

# Container for both tables

root = ttk.Frame(self)

root.place(relx=0.0015, rely=0.15, relwidth=0.98, relheight=0.75)

self.create\_table(root, df\_buy, "Buy Transactions")

self.create\_table(root, df\_sell, "Sell Transactions")

def go\_homepage(self):

self.navigate\_to(HomePage, self.user\_obj)

self.destroy()

class ProfilePage(BasePage):

def \_\_init\_\_(self, user\_obj):

super().\_\_init\_\_("Profile Page")

self.user\_obj = user\_obj

# Load account history and balance

account\_valid, account\_df = self.user\_obj.db.get\_account\_history(self.user\_obj.username)

root = ttk.Frame(self)

root.place(relx=0.0015, rely=0.15, relwidth=0.5, relheight=0.7)

self.parent = root

DisplayPictures.load\_image(self, user\_obj.username, "Profile")

# Display current account balance

tk.Label(self, text="Account Balance", font=("Arial", 30)).place(relx=0.8, rely=0.4, anchor="center")

account\_balance = round(self.user\_obj.db.get\_account\_balance(self.user\_obj.username), 2)

tk.Label(self, text=f"{account\_balance} $", font=("Arial", 30)).place(relx=0.8, rely=0.5, anchor="center")

# Plot equity curve if available

if account\_valid:

self.plot\_equity\_curve(account\_df)

else:

print("No history available")

def go\_homepage(self):

self.navigate\_to(HomePage, self.user\_obj)

self.destroy()

def plot\_equity\_curve(self, account\_df):

fig = Figure(figsize=(15, 10))

ax1 = fig.add\_subplot(111)

# Format and group data by date

account\_df["account\_date"] = pd.to\_datetime(account\_df["account\_date"])

account\_df = account\_df.sort\_values("account\_date").groupby(account\_df["account\_date"].dt.date).last()

account\_df.index = pd.to\_datetime(account\_df.index)

# Plot account balance over time

ax1.plot(account\_df.index, account\_df["account\_balance"], color="blue")

ax1.set\_title("Equity Curve")

ax1.set\_ylabel("Account Balance ($)")

# Format x-axis with readable dates

ax1.xaxis.set\_major\_formatter(mdates.DateFormatter("%d-%m-%Y"))

ax1.set\_xticks(account\_df.index)

for label in ax1.get\_xticklabels():

label.set\_rotation(45)

label.set\_horizontalalignment("right")

# Render the plot in the UI

self.canvas = FigureCanvasTkAgg(fig, master=self.parent)

self.canvas.draw()

self.canvas.get\_tk\_widget().pack(fill=tk.BOTH, expand=True)

class BotIndicatorsPage(BasePage):

def \_\_init\_\_(self, user\_obj):

super().\_\_init\_\_("Bot Indicators Page")

self.user\_obj = user\_obj

print("Bot Indicators Page", self.user\_obj.thresholds)

# Load profile image and set up chart canvas

DisplayPictures.load\_image(self, user\_obj.username,"Bot Indicators")

self.chart\_frame = tk.Frame(self)

self.chart\_frame.place(relx=0.5, rely=0.55, anchor="center", width=1350, height=800)

# Initialize indicators and chart

self.bi = BotIndicators(self.user\_obj)

self.chart = BTCCandlestickChart(self.chart\_frame, self.bi)

self.ab = ActivateBot(user\_obj, self, self.bi)

# Handle date cutoff if using live trading

to\_date = self.bi.to\_date

if self.user\_obj.trading\_preference == 1:

to\_date = Utils.get\_date()

# Calculate all indicators

self.bi.calculate\_macd(to\_date)

self.bi.calculate\_supertrend(to\_date)

self.bi.calculate\_historical\_rsi(to\_date)

# Fetch and plot data

self.chart.get\_ohlc\_data(self.user\_obj.trading\_preference, self.user\_obj.from\_date, self.user\_obj.to\_date)

print("historical rsi done")

self.chart.plot\_indicator\_graphs(self.bi)

def go\_homepage(self):

self.navigate\_to(HomePage, self.user\_obj)

class RiskPage(BasePage):

def \_\_init\_\_(self, user\_obj):

super().\_\_init\_\_("Risk Page")

self.user\_obj = user\_obj

print("risk page ", self.user\_obj.trading\_preference)

DisplayPictures.load\_image(self, user\_obj.username,"Risk Levels")

self.btn\_style = {"font": ("Arial", 30), "width": 12, "height": 2, "fg": "black", "bg": "white", "borderwidth": 2, "relief": "ridge"}

self.selected\_button = None

tk.Label(self, text="Note: Risk Increases from 1 to 3", font=("Arial", 50)).place(relx=0.5, rely=0.3, anchor="center")

# Create and place risk level buttons

self.btn1 = tk.Button(self, text="Level 1", \*\*self.btn\_style, command=lambda: self.level\_clicked(self.btn1))

self.btn2 = tk.Button(self, text="Level 2", \*\*self.btn\_style, command=lambda: self.level\_clicked(self.btn2))

self.btn3 = tk.Button(self, text="Level 3", \*\*self.btn\_style, command=lambda: self.level\_clicked(self.btn3))

self.btn1.place(relx=0.35, rely=0.5, anchor="center")

self.btn2.place(relx=0.5, rely=0.5, anchor="center")

self.btn3.place(relx=0.65, rely=0.5, anchor="center")

def level\_clicked(self, clicked\_button):

# Visual feedback for selected level

if self.selected\_button:

self.selected\_button.config(fg="black", relief="ridge")

clicked\_button.config(fg="green", relief="solid")

self.selected\_button = clicked\_button

self.selected\_level = int(clicked\_button.cget("text")[-1])

print("risk level:", self.selected\_level)

# Place activation buttons

tk.Button(self, text="Activate Bot", font=("Arial", 30), command=self.apply\_risk\_thresholds,

width=12, height=2, highlightbackground="green", bd=5, relief="solid").place(relx=0.40, rely=0.8, anchor="center")

tk.Button(self, text="Indicator Graphs", font=("Arial", 30), command=self.bot\_indicators,

width=12, height=2, highlightbackground="green", bd=5, relief="solid").place(relx=0.6, rely=0.8, anchor="center")

def apply\_risk\_thresholds(self):

# Set selected risk level and fetch its thresholds

self.user\_obj.risk\_level = self.selected\_level

self.user\_obj.thresholds = self.user\_obj.db.get\_risk\_thresholds(self.user\_obj.risk\_level)

print(self.user\_obj.thresholds)

account\_balance = round(self.user\_obj.db.get\_account\_balance(self.user\_obj.username), 2)

if account\_balance < 500:

messagebox.showerror("Error", "Account Balance should be greater than 500$")

return

self.navigate\_to(WorkStationPage, self.user\_obj)

self.destroy()

def bot\_indicators(self):

# Load indicator view based on selected risk

self.user\_obj.risk\_level = self.selected\_level

self.user\_obj.thresholds = self.user\_obj.db.get\_risk\_thresholds(self.user\_obj.risk\_level)

self.navigate\_to(BotIndicatorsPage, self.user\_obj)

self.destroy()

def go\_homepage(self):

self.navigate\_to(HomePage, self.user\_obj)

self.destroy()

class WorkStationPage(BasePage):

"""

WorkStationPage represents the main trading interface where bot indicators

are calculated, trades are executed, and results are visualized on a candlestick chart.

"""

def \_\_init\_\_(self, user\_obj):

"""

Initializes the WorkStationPage.

Args:

user\_obj: The user object containing preferences, thresholds, and trading history.

"""

super().\_\_init\_\_("WorkStation Page")

self.user\_obj = user\_obj

print("workstation page" , self.user\_obj.thresholds)

# Load user profile picture for display

DisplayPictures.load\_image(self, user\_obj.username, "WorkStation")

# Create a frame for candlestick chart

self.chart\_frame = tk.Frame(self)

self.chart\_frame.place(relx=0.3, rely=0.55, anchor="center", width=1200, height=800)

self.bi = BotIndicators(user\_obj) # Initialize trading indicators

self.chart = BTCCandlestickChart(self.chart\_frame, self.bi) # Candlestick chart instance

self.ab = ActivateBot(user\_obj, self, self.bi) # Trade execution logic

self.from\_date = self.bi.from\_date # Start date

self.to\_date = self.bi.to\_date # End date

self.running\_buy = True # Indicates whether buy loop is active

self.after(1000, self.run\_buy\_loop) # Start buy loop

# Load historical data for visualization

self.chart.get\_ohlc\_data(self.user\_obj.trading\_preference, self.user\_obj.from\_date, self.user\_obj.to\_date)

def run\_buy\_loop(self):

"""

Executes the buying logic in a loop, based on indicator flags and user preferences.

"""

if self.bi.continuous\_trade:

if not self.running\_buy:

return

else:

if self.from\_date > self.to\_date or not self.running\_buy:

return # Exit loop if date range ends or loop is inactive

print('buy activated for this date: ', self.from\_date)

print("btc value: ", BinanceAPI.get\_price\_on\_day(self.from\_date))

# Calculate trading indicators for the current date

self.bi.calculate\_rsi(self.from\_date)

self.bi.calculate\_macd(self.from\_date)

self.bi.calculate\_supertrend(self.from\_date)

# Update OHLC chart data for fixed date trading

if self.user\_obj.trading\_preference == 1:

self.chart.get\_ohlc\_data(self.user\_obj.trading\_preference, self.user\_obj.from\_date, self.user\_obj.to\_date)

print("rsi flag = ", self.bi.rsi\_flag)

print("macd flag = ", self.bi.macd\_flag)

print("supertrend flag = ", self.bi.st\_flag)

self.ab.buy\_btc(self.from\_date) # Attempt to buy BTC

self.display\_indicators() # Display updated indicators

# If a trade has been executed, switch to sell loop

if self.ab.buy\_btc\_price != 0:

self.running\_buy = False

self.after(1000, self.run\_sell\_loop)

return

# Update the date and re-run the loop

if self.bi.continuous\_trade:

self.from\_date = Utils.get\_date()

self.after(60000, self.run\_buy\_loop)

else:

self.from\_date += timedelta(days=1)

self.after(1000, self.run\_buy\_loop)

def run\_display\_loop(self):

"""

Runs a loop to continuously update and display indicators.

"""

self.display\_indicators()

self.after(2000, self.run\_display\_loop)

def run\_sell\_loop(self):

"""

Executes the selling logic in a loop, based on indicator flags and user preferences.

"""

if not self.bi.continuous\_trade and self.from\_date > self.to\_date:

return # Exit loop if date range ends

print('sell activated for this date: ', self.from\_date)

print("btc value: ", BinanceAPI.get\_price\_on\_day(self.from\_date))

# Calculate trading indicators

self.bi.calculate\_rsi(self.from\_date)

self.bi.calculate\_macd(self.from\_date)

self.bi.calculate\_supertrend(self.from\_date)

# Update OHLC chart data if in fixed mode

if self.user\_obj.trading\_preference == 1:

self.chart.get\_ohlc\_data(self.user\_obj.trading\_preference, self.user\_obj.from\_date, self.user\_obj.to\_date)

print("rsi flag = ", self.bi.rsi\_flag)

print("macd flag = ", self.bi.macd\_flag)

print("supertrend flag = ", self.bi.st\_flag)

self.ab.sell\_btc(self.from\_date) # Attempt to sell BTC

self.display\_indicators() # Display updated indicators

if self.ab.sell\_btc\_price != 0:

self.run\_display\_loop() # Start visual update loop

return

if self.bi.continuous\_trade:

self.from\_date = Utils.get\_date()

self.after(60000, self.run\_sell\_loop)

else:

self.from\_date += timedelta(days=1)

self.after(1000, self.run\_sell\_loop)

def display\_indicators(self):

"""

Displays real-time and historical indicators, trading status, and performance metrics.

Also updates the candlestick chart and flag icons.

"""

price = BinanceAPI.get\_ticker\_price()

if isinstance(price, float):

price\_text = f"Bitcoin Price: ${price:.2f}"

else:

price\_text = price # Show error message

# Create or update price label

if not hasattr(self, "price\_label"):

self.price\_label = tk.Label(self, text=price\_text, font=("Arial", 35), fg="black")

self.price\_label.place(relx=0.8, rely=0.2, anchor="center")

else:

self.price\_label.config(text=price\_text)

# Create and draw candlestick chart

fig = Figure(figsize=(16, 10))

ax = fig.add\_subplot(111)

self.chart.canvas = FigureCanvasTkAgg(fig, master=self.chart.parent)

self.chart.plot\_candlestick\_chart(fig, ax)

self.chart.canvas.draw()

self.chart.canvas.get\_tk\_widget().pack(fill=tk.BOTH, expand=True)

# Load status icons

self.truepic = self.load\_image("assets/true.png", (30, 30))

self.falsepic = self.load\_image("assets/false.png", (30, 30))

# Display current mode and flags

tk.Label(self, text=f"Date: {self.from\_date}", font=("Arial", 25), fg="black").place(relx=0.8, rely=0.16, anchor="center")

if self.running\_buy:

tk.Label(self, text="BUY Mode", font=("Arial", 25), fg="black").place(relx=0.8, rely=0.25, anchor="center")

rsi\_flag = self.bi.rsi\_flag == 1

macd\_flag = self.bi.macd\_flag == 1

st\_flag = self.bi.st\_flag == 1

else:

tk.Label(self, text="Sell Mode", font=("Arial", 25), fg="black").place(relx=0.8, rely=0.25, anchor="center")

rsi\_flag = self.bi.rsi\_flag == -1

macd\_flag = self.bi.macd\_flag == -1

st\_flag = self.bi.st\_flag == -1

# Show indicators with icons

tk.Label(self, text="RSI Flag: ", image=self.truepic if rsi\_flag else self.falsepic,

compound="right", font=("Arial", 25), fg="black").place(relx=0.8, rely=0.3, anchor="center")

tk.Label(self, text="MACD Flag: ", image=self.truepic if macd\_flag else self.falsepic,

compound="right", font=("Arial", 25), fg="black").place(relx=0.8, rely=0.4, anchor="center")

tk.Label(self, text="Supertrend Flag: ", image=self.truepic if st\_flag else self.falsepic,

compound="right", font=("Arial", 25), fg="black").place(relx=0.8, rely=0.5, anchor="center")

# Show trading metrics

if not hasattr(self.ab, "trading\_amount"):

tk.Label(self, text="Amount Bought: No data yet", font=("Arial", 25), fg="black").place(relx=0.8, rely=0.6, anchor="center")

else:

tk.Label(self, text=f"Amount Bought: {round(self.ab.trading\_amount, 2)}$ ", font=("Arial", 25), fg="black").place(relx=0.8, rely=0.6, anchor="center")

if not hasattr(self.ab, "buy\_btc\_price"):

tk.Label(self, text="Bought At: No data yet", font=("Arial", 25), fg="black").place(relx=0.8, rely=0.7, anchor="center")

else:

tk.Label(self, text=f"Bought At: {round(self.ab.buy\_btc\_price, 2)}$ ", font=("Arial", 25), fg="black").place(relx=0.8, rely=0.7, anchor="center")

if not hasattr(self.ab, "sell\_btc\_price"):

tk.Label(self, text="Sold At: No data yet", font=("Arial", 25), fg="black").place(relx=0.8, rely=0.8, anchor="center")

else:

tk.Label(self, text=f"Sold At: {round(self.ab.sell\_btc\_price, 2)}$ ", font=("Arial", 25), fg="black").place(relx=0.8, rely=0.8, anchor="center")

if not hasattr(self.ab, "profit\_loss"):

tk.Label(self, text="Profit/Loss: No data yet", font=("Arial", 25), fg="black").place(relx=0.8, rely=0.9, anchor="center")

else:

tk.Label(self, text=f"Profit/Loss: {round(self.ab.profit\_loss, 2)}$ ", font=("Arial", 25), fg="black").place(relx=0.8, rely=0.9, anchor="center")

def update\_price(self):

"""

Periodically updates the displayed real-time Bitcoin price.

"""

price = BinanceAPI.get\_ticker\_price()

if isinstance(price, float):

price\_text = f"Bitcoin Price: ${price:.2f}"

else:

price\_text = price

if not hasattr(self, "price\_label"):

self.price\_label = tk.Label(self, text=price\_text, font=("Arial", 35), fg="black")

self.price\_label.place(relx=0.825, rely=0.15, anchor="center")

else:

self.price\_label.config(text=price\_text)

self.after(2000, self.update\_price) # Schedule next update

def go\_homepage(self):

"""

Navigates back to the HomePage and destroys the current frame.

"""

self.navigate\_to(HomePage, self.user\_obj)

self.destroy()

def load\_image(self, path, size):

"""function to load and resize an image."""

img = Image.open(path)

img = img.resize(size, Image.LANCZOS)

return ImageTk.PhotoImage(img)

#### DisplayPictures Class

**Purpose**:  
This class deals with the loading, resizing, and displaying of images (e.g., logo, profile picture) within the UI of the platform. A static method is a method that belongs to a class rather than an instance of the class. In the DisplayPictures class, load\_image is a static method because it only needs the page\_obj, username, and page\_heading passed to it and it doesn’t rely on any internal state of a DisplayPictures object.

**Methods**:

load\_image(page\_obj, username, page\_heading) (staticmethod)

* Loads the BTB logo from the assets directory and resizes it.
* Makes the logo as a clickable button on the top-left corner of the UI, which redirects the user to the Home page
* Loads and resizes the profile image and displays it on the top-right, alongside the entered username.
* Displays the current page heading, at the center of the screen.
* Designed for reuse to not write the same code again

#### BasePage Class

**Purpose**:  
This class contains the common features for all other pages. All classes in the inherit from this class. It inherits from tk.Tk and sets the same features such as fullscreen mode and page navigation.

**Methods**:

\_\_init\_\_(self, title)

* Initializes the main window by calling the parent tk.Tk constructor.
* Sets the window title using the title parameter.
* Puts it in fullscreen mode

navigate\_to(self, page\_class, \*args)

* Destroys the current window
* Instantiates the specified page\_class with any additional arguments and starts its main loop
* Basically deals with moving to new pages

#### LoginPage Class

**Purpose**:  
This class makes the login page. It allows users to log in or register with input validation and navigates them to the next page if the login process happens properly. It inherits from BasePage.

**Methods**:

\_\_init\_\_(self)

* Initializes the login window with the title "Login Page"
* Instantiates the AuthManager
* Places help buttons (?) to display rules for making a proper username and password
* Displays a status label to show messages regarding to the stauts of the login/registration

show\_username\_rules(self)

* Displays a popup with the rules for creating a proper username

show\_password\_rules(self)

* Displays a popup window with the rules for creating a proper password

\_create\_popup(self, title, rules)

* Creates the actual popup window

handle\_login(self)

* Gets the user input from the text boxes and sends it to the AuthManager for validation.
* Updates the status label
* If login is successful, the authentication connection and navigates to the PreferencePage.

validate\_user(self, username: str, password: str) -> bool

* Validates the username and password input.
* Displays error messages if the input does not comply with the username and password rules

handle\_register(self)

* Validates user input
* Registers the user if valid and if the username is not already taken.
* Displays success or error messages in the status

on\_close(self)

* Closes the authentication connection shuts down the login window.

#### PreferencePage Class

**Purpose**:  
This class makes the page for selecting trading preference. It allows the user to choose between live or historical trading

**Methods**:

\_\_init\_\_(self, username)

* Initializes the page with the title "Account Page".
* Calls DisplayPictures.load\_image() to load all images and display the heading
* Sets up the Live Trading button and Historical Trading button

level\_clicked(self, clicked\_button)

* Highlights the selected button in green
* Sets a variable trading\_preference based on the selected option (1 for live, 0 for historical)
* Creates and displays a Confirm button after the trading preference is selected

go\_homepage(self)

* Instantiates a DatabaseManager object with the selected trading preference
* Navigates to the HomePage

#### AccountPage Class

**Purpose**:  
This class makes the account page. It allows users to add money to their account, which is then updated in the database.

**Methods**:

\_\_init\_\_(self, user\_obj)

* Initializes the page with the title "Account Page".
* Asks users to add balance to their account

add\_balance(self)

* Gets the value entered in the text box
* Calls update\_account\_balance() from the DB manager to update the user’s balance.
* If the update is successful, it navigates to the HomePage.
* If the balance is outside the accepted range ($500–$30000) or input is invalid, it displays error messages

go\_homepage(self)

* Navigates to the HomePage, along with the user\_obj

on\_close(self)

* Closes the window when needed.

#### HomePage Class

**Purpose**:  
This class makes the main home page. It acts as the main page to navigate through all the other pages

**Methods**:

\_\_init\_\_(self, user\_obj)

* Initializes the page with the title "Home Page".
* Adds buttons for Profile, Add Money, Trade History
* Displays a Start Trading button goes to either the Date page or directly to the Risk page

go\_homepage(self)

* Refreshes the current page

profile(self)

* Navigates to the ProfilePage with the current user\_obj.

add\_money(self)

* Navigates to the AccountPage with the current user\_obj

trade\_history(self)

* Opens the TradeHistoryPage with the current user\_obj

start\_trading(self)

* If the user selected Historical Trading, navigates to the DatePage.
* If the user selected Live Trading, navigates to the RiskPage.

#### DatePage Class

**Purpose**:  
This class makes the date page to allow the user to enter a date range for historical trading.

**Methods**:

\_\_init\_\_(self, user\_obj)

* Initializes the page titled "Account Page" and stores the passed user\_obj.
* Text boxes for From Date and To Date, prefilled with "YYYY-MM-DD".
* Adds a Confirm button

handle\_date\_entry(self)

* Gets input from the date fields.
* If the process is successful:
  + Stores the date rane in the user\_obj attributes from\_date and to\_date.
  + Navigates to the RiskPage.
* If process fails:
  + Displays an appropriate error messgae

go\_homepage(self)

* Navigates to the HomePage, along with the user\_obj.

go\_riskpage(self)

* Navigates to the RiskPage, along with the updated user\_obj.

#### TradeHistoryPage Class

**Purpose**:  
This class displays a user's full trade history, separating buy and sell transactions into two tables.

**Methods**:

\_\_init\_\_(self, user\_obj)

* Initializes the page titled "Trade History Page"
* Fetches the user’s trade history from the database using get\_trade\_history().
* If history is available, calls display\_tradehistory() to show it
* Otherwise, prints an error message in the terminal

create\_table(self, parent, df, title)

* Creates a scrollable Treeview table

display\_tradehistory(self, df)

* Separates the data into two tables: buy transactions and sell transactions.
* Creates and displays two tables (Buy and Sell) using create\_table()

go\_homepage(self)

* Navigates to the HomePage, along with along the user\_obj.

#### ProfilePage Class

**Purpose**:  
This class displays a user’s profile page, showing their current account balance and an equity curve (the growth of a portfolio over time)

**Methods**:

\_\_init\_\_(self, user\_obj)

* Initializes the page titled "Profile Page"
* Loads the user’s account history and balance from the database.
* Displays the current account balance
* If account history is available:
  + Calls plot\_equity\_curve() to plot the the equity curve
* If no data is available:
  + Prints "No history available" to the terminal

go\_homepage(self)

* Navigates to the HomePage, along with the user\_obj.

plot\_equity\_curve(self, account\_df)

* Creates a Matplotlib Figure
* Uses only the latest entry for each day.
* Embeds the plot into Tkinter using FigureCanvasTkAgg.

#### BotIndicatorsPage Class

**Purpose**:  
This class deals with the bot's indicator analysis using data such as candlestick charts and technical indicators: MACD, RSI, and Supertrend.

**Methods**:

\_\_init\_\_(self, user\_obj)

* Initializes the page titled "Bot Indicators Page"
* Initializes:
  + BotIndicators object to calculate indicators.
  + BTCCandlestickChart to plot the data
  + ActivateBot for the actual trade execution
* Calculates MACD, Supertrend, RSI (Historical)
* Fetches OHLC data based on user entered date range and trading preference.
* Plots the indicator graphs and BTC price chart

go\_homepage(self)

* Navigates to the HomePage, along with the user\_obj.

#### RiskPage Class

**Purpose**:  
This class creates the page to ask users to select the risk page.

**Methods**:

\_\_init\_\_(self, user\_obj)

* Initializes the page titled "Risk Page"
* Creates a heading that risk increases from Level 1 to Level 3.
* Creates three buttons for selecting risk levels

level\_clicked(self, clicked\_button)

* Stores the selected level from the button text.
* Displays the “Activate Bot” button to confirm risk and the “Indicator Graphs" button to view the four graphs

apply\_risk\_thresholds(self)

* Stores the selected risk level in user\_obj.
* Fetches and stores corresponding thresholds from the database.
* Prints the fetched thresholds for debugging.
* Checks if the account balance is greater than 500$.
* Navigates to WorkStationPage and closes the current window.

bot\_indicators(self)

* Stores the selected risk level in user\_obj.
* Fetches and stores corresponding thresholds from the database.
* Navigates to BotIndicatorsPage and closes the current window.

go\_homepage(self)

* Navigates to the HomePage, passing the user\_obj.
* Closes the current window.

#### WorkStationPage Class

**Purpose**:  
This class makes the workstation page, where users can see the bot’s progress and performance

**Methods**:

\_\_init\_\_(self, user\_obj)

* Initializes the page titled "WorkStation Page"
* Initializes BotIndicators for calculating technical indicators.
* Initializes BTCCandlestickChart for displaying the BTC chart
* Initializes ActivateBot for trade execution.
* Sets the trading date range using from\_date and to\_date.
* Starts the buy loop using self.after().

run\_buy\_loop(self)

* Calculates RSI, MACD, and Supertrend indicators.
* Attempts to buy BTC using self.ab.buy\_btc().
* Displays indicators using display\_indicators().
* Goes to the sell loop if the buy action is finished

run\_display\_loop(self)

* Continuously updates and displays indicator flags
* Schedules the next update after 2 seconds using self.after().

run\_sell\_loop(self)

* Calculates RSI, MACD, and Supertrend indicators.
* Attempts to sell BTC using self.ab.sell\_btc().
* Displays updated indicator flags using display\_indicators().
* Switches to the display loop if a trade has finished
* Updates the date and re-runs the loop

display\_indicators(self)

* Displays real-time progress of the current trade
* Refreshes flags for RSI, MACD, and Supertrend.
* Shows trading metrics like current price, amount bought, buy/sell price, and profit/loss.

update\_price(self)

* Updates the real-time Bitcoin price.

go\_homepage(self)

* Navigates back to the HomePage, along with the user\_obj.

load\_image(self, path, size)

* Loads and resizes the images

### activate\_bot.py

from api import BinanceAPI

class ActivateBot:

"""

ActivateBot controls the decision-making logic of when to buy or sell

Bitcoin based on RSI, MACD, and Supertrend.

"""

def \_\_init\_\_(self, user\_obj, ws\_class, bi\_class):

"""

Initializes the trading bot with user and indicator references.

Args:

user\_obj: The user class that includes username and database

access

ws\_class: class that has indicator flags (RSI, MACD,

ST).

bi\_class: class that has indicator values

"""

self.user\_obj = user\_obj

self.ws\_class = ws\_class

self.bi\_class = bi\_class

def buy\_btc(self, given\_date):

"""

Executes a buy order for Bitcoin if 2 or more technical indicators

are positive.

Args:

given\_date (str): Date string in 'YYYY-MM-DD' format

"""

self.buy\_btc\_price = 0

combined\_flag = 0

# Check individual buy signals

if self.ws\_class.bi.rsi\_flag > 0:

combined\_flag += 1

if self.ws\_class.bi.macd\_flag > 0:

combined\_flag += 1

if self.ws\_class.bi.st\_flag > 0:

combined\_flag += 1

#Checks if at least two out of three flags (RSI, MACD, ST) suggest buying.

if combined\_flag >= 2:

print("time to buy")

self.balance = self.user\_obj.db.get\_account\_balance(

self.user\_obj.username

)

print("account balance before", self.balance)

#calculate trading amount based on investment threshold.

self.trading\_amount = self.balance \* (

(self.user\_obj.thresholds["invest\_thres"][0]) / 100

)

price = BinanceAPI.get\_price\_on\_day(given\_date)

if isinstance(price, float):

self.buy\_btc\_price = price

print(

"bought",

self.trading\_amount,

"$ amount of bitcoin at",

self.buy\_btc\_price,

)

#Updates the trade history for a buy transaction.

self.user\_obj.db.update\_trade\_history\_buy(

given\_date, self.user\_obj, self, self.bi\_class

)

else:

self.buy\_btc\_price = -1

print("Error: ", price)

return

def sell\_btc(self, given\_date):

"""

Executes a sell order for Bitcoin if 2 or more technical indicators

are negative.

Args:

given\_date (str): Date string in 'YYYY-MM-DD' format

"""

self.sell\_btc\_price = 0

combined\_flag = 0

if self.buy\_btc\_price <= 0:

print("No buy transaction found")

return

# Check individual sell signals

if self.ws\_class.bi.rsi\_flag < 0:

combined\_flag += 1

if self.ws\_class.bi.macd\_flag < 0:

combined\_flag += 1

if self.ws\_class.bi.st\_flag < 0:

combined\_flag += 1

# Checks if at least two out of three flags (RSI, MACD, ST) suggest selling.

if combined\_flag >= 2:

print("time to sell")

price = BinanceAPI.get\_price\_on\_day(given\_date)

if isinstance(price, float):

self.sell\_btc\_price = price

#Calculates percentage difference and adjusts trading amount.

per\_diff = (

(self.sell\_btc\_price - self.buy\_btc\_price)

/ self.buy\_btc\_price

)

multiplier\_value = 1 + per\_diff

self.new\_trading\_amount = multiplier\_value \* self.trading\_amount

self.profit\_loss = (

self.new\_trading\_amount - self.trading\_amount

)

self.user\_obj.db.update\_account\_balance(

self.user\_obj.username, self.profit\_loss

)

self.balance = self.user\_obj.db.get\_account\_balance(

self.user\_obj.username

)

print(

"sold",

self.new\_trading\_amount,

"$ of bitcoin at",

self.sell\_btc\_price,

)

print("profit/loss=", self.profit\_loss)

print("account balance after", self.balance)

#Updates the user's account balance and trade history.

self.user\_obj.db.update\_trade\_history\_sell(

given\_date, self.user\_obj, self, self.bi\_class

)

else:

self.sell\_btc\_price = -1

print("Sell Error: ", price)

return

#### ActivateBot Class

**Purpose**:  
This class deals with the trading logic by using the three indicators

**Methods**

\_\_init\_\_(self, user\_obj, ws\_class, bi\_class)

* Initializes
  + user\_obj - has user data including thresholds and database access
  + ws\_class – has flags for current trade.
  + bi\_class - has historical indicator data.

buy\_btc(self, given\_date)

* Executes a buy order for Bitcoin if at least 2 out of 3 indicator flags are positive.
* Gets the user's current account balance from the database.
* Calculates how much money to invest based on the risk threshold.
* Gets the Bitcoin price for the required date using BinanceAPI.

sell\_btc(self, given\_date)

* Executes a sell order for Bitcoin if at least 2 out of 3 indicator flags are negative.
* Checks that a previous buy order has finished before selling
  + If no buy order exists, error is added to application logs and exits
* Gets the Bitcoin price for the required date using BinanceAPI.
* Updates the user's account balance and adds the trade in trade history.

### bot\_indicators.py

from api import BinanceAPI

from datetime import timedelta

from utils import Utils

class BotIndicators:

"""

BotIndicators calculates trading indicators (RSI, MACD, Supertrend)

Attributes:

user\_obj (object): User's settings for thresholds, dates, and trading preferences.

symbol (str): BTC

prices (list): List to store price from Binance API.

highs (list): List to store the highest prices in the given timeframe.

lows (list): List to store the lowest prices in the given timeframe.

thresholds (dict): Various risk thresholds for calculating indicators.

continuous\_trade (bool): Flag to determine whether historical trading is enabled.

from\_date (datetime): The start date for fetching historical data.

to\_date (datetime): The end date for fetching historical data.

"""

def \_\_init\_\_(self, user\_obj):

"""

Initializes the BotIndicators class with the user's settings

Args:

user\_obj (object): The user's settings

"""

self.user\_obj = user\_obj

self.symbol = "BTCUSDT"

self.prices = None

self.highs, self.lows, self.prices = None, None, None

self.thresholds = self.get\_risk\_thresholds(self.user\_obj.thresholds)

if self.user\_obj.trading\_preference == 1:

self.to\_date = Utils.get\_date()

self.from\_date = Utils.get\_date()

self.continuous\_trade = True

else:

self.continuous\_trade = False

self.from\_date = self.user\_obj.from\_date

self.to\_date = self.user\_obj.to\_date

def get\_risk\_thresholds(self, thresholds):

"""

Returns risk thresholds from the user's settings

"""

return {

"invest\_thres": thresholds["invest\_thres"][0],

"rsi\_oversold": thresholds["rsi\_oversold"][0],

"rsi\_overbought": thresholds["rsi\_overbought"][0],

"macd\_fast\_ema": thresholds["macd\_fast\_ema"][0],

"macd\_slow\_ema": thresholds["macd\_slow\_ema"][0],

"macd\_signal\_ema": thresholds["macd\_signal\_ema"][0],

"supertrend\_atr\_period": thresholds["supertrend\_atr\_period"][0],

"supertrend\_multiplier": thresholds["supertrend\_multiplier"][0]

}

def calculate\_rsi(self, given\_date):

"""

Calculates RSI for the given date based on historical price data.

"""

period = 14

from\_date = given\_date - timedelta(days=period+1)

to\_date = given\_date

self.rsi\_flag = 0

self.rsi = 0

self.rsi\_values = []

ret\_val = BinanceAPI.get\_prices\_day\_range(self, from\_date, to\_date)

if (ret\_val == 0):

print("Not enough data for RSI")

return

gains, losses = [], []

# Calculate initial gains and losses

for i in range(1, period + 1):

change = self.prices[i] - self.prices[i - 1]

gains.append(max(change, 0))

losses.append(abs(min(change, 0)))

# Calculate initial average gain and loss

avg\_gain = sum(gains) / period

avg\_loss = sum(losses) / period

change = self.prices[-1] - self.prices[-2]

gain = max(change, 0)

loss = abs(min(change, 0))

# Calculate smoothed averages

avg\_gain = ((avg\_gain \* (period - 1)) + gain) / period

avg\_loss = ((avg\_loss \* (period - 1)) + loss) / period

# Calculate RSI

rs = avg\_gain / avg\_loss if avg\_loss != 0 else 100 # Prevent division by zero

self.rsi = 100 - (100 / (1 + rs))

print("RSI ", self.rsi, self.thresholds["rsi\_overbought"], self.thresholds["rsi\_oversold"])

# Determine RSI status (Buy, Sell, or Hold)

if self.rsi > self.thresholds["rsi\_overbought"]:

status = "Sell"

self.rsi\_flag = -1

elif self.rsi < self.thresholds["rsi\_oversold"]:

status = "Buy"

self.rsi\_flag = 1

else:

self.rsi\_flag = 0

return

def calculate\_historical\_rsi(self, given\_date):

"""

Calculates the historical RSI based on closing prices over a given time period.

"""

period = 14

from\_date = given\_date - timedelta(days=100)

to\_date = given\_date

self.rsi\_values = []

ret\_val = BinanceAPI.get\_prices\_day\_range(self, from\_date, to\_date)

if (ret\_val == 0):

print("Not enough data for RSI")

return

gains = []

losses = []

for i in range(1, len(self.prices)):

change = self.prices[i] - self.prices[i - 1]

if change > 0:

gains.append(change)

losses.append(0)

else:

losses.append(abs(change))

gains.append(0)

# Compute initial average gains and losses

avg\_gain = sum(gains[:period]) / period

avg\_loss = sum(losses[:period]) / period

rsi\_values = []

# Calculate smoothed moving average method for the remaining RSI values

for i in range(period, len(self.prices)):

gain = gains[i - 1]

loss = losses[i - 1]

avg\_gain = ((avg\_gain \* (period - 1)) + gain) / period

avg\_loss = ((avg\_loss \* (period - 1)) + loss) / period

if avg\_loss == 0:

rs = float("inf")

else:

rs = avg\_gain / avg\_loss

rsi = 100 - (100 / (1 + rs))

rsi\_values.append((rsi))

self.rsi\_values = rsi\_values

def calculate\_macd(self, given\_date):

"""

Calculates MACD indicator.

"""

fast\_period = self.thresholds["macd\_fast\_ema"]

slow\_period = self.thresholds["macd\_slow\_ema"]

signal\_period = self.thresholds["macd\_signal\_ema"]

self.macd\_flag = 0

self.macd = 0

self.macd\_line\_filtered = []

self.signal\_line\_filtered = []

required\_period = 100

from\_date = given\_date - timedelta(days=required\_period)

to\_date = given\_date

ret\_val = BinanceAPI.get\_prices\_day\_range(self, from\_date, to\_date)

if (ret\_val == 0):

print("Not enough data for MACD")

return

if len(self.prices) < required\_period:

print("Not enough data for MACD")

return

# Calculate EMA multipliers

multiplier\_fast = 2 / (fast\_period + 1)

multiplier\_slow = 2 / (slow\_period + 1)

multiplier\_signal = 2 / (signal\_period + 1)

# Initialize EMA lists with None values

ema\_fast\_list = [None] \* len(self.prices)

ema\_slow\_list = [None] \* len(self.prices)

ema\_fast = sum(self.prices[:fast\_period]) / fast\_period

ema\_slow = sum(self.prices[:slow\_period]) / slow\_period

ema\_fast\_list[fast\_period - 1] = ema\_fast

ema\_slow\_list[slow\_period - 1] = ema\_slow

# Compute the remaining EMA values iteratively

for i in range(slow\_period, len(self.prices)):

ema\_fast = (self.prices[i] - ema\_fast) \* multiplier\_fast + ema\_fast

ema\_slow = (self.prices[i] - ema\_slow) \* multiplier\_slow + ema\_slow

ema\_fast\_list[i] = ema\_fast

ema\_slow\_list[i] = ema\_slow

# Calculate MACD line by subtracting fast EMA from slow EMA

macd\_line = [None] \* len(self.prices)

for i in range(len(self.prices)):

if ema\_fast\_list[i] is not None and ema\_slow\_list[i] is not None:

macd\_line[i] = ema\_fast\_list[i] - ema\_slow\_list[i]

# Generate Signal Line by applying EMA to MACD line

valid\_macd = [m for m in macd\_line if m is not None]

if len(valid\_macd) < signal\_period:

print("Not enough data for Signal Line")

signal\_ema = sum(valid\_macd[:signal\_period]) / signal\_period

signal\_line = [None] \* (slow\_period + signal\_period - 2) + [signal\_ema]

for i in range(0, len(valid\_macd)):

signal\_ema = (valid\_macd[i] - signal\_ema) \* multiplier\_signal + signal\_ema

signal\_line.append(signal\_ema)

if len(macd\_line) < 2 or len(signal\_line) < 2 or macd\_line[-1] is None or macd\_line[-2] is None:

print("Not enough data for MACD signals")

if (

macd\_line[-1] > signal\_line[-1] and macd\_line[-2] <= signal\_line[-2]

):

self.macd\_flag = 1

status = "Buy"

elif (

macd\_line[-1] < signal\_line[-1] and macd\_line[-2] >= signal\_line[-2]

):

self.macd\_flag = -1

status = "Sell"

else:

self.macd\_flag = 0

status = "Hold"

self.macd = macd\_line[-1]

self.macd\_line\_filtered = [m for m in macd\_line]

self.signal\_line\_filtered = [s for s in signal\_line]

def calculate\_supertrend(self, given\_date):

"""

Calculates the Supertrend indicator

"""

self.supertrend\_green = []

self.supertrend\_prices = []

self.supertrend\_red = []

min\_supertrend = 0

max\_supertrend = 0

self.st\_flag = 0

atr\_period = self.thresholds["supertrend\_atr\_period"]

multiplier = self.thresholds["supertrend\_multiplier"]

#period = atr\_period \* 3

period = 100

from\_date = given\_date - timedelta(days=period)

to\_date = given\_date

ret\_val = BinanceAPI.get\_prices\_day\_range(self, from\_date, to\_date)

if (ret\_val == 0):

print("Not enough data for supertrend")

return

tr\_list = []

atr = [None] \* len(self.prices)

supertrend = [None] \* len(self.prices)

for i in range(len(self.prices)):

tr = max(self.highs[i] - self.lows[i],

abs(self.highs[i] - self.prices[i - 1]) if i > 0 else 0,

abs(self.lows[i] - self.prices[i - 1]) if i > 0 else 0)

tr\_list.append(tr)

# Compute First ATR value using SMA

atr[atr\_period - 1] = sum(tr\_list[:atr\_period]) / atr\_period

# Compute Remaining ATR values using EMA formula

for i in range(atr\_period, len(self.prices)):

atr[i] = ((atr[i - 1] \* (atr\_period - 1)) + tr\_list[i]) / atr\_period

upper = [None]\* len(self.prices)

lower = [None]\* len(self.prices)

multiplier = 1

for i in range(atr\_period, len(self.prices)):

mid = (self.highs[i] + self.lows[i]) / 2

upper[i] = mid + (multiplier \* atr[i])

lower[i] = mid - (multiplier \* atr[i])

if i == atr\_period:

supertrend[i] = (lower[i])

else:

if self.prices[i] > supertrend[i-1]:

supertrend[i] = lower[i]

else:

supertrend[i] = upper[i]

#print("price -", self.prices[i], atr[i],upper[i], lower[i])

green\_line = [None] \* len(self.prices)

red\_line = [None] \* len(self.prices)

for i in range(atr\_period, len(self.prices)):

if supertrend[i] < self.prices[i]:

green\_line[i] = supertrend[i]

else:

red\_line[i] = supertrend[i]

self.supertrend\_green = [m for m in green\_line ]

self.supertrend\_prices = [s for s in self.prices ]

self.supertrend\_red = [p for p in red\_line]

min\_supertrend = min(s for s in supertrend if s != None)

max\_supertrend = max(s for s in supertrend if s != None)

#plot\_macd(min\_supertrend-5000, max\_supertrend+5000, macd\_line\_filtered, "buy", signal\_line\_filtered, "prices", price\_line, "sell")

signals = [None] \* len(self.prices)

for i in range(atr\_period+1, len(self.prices)):

# if supertrend[i] is None or supertrend[i-1] is None:

# continue

if i > 0 and self.prices[i] > supertrend[i] :# and self.prices[i - 1] < supertrend[i - 1]:

signals[i] = "BUY"

elif i > 0 and self.prices[i] < supertrend[i] :#and self.prices[i - 1] > supertrend[i - 1]:

signals[i]="SELL"

else:

signals[i]="HOLD"

#print(self.prices[i], supertrend[i], signals[i])

self.supertrend = supertrend[-1]

if signals[-1]=="BUY":

self.st\_flag = 1

elif signals[-1]=="SELL":

self.st\_flag=-1

else:

self.st\_flag=0

print("supertrend ", self.prices[-1] , supertrend[-1] , self.prices[-2] , supertrend[-2])

#### BotIndicators Class

**Purpose**:  
This class calculates the trading indicators and returns the flags

**Methods**

\_\_init\_\_(self, user\_obj)

* Initializes with the user’s setting (preference and risk level)
* Sets the trading range based on live or historical trading

get\_risk\_thresholds(self, thresholds)

* Gets the risk thresholds from the user's settings

calculate\_rsi(self, given\_date)

* Calculates the RSI for the given date
  + Uses a 14-day period to calculate average gains and losses.
* Stores the RSI value and creates a signal based on thresholds

calculate\_historical\_rsi(self, given\_date)

* Calculates the historical RSI for 100 days
  + Calculates gains and losses for the 100 days before the given date
* Stores the calculated RSI values for graph plotting

calculate\_macd(self, given\_date)

* Calculates the MACD indicator for the given date.
  + Uses fast, slow, and signal EMAs (Exponential Moving Averages) to calculate the MACD line and signal line.
  + Creates a signal based on crossovers (when the MACD line crosses the signal line).
* Stores the MACD values and filtered signal lines for graph plotting

calculate\_supertrend(self, given\_date)

* Calculates the Supertrend indicator based on the user’s Average True Range and multiplier (both pre-configured thresholds)
  + Calculates the ATR for 100 days and applies the multiplier to determine upper and lower bands.
  + Creates a signal based on the crossover between the price and the Supertrend line.
* Stores the Supertrend values and signals (green and red lines) for graph plotting

### btccschart.py

import matplotlib.pyplot as plt

import matplotlib.dates as mdates

from matplotlib.figure import Figure

from matplotlib.patches import Rectangle

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

import tkinter as tk

from api import BinanceAPI

from utils import Utils

from datetime import datetime, timedelta

class BTCCandlestickChart:

def \_\_init\_\_(self, parent):

"""

Initializes the candlestick chart inside the given Tkinter parent widget.

"""

self.parent = parent

def get\_ohlc\_data(self, trading\_preference, from\_date, to\_date):

"""Fetches the OHLC data for the specified date range.

If trading\_preference is 1, it fetches data for the last 30 days.

"""

if trading\_preference == 1:

from\_date = Utils.get\_date() - timedelta(days=30)

to\_date = Utils.get\_date()

self.ohlc\_data = BinanceAPI.get\_ohlc\_day\_range(from\_date, to\_date)

def plot\_indicator\_graphs(self, bi\_class):

"""Plots RSI, MACD, Supertrend, and Prices in a 2x2 Matplotlib figure

"""

fig = Figure(figsize=(12, 8))

ax1 = fig.add\_subplot(221) # RSI

ax2 = fig.add\_subplot(222) # MACD

ax3 = fig.add\_subplot(223) # Supertrend

ax4 = fig.add\_subplot(224) # Prices

fig.subplots\_adjust(hspace=0.4, wspace=0.3)

y1 = bi\_class.rsi\_values

y21 = bi\_class.macd\_line\_filtered

y22 = bi\_class.signal\_line\_filtered

y31 = bi\_class.supertrend\_green

y32 = bi\_class.supertrend\_prices

y33 = bi\_class.supertrend\_red

# Plot RSI

x1 = list(range(len(y1)))

ax4.plot(x1, y1, color="blue")

ax4.set\_title("RSI - 100 Days")

ax4.set\_xlabel("Days")

ax4.set\_ylabel("Percentage")

# Plot MACD & Signal Line

ax2.plot(y21, color="purple", label="MACD Line")

ax2.plot(y22, color="pink", label="Signal Line")

ax2.set\_title("MACD - 100 Days")

ax2.legend()

ax2.set\_xlabel("Days")

# Plot Supertrend

ax3.plot(y31, label="Buy Signal", color="green")

ax3.plot(y32, label="Price", color="blue")

ax3.plot(y33, label="Sell Signal", color="red")

ax3.set\_title("Supertrend - 100 Days")

ax3.legend()

ax3.set\_xlabel("Days")

ax3.set\_ylabel("Price($)")

# Plot Prices

# x4 = list(range(len(y32)))

# ax4.plot(x4, y32, color="purple")

# ax4.set\_title("Prices")

# # Hide X-Axis

# for ax in [ ax2, ax3]:

# ax.set\_xticks([]) # Remove tick marks

# ax.set\_xticklabels([]) # Remove tick labels

# ax.set\_xlabel('') # Remove axis label

# ax.spines['bottom'].set\_visible(False) # Hide x-axis line

self.canvas = FigureCanvasTkAgg(fig, master=self.parent)

self.plot\_candlestick\_chart(fig, ax1)

self.canvas.draw()

self.canvas.get\_tk\_widget().pack(fill=tk.BOTH, expand=True)

def plot\_candlestick\_chart(self, fig, ax):

"""

Plots the BTC/USDT candlestick chart inside Tkinter.

"""

self.fig = fig

self.ax = ax

if (self.ohlc\_data is None):

print("Not enough data to plot the chart")

return

# Bind Mouse Hover Event to handle tooltip

self.canvas.mpl\_connect("motion\_notify\_event", self.on\_hover)

# Format x-axis for dates

self.ax.xaxis.set\_major\_formatter(mdates.DateFormatter('%b %d'))

self.ax.xaxis.set\_major\_locator(mdates.DayLocator(interval=2))

for label in self.ax.get\_xticklabels():

label.set\_rotation(45)

label.set\_horizontalalignment('right')

# Plot each candlestick (OHLC) data

for x, o, h, l, c in self.ohlc\_data:

color = 'g' if c >= o else 'r'

self.ax.plot([x, x], [l, h], color=color)

self.ax.add\_patch(Rectangle((x - 0.3, min(o, c)), 0.6, abs(o - c), color=color))

self.ohlc\_annot = self.ax.annotate("", xy=(0, 0), xytext=(-50, 50), textcoords="offset points",

bbox=dict(boxstyle="round", fc="w"),

arrowprops=dict(arrowstyle="->"))

self.ohlc\_annot.set\_visible(False)

self.ax.set\_title("BTC/USDT")

self.ax.set\_ylabel("Price($)")

def is\_cursor\_on\_candlestick(self, event):

"""

Checks if the cursor is over a candlestick body.

"""

for i, (x, o, h, l, c) in enumerate(self.ohlc\_data):

if abs(x - event.xdata) < 0.3 and l <= event.ydata <= h:

return i

return None

def update\_ohlc\_annot(self, index):

"""

Updates the OHLC tooltip when hovering over a candlestick.

"""

x, o, h, l, c = self.ohlc\_data[index]

self.ohlc\_annot.xy = (x, c)

text = f"Date: {mdates.num2date(x).strftime('%b %d')}\nOpen: {o}\nHigh: {h}\nLow: {l}\nClose: {c}"

self.ohlc\_annot.set\_text(text)

self.ohlc\_annot.get\_bbox\_patch().set\_alpha(0.9)

def on\_hover(self, event):

"""

Handles the hover event for showing tooltips when hovering over candlesticks.

"""

if event.inaxes == self.ax:

candlestick\_index = self.is\_cursor\_on\_candlestick(event) # Check if hovering over a candlestick

if candlestick\_index is not None:

self.update\_ohlc\_annot(candlestick\_index)

self.ohlc\_annot.set\_visible(True)

else:

self.ohlc\_annot.set\_visible(False)

self.canvas.draw\_idle()

else:

self.ohlc\_annot.set\_visible(False)

self.canvas.draw\_idle()

#### BTCCandlestickChart Class

**Purpose**:  
This class is where all the graphs are plotted. There are four graphs that can be seen in the Bot Indicators page in the platform: the BTC/USDT chart, RSI, MACD, and Supertrend charts

**Methods**

\_\_init\_\_(self, parent, bi\_class)

* Initializes the class and embeds it into the Tkinter parent widget

get\_ohlc\_data(self, trading\_preference, from\_date, to\_date)

* Gets historical OHLC data for BTC/USDT.
* If the user selects live trading, it plots all the graphs for 30 days before the current date.

plot\_indicator\_graphs(self, bi\_class)

* Creates a subplot to display RSI, MACD, Supertrend, and the BTC price chart.
* Calls BotIndicators and plots from the values given from that class

plot\_candlestick\_chart(self, fig, ax)

* Plots all the four charts, with colour using matplotlib

is\_cursor\_on\_candlestick(self, event)

* Checks if the mouse cursor is hovering over a candlestick.
* If it is hovering over a candlestick, it displays the OHLC data for that particular candle

update\_ohlc\_annot(self, index)

* Updates the popup with OHLC data for the given candlestick.

on\_hover(self, event)

* Handles mouse hovers to display candlestick data popups
* Calls is\_cursor\_on\_candlestick() to check if a candle is under the cursor.

### auth.py

from database import DatabaseManager

class AuthManager:

"""

AuthManager handles user authentication, including user registration,

login validation, and password hashing with a custom algorithm.

"""

def \_\_init\_\_(self):

"""

Initializes the AuthManager instance by creating a connection

to the database using DatabaseManager.

"""

self.db = DatabaseManager(1)

def close(self):

"""

Closes the connection to the database.

"""

self.db.close\_connection()

def custom\_hash(self, password: str) -> str:

"""

Generates a custom hash for a given password

"""

salt = self.db.get\_salt\_key()

data = password + salt

hash\_value = 0

for char in data:

hash\_value = (hash\_value << 3) + (hash\_value >> 2) + ord(char)

hash\_value %= 2\*\*32

return hex(hash\_value)

def register\_user(self, username: str, password: str) -> bool:

"""

Registers a new user by hashing the password and storing it in the database.

"""

hashed\_password = self.custom\_hash(password)

return self.db.add\_user(username, hashed\_password)

def validate\_login(self, username: str, password: str) -> str:

"""

Validates user credentials

"""

user\_data = self.db.get\_user(username)

if user\_data.empty:

return "User does not exist"

stored\_hashed\_password = user\_data["hashed\_password"][0]

hashed\_password = self.custom\_hash(password)

if stored\_hashed\_password == hashed\_password:

self.db.log\_event(username, 'LOGIN', 'User logged in.')

return "Login Successful"

else:

self.db.log\_event(username, 'LOGIN', 'Login Unsuccessful/Wrong Credentials')

return "Wrong password"

#### AuthManager Class

**Purpose**:  
This class is used for authenticating the user by validating registration and login. It also contains the hashing algorithm used to encrypt the passwords

**Methods:**

\_\_init\_\_(self)

* Initializes the class and connects to the database with the selected trading preference

close(self)

* Closes the database connection.

custom\_hash(self, password: str) -> str

* Generates a custom hash of the given password.
* Gets the salt key from the database and concatenates it to the password.
* Applies a bitwise hashing algorithm using left and right shift operations.
* Returns the hashed password as a hexadecimal.

register\_user(self, username: str, password: str) -> bool

* Registers a new user in the database.
* Stores the username and hashed password in the database.

validate\_login(self, username: str, password: str) -> str

* Validates the login process by comparing the hashed password.
* Gets the stored hashed password from the database.
* Hashes the password using the hashing algorithm
  + If they both match, returns “Login Successful.”
  + If not, returns an error message.

### api.py

import requests

from datetime import datetime, timedelta

import matplotlib.dates as mdates

class BinanceAPI:

API\_BASE\_URL= "https://api.binance.com/api/v3"

API\_TIMEOUT = 30

@staticmethod

def get\_ticker\_price(symbol="BTCUSDT"):

"""

Fetch the latest price for a given symbol from Binance Testnet.

"""

endpoint = f"{BinanceAPI.API\_BASE\_URL}/ticker/price"

params = {"symbol": symbol}

try:

response = requests.get(endpoint, params=params, timeout=10)

response.raise\_for\_status()

data = response.json()

return float(data["price"])

except requests.exceptions.RequestException as e:

print(f"Request error: {e}")

return(f"API Request error")

except ValueError:

print( f"Error: Could not parse response. {response.status\_code}, {response.text}")

return(f"Error: Could not parse API response.")

except KeyError:

print( f"Error: Unexpected response format.{response.status\_code}, {response.text}")

return(f"Error: Unexpected response API format.")

@staticmethod

def get\_price\_on\_day(given\_date):

"""

Fetches the closing price for a given date from Binance API.

"""

endpoint = f"{BinanceAPI.API\_BASE\_URL}/klines"

from\_ts = given\_date - timedelta(days=1)

from\_ts = int(from\_ts.timestamp() \* 1000)

to\_ts = int(given\_date.timestamp() \* 1000)

params = {

"symbol": "BTCUSDT",

"interval": "1d",

"startTime": from\_ts,

"endTime": to\_ts

}

try:

response = requests.get(endpoint, params=params, timeout=BinanceAPI.API\_TIMEOUT)

response.raise\_for\_status()

data = response.json()

if data:

closing\_price = float(data[0][4])

return closing\_price

else:

print("No data found for that day.")

return None

except requests.exceptions.RequestException as e:

print(f"Request error: {e}")

return None

except (ValueError, KeyError, IndexError) as e:

print(f"Data processing error: {e}")

return None

@staticmethod

def get\_prices\_day\_range(data, from\_date, to\_date):

"""

Fetches closing, high, and low prices from Binance API for a date range.

"""

endpoint = f"{BinanceAPI.API\_BASE\_URL}/klines"

from\_ts = int(from\_date.timestamp() \* 1000)

to\_ts = int(to\_date.timestamp() \* 1000)

params = {

"symbol": "BTCUSDT",

"interval": "1d",

"startTime": from\_ts,

"endTime": to\_ts

}

try:

response = requests.get(endpoint, params=params, timeout=BinanceAPI.API\_TIMEOUT)

response.raise\_for\_status()

response\_data = response.json()

data.prices = [float(candle[4]) for candle in response\_data]

data.highs = [float(candle[2]) for candle in response\_data]

data.lows = [float(candle[3]) for candle in response\_data]

return(len(data.prices))

except requests.exceptions.RequestException as e:

print(f"Request error: {e}")

return(0)

except (ValueError, KeyError, IndexError) as e:

print(f"Data processing error: {e}")

return(0)

@staticmethod

def get\_ohlc\_day\_range(from\_date, to\_date):

"""

Fetches OHLC data from Binance API for a date range.

"""

endpoint = f"{BinanceAPI.API\_BASE\_URL}/klines"

from\_ts = int(from\_date.timestamp() \* 1000)

to\_ts = int(to\_date.timestamp() \* 1000)

params = {

"symbol": "BTCUSDT",

"interval": "1d",

"startTime": from\_ts,

"endTime": to\_ts

}

try:

response = requests.get(endpoint, params=params, timeout=BinanceAPI.API\_TIMEOUT)

response.raise\_for\_status()

response\_data = response.json()

ohlc\_data = []

for candle in response\_data:

timestamp = datetime.fromtimestamp(candle[0] / 1000)

ohlc\_data.append([

mdates.date2num(timestamp),

float(candle[1]),

float(candle[2]),

float(candle[3]),

float(candle[4])

])

return ohlc\_data

except requests.exceptions.RequestException as e:

print(f"Request error: {e}")

return None

except (ValueError, KeyError, IndexError) as e:

print(f"Data processing error: {e}")

return None

#### BinanceAPI Class

**Purpose:**  
This class contains all the API calls needed for the whole project.

**Methods:**

get\_ticker\_price(self, symbol: str = "BTCUSDT") -> float

* Gets the latest price for BTC/USDT from the Binance API.
* Returns the latest price
  + Otherwise returns an error message
* Deals with exceptions such as network errors or unexpected formats.

get\_price\_on\_day(self, given\_date: datetime) -> float

* Gets the closing price of BTC/USDT for a given date using the Binance API.
* Deals with exceptions such as network errors or unexpected formats.

get\_prices\_day\_range(self, data, from\_date: str, to\_date: str) -> int

* Gets prices for BTC/USDT over a required range.
* Converts the date range into timestamps
* Deals with exceptions such as network errors or unexpected formats.

get\_ohlc\_day\_range(self, from\_date: str, to\_date: str) -> list

* Gets OHLC data for BTC/USDT over a required range.
* Converts the date range into timestamps
* Returns the formatted OHLC data as a list
  + Otherwise, None if an error occurs
* Deals with exceptions such as network errors or unexpected formats.

### database.py

import sqlite3

import pandas as pd

from utils import Utils

# Path to the database files

DATABASE\_PATH = "assets/bitcoin.db"

DATABASE\_TEST\_PATH = "assets/bitcoin\_test.db"

class DatabaseManager:

"""

Handles database operations for user management and balances.

"""

def \_\_init\_\_(self, trading\_preference):

"""

Initializes the database connection based on the trading preference.

"""

self.connect(trading\_preference)

def connect(self, trading\_preference):

try:

print("Connecting to database...")

self.trading\_preference = trading\_preference

if trading\_preference:

self.conn = sqlite3.connect(DATABASE\_PATH) # Connect to main database

else:

self.conn = sqlite3.connect(DATABASE\_TEST\_PATH) # Connect to test database

self.cursor = self.conn.cursor()

print("Connected to database successfully.")

except sqlite3.Error as error:

print(f"Unable to connect database manager: {error}")

return

def check\_connection(self):

try:

self.cursor.execute("SELECT 1")

return True

except sqlite3.Error:

self.connect(trading\_preference=self.trading\_preference) # Reconnect if the connection is lost

return False

def close\_connection(self):

if self.conn:

self.conn.close()

def get\_salt\_key(self) -> str:

"""

Gets the SALT\_KEY from the database, which is used for password hashing.

"""

try:

self.check\_connection()

self.thresholds = pd.read\_sql\_query(

"SELECT config\_value FROM bitcoin\_config WHERE config\_code = 'SALT\_KEY'",

self.conn

)

return self.thresholds['config\_value'][0] if not self.thresholds.empty else ""

except sqlite3.Error as error:

print(f"Database error: {error}")

return ""

def add\_user(self, username: str, hashed\_password: str) -> bool:

"""

Adds a new user to the database

"""

try:

self.check\_connection()

self.thresholds = pd.read\_sql\_query(

"SELECT COUNT(\*) as user\_count FROM user\_profile WHERE username = ?",

self.conn, params=(username,)

)

user\_count = self.thresholds['user\_count'][0]

if user\_count > 0:

return False

self.cursor.execute(

"INSERT INTO user\_profile(username, hashed\_password, account\_balance) VALUES (?, ?, ?);",

(username, hashed\_password, 0)

)

self.conn.commit()

return True

except sqlite3.Error as error:

print(f"Database error: {error}")

return False

def get\_user(self, username: str) -> pd.DataFrame:

"""

Fetches user details from the database.

"""

try:

self.check\_connection()

return pd.read\_sql\_query(

"SELECT \* FROM user\_profile WHERE username = ?",

self.conn, params=(username,)

)

except sqlite3.Error as error:

print(f"Database error: {error}")

return pd.DataFrame()

def update\_account\_balance(self, username: str, amount: float):

"""

Updates the user's account balance by adding the specified amount.

"""

try:

self.check\_connection()

self.cursor.execute("SELECT account\_balance FROM user\_profile WHERE username = ?", (username,))

row = self.cursor.fetchone()

if row is None:

print("User not found.")

return False, 0

current\_balance = row[0]

new\_balance = current\_balance + amount

self.balance = new\_balance

if 500 < new\_balance < 30000: # Ensure balance is within valid range

self.cursor.execute(

"UPDATE user\_profile SET account\_balance = ? WHERE username = ?",

(new\_balance, username)

)

self.conn.commit()

self.log\_event(username, 'BALANCE\_UPDATE', f'topup = {amount} total balance = {new\_balance}')

self.update\_account\_history(username, new\_balance)

return True, new\_balance

else:

return False, 0

except sqlite3.Error as error:

print(f"Database error: {error}")

return False, 0

def get\_account\_balance(self, username: str):

"""

Fetches the current account balance for a user.

"""

try:

self.check\_connection()

self.cursor.execute("SELECT account\_balance FROM user\_profile WHERE username = ?", (username,))

row = self.cursor.fetchone()

if row is None:

print("User not found.")

return False, 0

current\_balance = row[0]

return current\_balance

except sqlite3.Error as error:

print(f"Database error: {error}")

return False, 0

def get\_risk\_thresholds(self, risk\_level):

"""

Fetches the risk thresholds for a given risk level from the database.

"""

try:

self.check\_connection()

self.thresholds = pd.read\_sql\_query(

f"SELECT \* FROM risk\_thresholds WHERE risk\_level = {risk\_level}",

self.conn

)

if not self.thresholds.empty:

self.invest\_thres = self.thresholds["invest\_thres"][0]

return self.thresholds

except sqlite3.Error as error:

print(f"Database error: {error}")

return

def update\_trade\_history\_buy(self, given\_date, user\_obj, ab\_class, bi\_class):

"""

Updates the trade history with a buy trade.

"""

try:

self.check\_connection()

user\_id\_df = pd.read\_sql\_query(

"SELECT user\_id FROM user\_profile WHERE username = ?",

self.conn, params=(user\_obj.username,)

)

user\_id = int(user\_id\_df['user\_id'][0])

# Insert trade data into trade history

self.cursor.execute("""

INSERT INTO trade\_history (

user\_id, money\_in, risk\_level, buying\_time, buy\_price, balance\_before,

rsi\_buy, macd\_buy, supertrend\_buy,

rsi\_flag\_buy, macd\_flag\_buy, supertrend\_flag\_buy

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?);

""", (

user\_id, ab\_class.trading\_amount, user\_obj.risk\_level, given\_date,

ab\_class.buy\_btc\_price, ab\_class.balance, bi\_class.rsi, bi\_class.macd,

bi\_class.supertrend, bi\_class.rsi\_flag, bi\_class.macd\_flag,

bi\_class.st\_flag

))

self.conn.commit()

return True

except sqlite3.Error as error:

print(f"Database error: {error}")

self.conn.rollback()

return False

def update\_trade\_history\_sell(self, given\_date, user\_obj, ab\_class, bi\_class):

"""

Updates the trade history with a sell trade.

"""

try:

self.check\_connection()

user\_id\_df = pd.read\_sql\_query(

"SELECT user\_id FROM user\_profile WHERE username = ?",

self.conn, params=(user\_obj.username,)

)

user\_id = int(user\_id\_df['user\_id'][0])

trade\_id\_df = pd.read\_sql\_query(

"SELECT max(trade\_id) as trade\_id FROM trade\_history WHERE user\_id = ?",

self.conn, params=(user\_id,)

)

trade\_id = int(trade\_id\_df['trade\_id'][0])

# Update trade data in trade history

self.cursor.execute("""

UPDATE trade\_history

SET selling\_time = ?, sell\_price = ?, profit\_loss = ?, balance\_after = ?,

rsi\_sell = ?, macd\_sell = ?, supertrend\_sell = ?,

rsi\_flag\_sell = ?, macd\_flag\_sell = ?, supertrend\_flag\_sell = ?

WHERE trade\_id = ?;

""", (

given\_date,

ab\_class.sell\_btc\_price,

ab\_class.profit\_loss,

ab\_class.balance,

bi\_class.rsi,

bi\_class.macd,

bi\_class.supertrend,

bi\_class.rsi\_flag,

bi\_class.macd\_flag,

bi\_class.st\_flag,

trade\_id

))

self.conn.commit()

return True

except sqlite3.Error as error:

print(f"Database error: {error}")

self.conn.rollback() # Rollback on error

return False

def log\_event(self, username, event\_type, comments=None):

"""

Logs an event into the application logs table.

"""

try:

self.check\_connection()

# Fetch user ID

user\_id\_df = pd.read\_sql\_query(

"SELECT user\_id FROM user\_profile WHERE username = ?",

self.conn,

params=(username,)

)

user\_id = int(user\_id\_df['user\_id'][0])

event\_time = Utils.get\_localtime()

# Insert log record into application\_logs table

query = """

INSERT INTO application\_logs (user\_id, event\_type, event\_time,

comments)

VALUES (?, ?, ?, ?)

"""

self.cursor.execute(query, (user\_id, event\_type, event\_time, comments))

self.conn.commit()

except sqlite3.Error as error:

print(f"Database error: {error}")

self.conn.rollback()

return False

def get\_trade\_history(self, username):

"""

Gets the trade history for a user.

"""

try:

self.check\_connection()

user\_id\_df = pd.read\_sql\_query(

"SELECT user\_id FROM user\_profile WHERE username = ?",

self.conn,

params=(username,)

)

user\_id = int(user\_id\_df['user\_id'][0])

trade\_df = pd.read\_sql\_query(

"SELECT \* FROM trade\_history WHERE user\_id = ?",

self.conn,

params=(user\_id,)

)

if len(trade\_df) <= 0:

print("There are no records")

return None

else:

return trade\_df

except sqlite3.Error as error:

print(f"Database error: {error}")

return None

def update\_account\_history(self, username, account\_balance):

"""

Updates the account history with the new account balance.

"""

try:

self.check\_connection()

# Fetch user ID

user\_id\_df = pd.read\_sql\_query(

"SELECT user\_id FROM user\_profile WHERE username = ?",

self.conn,

params=(username,)

)

user\_id = int(user\_id\_df['user\_id'][0])

account\_date = Utils.get\_localtime()

# Insert account balance history record

query = """

INSERT INTO account\_history (user\_id, account\_date,

account\_balance)

VALUES (?, ?, ?)

"""

self.cursor.execute(query, (user\_id, account\_date, account\_balance))

self.conn.commit()

except sqlite3.Error as error:

print(f"Database error: {error}")

self.conn.rollback()

return False

def get\_account\_history(self, username):

"""

Retrieves the account history for a user.

"""

try:

self.check\_connection()

user\_id\_df = pd.read\_sql\_query(

"SELECT user\_id FROM user\_profile WHERE username = ?",

self.conn,

params=(username,)

)

user\_id = int(user\_id\_df['user\_id'][0])

account\_df = pd.read\_sql\_query(

"SELECT \* FROM account\_history WHERE user\_id = ?",

self.conn,

params=(user\_id,)

)

if len(account\_df) <= 0:

print("No history available")

return False, None

else:

return True, account\_df

except sqlite3.Error as error:

print(f"Database error: {error}")

return None

#### DatabaseManager Class

**Purpose:**  
This class handles all database operations such as user management, account balances, trade history, and event logging.

**Methods:**

\_\_init\_\_(self, trading\_preference: bool)

* Selects the appropriate database based on the trading preference

close\_connection(self)

* Closes the database connection.

get\_salt\_key(self) -> str

* Gets the salt key from the bitcoin\_config table

add\_user(self, username: str, hashed\_password: str) -> bool

* Registers a new user by adding to the user\_profile table.
* Checks if the username already exists in the database
* Adds the username, hashed password, and an initial account balance of 0

get\_user(self, username: str) -> pd.DataFrame

* Gets user details from the user\_profile table for a given username.

update\_account\_balance(self, username: str, amount: float) -> tuple[bool, float]

* Gets the current balance, adds to calculate new balance, and checks that it meets the conditions (500 to 30,000$).
* Updates the user\_profile table with the new balance

get\_account\_balance(self, username: str) -> tuple[bool, float]

* Gets the current account balance for a user

get\_risk\_thresholds(self, risk\_level: int) -> pd.DataFrame

* Gets risk thresholds for a given risk level from the risk\_thresholds table.

update\_trade\_history\_buy(self, given\_date: str, user\_obj: object, ab\_class: object, bi\_class: object) -> bool

* Adds a buy trade in the trade\_history table.
* Gets the user ID for a username and inserts trade details
* Commits the action to the database.

update\_trade\_history\_sell(self, given\_date: str, user\_obj: object, ab\_class: object, bi\_class: object) -> bool

* Adds a sell trade in the trade\_history table
* Gets the user ID for a username and the trade ID from that and inserts trade details
* Commits the action to the database

log\_event(self, username: str, event\_type: str, comments: str = None) -> bool

* Logs an event into the application\_logs table
* Gets the user ID for a username and logs the event with the current timestamp
* Commits the action to the database.

get\_trade\_history(self, username: str) -> pd.DataFrame

* Gets the trade history for a user from the trade\_history table

update\_account\_history(self, username: str, account\_balance: float) -> bool

* Updates the account\_history table with the user’s new balance.
* Gets the user ID and adds the current timestamp and updated balance
* Commits the action to the database.

get\_account\_history(self, username: str) -> tuple[bool, pd.DataFrame]

* Gets the account history for a user from the account\_history table.

### utils.py

from datetime import datetime

import pytz

class Utils:

@staticmethod

def get\_localtime():

dubai\_tz = pytz.timezone("Asia/Dubai")

# Get the current local time in Dbia

dubai\_time = datetime.now(dubai\_tz)

# Return the current date and time as a datetime object

return dubai\_time

@staticmethod

def get\_localdate():

dubai\_tz = pytz.timezone("Asia/Dubai")

dubai\_time = datetime.now(dubai\_tz)

# formatted as 'YYYY-MM-DD'

return dubai\_time.strftime(

"%Y-%m-%d"

)

@staticmethod

def get\_date():

dubai\_tz = pytz.timezone("Asia/Dubai")

dubai\_time = datetime.now(dubai\_tz)

return dubai\_time

#### Utils Class

**Purpose:**This class handles all utility functions by managing all time operations. It uses Dubai/Asia time across the whole platform

**Methods:**

get\_localtime(self) -> datetime

* Gets the full unformatted datetime in Dubai

get\_localdate(self) -> str

* Gets only the date in Dubai, formatted as YYYY-MM-DD.

get\_date(self) -> datetime

* **Same as get\_localtime**

## Database

There are two databases being used for this application:

bitcoin.db – This is used for live trading

bitcoin\_test.db – This is used for historical trading/backtesting

**Each of these databases have identical tables and schema.**

### user\_profile

CREATE TABLE user\_profile (

user\_id INTEGER PRIMARY KEY AUTOINCREMENT,

username TEXT,

hashed\_password TEXT,

account\_balance FLOAT

)

|  |  |  |  |
| --- | --- | --- | --- |
| **user\_id** | **username** | **hashed\_password** | **account\_balance** |
| 1 | charlie1 | 0x32b54d8f | 8795.915606 |
| 2 | mamathag | 0xfdd0fad | 1000 |
| 3 | paulshyam | 0x32b54d8f | 1000 |

### trade\_history

CREATE TABLE trade\_history (

trade\_id INTEGER PRIMARY KEY AUTOINCREMENT, -- Unique trade identifier

user\_id INTEGER NOT NULL, -- Reference to the user who made the trade

money\_in FLOAT, -- Amount of money invested in the trade

risk\_level INT, -- Risk level setting at the time of trade (e.g., 1, 2, 3)

buying\_time DATETIME, -- Timestamp when the trade was opened (buy order executed)

selling\_time DATETIME, -- Timestamp when the trade was closed (sell order executed)

buy\_price FLOAT, -- Price at which the asset was bought

sell\_price FLOAT, -- Price at which the asset was sold

profit\_loss FLOAT, -- Profit or loss from the trade (sell\_price - buy\_price)

balance\_before FLOAT, -- Account balance before executing the trade

balance\_after FLOAT, -- Account balance after the trade was completed

rsi\_buy FLOAT, -- RSI (Relative Strength Index) value at the time of buying

macd\_buy FLOAT, -- MACD (Moving Average Convergence Divergence) value at buying

supertrend\_buy FLOAT, -- Supertrend indicator value at buying

rsi\_sell FLOAT, -- RSI value at the time of selling

macd\_sell FLOAT, -- MACD value at selling

supertrend\_sell FLOAT, -- Supertrend indicator value at selling

rsi\_flag\_buy INT, -- RSI buy signal flag (1 = buy, 0 = no signal)

macd\_flag\_buy INT, -- MACD buy signal flag (1 = buy, 0 = no signal)

supertrend\_flag\_buy INT, -- Supertrend buy signal flag (1 = buy, 0 = no signal)

rsi\_flag\_sell INT, -- RSI sell signal flag (1 = sell, 0 = no signal)

macd\_flag\_sell INT, -- MACD sell signal flag (1 = sell, 0 = no signal)

supertrend\_flag\_sell INT, -- Supertrend sell signal flag (1 = sell, 0 = no signal)

FOREIGN KEY (user\_id) REFERENCES user\_profile(user\_id)

)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **trade\_id** | **user\_id** | **money\_in** | **risk\_level** | **buying\_time** | **selling\_time** |
| 24 | 1 | 5217.549364 | 2 | 2025-04-05 18:13:08.494566+04:00 | 2025-04-05 18:15:08.559008+04:00 |
| 25 | 1 | 7826.324046 | 3 | 2025-04-05 18:17:39.555541+04:00 | 2025-04-05 18:19:52.983305+04:00 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **buy\_price** | **sell\_price** | **profit\_loss** | **balance\_before** | **balance\_after** | **rsi\_buy** |
| 43302.7 | 65300.63 | 2650.534162 | 8695.915606 | 8695.915606 | 60.3254496 |
| 41823.51 | 67840.51 | 4868.493168 | 8695.915606 | 8695.915606 | 54.58883938 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **macd\_buy** | **supertrend\_buy** | **rsi\_sell** | **macd\_sell** | **supertrend\_sell** |
| -358.5813169 | 40938.40564 | 48.20505312 | 4475.358781 | 71031.41143 |
| -592.0046004 | 39424.48288 | 60.73662338 | -76.90114467 | 69261.89873 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **rsi\_flag\_buy** | **macd\_flag\_buy** | **supertrend\_flag\_buy** | **rsi\_flag\_sell** | **macd\_flag\_sell** | **supertrend\_flag\_sell** |
| 0 | 1 | 1 | 0 | -1 | -1 |
| 0 | 1 | 1 | -1 | 0 | -1 |

### application\_logs

CREATE TABLE application\_logs(

user\_id INTEGER,

event\_type TEXT,

event\_time DATETIME,

comments TEXT,

FOREIGN KEY (user\_id) REFERENCES user\_profile(user\_id)

)

|  |  |  |  |
| --- | --- | --- | --- |
| **user\_id** | **event\_type** | **event\_time** | **comments** |
| 1 | LOGIN | 2025-04-03 13:27:55.958881+04:00 | User logged in. |
| 1 | BALANCE\_UPDATE | 2025-04-03 13:28:07.255059+04:00 | topup = 2000.0 total balance = 2000.0 |
| 1 | LOGIN | 2025-04-03 13:31:31.302698+04:00 | User logged in. |
| 1 | LOGIN | 2025-04-03 13:33:55.366730+04:00 | User logged in. |

### risk\_thresholds

CREATE TABLE risk\_thresholds (

risk\_level INT PRIMARY KEY, -- 1 (Low), 2 (Medium), 3 (High)

invest\_thres INT, -- Investment threshold (%)

rsi\_oversold INT, -- RSI Oversold threshold

rsi\_overbought INT, -- RSI Overbought threshold

macd\_fast\_ema INT, -- MACD Fast EMA (short-term EMA)

macd\_slow\_ema INT, -- MACD Slow EMA (long-term EMA)

macd\_signal\_ema INT, -- MACD Signal EMA

supertrend\_atr\_period INT, -- ATR Period for Supertrend

supertrend\_multiplier FLOAT -- Supertrend Multiplier

)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **risk\_level** | **invest\_thres** | **rsi\_oversold** | **rsi\_overbought** | **macd\_fast\_ema** | **macd\_slow\_ema** | **macd\_signal\_ema** | **supertrend\_atr\_period** | **supertrend\_multiplier** |
| 1 | 30 | 20 | 80 | 26 | 52 | 18 | 20 | 3.5 |
| 2 | 60 | 30 | 70 | 12 | 26 | 9 | 14 | 3 |
| 3 | 90 | 40 | 60 | 6 | 13 | 4 | 7 | 2 |

### bitcoin\_config

CREATE TABLE bitcoin\_config (

config\_code TEXT,

config\_value TEXT

)

|  |  |
| --- | --- |
| **config\_code** | **config\_value** |
| SALT\_KEY | od029gn2fb20 |
| API\_SECRET | NUqHbOQetrthl4JMcSRORxEFMgnQlctueiY |

### account\_history

CREATE TABLE account\_history (

user\_id INT,

account\_date DATETIME,

account\_balance FLOAT,

FOREIGN KEY (user\_id) REFERENCES user\_profile(user\_id)  
)

|  |  |  |
| --- | --- | --- |
| **user\_id** | **account\_date** | **account\_balance** |
| 1 | 2025-04-03 13:38:16.553656+04:00 | 2000 |
| 1 | 2025-04-03 13:49:50.506821+04:00 | 2200 |
| 1 | 2025-04-03 13:49:57.947803+04:00 | 2700 |
| 1 | 2025-04-04 22:56:26.289184+04:00 | 3000 |

## Image Files

### btblogo.png

### A grey and black circle with a person in the middle AI-generated content may be incorrect.profile.png

### true.png

### A red circle with a white x in it AI-generated content may be incorrect.false.png

## UI Implementation

### Login/Register Page

A screenshot of a login page

AI-generated content may be incorrect.

### Preference Page

A screenshot of a screen

AI-generated content may be incorrect.

### Home Page

A screenshot of a computer

AI-generated content may be incorrect.

### Profile Page

A screenshot of a graph

AI-generated content may be incorrect.

### A screenshot of a computer AI-generated content may be incorrect.Balance Page

### Trade History Page

A screenshot of a trade history

AI-generated content may be incorrect.

### A screenshot of a computer AI-generated content may be incorrect.Date Page

### A screenshot of a computer AI-generated content may be incorrect.Risk Page

### A screenshot of a graph AI-generated content may be incorrect.Bot Indicators Page

### Workstation Page

A screenshot of a graph

AI-generated content may be incorrect.

## GitHub Repository

I also uploaded my code to GitHub to track changes and compare different versions throughout the development process. This practice not only helped me monitor my progress and understand the impact of each modification, but also made debugging more efficient by allowing me to identify exactly when and where issues were introduced.

Here is the link to my GitHub: <https://github.com/SPBoss69/bitcoinbot>A screenshot of a computer

AI-generated content may be incorrect.

# Testing

Test video: <https://youtu.be/Q5ni4tGSIGM>

## Input Validation Tests

### Username

|  |  |
| --- | --- |
| Objective | Username Validation |
| Purpose of test | To check if the username entered is valid as per the rules in the code.  The first letter should be an alphabet  Username should be 8-12 characters long  No special characters allowed |
| Description of test | I will test the username with:  Starting with a number  Less than 8 and greater than 12 characters  Containing special characters |
| Test data | Testuser1 = “5alanjake”  Testuser2 = “alan”  Tesuser3 = “alanjakeharper”  Testuser4 = “alan@jake” |
| Expected Result | The program should display a error messagebox with the appropriate message |
| Actual result: Passed | Testuser1 result:    Tesuser2 result:A screenshot of a computer  AI-generated content may be incorrect.  Testuser3:  A screenshot of a computer  AI-generated content may be incorrect.  Testuser4:  A screenshot of a computer  AI-generated content may be incorrect. |

### Password

|  |  |
| --- | --- |
| Objective | Password Validation |
| Purpose of test | To check if the password entered is valid as per the rules in the code  At least 1 capital letter  At least 1 number  At least 1 special character  Minimum 8 characters and maximum 15 characters |
| Description of test | I will test the password with: Containing no capital letters  Containing no numbers  Containing no special charcters  Less than 8 and greater than 15 characters |
| Test data | Testpassword1 = “harper@123”  Testpassword2 = “Harper@alan”  Testpassword3 = “”Harper1234”  Testpassword4 = “Harp@2”  Testpassword5 = “Harper@123456789”  Note: For this test, I will remove the hiding of the password so that the test can be seen properly. |
| Expected Result | The program should display a error messagebox with the appropriate message |
| Actual result: Passed | Testpassword1 result:  A screenshot of a computer  AI-generated content may be incorrect.  Testpassword2 result: A screenshot of a computer  AI-generated content may be incorrect.  Testpassword3 result:  A screenshot of a computer  AI-generated content may be incorrect.  Testpassword4 result:  A screenshot of a computer  AI-generated content may be incorrect.  Testpassword5 result:  A screenshot of a computer  AI-generated content may be incorrect. |

### Add Money

|  |  |
| --- | --- |
| Objective | Balance Validation |
| Purpose of test | To check if the entered account balance is betweent the minimum and maximum values  The balance should be between 500 and 30000 |
| Description of test | I will test the balance with:  Less than 500  Greater than 30000 |
| Test data | Testbalance1: 200  Testbalance2: 32000 |
| Expected Result | The program should display a error messagebox with the appropriate message |
| Actual result: Passed | Testbalance1 result:  A screenshot of a computer  AI-generated content may be incorrect.  Testbalance2 result:  A screenshot of a computer  AI-generated content may be incorrect. |

### Trading Timeframe

|  |  |
| --- | --- |
| Objective | Date Range Validation |
| Purpose of test | To check if the entered to and from date is valid according to the rules  Date should be in YYYY-MM-DD  Dates should be valid  From date cannot be greater than To date  From date cannot be greater than current date  Date range cannot be greater than 60 days (so that number of API calls can be limited)  From date cannot be less than 2018-01-01 |
| Description of test | Test with:  Invalid date format  Invalid dates  From date>To date  From date or To date>Current date  Date range>60 days  From date < 2018 |
| Test data | Invalid date condition:  From = 10-06-2024  To = 10-07-2024  Invalid dates:  From = 2024-02-30  To = 2024-03-32  Fromdate>Todate:  From = 2024-04-01  To = 2024-03-01  From date or To date>Current date:  From = 2025-05-02  To=2025-06-02  Date range>60 days:  From = 2024-01-01  To = 2024-05-01  From date < 2018:  From = 2017-01-01  To = 2017-02-01 |
| Expected result | The program should display a error messagebox with the appropriate message |
| Actual result: Passed | Invalid date result:  A screen shot of a computer  AI-generated content may be incorrect.  Invalid dates result:  A screen shot of a computer  AI-generated content may be incorrect.  Fromdate>Todate result:  A screen shot of a computer  AI-generated content may be incorrect.  From date or To date>Current date result:  A screen shot of a computer  AI-generated content may be incorrect.  Date range>60 days result:  A screenshot of a computer  AI-generated content may be incorrect.  From date < 2018 result:  A screen shot of a computer  AI-generated content may be incorrect. |

## Exception Handling Tests

### Valid User

|  |  |
| --- | --- |
| Objective | User validity |
| Purpose of test | To check if the user exists in the database |
| Description of test | Test with non-existent user |
| Test data | alanharper1 |
| Expected result | Appropriate message should display |
| Actual result: Passed |  |

### Valid Password

|  |  |
| --- | --- |
| Objective | Wrong password |
| Purpose of test | To check if the password is correct or wrong |
| Description of test | Test with correct username but wrong password |
| Test data | Correct password: Harper1@  Test data (wrong) : Harper12 |
| Expected result | Appropriate message should be displayed |
| Actual result: Passed | A screenshot of a login box  AI-generated content may be incorrect. |

### Database Exception

|  |  |
| --- | --- |
| Objective | To recover from database disconnection |
| Purpose of test | To check if the program handles database disconnections and reconnects |
| Description of test | Log in, select preference, then rename the database name from bitcoin.db to eth.db to simulate the error. Click home page and other pages to verify the program is not crashing. Then revert back to the original file name to see if it the program reconnects |
| Expected result | The program should not crash and should recover from database disconnections |
| Actual result: Passed | Connecting to database...  Connected to database successfully.  Database error: attempt to write a readonly database |

### API Exceptions

|  |  |
| --- | --- |
| Objective | Handling API Errors |
| Purpose of test | To check if the program catches API errors and continues |
| Description of test | Disconnect the network to force the API calls to fail and to verify if the program would catch the exception and display proper messages or empty graphs |
| Expected result | Graphs should be empty  And error message should be displayed  The error will also be logged in the terminal |
| Actual result: Passed | Request error: HTTPSConnectionPool(host='api.binance.com', port=443): Max retries exceeded with url: /api/v3/ticker/price?symbol=BTCUSDT (Caused by NameResolutionError("<urllib3.connection.HTTPSConnection object at 0x1316a42c0>: Failed to resolve 'api.binance.com' ([Errno 8] nodename nor servname provided, or not known)"))  Not enough data to plot the chart |

|  |  |
| --- | --- |
| Objective | Handling API Errors |
| Purpose of test | To check if the program recovers from network disconnection, and runs properly |
| Description of test | Disconnect the network to force the API calls to fail.  The connect back to the network to see if the program is able to fetch data from API. |
| Expected result | Graphs should be displayed properly  Main functionality of the bot should be working |
| Actual result: Passed |  |

## Functionality Tests

### Register User

|  |  |
| --- | --- |
| Objective | Successful Registration |
| Purpose of test | To check if the registration process works |
| Description of test | I will register as a new user |
| Test data | Username – alanharper1  Password – Harper1$ |
| Expected result | Appropriate message should be displayed |
| Actual result: Passed | A screenshot of a login box  AI-generated content may be incorrect. |

### Trading Preference

|  |  |
| --- | --- |
| Objective | Successful Functionality of Preference Page |
| Purpose of test | To check if the page works properly |
| Description of test | Check that page opens with intended content and buttons work |
| Expected result | Should work appropriately |
| Actual result: Passed |  |

### Display Profile

|  |  |
| --- | --- |
| Objective | Successful Functionality of Profile Page |
| Purpose of test | To check if the page works properly |
| Description of test | Check that page opens with the eqiuty graph and that buttons work |
| Expected result | Should work appropriately |
| Actual result: Passed | A white background with black text  AI-generated content may be incorrect.  (There is no graph as the account balance hasn’t grown as it’s a new user. Here is the graph for an existing user)  A screenshot of a graph  AI-generated content may be incorrect. |

### Display Trade History

|  |  |
| --- | --- |
| Objective | Successful Functionality of Trade History Page |
| Purpose of test | To check if the page works properly |
| Description of test | Check that page opens with the trade history table that’s interactive and buttons work |
| Expected result | Should work appropriately |
| Actual result: Passed | A white background with black text  AI-generated content may be incorrect.  (There is no table as there are no completed trades as it’s a new user. Here is the table for an existing user)  A screenshot of a trade history  AI-generated content may be incorrect. |

### Balance Update

|  |  |
| --- | --- |
| Objective | Successful Updating of Balance |
| Purpose of test | To check if the money can be added in the text box and is correctly added to the account balance. And that all buttons work |
| Description of test | I will add 2000$ |
| Test data | 2000$ |
| Expected result | Balance should update when checked in the profile page |
| Actual result: Passed | A screenshot of a computer  AI-generated content may be incorrect.  A screenshot of a profile  AI-generated content may be incorrect. |

### Indicator Graphs – Live

|  |  |
| --- | --- |
| Objective | Successful Displaying of Indicator Graphs |
| Purpose of test | To check if the graphs are displayed correctly |
| Description of test | I will select live trading, level 2, and open indicator graphs |
| Expected result | Live graphs and indicators should be displayed |
| Actual result: Passed | A screenshot of a graph  AI-generated content may be incorrect. |

### Indicator Graphs – Historical

|  |  |
| --- | --- |
| Objective | Successful Displaying of Indicator Graphs |
| Purpose of test | To check if the graphs are displayed correctly |
| Description of test | I will select historical trading, enter timeframe of 2025 jan to 2025 feb, select level 2, and open indicator graphs |
| Expected result | Graphs and indicators for the entered timeframe should be displayed |
| Actual result: Passed | A screenshot of a graph  AI-generated content may be incorrect. |

### Activate Bot – Live

|  |  |
| --- | --- |
| Objective | Successful fuctionality of activating the bot in live trading |
| Purpose of test | To check if the bot works as intended |
| Description of test | I will select live trading, select level 2, and activate the bot. |
| Expected result: | The bot should open the workstation page and display the graph and all the metrics. It wont necessarily buy or sell. If the bot doesn’t find an entry, the page will stay still. It has completed its analysis for today. The next analysis will take place in 10min. The graph should be interactive, displaying the mini table with OHLC for the candle that is hovered on. |
| Actual result: Passed |  |

### Activate Bot – Historical

|  |  |
| --- | --- |
| Objective | Successful fuctionality of activating the bot in historical trading |
| Purpose of test | To check if the bot works as intended |
| Description of test | I will select historical trading, enter the timeframe 2024 Feb to 2024 March, select level 2, and activate the bot. |
| Expected result | The bot should open the workstation page and display the graph and all the metrics. It should run a simulation, going through each day in the timeframe and analysing when it’s a good day to buy and then sell. The graph should be interactive, displaying the mini table with OHLC for the candle that is hovered on. |
| Actual result: Passed | A screenshot of a graph  AI-generated content may be incorrect. |

### Application Log

|  |  |
| --- | --- |
| Objective | Application Log functionality |
| Purpose of test | To see if actions are recorded in the application logs table |
| Description of test | Log in and update balance |
| Expected result | The application logs table should have new entries |
| Actual result: Passed |  |

## Trading Tests

### Bullish Trend Test 1

|  |  |
| --- | --- |
| Objective | Bot performance under favorable conditions |
| Purpose of test | To see if the bot performs well under bullish conditions |
| Description of test | Enter the timeframe November 2024 to December 2024 (one of the bullish trends in 2024) |
| Expected result | The bot should make a profit |
| Actual result: Passed | A screenshot of a graph  AI-generated content may be incorrect.Made a profit |

### Bullish Trend Test 2

|  |  |
| --- | --- |
| Objective | Bot performance under favorable conditions |
| Purpose of test | To see if the bot performs well under bullish conditions |
| Description of test | Enter the timeframe February 2024 to March 2024 (one of the bullish trends in 2024) |
| Expected result | The bot should make a profit |
| Actual result: Passed | A screenshot of a graph  AI-generated content may be incorrect.  Made a profit |

### Bearish Trend Test 1

|  |  |
| --- | --- |
| Objective | Bot performance under unfavorable conditions |
| Purpose of test | To see if how the bot performs under bearish conditions |
| Description of test | Enter the timeframe June 2024 to July 2024 (one of the bearish trends in 2024) |
| Expected result | Bot should not enter a long position. Should wait for the next good entry point. |
| Actual result: Passed | A screenshot of a graph  AI-generated content may be incorrect.  Didn’t enter a buy position |

### Bearish Trend Test 2

|  |  |
| --- | --- |
| Objective | Bot performance under unfavorable conditions |
| Purpose of test | To see if how the bot performs under bearish conditions |
| Description of test | Enter the timeframe April 2022 to May 2022 (bearish trend in 2022) |
| Expected result | Bot should not enter a long position. Should wait for the next good entry point. |
| Actual result: Passed | A screenshot of a graph  AI-generated content may be incorrect.  Didn’t enter a buy position |

# Evaluation

## Objectives

|  |  |  |
| --- | --- | --- |
| Objective | Y/N | Comments |
| Develop an Intuitive User Interface with Relevant Features | N | While I did make a minimalistic interface, I could’ve added more features to increase the user engagement. I also should’ve added more design to make it easier to navigate for beginners. |
| Get Bitcoin Price Data from Binance API | Y | Connected to the Binance Testnet API and got all relevant data using requests |
| Make Algorithms for Technical Indicators | Y | Made functional algorithms for RSI, MACD and Supertrend which are used in technical analysis and graph production |
| Sustainability of Code and Readability | Y | Followed clean code principles by using clear variable names and breaking logic into modular functions. I also added comments to ensure the code is easy to maintain and understand for future users or collaborators. |
| Use of SQL Databases | Y | Used an SQLite database to store all relevant data such as user profiles, trade history, etc. |
| Applying a Risk Level System | Y | Applied risk level system, that affects trading decisions |
| Choosing Between Live and Historical Trading | Y | Included an option to select between live and historical trading so that users can see how the bot would perform in historical and live conditions |
| Trade Reasoning (Educational Feedback) | N | While the trade histroy does include the flags for different indicators, it does not really explain to the user why the trade happens. I could’ve added a description feature than explains in detail why each trade was exectued instead of just flags |

## Unmet Objectives

The few objectives that were not met could have made a significant impact on the platform’s usability and experience if met. The UI was too over simplified and minimalistic which didn’t allow the addition of many potential useful features. These are documented after this section. When oversimplifying anything, there’s always a loss of something no matter how much easier it is to use the platform. Therefore, moving forward, planning an UI that’s minimalistic but still feature rich enough so that beginner user enjoy the experience and can explore the different options on the platform is something that I’ve taken away from this. Oversimplifying is not always beneficial for the customer. Furthermore, I included indicator flags rather than the explanation of a trade from the bot’s reasoning in text form. If this text was added to the platforms, users could easier understand the trade reasoning. This is because a beginner is less likely to grasp lessons and reasoning logic from a trade history table with numbers compared to just reading the reason. Hence, another addition to this bot in the future would be to add explanations in word form rather than number form and potentially including generative explanations using AI so that it’s adaptive as well.

## Client Review and Feedback

This review is from Ashish Yeruva (one of the clients I interviewed at the beginning of my project):

There should be a way to view the indicator values via on-hover tooltip.

In the historical trading page, the start-trading button is misleading because you aren't trading anything, you are just seeing how the bot did in the past. So, a button called "View Back Test" or something like that would be more appropriate.

You should also display buy and sell arrows on the back test chart to see where it bought and where it sold rather than showing it numerically like it did on trading view.

There are 3 indicators, which are 3 different methods of trying to identify whether the graph is in an uptrend or in a downtrend. Ideally you want to be invested in an uptrend and not invested in a downtrend. To get the best signal, the bot takes an average of the 3 indicators. When 2/3 indicators believe it's an uptrend, the bot buys, when 2/3 indicators believe it's a downtrend, the bot sells. The user can benefit from an objective emotionless, and back testable mode of trading. They can understand what kind of moves the bot is able to capture well and what moves it isn't to make an informed decision about what investing style is best for them, what risk tolerance they are comfortable with and what timeframe they want to trade over (day-trading, medium-term, long-term investing).

They can also compare the trades a bot would have made to the trades they might have made and practice trying to refine their own methods of market analysis.

Back test is a very useful feature.

The ability to see the equity calculation is very useful, as it allows you to see visually your portfolio increases and decreases because there's a big difference between price moving up 20% vs seeing how that affects your portfolio. It's very difficult to visualize compounding and things like this so equity calculations are powerful

## Improvements

While the project has met its goals of introducing users to the world of trading Bitcoin, through a simple way, allowing them to select preferences, adjusting risk, and viewing technical analysis, there are still important areas where the platform could be improved. These changes will help create a more intuitive and user-friendly platform if this were to be visited again.

It would also be really helpful for the users if they could see the indicator values directly on the chart through hovering over that area. Also adding an option to merge the different indicators with the actual BTC chart would better help visualize the trends and crossovers. This would enhance their understanding of the bot’s logic without needing to manually go through each indicator graph.

While in the process of live or historical trading, it would be a good UI addition to ad buy/sell arrows on the chart at the respective candles when the bot has decided to buy/sell.

Also the button names should actually correlate to what it does. For example, the “Start Trading” button on the historical trading setting should be renamed to “View Backtest”. This is because nothing is actually being traded. It’s only running the algorithm on past data. This will reduce user confusion and reflect the actual functionality of each button.

Implementing these UI features would not only enhance the user experience but also make the platform more accessible to beginners who may not be familiar with raw technical analysis.

One important issue in the logic of the historical trading that was not accounted for was that if the bot initiates a buy order, but the timeframe ends before a sell order happens, the trade remain unfinished and causes a problem. This could make issues in the portfolio values and give false information about the bot’s performance.

To fix this, the bot should be modified in such a way that if sell signal has not yet been created after a buy signal within the entered timeframe, the timeframe is automatically extended until the bot exits the posision. This makes sure that all the trades are fully completed within the backtesting simulation and portfolio calcualtions remain meaningul.

By exposing more of the bot’s inner workings to the user, through clear visuals, and explanations of indicator decisions, users can better understand how the bot operates. This will help them compare its strategy to their own (when they start to make their own), and ultimately build more confidence in their trading approach.

If I were to start a new project, I would be mindful of the planning process and how I use my time. When working on this project, I found myself wasting a huge amount of time thinking about what to include and planning how the the platform should work, rather than actually working on the platform. I did not work on this project consistently throughout the alotted time which I know is crucial for success in the technology and programming industry. Therefore, in the future, I should be more efficient in my work, doing rather than wasting more than necessary time planning it, and putting the work each day consistently, eager to learn something new everyday.