Case Study: Automating Trading Data Collection and Interaction through Flask Web Application

Background:

In the financial sector, quick decision-making based on real-time data is crucial. A trading company dealing with mergers and acquisitions wanted to streamline their data collection and automation processes. The company used a trading platform, Takion, to manage their trades but had to manually gather data from various sources, including the platform's interface. The client needs a solution to automate the collection of trading data, perform real-time analysis, and enhance the process of executing trade decisions.

Objective

The goal of this project was to:

- The objective of this project was to automate the collection, processing, and storage of trading Automate the collection of trading data from a trading platform (Takion).
- Streamline data entry using an easy-to-use web form.
- Use OCR (Optical Character Recognition) to extract real-time data from the platform.
- Store and manage all trading data in a database for easy access.
- Allow users to filter and search trading records quickly.
- Improve efficiency and accuracy in decision-making for the trading team.

Tech Stack

Programming Language:

Python: Used for automating data collection, processing, and managing the web application.

Libraries:

- Flask: Used for building the web application and API to handle user interactions and manage trading data.
- SQLAlchemy: For handling database interactions, including saving, querying, and deleting trading data.
- PyAutoGUI: Used for automating interactions with the trading platform (Takion), like opening the application, typing, and clicking buttons.
- Pytesseract: For optical character recognition (OCR) to extract text from images or screen regions, particularly from Takion.
- Pillow (PIL): To handle image enhancements (like improving contrast) before OCR text extraction.
- NumPy: Assists in numerical operations, such as manipulating and calculating trading data values.
- OpenCV: Used for template matching on screen (to locate and interact with specific elements within Takion).
- Regex: Used for extracting decimal numbers from the OCR results.

Framework:

Flask: Used to build the web server for interacting with the database and handling trading data through a user-friendly interface.

Database:

MySQL: Used to store and manage trading data securely and efficiently.

Other Tools:

Flask-Migrate: For managing database migrations and schema changes.
pyautogui: For automating interactions on the desktop, such as clicking and typing in Takion.

Application Overview

The company developed a Flask-based web application that integrated with Takion's trading platform and automated several processes. The solution involved the following key components:

Web Interface for Data Entry: A custom web form powered by Flask allowed the user to enter trading data manually, such as acquirer stock, target stock, ratios, and more. The data was stored in a MySQL database using SQLAlchemy, ensuring that all entries were tracked over time.

Automated Data Collection and Processing: A Python script using pyautogui and pytesseract was developed to interact with the trading platform. The system automated the following:

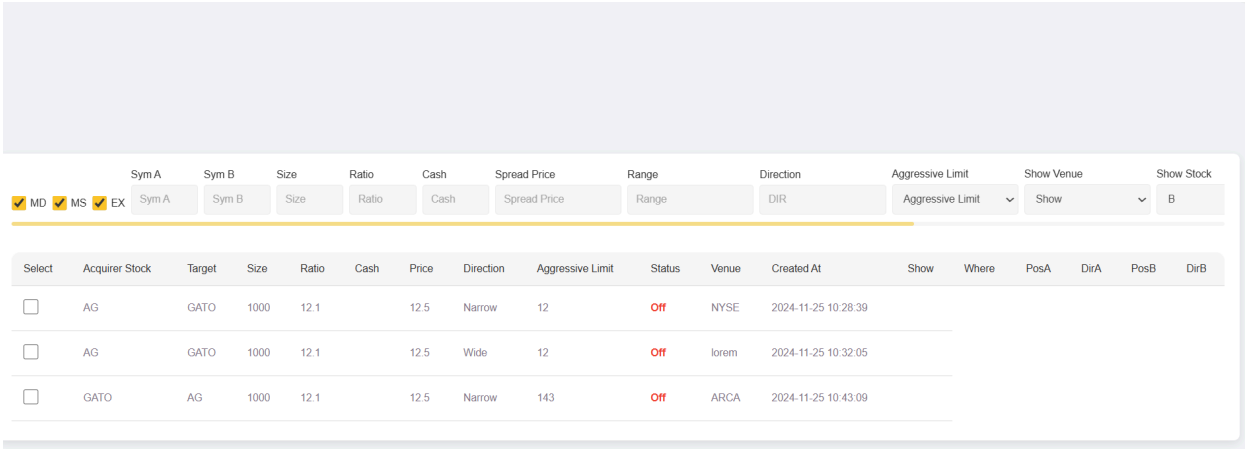
Login Automation: The application used pyautogui to automate the login process, reducing human error and ensuring that the process was repeated without variation.
Template Matching for Interaction: By utilizing template matching techniques (cv2.matchTemplate), the application identified and clicked on predefined areas of the trading platform, making it possible to extract data automatically.
OCR for Real-Time Data Extraction: The system captured screen regions and used OCR to extract trading-related data, such as financial ratios and stock prices, from the Takion interface in real-time.
Data Storage and Retrieval:

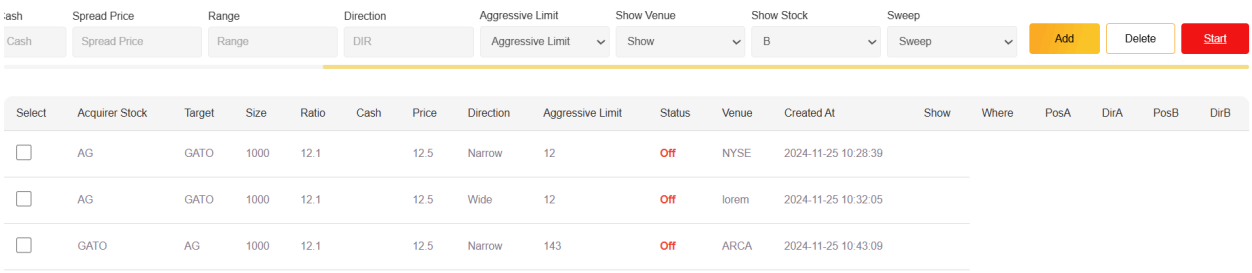
SQLAlchemy and Flask provided a backend interface to store and retrieve trading data. The TradingForm model handled data entries for acquirer stock, target, ratios, and other trade-related parameters.
The system allowed users to filter data via a dynamic search form, making it easy to view relevant data points and filter them based on specific criteria (e.g., acquirer stock, target, price).
CRUD Operations: The application supported basic CRUD operations, including creating new data entries, updating existing records, retrieving data via API calls, and deleting unwanted entries.
Scheduled Data Extraction: To ensure continuous monitoring and extraction of data from the platform, the system used Python’s Timer class to repeatedly capture and analyze the information extracted via OCR. This approach allowed the system to run indefinitely, capturing relevant data without requiring manual intervention.

User Interface: The data was displayed in a web interface with real-time updates, making it easy for users to interact with the data. The system offered:

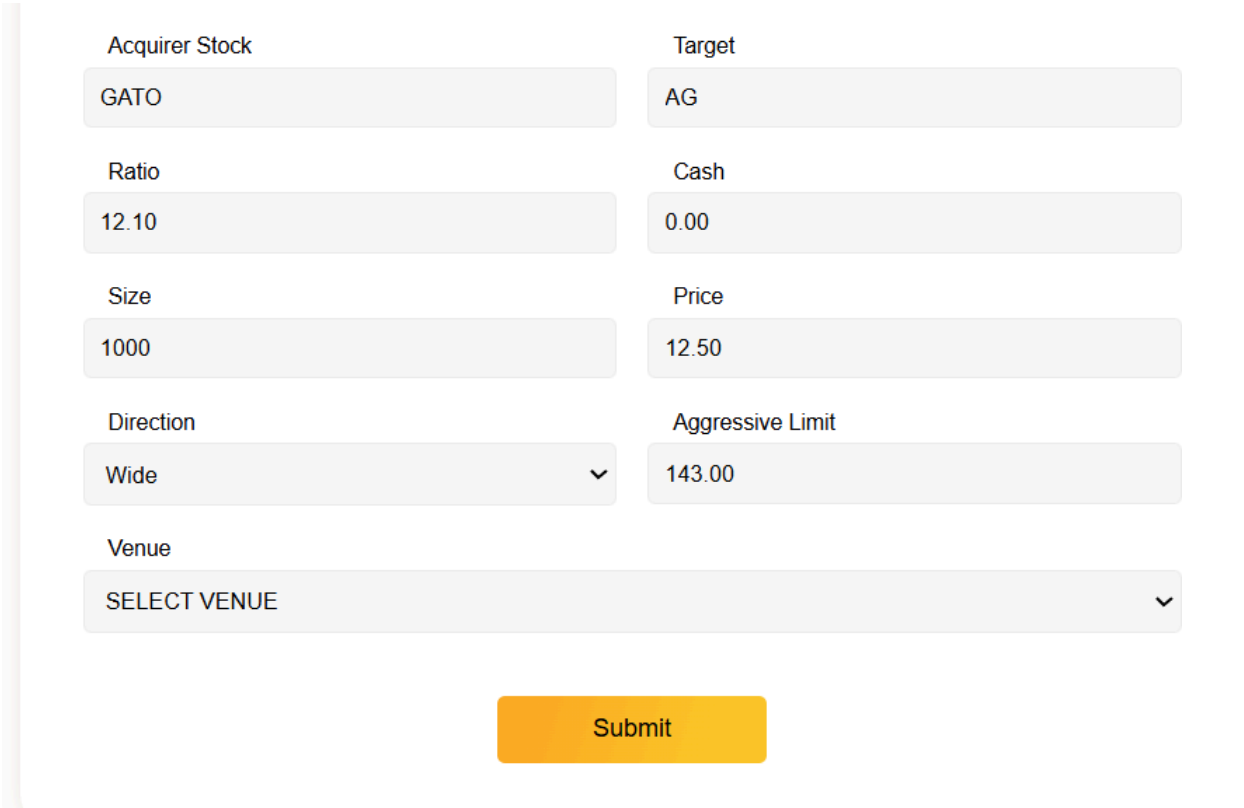
A searchable table for users to filter trading records by stock, price, ratio, etc.
Data pre-fill: The system could also pre-fill forms with the most recent entries, minimizing user effort when entering repetitive data.Below are the screenshot of the UI

DASHBOARD





Form to get details of stocks



Detailed Workflow Breakdown

Detailed Workflow Breakdown:

This Flask-based application, combined with several Python libraries and tools, is designed to automate the management of trading data and perform trading-related operations. Let's break down the workflow of each key component and its interactions:

---

1. Flask Web Application Setup:

- **Flask App Initialization**:
    - Flask(_name_) initializes the Flask application, creating an object app that will serve as the core of the web server.
    - **SQLAlchemy**: The app uses SQLAlchemy for managing database operations, specifically to interact with MySQL, storing trading form data in the TradingForm model.

2. Database Integration:

- **SQLAlchemy Configuration**:
    - The database URI is set in the Flask app config: 'mysql://root:root@localhost/takion_tr'. This connects the app to a MySQL database where the trading data will be stored.
    - **db.Model**: The TradingForm class defines the structure of the trading data (like acquirer stock, target, ratio, cash, size, price, etc.). It maps to the database table trading_form.
- **Migration**:
    - Flask-Migrate handles schema changes and migrations for the database.
    - **Migrate** connects Flask to the migration engine and allows the database schema to be updated seamlessly.

3. Trading Form API & Web Routes:

- **TradingFormAPI (MethodView)**:
    - **GET request**: When a GET request is made to /api/save_form, the get method fetches the most recent form data from the database and renders a form pre-filled with that data. If no data exists, it renders a blank form.
    - **POST request**: The POST method saves new data entered into the form. When the form is submitted, it stores the data in the TradingForm table.

- **show_data Route**:
    - Displays all stored trading data from the TradingForm table on the web page.
- **get_trading_data Route**:
    - This route allows retrieving all trading data in JSON format. It includes logic for filtering, where parameters (like acquirer_stock, ratio, etc.) can be passed in the URL to filter results.
- **search Route**:
    - Allows the frontend to filter data dynamically by applying various filters to the SQLAlchemy query, such as acquirer_stock, target, ratio, etc.
    - The data is then returned as a JSON response.
- **delete_data Route**:
    - Handles the deletion of selected rows from the database. It takes a list of IDs and deletes the corresponding rows.

4. Automation (Interacting with Takion):

This part involves automating trading tasks using **PyAutoGUI**, **OCR** (via pytesseract), and **OpenCV** to interact with a trading platform (Takion).

Steps:

- **VPN and Takion Automation Route**:
    - **Open Takion**: The script uses pyautogui to open Takion, type in the password, and log into the application.
    - **Open VPN Client**: The script begins by using pyautogui to open the VPN client, search for the "Lightpath" option, and interact with it by double-clicking. Once located, it types in the username and password to log in to the VPN.
    - **Login Automation**: After opening the VPN client and logging in, the script proceeds to open the Takion application. It uses pyautogui to input the Takion login credentials, entering the username "bigc" and password.
    - **Template Matching**: The find_and_click_template function uses **OpenCV** to locate specific templates (e.g., a green rectangle) on the screen and interact with it. It clicks on specific regions of the screen using coordinates found via template matching.
    - **Search and Extraction**: Once Takion is open, it searches for specific stock names and extracts financial data from defined regions using **OCR** (via pytesseract and Pillow).

OCR Processing:

- **capture_and_extract**:
    - Captures a screenshot of a defined region of the screen.
    - Converts the image to grayscale, enhances its contrast using Pillow, and then uses pytesseract to extract text (numbers) from the image.

    - Regular expressions (re.findall) are used to extract any decimal numbers from the OCR text, representing stock prices or trading information.
- **Continuous Timer**:
    - The script uses Timer (from Python's threading library) to call the OCR extraction periodically (every second), simulating real-time data collection from the Takion platform.

5. Trading Strategy (Merger Arbitrage):

- **MergerArbitrage Class**:
- The process starts when user click on the start button on theUI
    - The main class that handles the logic for merger arbitrage trading, specifically for the acquirer and target stock.
    - **Real-Time Price Fetching**: This would ideally involve fetching real-time stock prices from an API (though here it's a placeholder).
    - **Spread Calculation**: It calculates the spread based on whether it's a cash and stock deal or an all-stock deal.
        - **All-stock deal**: The spread is calculated by multiplying the acquirer's bid price by the exchange ratio and subtracting the target's offer price.
        - **Cash component deal**: If there's a cash component, it adds that value to the spread calculation.
- **Placing Orders**:
    - The place_order method simulates placing limit orders on the acquirer and target stocks, using an external API (though it's just printed in the placeholder code).
- **Handling Order Execution**:
    - It manages partial order fills by checking the remaining size and placing new orders if necessary.
- **Order Direction**:
    - Based on whether the spread is **narrowing** or **widening**, the system places buy or sell orders accordingly.
    - It checks the spread against the **narrow** or **wide** threshold, and depending on the condition, it places short or long orders on the acquirer and target stocks.
- **Continuous Loop**:
    - The run method continuously monitors real-time price changes, calculates the spread, and places orders based on the strategy logic.

6. Timer & Automation:

- **Timer for Continuous Operation**:
    - Both the capture_and_extract function and the trading strategy are set to run periodically (e.g., every 1 second), ensuring the system continuously monitors trading data and performs automated tasks such as capturing screen data and placing trades.

7. Final Workflow:

1. **User Interaction**:
    - The user accesses the web interface to view or submit trading forms.
2. **Data Storage**:
    - Submitted data is stored in the MySQL database via SQLAlchemy.
3. **Automation**:
    - The backend automates interactions with the Takion platform using pyautogui for GUI automation, pytesseract for OCR-based text extraction, and OpenCV for screen template matching.
4. **Trading Strategy**:
    - The MergerArbitrage class calculates spreads and places buy or sell orders based on real-time data fetched from the trading platform.
5. **Continuous Monitoring**:
    - The system continuously runs both the UI for managing trading data and the background automation for handling live data collection and trading.