‘WatchCord’

Research Project
Sponsored by
Silver Oak University

Submitted by
Harsh Awasthi
Jani Kathan Gopal
Jain Hardik
Pathan Danish Khan

Computer Engineering
Silver Oak College of Engineering and Technology

Silver Oak University

Silver Oak College of Engineering & Technology

Department of Computer Engineering

CERTIFICATE

This is to certify that the project entitled **"WatchCord"** has been carried out by **"HARSH AWASTHI (2101030400089)"** under my guidance in fulfillment of the **Minor Project** (1010043494) Subject of Bachelor of Engineering in **Computer Engineering** – **7th Semester** of Silver Oak University, Ahmedabad during the academic year **2024-2025**.

A/Prof. Parimal Patel
Internal Guide

Dr. Satvik Khara
Head of Department

SILVER OAK UNIVERSITY
EDUCATION TO INNOVATION

Silver Oak College of Engineering & Technology

Department of Computer Engineering

CERTIFICATE

This is to certify that the project entitled **"WatchCord"** has been carried out by **"JANI KATHAN GOPAL (2101030400101)"** under my guidance in fulfillment of the **Minor Project** (1010043494) Subject of Bachelor of Engineering in **Computer Engineering** – **7th Semester** of Silver Oak University, Ahmedabad during the academic year **2024-2025**.

A/Prof. Parimal Patel
Internal Guide

Dr. Satvik Khara
Head of Department

SILVER OAK UNIVERSITY
EDUCATION TO INNOVATION

Silver Oak College of Engineering & Technology

Department of Computer Engineering

CERTIFICATE

This is to certify that the project entitled **"WatchCord"** has been carried out by **"JAIN HARDIK (2101030400099)"** under my guidance in fulfillment of the **Minor Project** (1010043494) Subject of Bachelor of Engineering in **Computer Engineering** – **7th Semester** of Silver Oak University, Ahmedabad during the academic year **2024- 2025**.

A/Prof. Parimal Patel
Internal Guide

Dr. Satvik Khara
Head of Department

SILVER OAK UNIVERSITY
EDUCATION TO INNOVATION

Silver Oak College of Engineering & Technology

Department of Computer Engineering

CERTIFICATE

This is to certify that the project entitled **"WatchCord"** has been carried out by **"PATHAN DANISH KHAN (2101030400056)"** under my guidance in fulfillment of the **Minor Project** (1010043494) Subject of Bachelor of Engineering in **Computer Engineering** – **7th Semester** of Silver Oak University, Ahmedabad during the academic year **2024-2025**.

A/Prof. Parimal Patel
Internal Guide

Dr. Satvik Khara
Head of Department

ACKNOWLEDGEMENT

We are heartily thankful to our supervisor, A/Prof. Parimal Patel, whose encouragement, supervision and support from the preliminary to the concluding level enabled me to develop an understanding of the subject. At the end, we offer my regards and blessings to all of those who supported us in any respect during the completion of the project and to our college for providing resources and materials.

We would like to extend our gratitude to Dr. Satvik Khara, head of Computer Engineering Department, Silver Oak college of Engineering and Technology, Ahmedabad, for his continuous encouragement and motivation.

Last but not the least we would like to mention here that we are greatly indebted to each and everybody who has been associated with our project at any stage but whose name does not find a place in this acknowledgement.

Yours Sincerely,

Harsh Awasthi (2101030400089)

Jani Kathan (2101030400101)

Jain Hardik (2101030400099)

Pathan Danish Khan (2101030400056)

SILVER OAK UNIVERSITY
EDUCATION TO INNOVATION

INDEX

Sr NO	Content	Page No
01	Abstract	1
02	Introduction	2
03	Objective(s)	4
04	Methodology	5
05	Project Outcome(s)	30
06	Reference(s)	32

Technical Details of Project

1. **Abstract**
2. **Introduction**
3. **Objective**
4. **Methodology**
    - **Requirements Analysis**
    - **System Design**
    - **Software Development**
    - **Algorithm Development**
    - **Integration and Testing**
    - **Iterative Improvement**
    - **Photos**
5. **Outcomes**
6. **References**

1. ABSTRACT

Our solution aims to improve the situation in the following areas

Keeping track of one's favorite products can be quite a chore, and usually pretty annoying. A common problem everyone faces in their day-to-day lives is often missing out on some steal-deals due to a lack of attention. The aim of this project is to basically solve this very problem, along with providing several other features that would come in handy pertaining to situations like these. The basic idea is to create a Discord bot, that would allow a user to enter the URL of a product's webpage, which the bot will then web scrape, and start tracking, by scraping it periodically. The user can set a target price which the bot will respond to, by pinging the user or discord role that a sale has started for the required price. The bot also includes other useful features like utilizing platform-specific APIs wherever possible, for real-time data fetching, product history, checking currently tracked products, allowing search & filter commands, data logging for bot enhancement, automation scripts, etc. It also aims to include a machine learning model for product recommendation, based on the logged user-interaction data. Finally, a small, handy GUI based admin panel for accessing the bot's functionalities other than via textual commands. Having all these features offered by a single Discord bot is a uniquely convenient and consumer-friendly idea. Being able to track and gather some information at the convenience of one's discord server, and allowing the server members straightforward features with some form of customizability can prove to be quite an interesting project that actually helps save time and effort along with avoiding tedious scenarios.

2. INTRODUCTION

In today's fast-paced world, keeping track of favourite products and finding the best deals can be a tedious and time-consuming task. Many people miss out on attractive offers due to the sheer volume of information and lack of attention to detail. This project aims to address this problem by developing a comprehensive Discord bot that helps users track product prices easily and conveniently. The bot allows users to input the URL of a product's webpage, and it then scrapes the page periodically to track price changes. Users can set a target price, and the bot will notify them when the product price drops to or below their specified threshold, providing an efficient and personalized price-tracking solution.

This bot goes beyond basic tracking by integrating several additional features that enhance its functionality. These include utilizing platform-specific APIs if available, for historical data retrieval, logging product history, and offering search and filter commands to help users find and manage their tracked items easily. Furthermore, it aims to incorporate a machine learning algorithm to recommend products based on user interaction data. To simplify management, a GUI-based admin panel will allow administrators to access the bot's features without relying solely on textual commands.

By combining these features in a single tool, the bot provides a convenient, consumer-friendly solution to tracking product prices and managing purchases efficiently. Users can now gather and track valuable product information within their Discord servers, saving time, avoiding missed opportunities, and automating the entire process with a customizable experience.

Key Features:

- **Automated Price Tracking:** Allows users to input product URLs and periodically scrape the pages to track price changes.
- **Customizable Alerts:** Users can set target prices, and the bot will notify them when a deal meets their set criteria.
- **Historical Data Integration:** Uses platform-specific APIs if available, to fetch historical product information.
- **Additional Functionalities:** Includes search and filter commands, user interaction logging, and more.
- **Machine Learning Integration:** To implement a recommendation model based on logged user interaction data.
- **GUI-Based Admin Panel:** A user-friendly graphical interface for administrators to manage bot functionalities beyond textual commands.

This project combines convenience, automation, and personalized experiences in a way that can save users time, effort, and missed opportunities, all within the comfort of their Discord server.

3.OBJECTIVE

- **Automated Price Tracking:** Develop a Discord bot that allows users to track product prices by inputting product URLs, using periodic web scraping to monitor changes and notify users when the price reaches their set target.
- **Data Integration:** Utilize platform-specific APIs to fetch product information, ensuring timely and accurate updates for users across different web platforms.
- **Customizable Notifications:** Implement features that allow users to set specific target prices and notify them via Discord pings.
- **Enhanced User Features:** Provide additional functionalities like search and filter commands for easier management of tracked products, logging product history for reference, and implementing automation scripts to customize user preferences.
- **Machine Learning Integration:** Incorporate machine learning algorithms to recommend products based on user interaction data and timestamp, making the bot not only a price-tracking tool but also a smart product discovery assistant.
- **User-Friendly Management:** Develop a graphical admin panel for easier bot management, allowing users and administrators to access features beyond textual commands, making the bot more accessible and user-friendly.
- **Data Logging for Improvement:** Ensure data logging of user interactions, enabling continuous enhancement of the bot's features based on user behaviour and feedback.

4.METHODOLOGY

4.1 REQUIREMENT ANALYSIS

The requirement analysis for this project involves identifying the functional and non-functional requirements needed to develop a comprehensive Discord bot for tracking product prices, sending notifications, and offering additional features such as search, filtering, and machine learning recommendations.

Functional Requirements:

1. **User Authentication:**
    - Users must be able to authenticate through Discord, linking their accounts for personalized product tracking.
2. **Product Tracking:**
    - Users can input a product URL, and the bot will periodically scrape the webpage for price updates.
    - Allow users to track multiple products simultaneously.
3. **Customizable Alerts:**
    - Users can set target prices, and the bot will notify them when the product price falls below or matches the set price.
    - Notifications should be delivered via Discord pings.
4. **Data Fetching:**
    - Use platform-specific APIs (where available) to fetch product data.
5. **Search and Filter Features:**
    - Allow users to search for tracked products by name, category, or price.
    - Provide filtering options based on price, availability, and discount percentages.

6. **Tracking History Logging:**
    - Enable users to view tracking history.
7. **Machine Learning Recommendations:**
    - Implement a machine learning model to recommend similar or related products based on user interaction data.
8. **GUI-Based Admin Panel:**
    - Provide a graphical interface for administrators to manage bot settings, access logs, and monitor tracked products beyond Discord's textual interface.
9. **Data Logging and Analytics:**
    - Store logs of user interactions and product tracking data to enable analytics and improvement of the ML model's functionality.

Non-Functional Requirements:

1. **Performance:**
    - The bot should be able to handle multiple concurrent users tracking numerous products without performance degradation.
    - Updates should be provided efficiently, with minimal delay for users receiving notifications.
2. **Scalability:**
    - The system should be scalable to support many users and products.
    - It should be designed to handle increasing amounts of user interaction data for machine learning algorithms.
3. **Security:**
    - Secure authentication methods must be implemented to protect user data and product tracking preferences.

- The bot should safeguard sensitive information like Discord IDs and product tracking history.

4. **Reliability:**
   - The bot must be available 24/7, with minimal downtime for users relying on product price tracking.
5. **Usability:**
   - The Discord bot should be intuitive and easy to use for both novice and advanced users, offering clear commands for tracking products and receiving notifications.
   - The GUI admin panel should be designed with user-friendly navigation and management features.
6. **Maintainability:**
   - The codebase should be modular and well-documented to enable future updates and enhancements, especially as new web platforms or features are integrated.
   - Logging should be structured in a way that allows for easy debugging and performance monitoring.

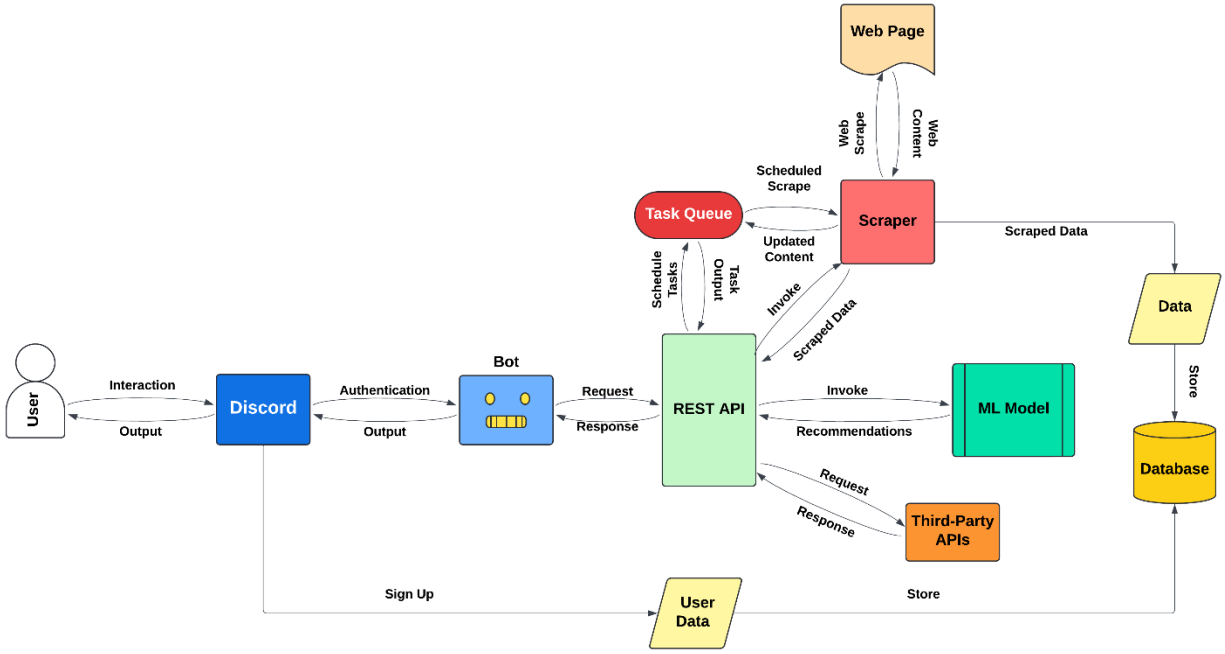
Technology Requirements:

1. **Frontend:**
   - Discord's messaging interface for primary bot interactions.
   - A GUI-based admin panel using web technologies like Tailwind CSS for additional functionality.
2. **Backend:**
   - Python for Discord bot development (using Discord.py).
   - Web scraping libraries like Scrapy for product data extraction.

- Platform-specific third-party APIs if available, integrations for historical data fetching (e.g., Amazon Product API).
- Databases like MongoDB to store product, user, and interaction data.

3. **APIs:**
   - REST APIs for fetching historic product data from web platforms.
4. **Machine Learning:**
   - PyTorch,, Scikit-learn or any other prominent ML-based library for building recommendation systems based on user behavior and interaction data.

4.2 SYSTEM DESIGN



4.2.1: System Design Diagram

The various system components and their respective behaviour is depicted in this diagram.

4.3 SOFTWARE DEVELOPMENT

1. User Interface and Interaction

User

- **Function:** Represents the end-user who interacts with the bot through Discord.
- **Interaction:**
    - Users can enter product URLs for tracking, set target prices, and customize their notification preferences.
    - Users receive alerts through Discord messages or pings when tracked products meet their specified conditions, providing updates without requiring them to check the site manually.

Discord

- **Function:** The bot uses Discord as the primary platform for user interaction and notifications.
- **Usage:**
    - **Alerts:** Sends real-time notifications about price drops, stock availability, and other important updates directly to the user through Discord messages.
    - **Commands:** Users can interact with the bot using text commands to manage their tracked products, configure settings, and retrieve information, making it user-friendly and accessible.

2. Data Collection and Processing

Web Page (Product Source)

- **Function:** The webpage serves as the primary source for web scraping, allowing users to input URLs of products they wish to track.
- **Features:**
  - **Data Scraping:** Users provide URLs of product pages from various e-commerce websites directly through Discord commands.
  - **Status Updates:** Users can receive real-time updates on the status of their tracked products via the Discord bot, including notifications of price changes or availability.

Scraper

- **Function:** This component is responsible for collecting product data from the provided product URLs.
- **Types:**
  - **Web Scraping:** Utilizes libraries such Scrapy to extract product information, including price, title, description and images from user-provided webpages.

Task Queue

- **Function:** Manages the scheduling and execution of scraping tasks.
- **Features:**
  - **Task Management:** Implements a queuing system (like Celery) to manage and prioritize scraping tasks based on user requests and intervals set by users for price checks.

- **Status Monitoring:** Users can check the status of their scraping tasks through Discord commands to ensure everything is functioning smoothly and that their tracked products are being monitored.

Scraped Data

- **Function:** Represents the raw data collected by the scraper, which is subsequently processed for user notifications and analysis.
- **Types of Data:**
  - Product details, including name, price, title, product description, and image URLs.
  - Historical pricing data that helps in determining pricing trends and informing users when to make a purchase.

Database

- **Function:** A structured collection of data used to store user profiles, tracked product information, and logs.
- **Types of Data Stored:**
  - **User Data:** Stores authentication credentials, preferences, and interaction history to provide a personalized experience.
  - **Tracked Data:** Holds raw and processed data for each tracked product, including timestamps for historical reference.

3. Core Logic

Bot

- **Function:** The bot serves as the main interface through which users interact with the system, executing commands and performing necessary tasks.
- **Capabilities:**
  - **Notification System:** Sends alerts to users via Discord whenever tracked products reach the desired price, ensuring that users don't miss out on deals.

REST API

- **Function:** Facilitates communication between the bot and other components, allowing different services to interact seamlessly.
- **Usage:**
  - **Data Exchange:** Enables the bot to retrieve data from the scraper and send processed data back to users. It allows for a modular architecture where components can operate independently yet communicate efficiently.

4. External Interactions

Third-Party APIs

- **Function:** External services that the bot can interact with to enrich its functionality and provide additional features.

- **Examples:**
    - **E-commerce APIs:** Accesses historical data about products, such as pricing, which can enhance the accuracy of scraped data.

5. Recommendations

ML Model

- **Function:** A machine learning model analyzes user interaction data to provide smarter product recommendations.
- **Applications:**
    - **Personalized Recommendations:** Uses logged user interactions to suggest products that may interest the user based on their past behaviour and preferences.

4.4 ALGORITHM DEVELOPMENT

1. OAuth2 for Authentication and Authorization

- **Objective:** Implement OAuth2 to securely authenticate users and authorize access to the bot's services, including tracking products, setting price alerts, and fetching historical data.
- **Implementation:**
    - OAuth2 Protocol: The bot uses OAuth2 to authenticate users, allowing them to securely log in using their Discord account.
    - Token Generation: After successful login, an access token is generated for the user. This token is used to authenticate all future requests from the user to the bot.
    - Scope Control: Different levels of access (scopes) are provided, ensuring that users can perform actions like setting product tracking or receiving notifications while maintaining data privacy.
    - Token Refreshing: OAuth2 handles refreshing tokens to ensure that the user remains authenticated without the need for frequent logins.

2. RESTful API for Bot Communication

- **Objective:** Provide an interface for external communication with the bot, allowing users to interact via Discord commands or the GUI, and enabling the bot to fetch product data or trigger actions.
- **Implementation:**
    - Endpoints Creation: Define RESTful API endpoints for key functionalities such as adding a product URL, setting price alerts, checking tracked products, and fetching product data.
    - HTTP Requests: Use standard REST methods (e.g., GET to fetch

tracked products, POST to set new alerts) to interact with the bot.

- Data Fetching: REST APIs are integrated with third-party platform-specific APIs if available, to fetch historical product data using product URLs, ensuring up-to-date information for users.
- Response Handling: The bot handles API responses and formats the data (e.g., product price) before sending it back to the user on Discord.

3. DistilBERT for Embeddings

- **Objective:** Use **DistilBERT**, a lightweight transformer model, to generate embeddings for product names and categories, enabling semantic understanding and product recommendation based on textual data.
- **Implementation:**
  - **Text Preprocessing:** Clean the product name and category data (e.g., remove special characters, lowercase text) before feeding it into DistilBERT.
  - **Embedding Generation:** Use DistilBERT to transform product names and categories into high-dimensional embeddings. These embeddings capture the semantic meaning of the product names and categories, making it possible to compare products based on their textual similarity.
  - **Storage:** Store the resulting embeddings in the dataset, ensuring they are accessible for future comparison or clustering tasks. These embeddings will have high dimensions after transformation.
  - **Use:** Whenever a new product URL is added, generate embeddings and store them in the system for future use in product recommendations.

4. PCA for Dimensionality Reduction of Embeddings

- **Objective:** Reduce the dimensionality of the product name and category embeddings generated by DistilBERT, improving efficiency and performance in downstream tasks.
- **Implementation:**
    - **Dimensionality Reduction:** After generating high-dimensional embeddings from DistilBERT, PCA (Principal Component Analysis) is applied to reduce these embeddings to a lower-dimensional space.
    - **Storage & Efficiency:** By reducing the dimensionality, the computational load is minimized for tasks like search, clustering, and storage, making the embeddings more efficient for comparison and retrieval.
    - **Preserving Information:** PCA ensures that most of the essential variance (information) in the original high-dimensional embeddings is retained in the lower-dimensional space.
    - **Usage:** The reduced embeddings can be used for faster computation and retrieval, especially when performing tasks like clustering products, similarity searches, or further indexing. This step ensures that the embeddings remain meaningful while significantly reducing the memory and processing requirements.

5. FAISS for Indexing and Fast Product Search

- **Objective:** Use **FAISS (Facebook AI Similarity Search)** to efficiently index and retrieve products based on the output column (tuple of (domain_name, product_id)), enabling quick similarity searches and recommendations.

- **Implementation:**
  - **Index Creation:** Build a FAISS index using the output column of the dataset. The output column stores tuples like (domain_name, product_id), and FAISS is used to enable fast similarity searches based on this column.
  - **Query Processing:** Whenever a user searches for similar products or the system needs to recommend products, FAISS is queried to find the closest matching entries in the output column.
  - **Approximate Nearest Neighbor (ANN) Search:** FAISS uses efficient algorithms to perform ANN searches, allowing the bot to retrieve relevant products quickly, even from a large pool of items.
  - **Product Addition:** New products can be added to the FAISS index as the dataset grows, ensuring that the system remains scalable.

4.5 INTEGRATION AND TESTING

4.5.1 INTEGRATION

1. User Interface and Interaction

This layer focuses on how users interact with the system and receive feedback.

- **User (End-User Interaction)**
  - Users initiate the process by entering product URLs and setting preferences (e.g., target prices) through **Discord** commands.
  - Once a product is being tracked, users receive notifications (like price changes) via the **Discord Bot**. These notifications make it easy for users to stay informed without having to manually check product pages.
  - Example: A user sends a Discord command to track an Amazon product. The bot confirms the request, and after scraping the relevant page, the user is notified if the product's price drops below their target.
- **Discord (Platform)**
  - Discord serves as the central interface for interaction. The bot responds to user commands, processes requests, and sends real-time alerts to users via Discord messages.
  - All data input (e.g., product URLs, target prices) and output (notifications, status updates) flows through this platform.

2. Data Collection and Processing

This layer focuses on scraping product information and managing tasks.

- **Web Page (Product Source)**
  - Users provide product URLs via Discord. The bot uses these URLs to scrape relevant product information, including the current price, title, and description.
  - The **Scraper** component kicks in to collect this data, which is triggered by the bot when a user adds a new product for tracking.
- **Scraper**
  - The scraper uses libraries like Scrapy to extract product details from the provided URLs. This includes product price, name, and description. If required, it also collects product images.
  - The scraped data is then passed to the next stage for processing or storage.
- **Task Queue**
  - To manage multiple user requests efficiently, the task queue component helps prioritize scraping tasks and ensures the bot runs in an organized manner.
  - Also manages scheduled scraping tasks.
  - For example, if many users are tracking different products at the same time, a system like Celery will manage which tasks are processed first and how frequently products are checked for updates (e.g., hourly or daily price checks).

- **Scraped Data**
    - The raw product data obtained from the scraper (such as price, description, and title) is processed and stored for further actions.
    - Helps users make informed purchasing decisions.

3. Core Logic

This layer manages bot functionality and communication between components.

- **Bot**
    - The Discord bot is the core logic layer responsible for receiving commands, triggering the scraper, and sending updates.
    - Once the product URL is scraped and relevant data is gathered, it sends a notification to the user.
    - Example: If a user sets a target price of 500 INR, the bot will monitor the product and only send an alert if the price drops below 500 INR.
- **REST API**
    - The bot uses REST APIs to communicate with other services or components like the **Scraper** or **Database**.
    - The API ensures smooth data flow between the scraper (responsible for gathering raw data) and the bot (which handles user requests and notifications).
    - This modular approach allows different parts of the system to function independently but still communicate efficiently.

4. External Interactions

This layer enhances the bot's functionality with external services.

- **Third-Party APIs**
  - The bot can interact with external e-commerce APIs (like the Amazon Product API) to fetch more accurate or detailed product data, such as historical pricing.

5. Recommendations

This layer focuses on improving user experience through smart suggestions.

- **ML Model**
  - The machine learning model helps make the system smarter by learning from user interactions. Over time, the model can suggest new products that the user might find interesting based on their past interactions (e.g., previously tracked items).
  - The ML model analyzes the user's historical behavior and the product data stored in the **Database** to generate personalized product recommendations.
  - Example: If a user has tracked multiple gaming accessories, the bot might recommend a new gaming mouse.

4.5.2 TESTING

Test Scenario: Product Tracking Bot Functionality

Objective: Verify the functionality of the product tracking bot on Discord, ensuring that it successfully tracks product prices and provides user notifications.

Test Steps:

1. **User Input via Discord**
    - **Action:** A user sends a command to the Discord bot, entering a product URL (e.g., a product page from Amazon).
    - **Expected Outcome:** The bot confirms receipt of the URL, acknowledging that it will begin tracking the product.
2. **Scraping & Task Queue**
    - **Action:** The bot queues the scraping task and invokes the scraper to fetch product information from the provided URL.
    - **Expected Outcome:** The scraper retrieves relevant details such as current price, product title, and description, confirming successful data extraction.
3. **Data Storage**
    - **Action:** The bot stores the scraped data in the database.
    - **Expected Outcome:** The database reflects the newly added product information, including the initial price and timestamp of when it was scraped.
4. **Price Monitoring**
    - **Action:** The bot periodically checks the scraped data for changes in price or availability.

- **Expected Outcome:** If the price drops to or below the target price set by the user, the bot prepares to send a notification.

5. **Recommendation**
   - **Action:** Over time, the ML model analyzes user interactions and preferences (e.g., products tracked, timestamp).
   - **Expected Outcome:** The model generates recommendations for similar products or trending items that match the user's interests.
6. **User Receives Notifications**
   - **Action:** When the price condition is met, the bot sends a notification via Discord.
   - **Expected Outcome:** The user receives a real-time alert in Discord, informing them that the tracked product is now available at the desired price.

Postconditions:

- The product tracking bot maintains a log of interactions, notifications, and recommendations.
- User preferences and product data are updated in the database for future reference.

Acceptance Criteria:

- The bot successfully tracks products and notifies users of price changes.
- Recommendations are relevant to the user's tracked products.
- All components (scraper, database, notification system) work together seamlessly to enhance user experience.

4.6 ITERATIVE IMPROVEMENT

Iterative improvement ensures the iterative refinement and optimization of the project and incorporating user feedback over time. Below is an iterative development plan for improving the project:

Version 1: Initial Version

- **Features:**
  - Basic Discord bot functionality with URL input for product tracking.
  - Web scraping product data (price) at regular intervals.
  - Notification system that alerts users when a product's price drops below the set threshold.
  - Basic logging and error handling for product URLs.
- **Challenges:**
  - Limited scalability for handling multiple users and products.
  - Only basic notifications and single currency support.
  - No product history or recommendation system.
- **Goals:**
  - Create a fully functional bot that meets the basic use case of tracking product prices.
  - Ensure stable scraping and notification functionalities.

Version 2: Enhanced Functionality and API Integration

- **Improvements:**
  - Integration of third-party platform-specific APIs if available, for historical price fetching, reducing the reliance on web scraping.
  - Enhanced notification options, such as scheduling periodic notifications.

- **Challenges:**
  - Ensuring the accuracy of API data fetching.
- **Goals:**
  - Provide a more reliable and scalable system by replacing scraping with APIs wherever possible.
  - Offer more flexible notification options, making the bot more user-friendly.

Version 3: Introduction of Machine Learning for Recommendations

- **Improvements:**
  - Integrating a **DistilBERT-based** recommendation system by embedding product names and categories and suggesting similar products based on user preferences and interaction data.
  - Incorporate user interaction data into a recommendation engine, providing personalized suggestions to users.
  - Initial implementation of a machine learning-based product recommendation system using **FAISS indexing** to retrieve similar products based on the output column.
- **Challenges:**
  - Optimizing the training and evaluation of the recommendation engine.
  - Ensuring the embeddings accurately represent product similarities.
- **Goals:**
  - Enhance user engagement by offering personalized product recommendations based on their tracked products.
  - Improve the efficiency of the recommendation system by utilizing FAISS for fast similarity searches.

Version 4: Scalability and Performance Optimization

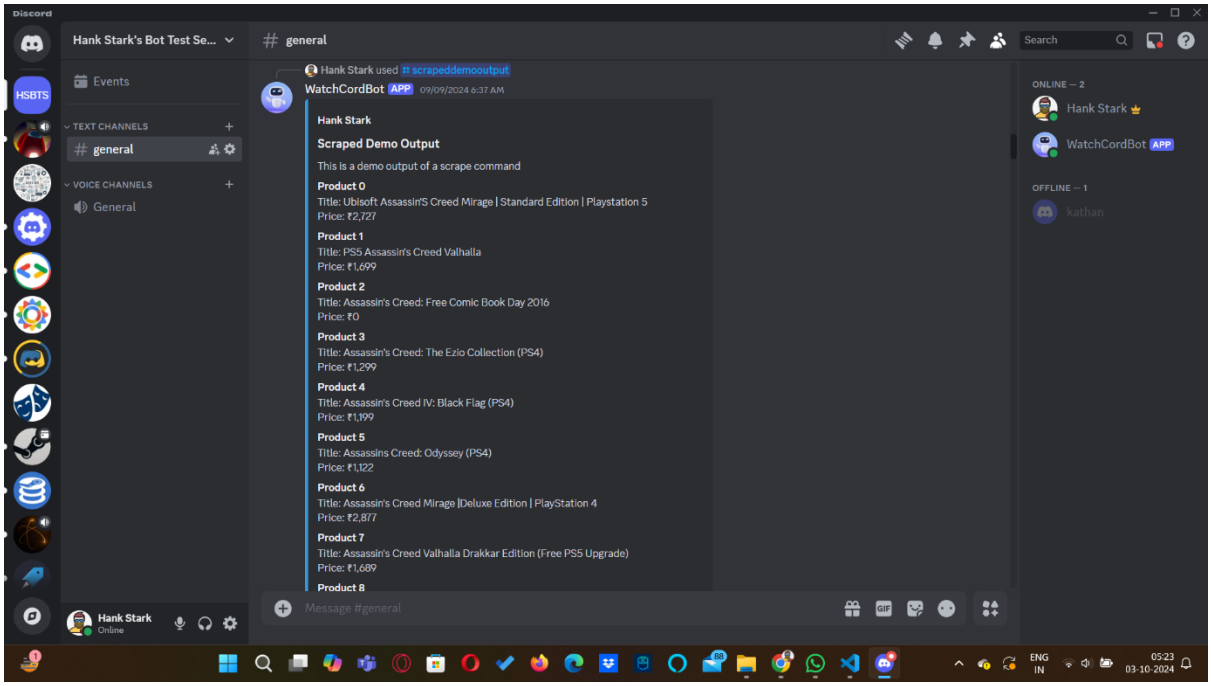
- **Improvements:**
  - Implement **PCA** to reduce the dimensionality of the DistilBERT embeddings for product name and category, improving the speed and scalability of the recommendation system.
  - Further optimize FAISS indexing for large datasets, ensuring fast retrieval even as the number of products grows (e.g., millions of items).
- **Challenges:**
  - Maintaining the accuracy of recommendations while reducing dimensionality through PCA.
  - Scaling the system to handle thousands of users and products efficiently.
- **Goals:**
  - Ensure that the system remains responsive and efficient as user and product volumes increase.
  - Balance the trade-off between performance and accuracy when reducing the dimensionality of embeddings.

Version 5: User Interface and Usability Enhancements

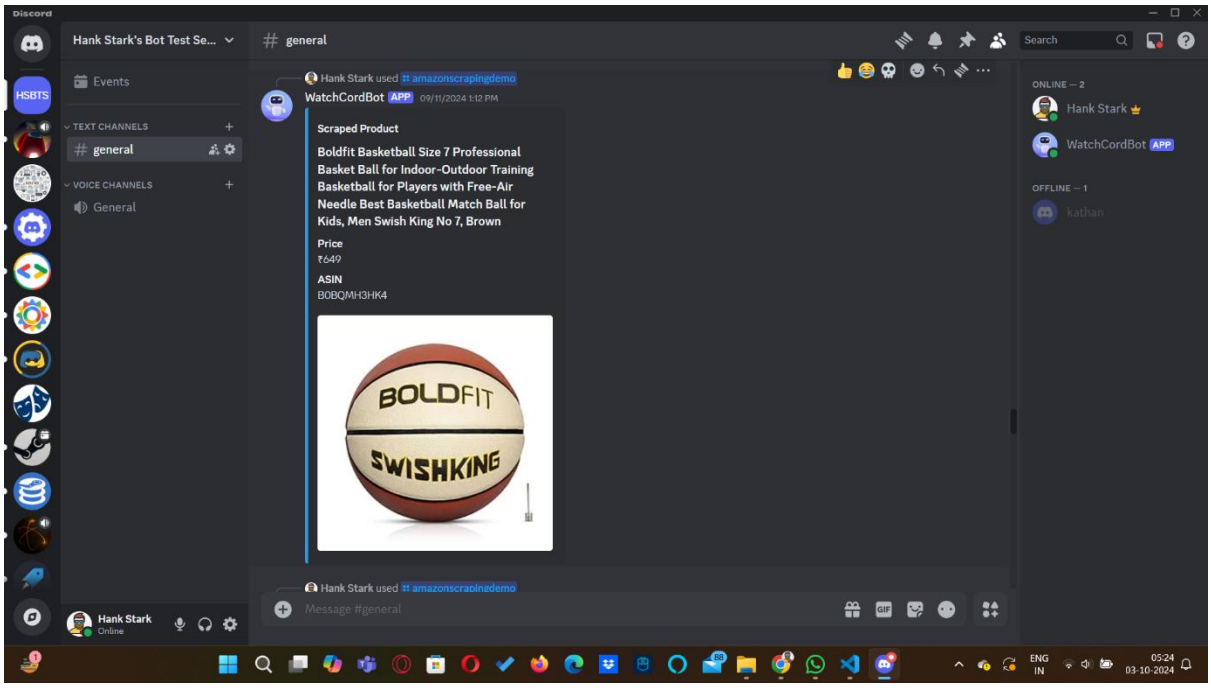
- **Improvements:**
  - Develop a **GUI-based admin panel** for managing tracked products, setting up alerts, and interacting with the bot's core features outside of Discord commands.
  - Improve user onboarding with help commands for using features like search and filter.
  - Allow users to modify and customize notifications more easily.

- **Challenges:**
  - Developing a responsive and intuitive GUI that can handle dynamic data from the bot.
  - Ensuring synchronization between the GUI and Discord command interface for seamless interaction.
- **Goals:**
  - Make the bot more accessible to non-technical users by providing an easy-to-use GUI.
  - Improve user satisfaction through better customization and control over bot features.

4.4 PHOTOS



4.4.1: Product List



4.4.2: Scraped Product Details

5.OUTCOMES

1. **Efficient Product Price Tracking System:**
    - A fully functional Discord bot capable of tracking product prices across multiple platforms using web scraping and APIs.
    - Periodic updates and notifications when prices drop below the user-specified threshold.
2. **Notification System:**
    - Customizable notification settings, including scheduled daily or weekly pings on price changes.
3. **Product Recommendation System:**
    - Machine learning-powered product recommendations using **DistilBERT** embeddings for product name and category.
    - Efficient **FAISS index** to retrieve similar products based on user interaction and timestamp.
4. **Dimensionality Reduction for Optimized Performance:**
    - Use of **PCA** to reduce embedding dimensionality, improving the speed and efficiency of the recommendation system, especially with large datasets.
5. **Enhanced User Experience through GUI:**
    - Development of a user-friendly GUI-based admin panel, allowing users to manage tracked products and notification preferences outside of Discord commands.
6. **Scalable and Robust System:**
    - Scalable infrastructure capable of handling multiple users and products efficiently, ensuring reliable performance as the user base and dataset grow.

7. **Data-Driven Continuous Learning:**
    - Implementation of a continuous learning model that adapts recommendations based on interaction data, improving accuracy over time.
8. **Improved Shopping Experience:**
    - Overall, the bot helps users save time and money by automating price tracking, providing deal alerts, and recommending relevant products based on their behaviour.
9. **Customizability and Flexibility:**
    - Offering various customization options for notifications and search & filter commands making the bot versatile and user-friendly.

6.REFERENCES

Discord.py https://discordpy.readthedocs.io/

Top.GG https://top.gg/tag/marketplace

FastAPI https://fastapi.tiangolo.com/

BERT https://huggingface.co/docs/transformers/en/model_doc/bert

FAISS https://ai.meta.com/tools/faiss/

PCAhttps://www.analyticsvidhya.com/blog/2020/12/an-end-to-end-comprehensive-guide-for-pca/