Product Price Tracker for Multivendor Marketplaces

MATEEH ULLAH ABBASI
01-135221-066
FARHAN TARIQ
01-135221-013

Supervisor: Ms. Aliya Amir

Bachelor of Science in Information Technology

Department of Computer Science
Bahria University, Islamabad

December 2025

Certificate

We accept the work contained in the report "Product Price Tracker for Multi-Vendor Marketplaces", written by Mr. Mateeh Ullah Abbasi and Mr. Farhan Tariq as a confirmation to the required standard for the partial fulfillment of degree of Bachelor of Science in Information Technology.

Approved by . . . :

Supervisor: Ms. Aliya Amir

_____

Internal Examiner:

_____

External Examiner:

_____

Project Coordinator: Dr Kabeer Ahmed Bhatti

_____

Head of Department: Dr. Muhammad Khurram Ehsan

_____

Abstract

In the evolving landscape of e-commerce, consumers often face difficulties in identifying the most cost effective platform for purchasing identical products.Manually comparing prices across various online marketplaces is not only inefficient but also prone to outdated information due to the dynamic nature of pricing. This project introduces a web-based Product Price Tracker for Multi-Vendor Marketplaces designed to aggregate and present real time pricing data from e-commerce platforms such as Daraz, PriceOye, and Telemart. The system is developed using a modern technology stack comprising React.js for thefrontend, Node.js with Express.js for backend logic, and MongoDB for data storage. Real time data retrieval is achieved through the integration of scrappers and webhooks.Key features include price comparison across platforms, and user defined alerts for price reductions. By providing a centralized and intelligent solution for price monitoring, this application aims to support informed decision making for consumers, assist retailers in competitive analysis, and offer valuable insights for market researchers. The project contributes to enhancing transparency and efficiency within the digital commerce ecosystem.

Acknowledgements

The success of this project would not have been possible without the assistance of a great number of people.Above all, gratitude goes to Allah Almighty, who provided the understanding and abilities needed to fully grasp every part of the task; His guidance is endless, for which we remain deeply thankful. Next, recognition is due to our project supervisor, Ms. Aliya Amir led us through all difficulties we faced. Her support motivated us, also shaping our mindset positively.Lastly, we would like to thank each and every teacher who supported us throughout the entire degree program.

Mateeh Ullah Abbasi,Farhan Tariq
Bahria University
E-8, Islamabad, Pakistan

December 2025

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

API	Application Programming Interface
URL	Uniform Resource Locator
UI	User Interface
UX	User Experience
DBMS	Database Management System
FYP	Final Year Project
MERN	MongoDB, Express.js, React.js, Node.js
JSON	JavaScript Object Notation
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
OCR	Optical Character Recognition
CSV	Comma-Separated Values
AI	Artificial Intelligence
ML	Machine Learning
NLP	Natural Language Processing
GUI	Graphical User Interface
CPU	Central Processing Unit
RAM	Random Access Memory

Chapter 1

Introduction

In today's digital marketplace, consumers are presented with countless online platforms offering similar products at varying prices. Manually reviewing each platform to find the best deal is both time-consuming and prone to human error in order to address this challenge, our project, Product Price Tracker for Multi-Vendor Marketplaces, offers a web-based solution for automating the process of comparing prices from multiple e-commerce stores such as Daraz, PriceOye and Telemart. The system retrieves real-time product data via web scraping techniques, and ensuring the accuracy and up-to-date nature of price information. The tracker enables users to see price comparisons side by side with each other, track price changes, and to be notified when the price of a product falls below some specifies threshold. [1, 2].

1.1 Objective

The key objective for this project is to develop an automated, real-time price comparison system that gathers and analyzes product data from several e-commerce vendors.

Key objectives include:

- **Implementing a powerful scraping data extraction pipeline:** for efficient and accurate data retrieval.
- **Developing a responsive React.Js frontend:** to provide an intuitive interface for product search and comparison.
- **Designing a Node.js and Express.js backend:** Since we have to manage data processing, API integration,and server operations.
- **Utilizing MongoDB:** for efficient storage of product information and historical price tracking.

- **Implementing an automated notification system:** in order to alert the users on the price drop below specific threshold level.

1.2 Problem Description

Consumers find it very difficult to determine how to locate the cheapest prices for items across various online stores. Efficient checking of multiple e-commerce platforms is not feasible with the manual process, Its time-consuming, and often provides unsuitable information in the form of outdated or incomplete information. Moreover, existing price comparison websites tend to not include local marketplaces, have limited use of APIs, or lacking real-time updates.

This project overcomes these limitations by providing a unified system which can automatically collect and compare prices of product from various vendors in real-time. The system ensures data accuracy and comprehensiveness; offering users a sound platform to make well informed purchasing decisions.

1.3 Methodology

The system architecture follows a modular design that follows web-scraping techniques approach to achieve comprehensive and real-time data collection.

- **Web Scraping:** Puppeteer, Playwright to Scrape Product Data from Websites.[1, 2].
- **Backend (Node.js/Express.js):**Responsible for scraping operations, data normalization and server-side logic[3].
- **Frontend (React.js):**Has a user-friendly interface to search the products, compare prices, and visualising trends in prices. [4].
- **Database (MongoDB):** Stores product details, user preferences and historical price records to enable data persistence and data analytics. [5].
- **Notification Service:** Sends automated email alerts when a product's price drops below the price that user would prefer.

1.4 Project Scope

The scope of this project involves real time product price comparison, visualization of price trends, automatic email alert generation.System allows user to register, manage his tracked products, and receive notification of fluctuation in price of product.

The project currently helps in integration with popular market places like Daraz, Telemart, and PriceOye.In the future,the system will incorporate international platforms like aliexpress and amazon allowing it to become globally adaptive solution.[6].

1.5 Feasibility Study

- **Technical Feasibility:**The project is developed using MERN(MongoDB, Express.js, React.js, Node.js) stack, a reliable, scalable, and open-source technology set for full-stack web applications.
- **Operational Feasibility:** The design of the system and its functionality is consistent with academic and practical implementation standards, enabling implementation on both local and cloud environments.
- **Economic Feasibility:** All tools and frameworks used are open-source, making the project cost-effective without requiring paid software or services.
- **Legal Feasibility:**The system complies with the fair use and ethical scraping policies by giving priority to official APIs and having compliance regarding regulations on data access.

1.6 Application Areas

- **Consumers:** Save time and effort enlisting the easiest way to find the lowest available prices for products.
- **Retailers:** Retaillers will be able to analyze competitor pricing strategies and adjust their own pricing models accordingly.
- **Market Analysts:**Analysts will be able to track and evaluate market trends and pricing patterns easily across multiple vendors.
- **Developers:** Developers will be able to extend the system by adding new market-places, improving data pipelines,or integrating AI based recommendation modules.

1.7 Tools & Technologies

- **Frontend:** React.js
- **Backend:** Node.js and Express.js
- **Database:** MongoDB

- **Web Scraping:** Puppeteer, Playwright
- **Data Visualization:** Chart.js
- **Communication:** RESTful APIs
- **Hosting:** Backend on AWSCloud and Frontend on Vercel

Chapter 2

Literature Review

This chapter examines those systems and research articles currently existing with regards to price comparison tools and ecommerce tracking solutions. Understanding such studies helps reveal the deficiencies of the current systems and prove the need for a more integrated and real time system [7, 8].

2.1 API-Based Price Comparison Systems

Early price tracking platforms heavily utilized vendor APIs to get products data. Although these APIs were providing structured information, they were often limited in their access frequency, vendor coverage, and their speed of update. As many retailers started limiting the use of API, there was an increase in dependency on web scraping by a large magnitude[8].

2.2 Web-Scraping Platforms

Web scraping served as the alternative of choice when APIs were not available. There are some popular systems such as CamelCamelCamel and Keepa collect pricing information directly from websites to maintain historical data and graphing trends.However, these systems are often restricted to a single vendor,such as Amazon, which limits their global or multi-market applicability[9, 10].

2.3 Browser Extensions and Shopping Assistants

Extensions such as Honey and PriceBlink operate directly within browsers to monitor product prices and provide coupon recommendations . While convenient for end-users, these tools are limited by their partnership networks and browser dependencies. They also lack centralized dashboards for managing tracked products[11, 12].

2.4 Commercial Monitoring Tools

Enterprise solutions like Prisync and Price2Spy offer extensive competitor analysis and automation capabilities. However, these tools are commercial and closed-source, making them unsuitable for academic research or open innovation. Our project draws inspiration from their efficiency but aims to create a transparent and freely accessible academic alternative [13, 14].

2.5 Academic Studies

Several academic prototypes have attempted real-time price tracking using web scraping and automation. Most focus on a small number of vendors and lack real-time updates or scalable architecture .Our system is an extension of this work, bringing together multiple vendors, the ability to update dynamically, and defining user alerts. These prototypes also show difficulties in dynamic websites and the need to have powerful crawlers[7].

2.6 Comparative Analysis

Table 2.1: Comparative Analysis of Existing Price Tracking Systems

Feature	CamelCamel	Keepa	Honey	PriceBlink	Prisync	Price2Spy
Vendor Coverage	Amazon-only	Amazon-only	Multiple stores (limited)	Multiple stores (limited)	Global wide coverage	Global wide coverage
Data Source	Web Scraping	API + Scraping	Browser Extension	Browser Extension	API + Scraping	API + Scraping
Real-Time Updates	No	Limited	Yes (browser-based)	No	Yes	Yes
Historical Price Charts	Yes	Yes	No	No	Yes	Yes
Alerting System	Yes	Yes	Yes	Yes	Advanced Alerts	Advanced Alerts
User Dashboard	Yes	Yes	No	No	Yes	Yes
Supports Multiple Marketplaces	No	No	Partially	Partially	Yes	Yes

2.7 Challenges and Limitations

Across reviewed systems, there are a number of common limitations that emerged:

- **Narrow platform scope:** Many tools have a small marketplace or vendor scope which means they leave the major and important platforms unmonitored.
- **Delayed or infrequent updates:** API-dependent systems update only according to API limitations, and many scraping systems run at intervals that may miss short-term price fluctuations.
- **Lack of historical tracking:** While some systems store price history, many simpler tools only notify on current changes without maintaining multi-day archives.
- **Limited alerting options:** Existing alert mechanisms are often basic and may lack flexibility or integration into centralized dashboards.
- **Technical barriers:** Anti-bot measures such as CAPTCHA, dynamic content, and request frequency limits hinder continuous scraping .
- **Legal and policy constraints:** Unsolicited scraping can violate terms of service; some systems mitigate this using official APIs where available.

In summary, the surveyed literature demonstrates that although many price-tracking solutions exist, none fully provide the combination of real-time updates, multiple marketplace coverage, historical analytics, and robust alerting that our proposed system aims to deliver at free of cost .

Chapter 3

Requirement Specification

System requirements are configurations that a system needs in order to function properly and effectively. Failure to comply with these requirements may cause installation issues or performance issues. System requirements are documents or sets of documents that outline the features and behaviour of a system or software application.

3.1 Existing System

E-commerce platforms and price tracking tools exist in the market, such as CamelCamelCamel, Keepa, Honey, PriceBlink, Prisync, and Price2Spy. While these platforms provide some features for tracking and comparing product prices, they have limitations:

- Limited vendor coverage or marketplace support.
- Restricted functionality in free versions (some require premium subscriptions).
- Poor usability and non-intuitive interfaces for first-time users.
- Lack of personalized notifications or historical price visualization.

Currently, users experience issues including checking the prices in the several different platforms manually, not getting the timely alerts and trouble in tracking the past trends. This makes it clear that there needs to be a comprehensive and easy-to-use price tracking system

3.2 Proposed System

The Product Price Tracker is a web-based system that collects real-time pricing information from various e-commerce platforms using web-scraping (Playwright, Puppeteer). It is intended to make price monitoring and comparison easier and therefore allow users to make informed purchasing decisions in an efficient way.

Key highlights:

- **Free and Accessible:** There are no barriers to subscription in order to access the system.
- **User-Friendly:** User-friendly intuitive frontend of React.js with responsive build for desktop and mobile.
- **Real-Time Comparison:** Users can take price comparison across the marketplaces such as Daraz, PrieOye, and Telemart.
- **Secure Accounts:** Personalised accounts , saved watchlists & encrypted login.
- **Automated Notifications:** Email notifications of price drops or reaching a particular price.
- **Historical Records:** Time-stamped price charts in the form of graphs for making informed decisions.

3.3 Requirements Specification

A requirement specification is a way to ensure that all the stakeholders have a common understanding of what the system is required to do, and what its constraints and expectations are. It is a reference to use for implementation, testing and maintenance.

3.4 Functional Requirements

3.4.1 User Management

- **Registration & Authentication:** Users can register, authenticate their email, login & log out securely with encrypted connections.
- **Profile Management:**Users can edit and update the personal information.

3.4.2 Product Search & Comparison

- Search products by name from various marketplaces.
- Provide price comparison in real time.

3.4.3 Tracking of Price & Notifications

- Put products on watchlists to track them on an ongoing basis.
- Receive automated email notifications as a result of conditions being met.

3.5 Non-Functional Requirements

3.5.1 Security

- HTTPS/TLS encryption of all data transmissions.
- Passwords before being stored (hashed) e.g. bcrypt.

3.5.2 Reliability & performance

- Quick page loading, Responsible interface (desktop and mobile).
- Background jobs update the price data periodically which can be specified.
- Wildcard successful scraping attempts are retried with automatic exponential backoff on failed attempts.

3.5.3 Usability

- Easy to navigate intuitively and needless learning curve.
- Graphical representation of price history and alerts.

3.5.4 Maintainability

- Modular, frontend, backend and data code structure.
- Documentation and adherence to coding standards.

3.5.5 Portability & Scalability

- Easily add more marketplaces, products and features without major changes.
- Cross terminal compatibility (React.Js frontend - Node.Js backend) Self hosted or cloud Linux servers.

3.5.6 Software Quality Attributes

- **Usability:** Easy to navigate and simple interface.
- **Performance:** Good response times even when a large number of users..
- **Security:** Secured user data & secure authentication.

3.6 Use Case Diagrams

This section gives the use case diagrams for Smart Price Tracker system.

3.6.1 User and System Interaction

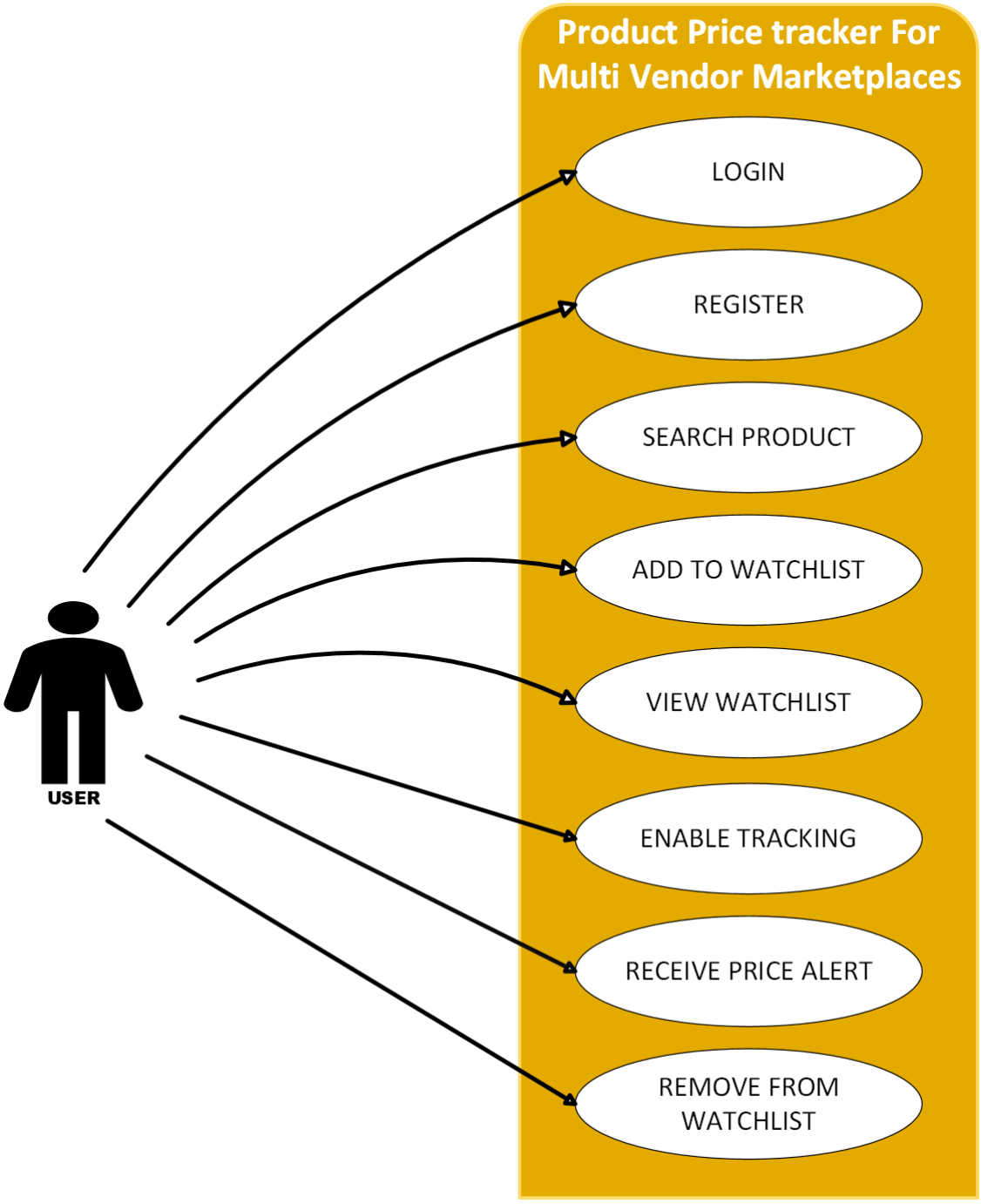

Figure 3.1: User and System Interaction

Figure 3.1 illustrates the interaction of a User with the Smart Price Tracker.A user can register or log in, search the products, add products to the Watchlist, turn on price tracking from the Watchlist, get alert when prices drop and remove products from the Watchlist.

3.6.2 External E-commerce Website Interaction

Figure 3.2 is a diagram of how a External E-Commerce Websites interact with the system. These websites will contain information about the product and updates of new prices which are used by the Smart Price Tracker which will monitor for changes and notify users.

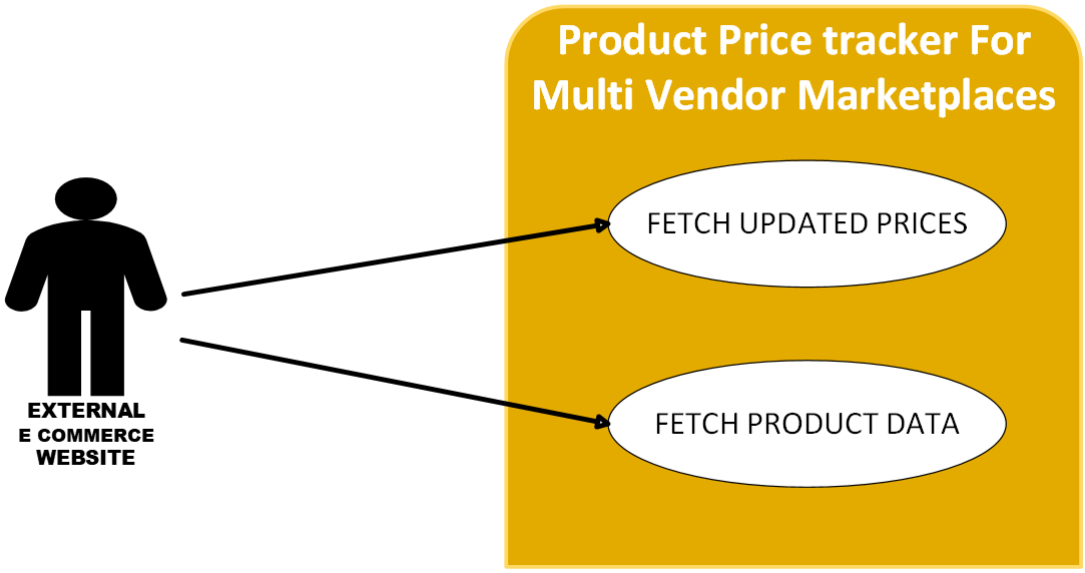

Figure 3.2: External E-commerce Website Interaction

All external e-commerce websites have dynamically displayed product details such as product names, price, status of availability, discounts offered and information about the seller. The Smart Price Tracker communicates with these sites after every few seconds in an automated scraping fashion in order to obtain the current price data and details.The system responds to scheduled requests that your computer transmits to these Web sites using background services. Once the product pages are accessed, the system takes relevant attributes such as current price, product title, and vendor name. This extracted data is then validated, normalized and is stored in the database of the system for further processing. One of the main objectives of this interaction is constant price monitoring. Whenever the price is detected to change on an external e-commerce website, Smart Price Tracker will update its internal records and determine the change against user-defined thresholds. If the conditions for a price alert are met, several notifications are automatically triggered.

3.6.3 Combined Use Case Diagram

Figure 3.3 shows the full integrated use case diagram with User and External E-Commerce Website integrated as a unified view of the Smart Price Tracker system.

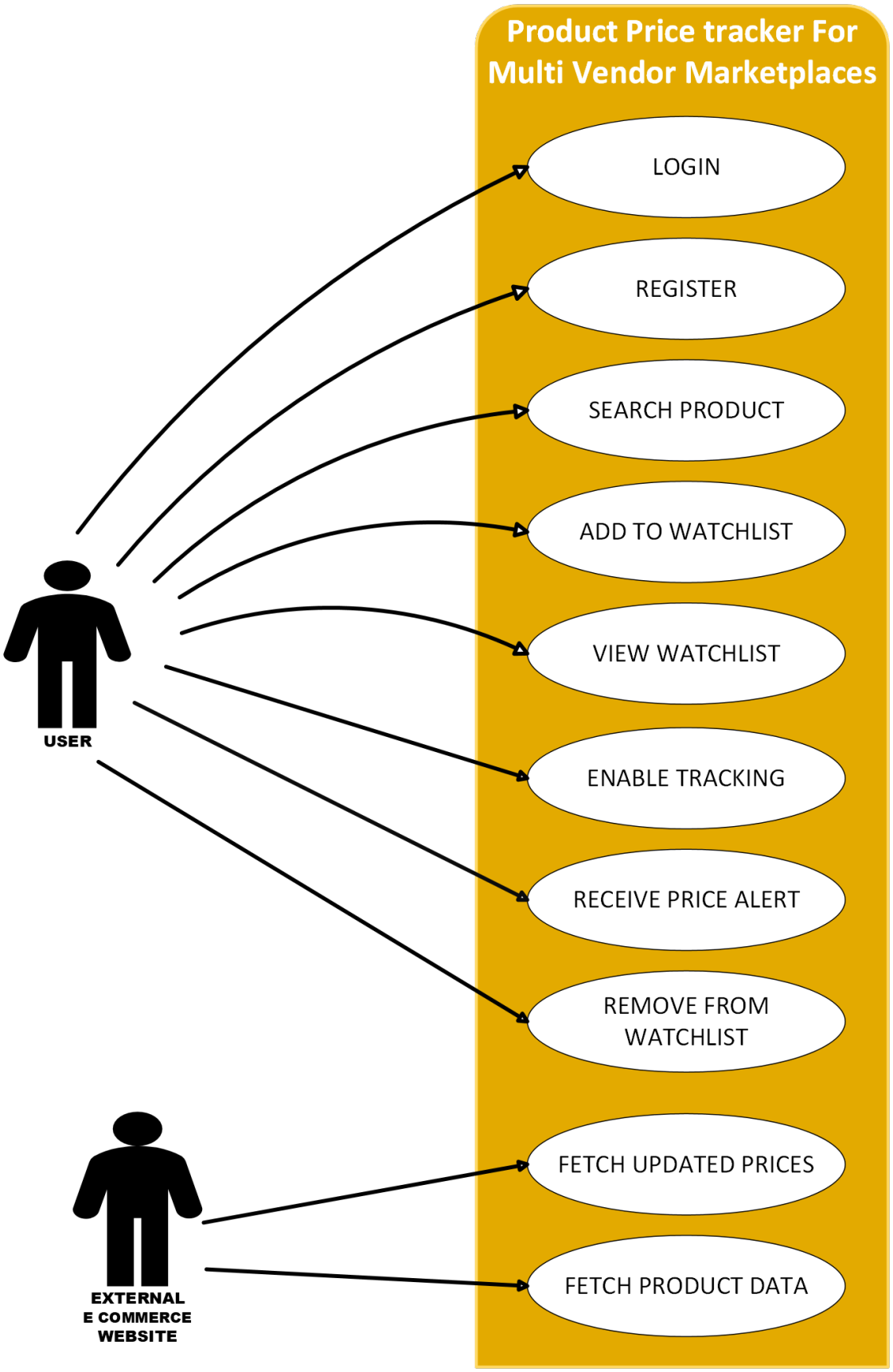

Figure 3.3: Use Case Diagram for Smart Price Tracker

3.7 Descriptive Use Cases

This presents the detailed descriptive use-cases in tabular forms for Smart Price Tracker system. Each table describes a purpose, the actors involved, the system behaviour, pre-condition, post-condition and flow of events.

3.7.1 Use Case: User Registration / Login

Table 3.1: User Registration / Login

Use Case ID	UC-1
Use Case Name	Register / Login
Actor(s)	User
Priority	High
Pre-Conditions	• User has opened the Smart Price Tracker application. • User is not authenticated.
Post-Conditions	• User is authenticated. • User is redirected to dashboard.
Basic Flow	1. User clicks “Register” or “Login”. 2. User enters credentials. 3. System validates information. 4. User is logged in and redirected to dashboard.
Actor Actions	User enters and submits credentials.
System Response	Validates, authenticates, and loads user dashboard.
Alternative Flow	• Invalid credentials → Error message. • Email already registered → Suggest login instead.

This use case represents the initial interaction between the user and the Smart Price Tracker system. User registration and login is critical for us to deliver personalized services such as watchlists, price alerts and saved preferences. The system secures the authentication process by checking the user credentials before they access the dashboard.

During this process, the system checks user input in order to prevent unauthorized access as well as creating duplicate accounts. In the case of wrong credentials or a previously registered email, relevant error messages are shown to instruct the user. Successful authentication creates a secure session and redirects the user to the dashboard allowing access to all the core features of the system.

3.7.2 Use Case: Search Product

Table 3.2: Search Product

Use Case ID	UC-2
Use Case Name	Search Product
Actor(s)	User
Priority	High
Pre-Conditions	• User is logged in.
Post-Conditions	• System displays search results fetched from all supported e-commerce platforms.
Basic Flow	1. User enters product name in search bar. 2. System fetches data from external e-commerce websites. 3. System displays consolidated search results.
Actor Actions	User initiates a search query.
System Response	Retrieves product details and returns results.
Alternative Flow	• No matching product found → System shows “No results found”.

This use case covers the search of products among multiple supported ecommerce platforms. It enables users to easily access product price and size comparison without having to visit individual websites. The search functionality serves as the basis for the Smart Price Tracker to make informed purchasing decisions.

Upon receiving a search request, the system goes to other e-commerce websites and consolidates the responses into a single format. The results are then shown in a consolidated view and thus making it easy for the users to compare the prices, Vendors and availability. If nothing relevant is found, the system clearly informs the user so that the experience of the system is also smooth and user-friendly.

3.7.3 Use Case: Add Product to Watchlist

Table 3.3: Add Product to Watchlist

Use Case ID	UC-3
Use Case Name	Add Product to Watchlist
Actor(s)	User
Priority	High
Pre-Conditions	• User is logged in. • User has searched for a product.
Post-Conditions	• Product is stored in user's Watchlist.
Basic Flow	1. User selects a product from search results. 2. User clicks "Add to Watchlist". 3. System stores product in Watchlist.
Actor Actions	User selects and adds product to Watchlist.
System Response	Saves product into Watchlist.
Alternative Flow	• Product already in Watchlist $\rightarrow$ System informs user.

This use case allows users to save products of user interest for continuous monitoring. The watchlist feature gives the users the ability to keep track of products of interest over time without having to constantly search for these products. It is instrumental in the automation of price monitoring and alert generation.

When the user adds a product to the watchlist, the system stores the details of the product and links it with the user's account. The system also ensures the prevention of duplication by checking if the product is already there in the watchlist. This ensures efficient storage, ensuring there are no redundancies and that there is proper tracking of selected products from users.

3.7.4 Use Case: Enable Price Tracking

Table 3.4: Enable Price Tracking

Use Case ID	UC-4
Use Case Name	Enable Price Tracking
Actor(s)	User
Priority	High
Pre-Conditions	• Product exists in user's Watchlist.
Post-Conditions	• System begins automated scheduled price tracking.
Basic Flow	1. User opens Watchlist. 2. User selects a product. 3. User clicks "Enable Tracking". 4. System activates price tracking for that product.
Actor Actions	User enables tracking for selected product.
System Response	Starts cron-based price checks.
Alternative Flow	• System cannot scrape website → Logs failure and retries.

This use case focuses on activating the automated price monitoring of products stored in the watchlist. Once enabled the system periodically looks up the latest prices of the selected product, using background scheduling mechanism.

The system is constantly making comparisons between present prices and past values and user-defined conditions. If any issue is encountered with the scraping process due to network problems or website restrictions, the system logs the error and retries it automatically. This helps in maintaining reliable and consistent tracking of the prices and eliminates manual intervention on the part of the user.

3.7.5 Use Case: Receive Price Alerts

Table 3.5: Receive Price Alerts

Use Case ID	UC-5
Use Case Name	Receive Price Alerts
Actor(s)	User
Priority	High
Pre-Conditions	• User has enabled tracking for at least one product.
Post-Conditions	• User receives alert notification.
Basic Flow	1. System checks product prices through scheduled jobs. 2. Detects price change or drop. 3. Sends email or in-app alert to user.
Actor Actions	Passive actor—receives notifications.
System Response	Sends alerts when conditions met.
Alternative Flow	• No drop → No alert sent.

This use case outlines how users receive updates about price changes for products that they are tracking. It shows how automated the Smart Price Tracker is, where it will generate alerts, without requiring an active involvement from the user.

The system reviews updated product prices against predetermined alert conditions on a regular basis. When a drop in the price or any related changes are identified, notifications are sent through emails or in-app notifications. If there is no change in the price, the system is idle so users can have meaningful and relevant notifications.

3.7.6 Use Case: Delete Tracked Product

Table 3.6: Delete Tracked Product

Use Case ID	UC-6
Use Case Name	Delete Tracked Product
Actor(s)	User
Priority	Medium
Pre-Conditions	• User is logged in. • Product is present in Watchlist.
Post-Conditions	• System removes product and disables tracking.
Basic Flow	1. User selects product in Watchlist. 2. User clicks “Remove”. 3. System deletes product entry.
Actor Actions	User deletes product.
System Response	Removes product and updates dashboard.
Alternative Flow	• System error → “Unable to delete product”.

This use case enables users to maintain their watchlist by eliminating those products that are no longer of interest. Removing a product automatically disables its price tracking and no further notifications will be made.

The system then updates the database accordingly and shows these changes in the dashboard in real time. In the event of mistake while deleting, the system gives a notification and maintains the data consistency. This use case creates a better usability, it gives users total control over their tracked products.

3.7.7 Use Case: External Website Provides Product Data

Table 3.7: Provide Product Data (External Website)

Use Case ID	UC-7
Use Case Name	Provide Product Data
Actor(s)	External E-commerce Website
Priority	High
Pre-Conditions	• Product page is accessible.
Post-Conditions	• System receives updated product data.
Basic Flow	1. System sends scraping request. 2. Website returns product details. 3. System parses and stores information.
Actor Actions	Website responds with product data.
System Response	Updates product database.
Alternative Flow	• Scraping fails → System retries or logs error.

This use case defines the interaction between Smart Price Tracker system and external e-commerce websites. These websites serve as informational data providers, giving real-time access to product information that is needed to compare prices and track product prices.

The system sends automated scraping requests periodically to external platforms and extracts relevant information about the products such as price, product name, availability, etc. The data that is retrieved is then processed and stored in the database. As scraping operations are unable to get through access restrictions or connectivity issues, the system will retry the operation or record the failure for monitoring purposes. This not only assures uninterrupted and predictable data acquisition.

Chapter 4

Design

This chapter describes the structural and architectural design of the Smart Price Tracker system. The main goal of system design is to transform the requirements into the defined components, relationships between components, and flow of information in the system. The design process is taken care of to ensure that the system can be made scalable, maintainable, secure, and capable of supporting the actual tracking of commodity prices across different e-commerce platforms in real time. The chapter includes the context diagram, high-level architecture, design constraints, and design methodology that is followed during the system development.

4.1 Context Diagram

The context diagrams is used to provide a high-level overview of how the Smart Price Tracker system interacts with its external entities.

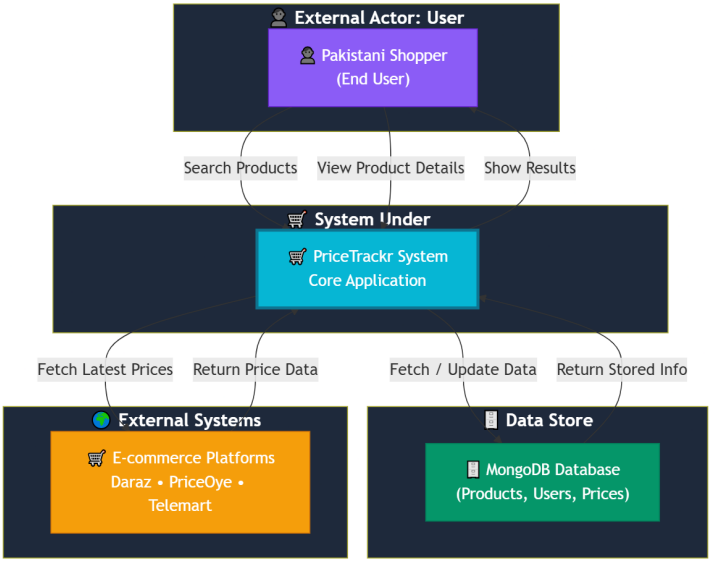

Figure 4.1: Context Diagram for Smart Price Tracker

It is presenting the system as a single unit with consideration of major data flows between the user, the database, and the external e-commerce platforms. This is a simplified representation of the theory to explain how different actors communicate with the system without exposing internal components. The overall context of the system is shown in Figure 4.1

4.2 High Level System Architecture

The high-level system architecture specifies the major software components and the interactions among them for product tracking, alert generation and user management. It shows the logical grouping of modules into three primary layers: the Users Layer, Interface Layer, and Backend/Storage Layer.

Users Layer

This layer consists of the end-users of the system. A user can search for products, add items to the Watchlist, enable price tracking, and receive alerts. All actions occur through the web interface.

Interface Layer

This layer includes the web-based frontend of the system, developed using React and Vite. It provides:

- Dashboard and Watchlist interface
- Product search and add functionality
- Authentication and sessions
- Real-time updates and notifications

The frontend communicates with the backend server using REST APIs.

Backend / Storage Layer

This layer contains the system's server-side logic and permanent data storage.

- **Node.js + Express Backend** manages routing, authentication, product tracking, and scheduled tasks.
- **MongoDB Database** stores user accounts, watchlist products, price data, and system logs.

External Integrations

The backend interacts with external services like Daraz, Telemart, PriceOye to get the product prices using the scraping technques. These platforms serve as data providers when scheduled price monitoring is performed.

The high level architecture is shown in Figure 4.2.

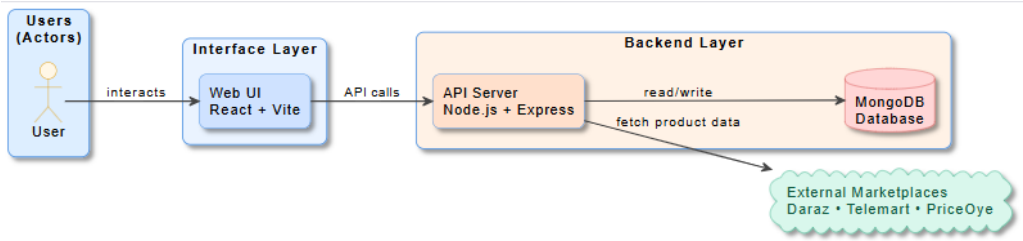

Figure 4.2: High Level System Architecture

4.3 Design Constraints

The system design must adhere to various constraints to ensure correct functionality, performance, and reliability. Table 4.1 summarizes the key constraints considered during design.

Table 4.1: Design Constraints

Constraint	Description
Reliability	The connection between the frontend, backend, and database must remain stable. Scheduled scrapers should execute reliably without failure.
Criticality of Application	Requests such as login, price fetching, and alert generation must be processed immediately to maintain real-time user experience.
Security Considerations	Sensitive data such as user credentials must remain encrypted. API routes must be protected to prevent unauthorized access.
Internet Connectivity	The system relies on stable internet access for scraping real-time product prices and serving frontend content.

4.4 Design Methodology

The process of developing Smart Price Tracker system is following the Agile Software Development Methodology. Agile has been chosen since the system demands constant improvement, frequent testing and the ability to iterate, particularly of the components of the system that include web scraping, automated tracking of prices, automated alerts

in real-time and improved user interfaces. Agile allows the team to divide the project into smaller, manageable development cycles (sprints), where each sprint phase will be dedicated to the designing, implementation, and testing of a specific system module. This approach provides a level of adaptability if system needs change or if external factors such as the structure of the e-commerce websites change and add new constraints.

Figure 4.3 illustrates the Agile methodology that is used in the course of system development.

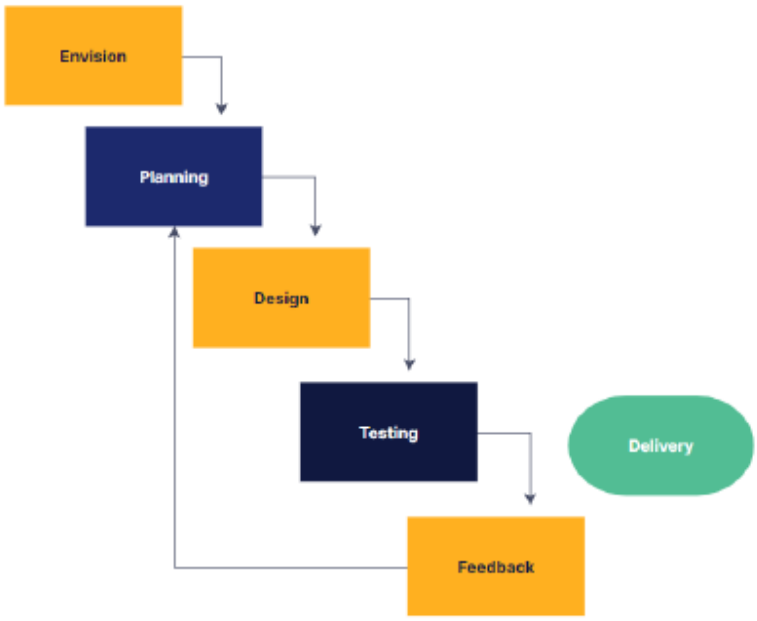

Figure 4.3: Agile Methodology Used for Smart Price Tracker Development

The agile framework gave the following advantages to the Smart Price Tracker project:

- **Iterative Development:** Allowed for constant improvement of scraping scripts, API endpoints and UI components depending on the results of the testing.
- **Flexibility:** Price tracking logic, database schema, and UI modules were modified as the system changed and new requirements were identified.
- **Rapid Issue Resolution:** Bugs relating to scraping failures, authentication or UI responsiveness were identified and solved within each sprint.
- **Continuous User Feedback:** Feedback was given by the test users on the behavior of the dashboard, watchlist and price alert mechanism, which was integrated in the next couple of sprints.
- **Parallel Development:** Frontend, backend and scraping modules were developed simultaneously leading to an improvement in overall efficiency.

Agile made sure the Smart Price Tracker was created incrementally, well tested at each stage and stayed in touch with the functional and non-functional requirements of the project.

4.5 High Level Design

This section discusses the high-level design of Smart Price Tracker system. High-level design addresses the modelling of fundamental system elements, their interaction, and the flow of information architectures from one module to another. The diagrams in this section represent the behavior of the system from different perspectives, such as the organization of components, the deployment architecture, and the sequential work flows.

4.5.1 Component Diagram

The Smart Price Tracker has several major components, User module, Product module, Watchlist module, Price Alert module and MongoDB data store. These components interact with the central system using their functionality to perform operations like product tracking, watchlist item storage, alert generation, etc. The high-level component diagram of the system is given in Figure 4.4.

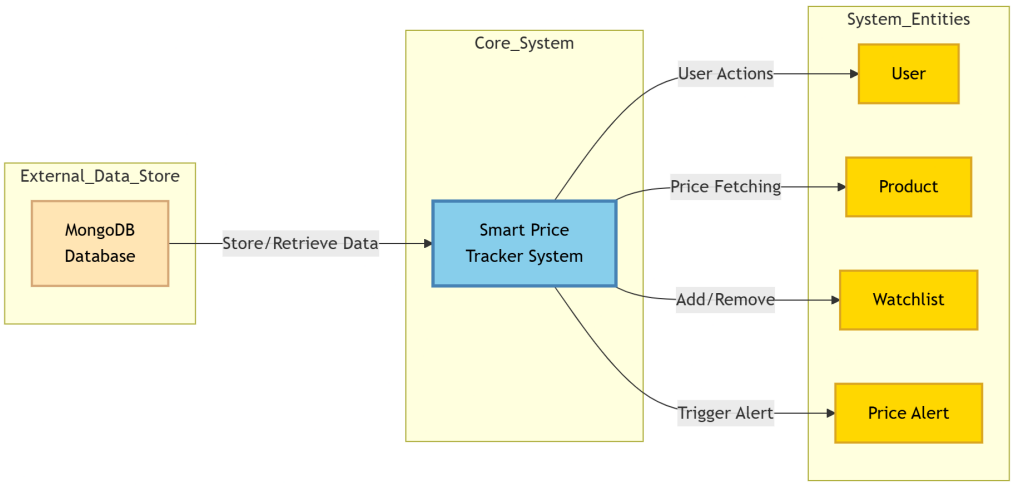

Figure 4.4: High Level Component Diagram

4.5.2 Deployment Diagram

The Smart Price Tracker web system works on a robust, multi-tier deployment model for a high availability/scalability. There is a logical level of separation between the presentation, application logic and data layers to maintain maintainability and efficient

real-time processing using this architecture. This separation is fundamental for facilitating future upgrades of the service as well as horizontal scaling.

User interaction starts from the Client Devices layer, where the user uses a web browser or mobile system to access the system. The frontend application is created using React.js and only communicates with the server using standard applications (http/https).

The processing for the requests is done using the backend server implemented in Node.js using the Express.js framework. This is the one most important application layer which handles the entire core transformation of the business logic, such as managing the session of the user, processing the API requests, managing data retrieval processes, and so on. There are two main functions of the backend with respect to data:

1. **Data Persistence:** Direct interaction with the MongoDB database for querying, storing and updating of persistent data such as user profiles, watchlists and historical price records. MongoDB has a flexible designing structure of document database and you can use it for efficient storing of dynamic product data schema.

2. **Real-Time Data Acquisition:** Using dedicated web scraping tasks to obtain up to date price data from external multi-vendor marketplaces.

Furthermore, the server is the host of important Background Jobs (automated schedulers). These jobs are persistent and not tied to active user sessions, so that they can track the prices of products against thresholds set by users and automatically send email or in-app notifications. The full deployment model showing the distribution of these components across various nodes and the communication links is shown in Figure 4.5.

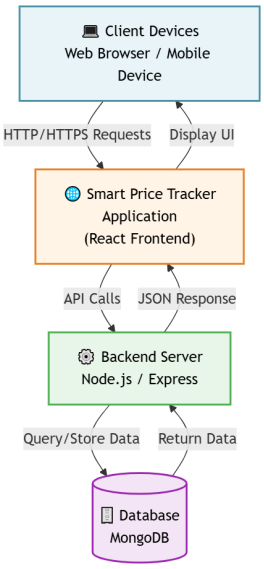

Figure 4.5: High Level Deployment Diagram

4.5.3 Sequence Diagrams

Sequence diagram shows the step-by-step interaction among the system components. The next few sub-sections describe the operation of the Smart Price Tracker with respect to operations such as product searching, watchlist management, background monitoring, and authentication, showing the flow of messages among the user, frontend, backend, and external systems.

4.5.3.1 Sequence Diagram for Product Search

Figure 4.6 shows how a user searches for a product and how the system gets the real-time information using scraping services.

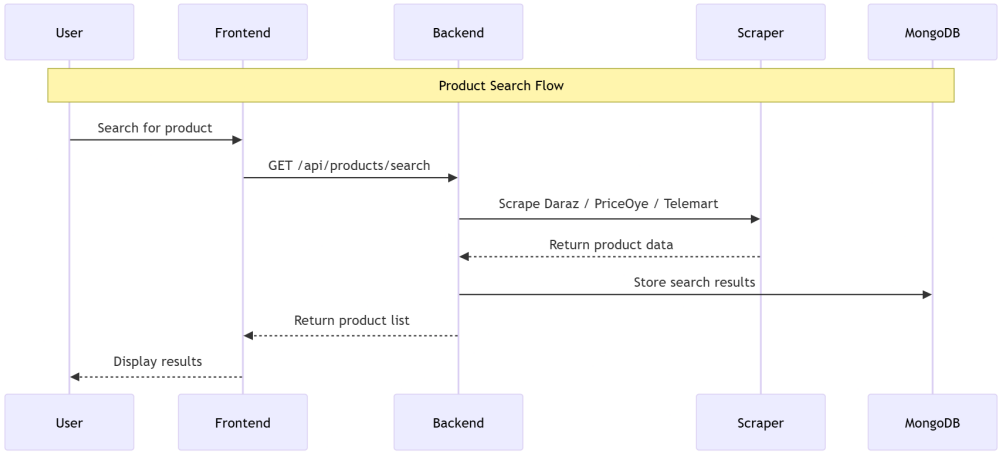

Figure 4.6: Sequence Diagram for Product Search

4.5.3.2 Sequence Diagram for Watchlist Management

Flow when a user adds a product to the watchlist is illustrated in Figure 4.7 . The backend checks the validity of the request and saves the watchlist item in the database.

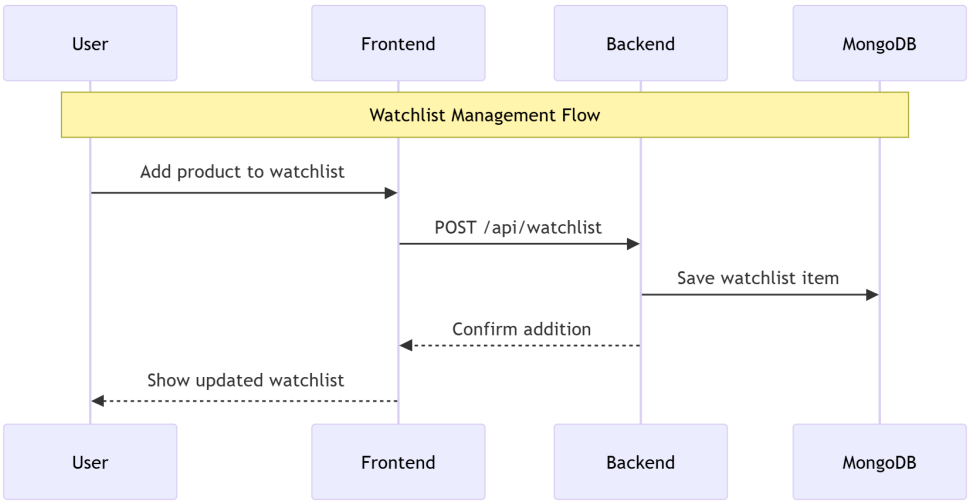

Figure 4.7: Sequence Diagram for Watchlist Management

4.5.3.3 Sequence Diagram for Background Price Monitoring

Figure 4.8 shows how there is a process in the backend that scrapes the product prices periodically and updates the database and also sends alert notification in the form of email or WebSocket.

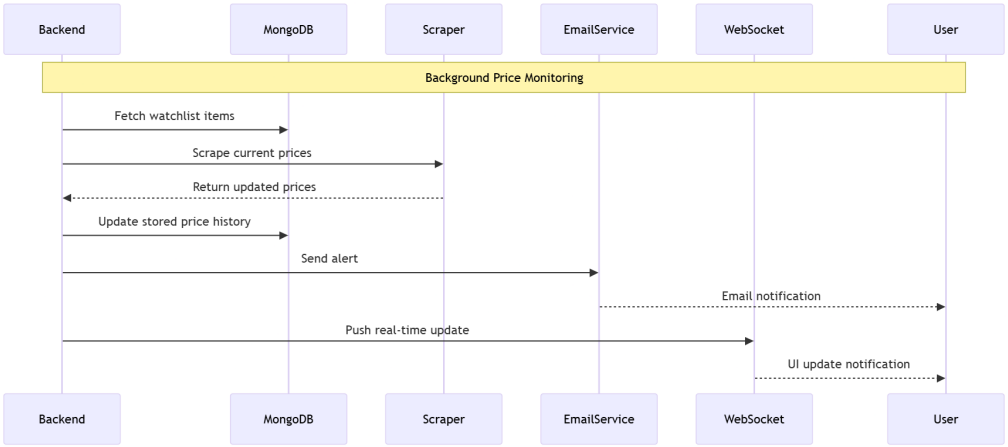

Figure 4.8: Sequence Diagram for Background Price Monitoring

4.5.3.4 Sequence Diagram for User Authentication

Figure 4.9 describes the login process using Google OAuth. The backend is used to validate the token, create/ update the user, and issue a JWT for safe handling of the user session. The entire flow is made to be secure, efficient and based on the strong authentication

offered by Google, while the formed JWT ensures the secure handling of a session in later interactions.

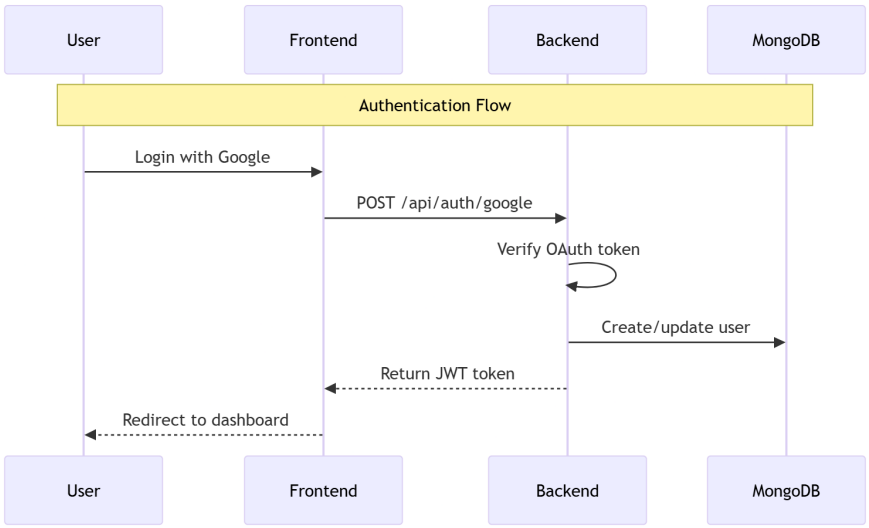

Figure 4.9: Sequence Diagram for Authentication Flow

Sequence Diagram for User Authentication: Detailed Explanation

1. Initiation of Login

The user starts the authentication process from the login screen (Figure A.2), by choosing the Google login option.

- **User → Frontend:** The User initiates the login process by logging into by clickng "Login with Google".
- **Frontend → Backend:** The Frontend will send the Google OAuth token to the Backend using a POST request to the `/api/auth/google`endpoint.

2. Backend Validation and User Management

The Backend takes care of the safe validation of the token provided to it and acting on the user data in the system.

- **Backend (Internal):** The Backend then uses the authorization servers of Google to safely check the validity of the OAuth token.
- **Backend → MongoDB:** The Backend then moves on to depose or modify the user record in the MongoDB database by connecting the external Google account with a local profile.

3. Session Creation and Finalization

Upon successful verification a secure session is created and passed back to the client.

- **Backend → Frontend:** Backend provides a security token in the form of a web token (JWT) to the Frontend.
- **Frontend → User:** The Frontend takes the JWT and immediately triggers a redirect to the dashboard for the user so that he has access to the main features of the application.

4.6 Low Level Design

In this section the low level design of Smart Price Tracker application is provided. Low-level design descriptions that directly enable the creation of modules are provided here. with the high-level architecture already in place, this final critical phase of design translates these architectural designs into concrete, implementation ready specifications. The individual components are broken down into their underlying classes, attributes, methods and relationships. There is a particular emphasis on describing the exact internal structure and behaviour of each unit of software. The set of diagrams provided in this section provide a detailed picture of the data structures and service interfaces that will be used throughout the application.This Section also involves defining of concrete parameters and types of return values for all the inter module communication. This is a direct blueprint to be used in coding. When the blueprint is followed, modularity and maintainability of the application are effective as well as the system gets integrated correctly.

The class diagram for the User and Watchlist relating classes are shown in Figure 4.10 along with the Backend service interface. Figure 4.11 illustrates the classes Scraper, Product and Price History and their relationship.

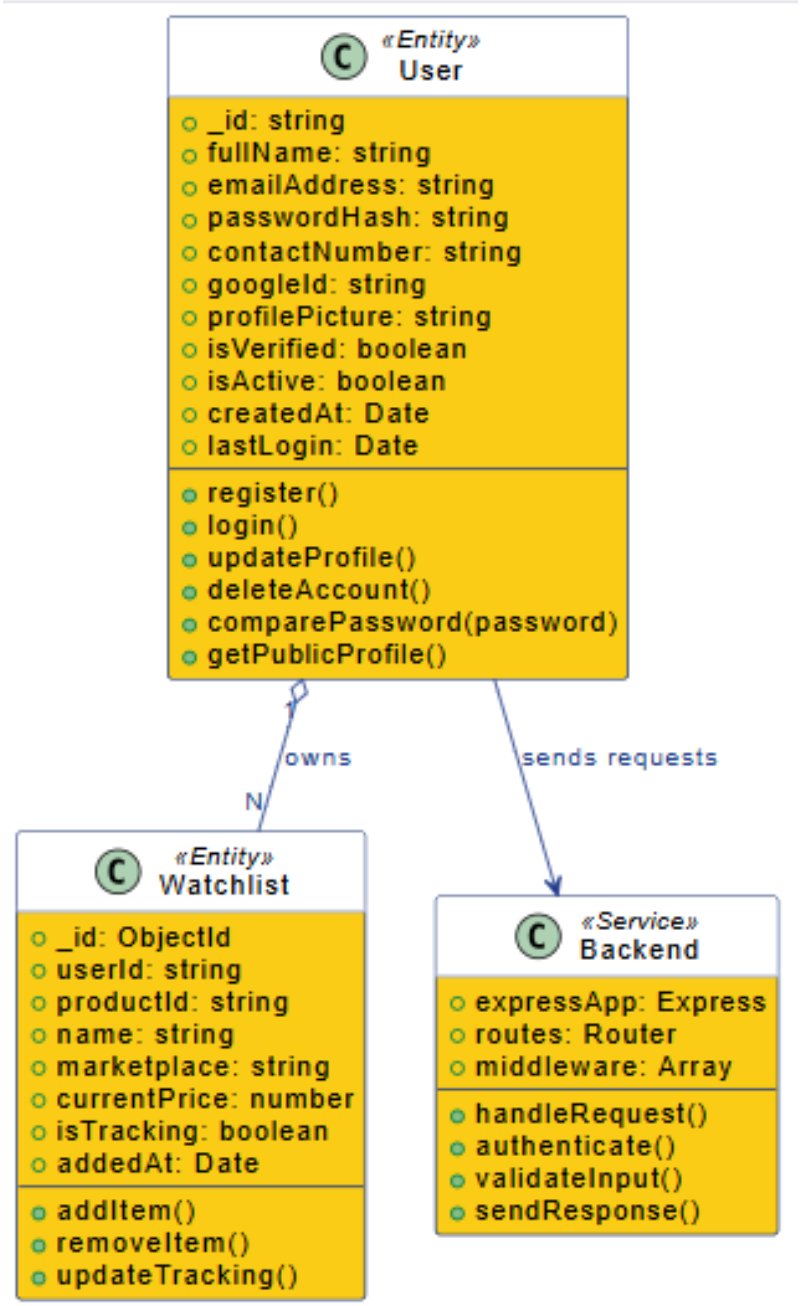

Figure 4.10: User, Watchlist and Backend module UML class diagram

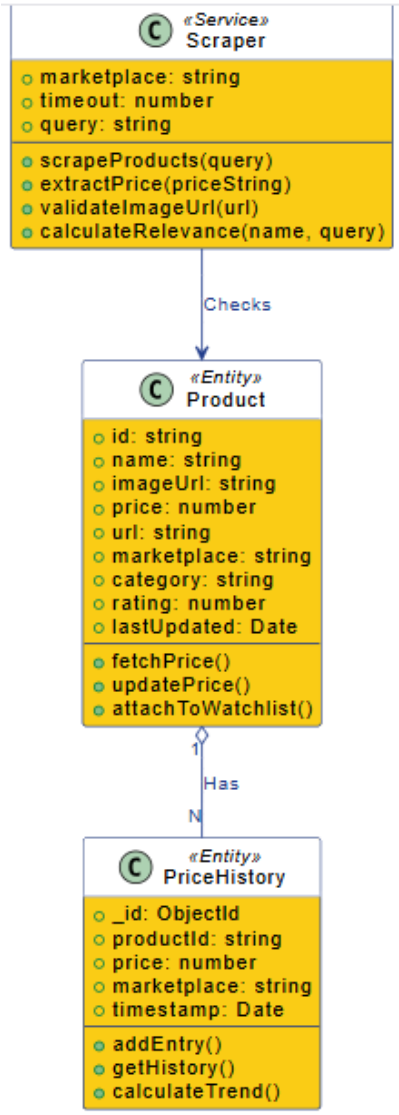

Figure 4.11: UML class diagram for Scraper, Product and PriceHistory module

4.7 GUI Design

We have designed our Smart Price Tracker web application user interface keeping in mind usability, clarity and aesthetics. The aim is to give an intuitive and visually appealing interface to help users easily search for products, monitor price changes, analyze trends and manage their watchlist. The design uses a clean responsive approach which ensures that the layout works best and can be read easily on both stand-alone and mobile devices. Consistency, both color schemes and typography, are retained across all screens so as to reduce the learning curve. With so much information, key information such as price comparisons and alert statuses, are prioritized to aid rapid decision-making. Furthermore, data visualization components such as charts are part of the design, making it possible to

gain immediate insights about the historical price volatility. This part shows the major GUI screens of the system with their specific layouts and functionality in detail.

4.7.1 Landing Page

When the user goes to the application, the landing page is shown as the first page to display. It emphasizes the main meaning of the system through an introductory slogan, text, and buttons.

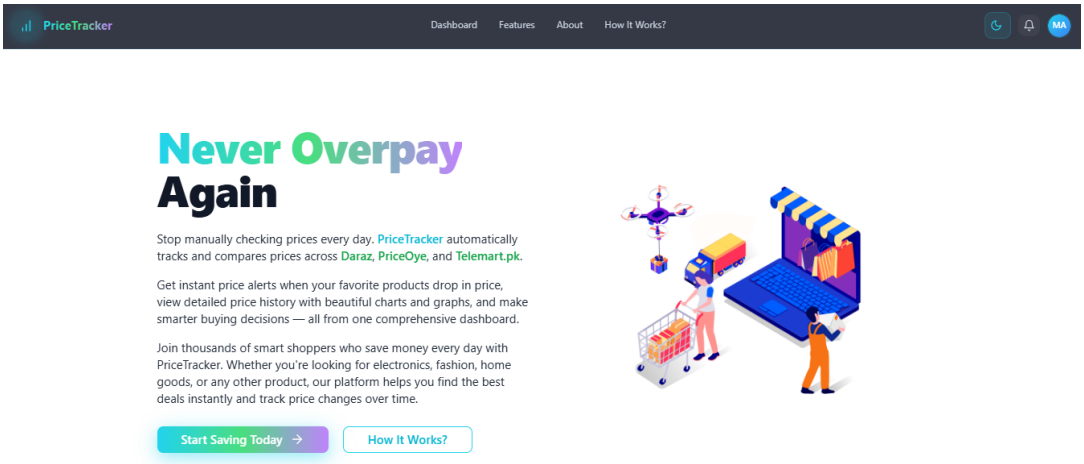

Figure 4.12: Landing Page of Smart Price Tracker

4.7.2 Login Screen

Users are able to sign in with their Google account or provide their email and password. The login interface is simple, modern and easy to navigate. The login screen is displayed in figure 4.13.

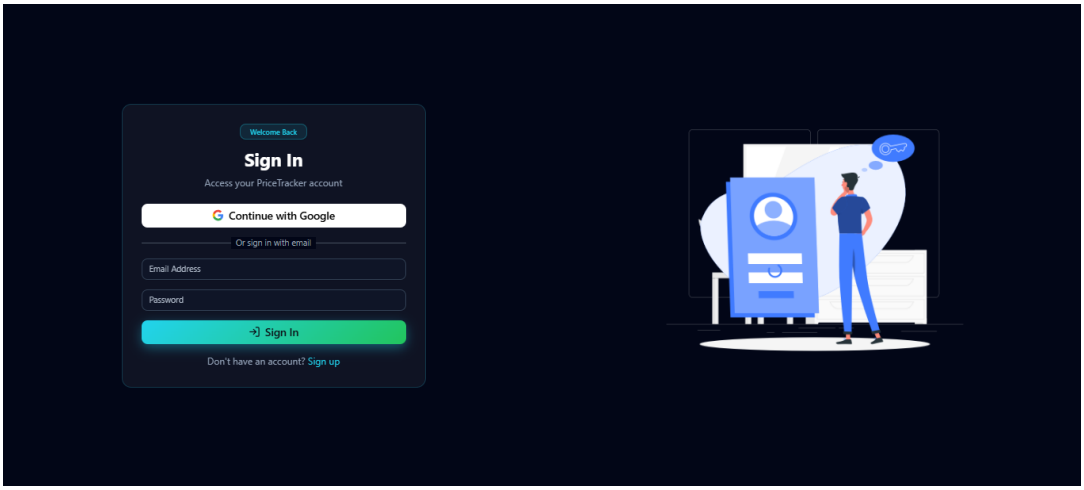

Figure 4.13: Login Screen

4.7.3 Dashboard

After logging in, it is sent back to the dashboard. It shows the best deals, price difference, potential saving and graphical comparisons between the marketplaces. The dashboard interface can be seen in Figure 4.14.

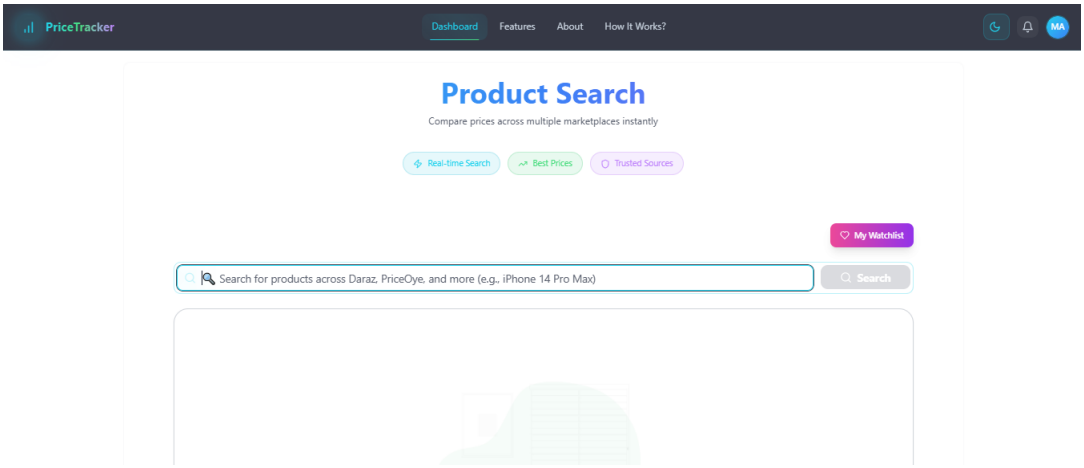

Figure 4.14: Main Dashboard

4.7.4 Price Analysis Charts

The price analysis section shows some visual charts of the lowest, median, and highest prices for each marketplace. This helps the users to take an informed decision about the purchase of product and services. these charts will give an instant visual sense of price volatility and market competitiveness between the vendors. users can quickly understand which vendor provides the lowest price range routinely for products they are following. The charts of the price analyses are shown in Figure 4.15 [15].

Figure 4.15: Price Range Analysis Charts

4.7.5 Product Search Interface

Users can search out products across Daraz, PriceOye and Telemart using the main search bar. The interface offers options for real-time search as well as multi-markets search.Figure 4.16 shows the product search screen.

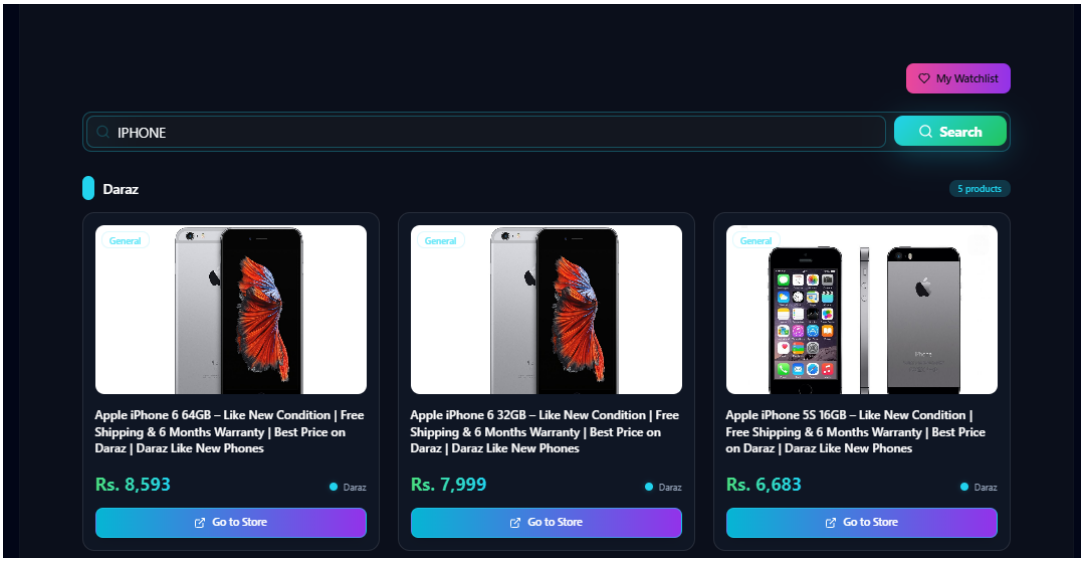

Figure 4.16: Product Search Interface

4.7.6 Watchlist

TThe watchlist lets the user store and keep track of selected items. Users can check product details, delete product, or open the product in the marketplace. The watchlist screen is shown in Figure 4.17.

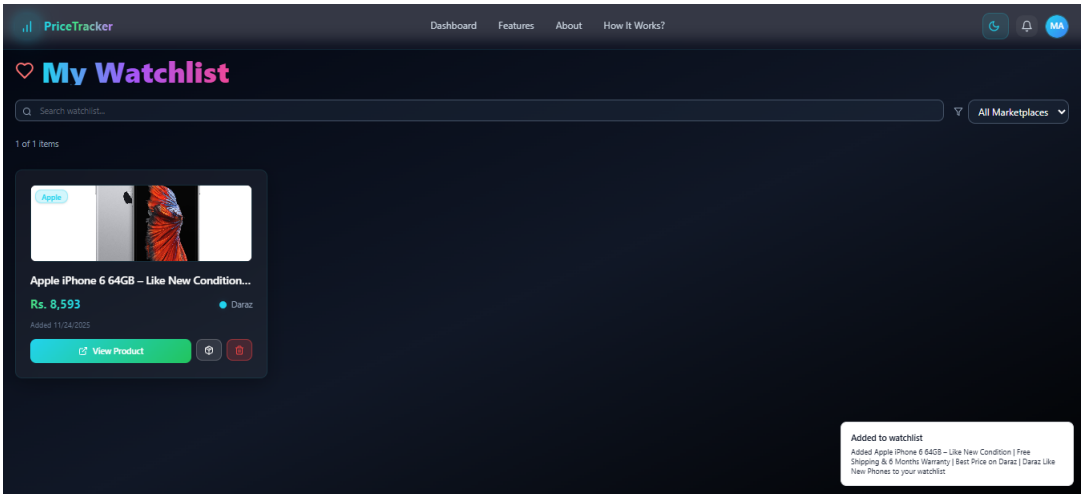

Figure 4.17: User Watchlist

4.7.7 User Profile

The profile screen shows user information as name, email, account status, date of membership, and last login. Users can also make edits to their profile from this section. The profile interface is shown in Figure 4.18.

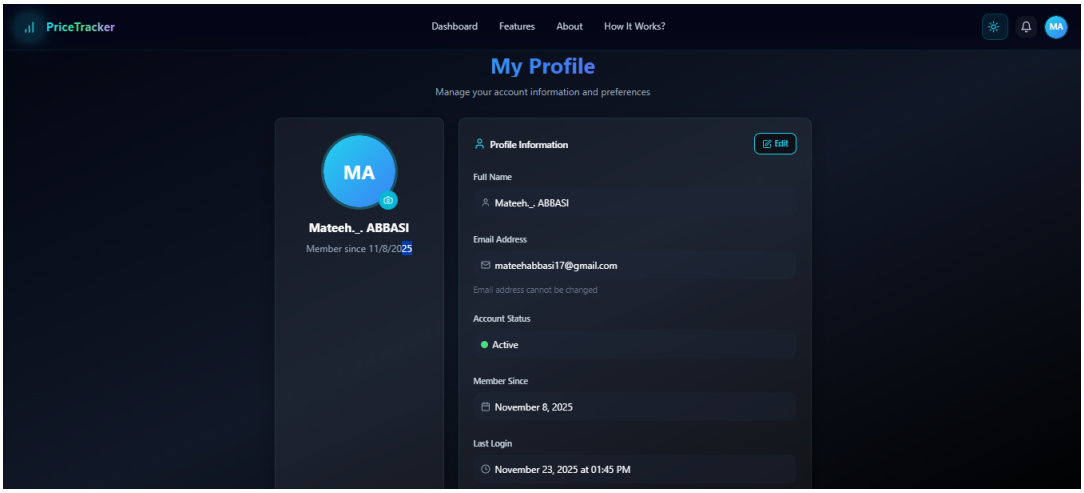

Figure 4.18: User Profile Screen

Chapter 5

System Implementation

The system implementation stage worked on turning the detailed design into a fully functional and deployable software solution.This was an essential part of the project and it focused on bringing together all the required parts in order to keep frontend and backend services, database and scraping utilities in strict alignment with the system architecture and requirements.

5.1 System Architecture

The Smart Price Tracker is developed using a multi-tier architecture in order to ensure modularity, scalability, and ease of maintenance. The system consists of the following major parts:

- **Frontend (React.js):** React.js Deals with user interactions, dashboards, watchlist management, and display of alerts.
- **Backend (Node.js and Express):** Provides the application with the means to offer the user some type of Business Logic, Routing, Authentication,and integration with web scraping modules.
- **MongoDB Database:** Stores User Information, Watchlist items, Product Meta data, Price,history, and notifications.
- **Scraping Layer:** Gets live product prices from ecommerce websites with Puppeteer and related tools.
- **Cron-Based Schedulers:** Periodic price checks and triggers if set price conditions are met.

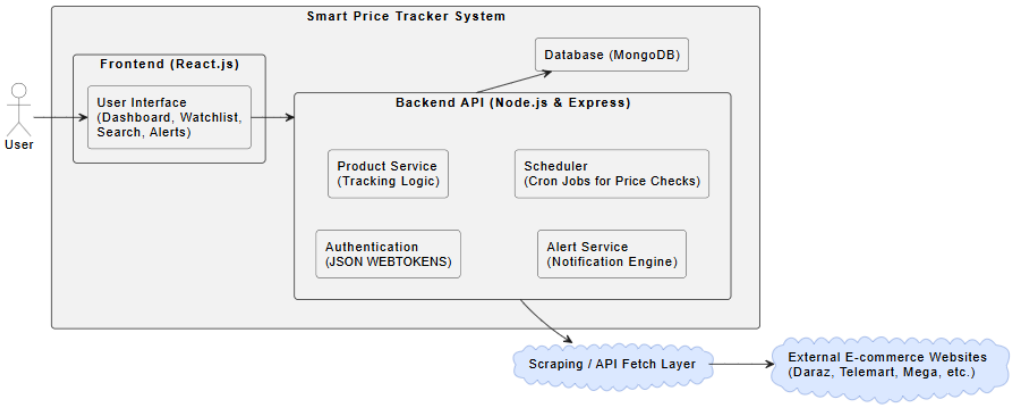

Figure 5.1: High Level System Architecture

5.2 Tools and Technologies

LaTeX

Used in the creation of structured documentation, being consistent, professional formatting, and easy reference handling.

MS PowerPoint

Used during project reviews and final evaluations in order to present system functionalities and workflows and results.

Visual Studio Code

Provides efficient space for code writing for react components, backend APIs and debugging JavaScript modules.

Postman

Widely used to test the backend endpoint such as authentication, products tracking, price fetching.

React.js

Chosen to have an interactive and responsive user interface. Its component driven structure allowed for the dashboard, watchlist, alerts, and authentication screens to be built in modules.

Node.js and Express.js

Acted as backend engine which handles request, routing, user management, price monitoring, and interaction with databases.

MongoDB

Selected as a result of its scalability as well as schema flexibility Ideal for Superior dynamic product data and user-specific watchlists.

JWT Authentication

Provides security of login sessions. Only authorized users are allowed to access protected resources such as watchlist and alert management.

Web Scraping Tools

Libraries such as Puppeteer or Cheerio were implemented to get live price data from supported e-commerce platforms.

Node-Cron

Used to perform planned background activities like price comparison and alert creation.

NodeMailer

Sends alert notification to users when the price of a product reaches or exceeds their specified target.

5.3 Security

Security is one of the main architectural factors that the Smart Price Tracker takes on board. Multiple measures were implemented to guarantee confidentiality and integrity of data:

- JWT-based authentication for user safe log in.
- Passwords were hashed before storage using industry standard hashing algorithms.
- All critical API routes are protected from unauthorized access.
- Safe communication between frontend and backend so as to avoid exposing data.

So, these measures are there for user accounts to keep them secure,Also for watchlist entries and price data to keep them secure.

Table 5.1 summarizes the software tools, frameworks, and libraries that are used in the system development.

Table 5.1: Tools and Technologies Used in the System

Tool / Technology	**Version**	**Purpose / Rationale**
LaTeX	2021	Used for professional documentation and report generation.
MS PowerPoint	2016	Used to prepare project presentation slides.
Visual Studio Code	Latest	IDE used for frontend and backend development.
Postman	Latest	API testing, debugging, and validation.
Git / GitHub	Latest	Version control and collaborative development.
React.js	18	Frontend framework used to build UI components and dashboards.
Node.js	18	Backend runtime environment for server logic.
Express.js	4	Framework used to define routes, middleware, and backend APIs.
MongoDB	6	NoSQL database storing users, products, and price tracking data.
JWT Authentication	Latest	Provides secure authentication and protected API routes.
Axios	Latest	Enables communication between frontend and backend.
Puppeteer / Cheerio	Latest	Used for scraping real-time product price information.
Node-Cron	Latest	Automates background tasks such as periodic price checks.
NodeMailer	Latest	Sends email alerts to users when prices drop.

5.4 Conclusion

The implementation of Smart Price Tracker required in-depth knowledge of modern web technologies, such as React.Js, Node.Js, Express.Js, MongoDB and asynchronous Java script. The successful integration of these technologies led to a fully functional system that is capable of monitoring product prices in real-time and providing timely notifications about the product price drop to its users.

Chapter 6

System Testing and Evaluation

System testing is performed on a complete and integrated system to verify a system's behavior to meet all specified requirements. Every software needs to be tested before being deployed to confirm on reliability, usability, stability and performance. Testing assists in finding the bugs and the problems at an early stage and thereby we get to know that we have a stable system under various operational conditions.

Our approach to the development of the Smart Price Tracker system was an iterative one:

- Code each module
- Executing a test on the module individually.
- Integrate it with the system
- Perform other kind of tests after integration.

To perform the testing we have done different types of testing depending on the nature of our web application. The following testings is performed on us:

- GUI Testing
- Usability Testing
- Software Performance Testing
- Compatibility Testing
- Exception Handling
- Load Testing
- Security Testing
- Installation (Deployment) Testing

6.1 Graphical User Interface Testing

In the case of GUI testing, the web interface of the application was tested to make sure that all the visual elements (buttons, menus, forms, links, dashboard widgets) of the application perform accurately. This includes how components have reactions to user interactions such as clicks, input entries and navigation.

Test Case – General GUI

Table 6.1: General GUI Test Case

Test Case ID	Test Case 1
Description	Tests the graphical user interface of the Smart Price Tracker web app
Environment	Browser: Chrome, Edge
Initial Condition	System deployed on localhost / production server; user not logged in
Steps	1. Open the web application. 2. Verify navigation bar links. 3. Check routing between pages. 4. Verify component loading (Dashboard, Watchlist, Search). 5. Check responsiveness on various screen sizes.
Result	PASS for all GUI components

Forms GUI Test Case

Table 6.2: Form GUI Test Case

Test Case ID	Test Case 2
Description	Tests form handling and validation in Login, Registration, and Add Product forms
Environment	Browser: Chrome 129+, Firefox 120+
Initial Condition	Backend connected; database running; user logged out
Steps	1. Open login/register page. 2. Check all form buttons that they are used to initiate the right actions. 3. Test required fields and validation messages. 4. In Add Product form enter valid and invalid URLs of product. 5. Submit without filing in required data.
Result	PASS — all validations and form components working correctly

6.2 Usability Testing

Usability testing is done to check the application is intuitive and efficient and will it ensure that the user can easily move through the dashboard, watch list and search pages to perform tasks.

Table 6.3: Usability Test Case

Test Case ID	Test Case 3
Description	Tests system usability and user experience
Environment	Standard keyboard/mouse; normal browser usage
Initial Condition	User logged in; dashboard loaded
Steps	1. Go in and out of Dashboard, Watchlist, Search. 2. Check button icons/texts are intuitive. 3. Check for simple and consistent navigation 4. Check important functions (For example : Add to Watchlist, Enable Tracking). 5. Complete full work flow (Search→Add→Track→View Alert).
Result	PASS — users were able to complete tasks smoothly

6.3 Software Performance Testing

Performance testing is a test conducted to check the speed, responsiveness, and stability of the web app. The backend logic of the scraper, API response time, cron jobs and database operations are tested.

Performance Test Case

Table 6.4: Performance Test Case

Test Case ID	Test Case 4
Description	Tests backend and frontend performance
Environment	Node.js server, MongoDB Atlas
Initial Condition	System running in production mode
Steps	1. Load the web app. 2. Measure dashboard load time. 3. Measure response time of product search and tracking. 4. Simulate multiple users Dashboard refresh. 5. Test under slow connection (throttled out to 1Mbps).
Result	PASS for all tests except slow-connection responsiveness, which was slightly delayed.

6.4 Compatibility Testing

Compatibility testing was conducted to ensure that Smart Price Tracker system is running properly on different browsers and devices. The application was tested on:

- Google Chrome
- Mozilla Firefox
- Microsoft Edge
- Safari (Mac/iOS)
- Android (Chrome Mobile)

The system was proven to be fully compatible on all the tested platforms.

6.5 Exception Handling

Several exceptions were met and resolved during development :

- **Duplicate Registration:** When user attempt to register with already registered email then system show "Email already exists
- **Invalid Product URL:** In case scraper fails to retrieve data, message "Invalid or unsupported product link" is displayed.
- **Unauthorized Access:** When token is missing/ invalid then backend will return "401 Unauthorized"
- **Scraping Failure:** Errors will block the requests from happening or network error will be logged and then retried later.

6.6 Load Testing

Load testing is a way to test the system to see how it operates when many different users are interacting with it at the same time. The application was tested using:

- Concurrent dashboard requests - 50.
- Simultaneous product tracking requests (20).
- Simultaneous execution of 5 cron price checks.

The system was found to be stable and responsive to the load tested.

6.7 Security Testing

Security testing ensures that the system is secure and meets user data protection requirements and provides protection from unauthorized access. Following security test case has been performed.

Security Test Case

Table 6.5: Security Test Case

Test Case ID	Test Case 5
Description	Tests application security and data protection
Environment	Node.js backend, JWT authentication, MongoDB encryption
Initial Condition	User logged in; backend server running
Steps	1. Login and logout. 2. Make sure that no user data is presented after user logouts. 3. Check storage of passwords, to ensure that passwords are stored in hashed form (bcrypt). 4. Attempt log in using invalid credentials. 5. Attempt to get access to protected API without token.
Result	PASS — all security checks successful

6.8 Installation Testing

Installation (deployment) testing checks that the system is installed properly and that the features of the system are functional after deployment.

The Smart Price Tracker was implemented on the environment of the cloud hosting. After deployment:

- All frontend pages were loaded correctly
- All APIs having a response as per expectation.
- MongoDB Atlas Connector was successful.
- Scraping, cron jobs, and notification services worked as intended.

The system passed the installation test successfully.

Chapter 7

Conclusions

The Smart Price Tracker system simplifies and modernizes the process of prices monitoring across multiple e-commerce platforms. Instead of having to manually check different websites, this is helpful for users because an automated platform that monitors the changes in prices, and that sends timely alerts, this saves time and allowing informed purchasing decisions.

This report described the motivation for the system, including the need for an unified price-tracking solution. The literature review indicated shortages with current tools and explained how Smart Price Tracker makes automating, multi-platform support, and usability better through its Watchlist based tracking model.

The development was done in a structured software engineering process that included requirement analysis ,System design and implementation, and testing. Some basic components included are frontend development, web scraping modules, scheduled tracking, authentication and alert workflows. Documentation assisted the development team in each stage.

The application was successfully completed within the allotted timeframe with a strong focus on usability. Users can search products, save them to a Watchlist, as well as track their prices in real-time, and receive notiffications on thresholds being met. The dashboard gives a good overview of tracked down items, thus less surprising need to check by hand.

Overall, Smart Price Tracker reveals the usefulness of automated price monitoring tools and has good prospects of further expansion in a comprehensive, multi-platform price intelligence system.

7.1 Contributions

The project has met functional as well as non-functional requirements successfully. Major contributions include:

- Development of a complete Watchlist system which enable users to add, manage and remove tracked products.
- Implementation of real time price monitoring with automated background processes.
- Integration with multiple e-commerce platforms to support cross website product comparison.
- A user centred interface made specifically to make search, select and keep track of items easier.
- A strong backend architecture that can work with product updates, alerts, scraping assignments and scheduled executions.
- Comprehensive documentation includes UML diagrams ,design models prepared during the system analysis process,

7.2 Disciplined Project Management

The successful implementation of the project was enabled through a structured planning and adherence to principles of software engineering. Effective time management was the key, particularly when coordinating the development of frontend and backend APIs and web scraping logic and deployment configurations. Breaking work into milestones, specifying responsibilities, and putting the core modules first, ensured deadlines were met even in times of technical challenges emerged. Poor management of time/poor allocation of tasks would have led to delays and unresolved dependencies.

7.3 Team Communication

Consistent and effective team member communication was a central part of resolving problems in the most efficient way and keeps the development in the same direction. Regular discussions contributed to frontend and backend workflow synchronization, mismatch implementation prevention and ensure that all members developed a common understanding of the system architecture including Watchlist operations alert mechanisms Data flow between scraping modules and the database. Open communication was a contributing factor to continued motivation and productivity throughout the project lifecycle.

7.4 Future Work

While the Smart Price Tracker is fully functional in the current version, a number of enhancements can stand to further improve its capabilities and user experience

7.4.1 Cross-Platform Availability

The system could be upgraded in the later version to include the following:

- an exclusive Android application
- an iOS application,
- and browser extension for on-page instant price detection.

Expanding the system to multiple platforms will help improve accessibility and usability by a wide margin.

7.4.2 Additional Features

A number of advanced features can make the platform even cooler:

- Interactive price graphs to see historical price graphs.
- Summary dashboards show long-term trends in prices.
- A.I. based predictions to advise best time to buy.
- Integration of other e-commerce websites to increase coverage.
- Improved filtering and categorization control to make it easier to navigate through the Watchlist.

Appendix A

User Manual

This User Manual gives a full explanation on how to use the Smart Price Tracker web application. It describes all the key features, workflow of navigation, interface components to help the users effectively track product prices across multiple e-commerce platforms.The information contained within will make sure you can get the most out of the system for making informed purchasing decisions.

A.1 Introduction

The Smart Price Tracker is a web-based system that helps users to compare prices, view historical trends, and track prices of products in real time across popular marketplaces like Daraz, PriceOye, and Telemart. It aims to save time and money to users by automating the process of price monitoring and price comparison.

This manual is intended for:

- New users that are interacting with the system for the first time including essential setup and navigation.
- Registered users who wish to take a deeper dive into such advanced features as watchlist management, and price alerts, as well as detailed analytics.

A.2 System Requirements

To ensure and ensure optimum performances and smooth user experience, please note the following requirements:

- **Browser Support:** Google Chrome 95+, Firefox 90+, Edge 100+. The application is optimized for these modern browsers.
- **Internet Connection:** Stable Connection suggests for real-time fetching of price data.

- **Account Requirements:** Optional Log in with Google, Standard Account Creation (email) is required to use personalized features such as Watchlist.

A.3 Getting Started

A.3.1 Landing Page

When users open the application, they are greeted with an attractive homepage introducing the features of the system and navigation options.

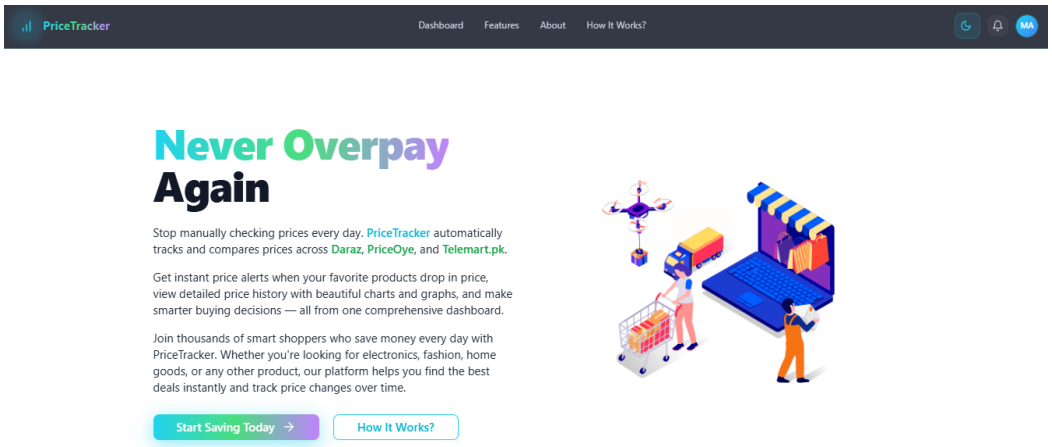

Figure A.1: Landing Page

A.3.2 Sign In

Users can use Google or email - password login.

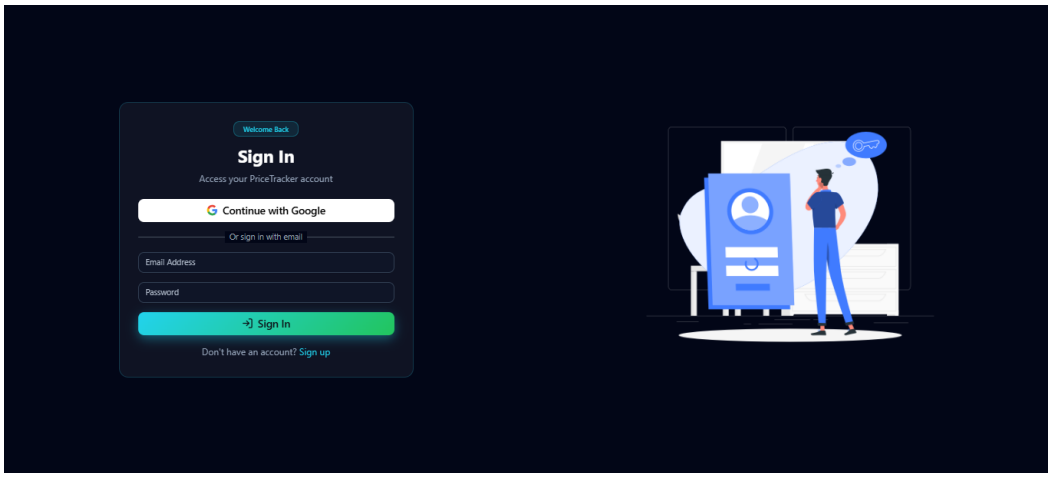

Figure A.2: Sign In Interface

The interface is designed to be maximally comfortable and secure for the user - emphasizing the following aspects:

- **Flexible Entry:** : Log in instantly with the use of Google or your registered e-mail and password.
- **Secure Access:** A Secure Access Make sure you have a path for password recovery.

A.4 Dashboard Interface

The dashboard allows users to get a real-time visualization of trends in product data including:

- Best deal across markets.
- Price difference highlights.
- Savings opportunities.
- Price range comparison charts.

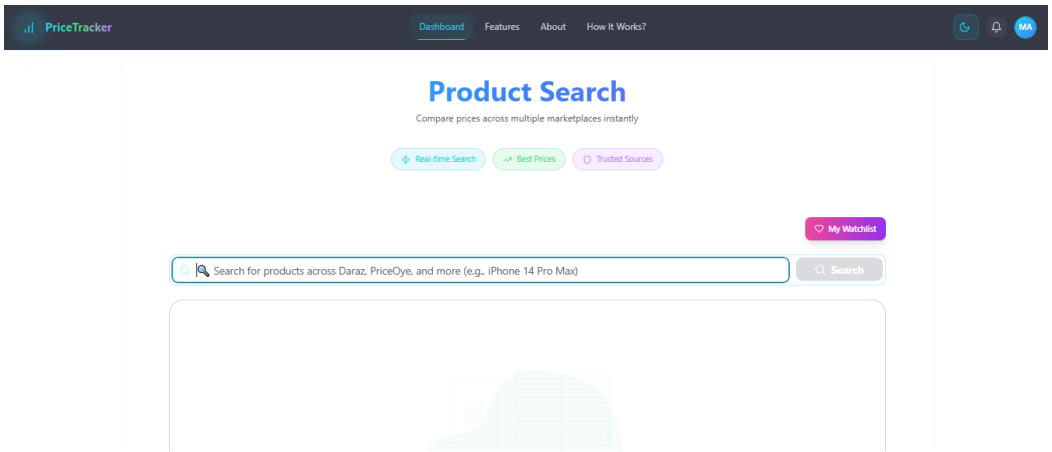

Figure A.3: Dashboard Overview

A.5 Product Search

Users can search for any of the products on supported platforms in real time.

- Enter keyword (e.g., “iPhone 14 Pro”).
- Get the price comparison results instantly.

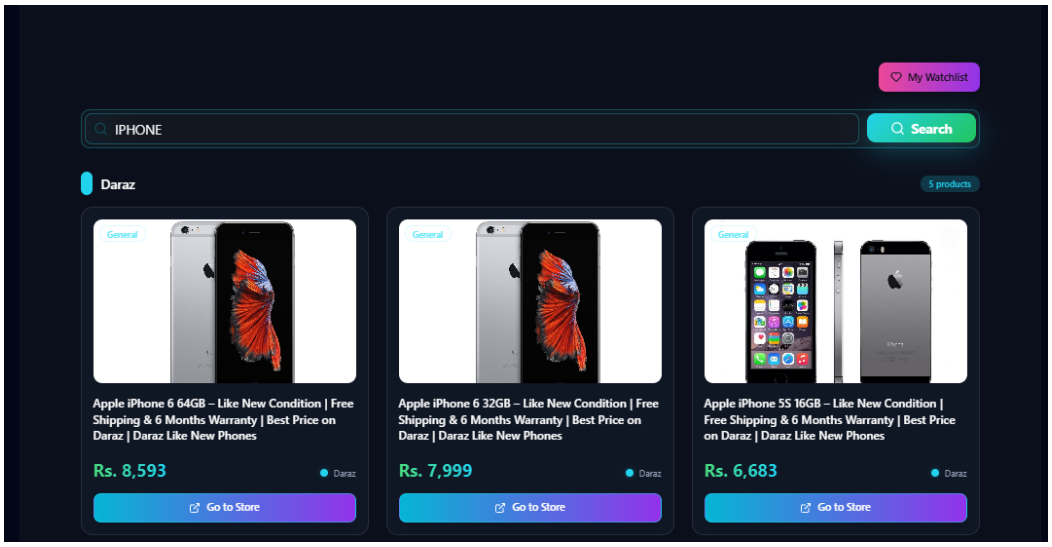

Figure A.4: Product Search Interface

A.6 Search Results Display

After the search is done, the system shows:

- Product image and name.
- Price from each marketplace.
- “Go to Store” button for the direct purchase.

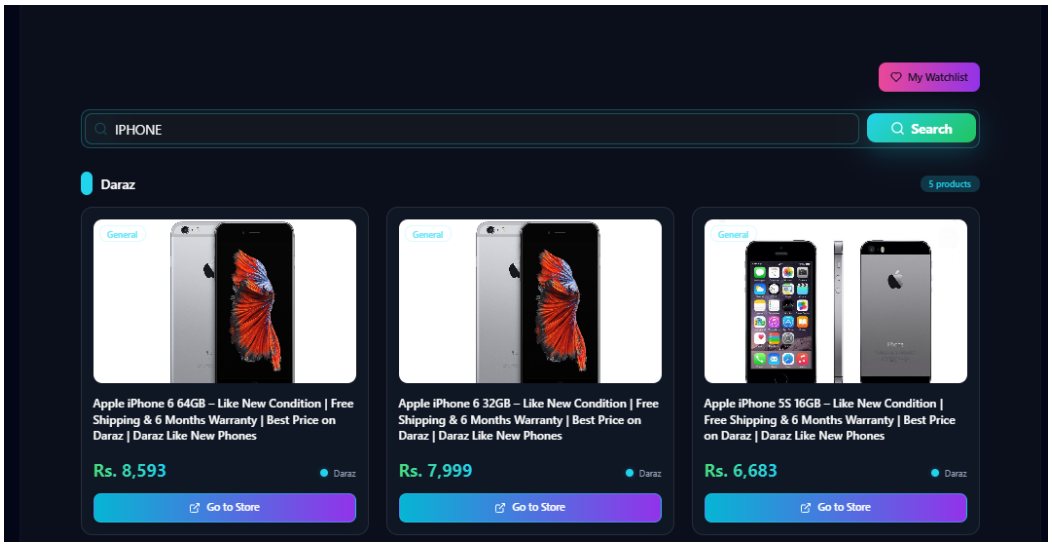

Figure A.5: Search Results Page

A.7 Watchlist Management

Users can add any product to their watchlist, where it will be used to track its price automatically.

- Add/remove items.
- View latest price updates.

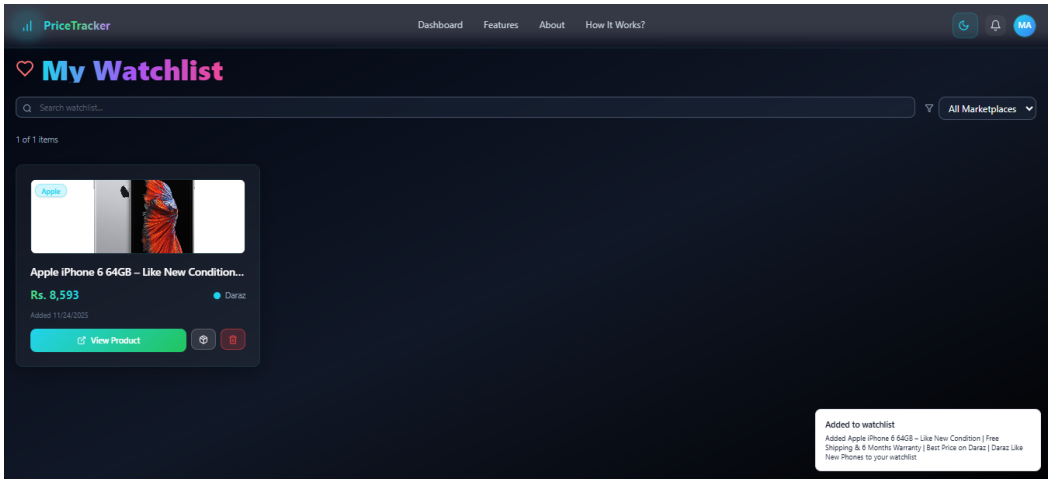

Figure A.6: User Watchlist

A.8 Profile Management

Users can manage their account details, get membership date and change allowed fields.

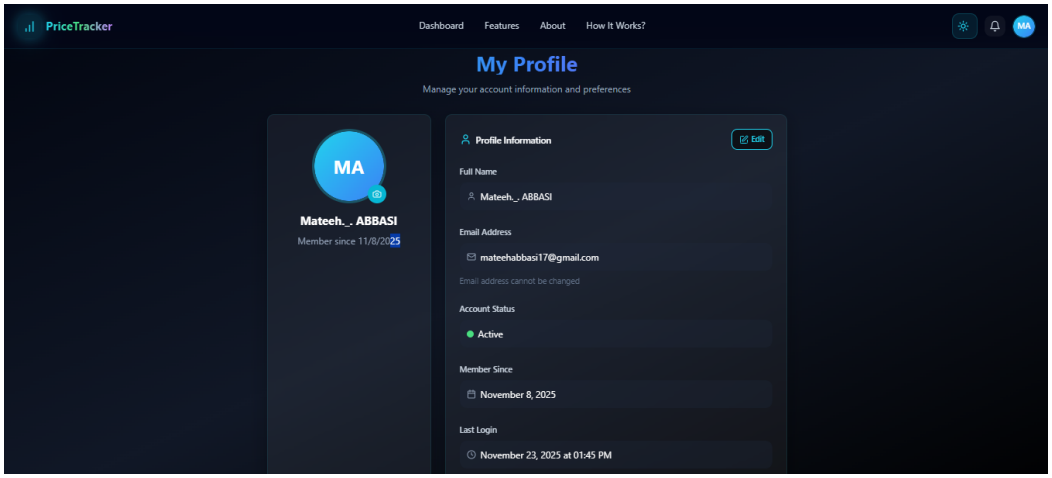

Figure A.7: Profile Page

A.9 Notifications and Alerts

The system allows automatic alerts in the case of:

- A product's price gets below target.
- New deals are detected.
- Watchlist items receive updates.

The notifications are displayed on-screen and via email if desired.

A.10 Troubleshooting

Login Issues:

- Ensure correct email/password.
- For Google login, verify Gmail session is active.

Products Not Appearing:

- Check spelling variations.
- Ensure stable internet connection.

No Price Alerts:

- Confirm watchlist item is enabled for tracking.
- Check spam folder for email expected alert.

A.11 Conclusion

This manual presented a full overview of the user-interface and features of the Smart Price Tracker system. It works as a quick guide for the new and returning user to help them fully utilize the capabilities of the system.

References

[1] Google Developers. Puppeteer documentation. https://pptr.dev/api, 2025. Accessed: 30-Nov-2025. Cited on pp. 1 and 2.

[2] Microsoft Corporation. Playwright testing and automation guide. https://playwright.dev/docs/writing-tests, 2025. Accessed: 30-Nov-2025. Cited on pp. 1 and 2.

[3] Express.js Foundation. Express.js framework documentation. https://expressjs.com/, 2025. Accessed: 30-Nov-2025. Cited on p. 2.

[4] React.js Team (Meta Platforms Inc.). React official documentation. https://react.dev/, 2025. Accessed: 30-Nov-2025. Cited on p. 2.

[5] MongoDB Inc. Mongodb manual. https://www.mongodb.com/docs/, 2025. Accessed: 30-Nov-2025. Cited on p. 2.

[6] Telemart.pk. Telemart api and product catalog. https://www.telemart.pk, 2025. Accessed: 30-Nov-2025. Cited on p. 3.

[7] D. Khairkar, S. Patel, A. Sharma, and M. Gupta. Real-time online price tracker across multiple e-commerce websites. *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*, 11(V):216–222, 2023. Cited on pp. 5 and 6.

[8] L. Jin and L. Chen. Exploring the impact of computer applications on cross-border e-commerce performance. *IEEE Access*, 12:49302–49313, January 2024. Cited on p. 5.

[9] CamelCamelCamel Team. Amazon price history tracker documentation. https://camelcamelcamel.com/, 2025. Accessed: 30-Nov-2025. Cited on p. 5.

[10] Keepa GmbH. Amazon price tracking api. https://keepa.com/, 2025. Accessed: 30-Nov-2025. Cited on p. 5.

[11] Honey Science LLC (PayPal Inc.). Honey browser extension documentation. https://help.joinhoney.com/, 2025. Accessed: 30-Nov-2025. Cited on p. 5.

[12] PriceBlink Team. Shopping assistant extension guide. https://www.priceblink.com/, 2025. Accessed: 30-Nov-2025. Cited on p. 5.

[13] Prisync Inc. Competitor price monitoring platform. https://prisync.com/competitor-price-tracking/, 2025. Accessed: 30-Nov-2025. Cited on p. 6.

[14] Price2Spy (XMLShop). Price tracking and analytics documentation. https://www.price2spy.com/, 2025. Accessed: 30-Nov-2025. Cited on p. 6.

[15] Chart.js Contributors. Chart.js data visualization library documentation. https://www.chartjs.org/docs/latest/, 2025. Accessed: 30-Nov-2025. Cited on p. 34.