Arzu Vats, 2026, 14:6
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology
An Open Access Journal

SHOP SYNC: An Asynchronous E-Commerce Product Aggregation System

Arzu Vats[1], Saksham Aggarwal[2], Sanjay[3], Aakash Raghav[4]
[1,2,3,4]Department of Computer Science and Engineering, HMRITM, Hamidpur, New Delhi, Delhi, India

Abstract- The rapid growth of e-commerce has changed the way people shop online, providing access to a wide range of products through different shopping platforms. However, searching for products across multiple websites and comparing their details can be time-consuming and inconvenient. Users often need to visit several platforms individually to find suitable products, check their prices and manage their shopping choices. SHOPSYNC is a web-based e-commerce product aggregation system developed to simplify the online shopping experience by bringing product information together through a single platform. The primary objective of the project is to provide users with a convenient way to search for products, view relevant product information and manage their shopping activities without repeatedly navigating between different websites. The system is developed using React for the frontend and FastAPI for backend services. It uses web scraping techniques to retrieve product information from e-commerce platforms and asynchronous processing to handle requests efficiently. Redis is integrated as a caching mechanism to store previously retrieved data and improve the efficiency of repeated searches. The application also includes user authentication and shopping-related features such as wishlists and cart management. The project focuses on creating a structured, user-friendly and efficient platform that simplifies product discovery and information retrieval. The current implementation integrates with Meesho, while the system architecture provides scope for incorporating additional e-commerce platforms in the future. SHOP SYNC demonstrates the practical application of modern web technologies, data extraction techniques and caching mechanisms in developing a centralized product aggregation platform. It aims to make online product searching more convenient, organized and accessible for users

Keywords- E-commerce, product aggregation, web scraping, FastAPI, Redis, product discovery, etc.

I. INTRODUCTION

The growth of the internet and digital technologies has transformed the way businesses operate and consumers purchase products. Online shopping has become an important part of modern life because it provides convenience, accessibility and a wide range of choices. Consumers can explore products from different categories, examine specifications and place orders without visiting physical stores. The increasing popularity of e-commerce has also encouraged businesses to establish their presence on digital marketplaces, resulting in a highly competitive online shopping environment.

Although online shopping provides numerous advantages, finding the right product can still be challenging. Consumers frequently visit multiple e-commerce platforms to explore available options and compare prices, features and specifications. Different platforms may display similar products with variations in pricing, descriptions, discounts and availability. Consequently, users often need to repeat their searches across several websites, increasing the time and effort required to make purchasing decisions. The availability of incomplete or outdated product information can further complicate this process.

Another challenge is the fragmentation of product information across online marketplaces. Each platform generally maintains its own product listings, search interface and presentation format, making direct comparisons difficult. Research on online consumer behaviour and price comparison highlights the importance of accessible product information in helping consumers evaluate alternatives and make informed purchasing

© 2026 Arzu Vats. his is an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly credited.

decisions [11–13]. Product aggregation systems can address this issue by collecting information from different sources and presenting it through a common interface. Techniques such as web scraping and caching provide a foundation for collecting and managing this information efficiently [10].

The motivation behind this research is to make online product discovery more convenient through a centralized platform. SHOP SYNC is developed as a web-based product aggregation application that allows users to search for products and view retrieved information in an organized manner. The current implementation integrates Meesho as a product source and also provides supporting features such as user accounts, wishlists and cart management. The system aims to reduce the need for users to navigate between multiple websites while exploring products. It also focuses on presenting relevant product details in a consistent format to make browsing and evaluating available options easier.

This research focuses on the design and development of a product aggregation system that combines data extraction, backend processing and caching to support product discovery. Its behaviour is examined through searches across different product categories, including observations of response times and cached data sizes. The project also provides a foundation for future improvements, such as integrating additional e-commerce marketplaces, enhancing product comparison and introducing more advanced search and filtering capabilities.

II. PROBLEM STATEMENTS AND OBJECTIVES

Problem Statement

The increasing number of e-commerce platforms has made online product searching and comparison a time-consuming process. Users often need to visit multiple websites separately to find products, compare prices, examine specifications and check availability. Research on online consumer behaviour and price comparison highlights the importance of accessible product information in helping users evaluate different purchasing options [11–13].
Most online shopping platforms primarily provide information about products listed within their own marketplace. This limits users who want to explore products from different sources without repeatedly switching between websites. Additionally, retrieving product information through repeated requests can increase response time and create unnecessary processing overhead.

Therefore, there is a need for a centralized product aggregation system that can collect and organize product information through a unified interface. Existing research on web data extraction and caching provides a foundation for developing such systems and reducing repeated retrieval operations [10]. Such a system should simplify product discovery and provide basic shopping management features to improve the overall user experience.

Objectives

- Provide a unified interface for searching products from a supported marketplace.
- Use asynchronous backend operations so network-bound retrieval does not block requests.
- Cache suitable results in Redis to avoid repeated retrieval and support prefetching.
- Provide authentication and personal wishlist, cart and comparison-list operations.

III. LITERATURE REVIEW

E-Commerce and Product Discovery

Research on e-commerce recommendation systems has explored ways to help users identify relevant products from large online catalogues. Wei et al. [1] examined recommendation systems in e-commerce, while Schafer et al. [2] discussed their applications in online shopping. Adomavicius and Tuzhilin [3] provided an overview of recommendation techniques and their development. Other studies have examined hybrid and advanced recommendation approaches [4, 5, 14–17]. Although recommendation systems focus on suggesting products based on user preferences, product aggregation focuses on collecting and presenting

product information from available sources. This distinction is important to the design of SHOP SYNC.

Web Data Extraction

Web data extraction is used to collect information from websites and convert it into a structured format. Ferrara et al. [6] reviewed web data extraction techniques and their applications, while Laender et al. [7] examined different tools for extracting information from web pages. Chang et al. [8] discussed web information extraction systems, and Cafarella et al. [9] explored the use of structured information available on the web. These studies provide a foundation for developing product extraction systems. In an aggregation application, extracted information must be organized consistently so that product details from a source can be displayed in a common format.

Price Comparison and Online Shopping

Previous studies have examined price differences across online retailers and the role of price information in consumer decisions. Bock et al. [11] investigated price comparison and price dispersion among products and retailers. Akimoto and Takeda [12] studied price movements in the online home electronics market, while Bodur et al. [13] examined the influence of price comparison websites on consumer price evaluations. These studies highlight the relevance of accessible product and pricing information when evaluating purchasing options. A product aggregation system can support this process by presenting retrieved product details along with their source information.

Caching and Performance Optimization

Caching is a technique used to store previously retrieved information so that it can be accessed without repeating the original retrieval process. Podlipnig and Böszörményi [10] reviewed web cache replacement strategies and discussed different approaches to managing cached information. Their work provides a basis for understanding cache management in web applications. In product aggregation systems, caching can help reduce repeated requests to external sources. However, cached information must also be managed carefully because product prices and availability can change over time.

Research Gap

The reviewed studies address recommendation systems, web data extraction, price comparison and caching as separate areas of research. A practical product aggregation application needs to combine these concepts to support product discovery through a common interface. SHOP SYNC applies these concepts in a web-based prototype using product extraction, asynchronous backend processing and Redis caching. Its current implementation supports Meesho, while integration with additional marketplaces and completion of the full product comparison interface remain future development areas [18].

IV. METHODOLOGY

Research Approach

This study follows a design-oriented system development approach. It focuses on the design and implementation of SHOP SYNC, a web-based product aggregation application that retrieves product information from an e-commerce marketplace and presents it through a centralized interface. The system is developed using separate frontend, backend, product extraction and caching components. The methodology describes the system requirements, technologies, architecture and implemented functions based on the project documentation [18].

System Requirements

The functional requirements define the main operations supported by SHOP SYNC. The system allows users to search for products using a search query and view the retrieved product information through a paginated interface. It supports user registration, login, logout and session validation. Authenticated users can add or remove products from their wishlist and cart.

The backend also provides comparison-list operations, along with health-check and API documentation endpoints. Product retrieval and

reuse of cached results are included in the search workflow [18].

The non-functional requirements focus on the reliability, responsiveness and security of the application. Asynchronous input/output operations are used to handle network requests without unnecessarily blocking backend processing [19, 20]. The system includes input validation and handling of malformed data, while password hashing, session management and CORS configuration provide security-related safeguards. It also accounts for errors from the external product source and possible request limitations [18].

Technology Stack

SHOP SYNC uses React 19, Hooks and CSS3 for frontend development and interface management. The backend is developed using Python and FastAPI, with Uvicorn used to run the application server. Asynchronous request handling is supported through Python's asyncio framework and FastAPI's async capabilities [19, 20].

The retrieval module uses httpx to perform asynchronous HTTP requests and extract product information from the supported marketplace. Redis is used as the caching layer to store retrieved results and support their reuse and prefetching [18, 21]. Pydantic is used for request and data validation, while bcrypt supports password hashing. HTTP-only cookies are used for session-token handling. Docker and Docker Compose provide the environment for running and managing the application services [18].
System Architecture

SHOP SYNC follows a layered request-processing architecture, as illustrated in Fig. 3. When a user enters a product search query, the React frontend sends a request to the FastAPI backend. The backend validates the input and checks Redis for a suitable cached result. If a valid result is available, it is returned to the frontend without requiring another external retrieval request.

If the requested information is not available in the cache, the backend uses the scraper module to retrieve product information from Meesho through asynchronous HTTP requests. The extracted information is processed, organized into a consistent product structure and stored in Redis for possible reuse. The backend then returns the processed information to the frontend for display. The success of retrieval depends on the availability of the external source, successful data extraction and cache freshness [18].

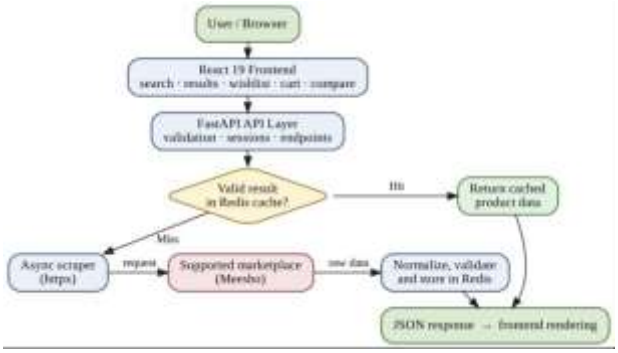

Figure 1: Request flow and architecture

Authentication and Personal Lists

The application provides user registration, login, logout and session-validation functionality. Passwords are hashed using bcrypt, while session tokens are handled through HTTP-only cookies. Request validation, CORS configuration and session-expiry settings provide additional safeguards for user-related operations [18].

Authenticated users can manage their personal wishlist and cart through the available backend operations. The system also provides comparison-list operations at the API level; however, the complete user-facing comparison interface remains planned for future development. These functions allow SHOP SYNC to combine product discovery with basic personal shopping management within a single application.

User Interface Implementation

The frontend of SHOP SYNC is developed using React 19, Hooks and CSS3. The home page provides a search bar, popular search queries, marketplace selection and feature cards for navigating the application.

The search results page displays retrieved products through individual product cards containing available information such as product title, price, rating and product actions. Users can navigate through paginated results and use the available wishlist and cart features.

The application also provides registration and login interfaces for user authentication. Comparison-list operations are available through the backend API, although the complete user-facing comparison interface remains planned for future development [18].

Asynchronous Backend and Caching

The backend is developed using FastAPI and Uvicorn, with httpx supporting asynchronous HTTP requests. Python's asyncio framework enables concurrent network operations without unnecessarily blocking request processing [18–20]. Redis is used as the caching layer to store retrieved product information and reuse suitable results during subsequent searches. The application also uses asynchronous Redis operations, connection pooling, pagination and prefetching to support product retrieval. Retry handling with exponential backoff helps manage temporary request failures. Caching reduces the need for repeated retrieval, although cached information must be refreshed to account for changes in product prices and availability [18, 19, 21].

Product Extraction and Data Model

The product extraction module is maintained separately from the main API routes and currently supports Meesho. It uses asynchronous HTTP requests to retrieve product information and organizes the extracted details into a structured format. Available product fields include the title, price, image URL, source URL and availability. This consistent structure allows the frontend to display retrieved products in an organized manner [18].

Error Handling and Security

The application includes input validation and error handling for search requests and external data retrieval. It accounts for issues such as request timeouts, unavailable pages and malformed product information. For user authentication, passwords are hashed using bcrypt and session tokens are handled through HTTP-only cookies. CORS configuration and session validation provide additional safeguards for user-related operations [18].

Deployment

Docker and Docker Compose are used to configure and run the application services in a containerized environment. The backend provides interactive API documentation through Swagger UI and ReDoc, along with a health-check endpoint for checking service availability. Environment-based configuration and logging also support application setup and monitoring [18].

V. RESULTS AND DISCUSSION

The project primarily uses secondary data obtained from external e-commerce marketplaces, including product details such as names, prices, images, ratings and availability. This information is retrieved and presented through SHOP SYNC in a centralized interface. Primary data is also collected during system testing in the form of product search response times and Redis cache storage measurements.

Overall System Results

SHOP SYNC was developed as a web-based product aggregation application that combines product discovery and personal shopping operations in one interface. The system integrates a React frontend with a FastAPI backend and uses asynchronous requests and Redis caching for product retrieval. It currently supports Meesho product searches, user registration, login, wishlist and cart operations.

The backend also provides API documentation, health checks and containerized deployment. Comparison-list operations are available through the API, while the complete comparison interface and additional marketplace integrations remain planned for future development.

Response-Time Evaluation

Response-time testing was conducted across five product categories: bags, mobile phones, earbuds,

wireless headphones and air purifiers. Initial search times ranged from approximately 6.45 to 8.08 seconds, while average subsequent search times ranged from 757.5 to 848 milliseconds. The overall average for subsequent searches was approximately 800 milliseconds.

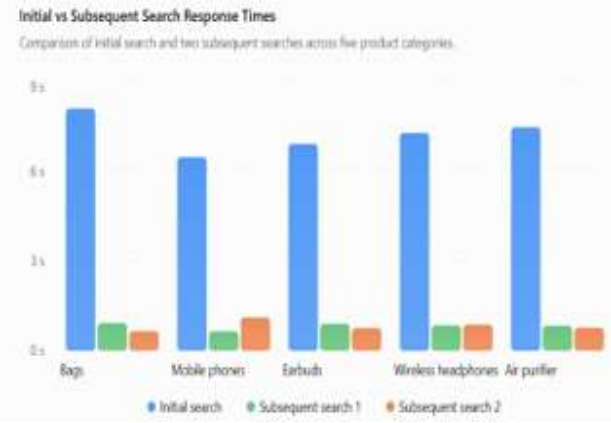

Figure 2: Comparison of Initial and Average Subsequent Search Response Times.

The results show considerably faster subsequent searches under the tested conditions. This is consistent with the expected benefits of Redis caching and asynchronous processing [19–21]. The response-time evaluation was conducted using Meesho across five product categories.

Redis Cache Storage Analysis

The cache storage measurements varied across the five categories. Bags occupied 312,849 bytes, mobile phones 273,997 bytes, earbuds 289,163 bytes, wireless headphones 309,270 bytes and air purifiers 1,366,237 bytes.

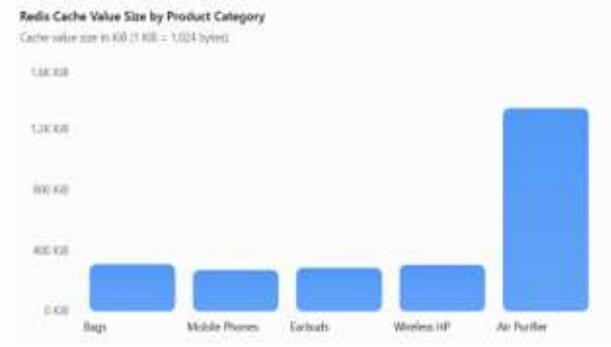

Figure 3: Redis Cache Value Size by Product Category.

Air purifiers occupied the largest cache size, while mobile phones occupied the smallest. These measurements demonstrate that Redis stores retrieved product information for reuse [21]. Redis cache storage measurements were collected from the Meesho-related cache entries for the selected product categories.

Discussion

The implementation and testing demonstrate that SHOP SYNC combines product aggregation, asynchronous processing and caching in a single application. The response-time observations indicate faster repeated searches, while the cache measurements show that retrieved product information is stored for reuse.

However, testing was limited to five product categories and a small number of searches. Further evaluation under different network conditions, larger datasets and higher request loads would be required to assess performance more comprehensively.

Future Scope

SHOP SYNC can be further developed by integrating additional e-commerce marketplaces to provide product information from multiple sources through a common interface. The product comparison feature can also be completed to allow users to compare prices and other product details more conveniently. Additional features such as advanced search filters, price alerts, user profiles and order history can further improve the application's functionality.

Future development can also focus on improving product extraction accuracy, cache management and system performance under higher user loads. More extensive testing with larger datasets and different network conditions would help evaluate the reliability and scalability of the application. These improvements can make SHOP SYNC a more comprehensive product aggregation platform.

VI. CONCLUSION

SHOP SYNC was developed as a web-based product aggregation system to simplify product discovery by

presenting product information through a centralized interface. The application combines a React frontend with a FastAPI backend and uses asynchronous request handling and Redis caching to support product retrieval. It also provides user authentication, wishlist and cart operations, making it useful for basic online shopping activities. The response-time evaluation showed considerably faster subsequent searches compared with initial searches across the five tested product categories. The cache storage measurements also demonstrated the storage of retrieved product information in Redis. The system includes marketplace-related data for Flipkart, Meesho, Shopsy and Myntra. Overall, SHOP SYNC demonstrates how web technologies, asynchronous processing and caching can be combined to build a practical product aggregation application.

REFERENCES

1. K. Wei, J. Huang, and S. Fu, "A Survey of E-Commerce Recommender Systems," in Proceedings of the International Conference on Service Systems and Service Management (ICSSSM), 2007, doi: 10.1109/ICSSSM.2007.4280214.
2. J. B. Schafer, J. A. Konstan, and J. Riedl, "E-commerce recommendation applications," Data Mining and Knowledge Discovery, vol. 5, nos. 1–2, pp. 115–153, 2001, doi: 10.1007/978-1-4615-1627-9_6.
3. G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 6, pp. 734–749, 2005, doi: 10.1109/TKDE.2005.99.
4. R. Burke, "Hybrid recommender systems: Survey and experiments," User Modeling and User-Adapted Interaction, vol. 12, pp. 331–370, 2002, doi: 10.1023/A:1021240730564.
5. P. Malekpour Alamdari, N. Jafari Navimipour, M. Hosseinzadeh, A. A. Safaei, and A. Darwesh, "A systematic study on the recommender systems in the e-commerce," IEEE Access, vol. 8, pp. 115694–115716, 2020, doi: 10.1109/ACCESS.2020.3002803.
6. E. Ferrara, P. De Meo, G. Fiumara, and R. Baumgartner, "Web data extraction, applications and techniques: A survey," Knowledge-Based Systems, vol. 70, pp. 301–323, 2014, doi: 10.1016/j.knosys.2014.07.007.
7. A. H. F. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira, "A brief survey of web data extraction tools," SIGMOD Record, vol. 31, no. 2, pp. 84–93, 2002, doi: 10.1145/565117.565137.
8. 8. C.-H. Chang, M. Kayed, M. R. Girgis, and K. F. Shaalan, "A survey of web information extraction systems," IEEE Transactions on Knowledge and Data Engineering, vol. 18, no. 10, pp. 1411–1428, 2006, doi: 10.1109/TKDE.2006.152.
9. M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang, "WebTables: Exploring the power of tables on the web," Proceedings of the VLDB Endowment, vol. 1, no. 1, pp. 538–549, 2008, doi: 10.14778/1453856.1453916.
10. 10. S. Podlipnig and L. Böszörményi, "A survey of web cache replacement strategies," ACM Computing Surveys, vol. 35, no. 4, pp. 374–398, 2003, doi: 10.1145/954339.954341.
11. G.-W. Bock, S.-Y. T. Lee, and H. Y. Li, "Price comparison and price dispersion: Products and retailers at different internet maturity stages," International Journal of Electronic Commerce, vol. 11, no. 4, pp. 101–124, 2007, doi: 10.2753/JEC1086-4415110404.
12. T. Akimoto and F. Takeda, "Price movements in the Japanese online home electronics market," Electronic Commerce Research and Applications, vol. 8, no. 1, pp. 28–36, 2009, doi: 10.1016/j.elerap.2008.07.001.
13. H. O. Bodur, N. M. Klein, and N. Arora, "Online price search: Impact of price comparison sites on offline price evaluations," Journal of Retailing, vol. 91, no. 1, pp. 125–139, 2015, doi: 10.1016/j.jretai.2014.09.003.
14. J. Lu, D. Wu, M. Mao, W. Wang, and G. Zhang, "Recommender system application developments: A survey," Decision Support Systems, vol. 74, pp. 12–32, 2015, doi: 10.1016/j.dss.2015.03.008.
15. A. Valencia-Arias, H. Uribe-Bedoya, J. D. González-Ruiz, G. Sánchez Santos, E. Chapoñan Ramírez, and E. Martínez Rojas, "Artificial intelligence and recommender systems in e-

commerce: Trends and research agenda," Intelligent Systems with Applications, vol. 24, article 200435, 2024, doi: 10.1016/j.iswa.2024.200435.
16. C. S. Rajpoot, V. Tiwari, and S. K. Vishwakarma, "Emerging trends of recommender system for e-commerce: A comprehensive review," Discover Computing, vol. 29, article 63, 2026, doi: 10.1007/s10791-026-09923-z.
17. F. Karimova, "A survey of e-commerce recommender systems," European Scientific Journal, vol. 12, no. 34, p. 75, 2016, doi: 10.19044/esj.2016.v12n34p75.
18. Raghav-27, "SHOP SYNC 2.0: E-commerce Product Aggregator," project source-code repository. Available: https://github.com/Raghav-27/SHOP SYNC2.0 (accessed 20 October 2026).
19. Python Software Foundation, "Coroutines and Tasks — asyncio," Python Documentation. Available: https://docs.python.org/3/library/asyncio-task.html (accessed 20 October 2026).
20. FastAPI, "Concurrency and async / await," FastAPI Documentation. Available: https://fastapi.tiangolo.com/async/ (accessed 20 October 2026).
21. Redis, "Caching," Redis Documentation. Available: https://redis.io/docs/latest/develop/use/client-side-caching/ (accessed 20 October 2026).
22. J. Nielsen, "10 Usability Heuristics for User Interface Design," Nielsen NormanGroup. Available: https://www.nngroup.com/articles/ten-usability-heuristics/ (accessed 20 October 2026).