

PriceSense: A Quick-Commerce Price Comparison Platform

Omkar Anand Sharma¹, Divij Nagpal², Akhand Pratap Singh Sengar³, Devansh Shringi⁴

Department of Computer Science and Business Systems
B.M.S. College of Engineering, Bengaluru, India

Abstract The rapid growth of quick-commerce platforms has provided consumers with faster access to groceries, household products, and everyday essentials. However, users often need to search across multiple platforms to determine where a product is available at the lowest price. This manual comparison process becomes inefficient when product names, package sizes, availability, and prices differ across platforms. PriceSense is a web-based quick-commerce price comparison platform developed to consolidate product pricing information from Blinkit, Zepto, Swiggy Instamart, and BigBasket through a unified interface. The system uses a Flask backend and Playwright-based browser automation to interact with dynamically rendered web pages. Product information is processed using product-name normalization, package-size extraction, token-based similarity matching, and sequence-based similarity calculation. The updated system additionally incorporates user authentication and location context. Users can register and log in using an email address and password, while location information can be provided through a delivery address or pincode and browser geolocation. Geographic coordinates can be reverse-geocoded to obtain a delivery postcode, after which the location context is passed to the scraping engine. Browser automation sessions are configured with the corresponding geographic coordinates. To improve data reliability, the system distinguishes directly extracted live prices from estimated and unavailable values. Estimated values are calculated only when at least two legitimate live price observations are available and are not used to determine the winning platform. The resulting comparison is presented through an interactive dashboard with platform-wise prices, summary indicators, and graphical visualization.

Keywords Quick-commerce, price comparison, Playwright, Flask, web automation, location-aware systems, product matching, web scraping, price intelligence

I. INTRODUCTION

The growth of quick-commerce services has changed the way consumers purchase groceries, food products, household items, and other essentials. Platforms such as Blinkit, Zepto, Swiggy Instamart, and BigBasket provide rapid access to products through web and mobile applications.

Although these platforms improve convenience, consumers generally need to search multiple applications when they want to compare prices. A product may have different prices, package descriptions, and availability conditions across platforms. Consequently, manually checking several applications can increase the time required to make a purchase decision.

Another challenge is that modern e-commerce websites rely heavily on dynamic rendering and

JavaScript. Static HTTP-based scraping may therefore fail to obtain information that

becomes available only after the webpage has been rendered in a browser.

PriceSense addresses these challenges through a centralized quick-commerce price comparison platform. The system accepts a product query and uses browser automation to collect available information from multiple platforms. Extracted product information is subsequently normalized and matched before being presented in a unified dashboard.

The updated implementation also introduces user authentication and location context. Location is particularly relevant to quick-commerce because product availability and serviceability can depend on geographical conditions. PriceSense therefore

accepts address or pincode information and can also use browser-provided geographic coordinates.

The main contributions of the system are:

- Development of a unified quick-commerce price comparison platform.
- Browser-based extraction of dynamically rendered product information using Playwright.
- Product normalization and package-size extraction for improved cross-platform matching.
- Similarity-based matching of comparable products across platforms.
- Session-based user authentication with SQLite persistence.
- Location-aware search using address, pincode, and browser geolocation context.
- Reliability-aware classification of prices as Live, Estimated, or Unavailable.
- Interactive visualization of platform-wise price comparisons.

II. LITERATURE REVIEW

Traditional price comparison systems generally depend on structured APIs or web scraping mechanisms to collect product information from multiple sources. APIs provide structured data but may not be available for every commercial platform. Conventional HTML scraping approaches can also become unreliable when websites depend on JavaScript-based rendering. Browser automation frameworks provide an alternative approach by allowing software to operate a real browser environment. Playwright is particularly suitable for this purpose because it supports browser navigation, JavaScript execution, page interaction, browser contexts, geolocation permissions, and automated waiting mechanisms.

Product matching is another important component of price comparison. The same product may be represented differently on different platforms. Differences may include capitalization, punctuation, brand descriptions, package quantities, and promotional text.

Text similarity techniques can therefore be used to determine whether two product descriptions represent comparable products. However, matching solely on product names may incorrectly associate products with different package sizes. PriceSense consequently incorporates package-size extraction into the matching process.

Location is also relevant to quick-commerce systems. Product availability can depend on the user's geographical region and delivery serviceability. A location-aware comparison architecture can therefore provide additional context to the price collection process.

PriceSense combines these concepts into a single browser-based price comparison workflow.

III. PROBLEM DEFINITION

Users searching for products through quick-commerce platforms commonly face the following problems:

1. Prices must be checked individually across multiple platforms.
2. Product names can differ between platforms.
3. Products with different package sizes can have similar names.
4. Modern platforms use dynamically rendered web interfaces.
5. Some platforms may block or restrict automated requests.
6. Product availability can depend on geographical location.
7. Missing price information should not be replaced with fabricated values.

Therefore, the problem addressed by PriceSense is the development of a centralized platform capable of collecting available product prices from multiple quick-commerce services while incorporating product matching, location context, and reliability-aware price handling.

IV. OBJECTIVES

The primary objectives of PriceSense are:

- To develop a centralized platform for comparing quick-commerce product prices.

- To integrate Blinkit, Zepto, Swiggy Instamart, and Big-Basket.
- To use Playwright for browser-based interaction with dynamically rendered websites.
- To normalize product names before comparison.
- To extract and consider package-size information.
- To match comparable products across platforms using similarity measures.
- To provide user registration and login functionality.
- To associate product searches with location context.
- To distinguish live, estimated, and unavailable prices.
- To identify the lowest reliably available live price.
- To visualize price comparisons through an interactive dashboard

V. SYSTEM ARCHITECTURE

The PriceSense architecture consists of several interconnected components. The overall workflow begins with the authenticated user, followed by location acquisition, Flask request processing, browser automation, product extraction, product matching, price processing, and dashboard visualization.

The logical workflow can be represented as:

User → Authentication → Location Context → Flask Backend → Playwright

The system consists of the following major components:

1. User Interface
2. Authentication Module
3. SQLite Database
4. Location Processing Module
5. Flask Backend
6. Playwright Scraping Engine
7. Platform-Specific Scrapers
8. Product Matching Engine
9. Price Estimation Engine
10. Comparison and Visualization Layer

VI. USER AUTHENTICATION AND DATABASE

The updated PriceSense implementation introduces user authentication using Flask sessions and a lightweight SQLite database.

The system maintains a user database containing the following fields:

Table User Database Structure

Field	Description
ID	Unique user identifier
Name	User's full name
Email	User email address
Password Hash	Securely hashed password
Created At	Account creation timestamp

During registration, the password is transformed into a secure hash using Werkzeug security utilities. The original plaintext password is not stored in the database.

After successful authentication, the application maintains the user's authenticated session. Protected routes such as the dashboard and search endpoint require an authenticated user.

The authentication workflow is:

Registration → Password Hashing → SQLite → Login → Session

(2)

This provides persistent user management while keeping the local development architecture lightweight.

VII. LOCATION PROCESSING

Location information is incorporated into the updated system because quick-commerce services can depend on geo-graphical context.

PriceSense supports the following forms of location information:

- Selected city or location.
- User-provided delivery address or pincode.
- Browser geographic coordinates.

The user can provide a delivery address or pincode through the interface. A six-digit Indian pincode can be extracted from the provided address.

The application can also request browser geolocation. When latitude and longitude are available, the backend can use reverse geocoding to obtain a delivery postcode.

The reverse-geocoding process is represented as:

(latitude, longitude) → Reverse Geocoding → Pincode

(3)

The resulting location information is passed to the scraping engine.

It is important to distinguish between providing geographic context to the browser automation environment and explicitly setting a delivery address inside every commercial platform. The current implementation provides geographic context to Playwright but does not guarantee that every supported platform accepts or applies the location through its own delivery-location interface.

VIII TECHNOLOGY STACK

Table II Pricesense Technology Stack

Component	Technology
Programming Language	Python
Backend Framework	Flask
Browser Automation	Playwright
Database	SQLite
Authentication	Flask Session and Werkzeug Security
Frontend	HTML, CSS, JavaScript
Visualization	Chart.js
Reverse Geocoding	OpenStreetMap Nominatim
Concurrency	Python ThreadPoolExecutor
Platforms	Blinkit, Zepto, Instamart, BigBasket

IX. DATA COLLECTION AND WEB AUTOMATION

The scraping engine uses Playwright to launch independent browser contexts for the supported platforms.

Each browser context can be configured with:

- User-agent information.
- Indian locale settings.
- Asia/Kolkata timezone.
- Browser viewport dimensions.

- Geographic coordinates.
- Geolocation permissions.

The browser automation process follows these stages:

1. Launch a Chromium browser.
2. Create a browser context.
3. Configure browser and location information.
4. Open the platform search page.
5. Wait for dynamically rendered content.
6. Detect potential blocking conditions.
7. Scroll the page.
8. Locate product cards.
9. Extract product names and prices.
10. Return structured product records.

The system checks for indicators such as access denial, CAPTCHA-related messages, unusual traffic messages, and automated-request restrictions.

When a platform cannot provide reliable information, the platform is represented as unavailable rather than generating a synthetic value.

Multi-Platform Price Collection

PriceSense currently targets four quick-commerce platforms:

TABLE III supported Quick-Commerce Platforms

Platform	Collection Approach
Blinkit	Playwright browser navigation and DOM extraction
Zepto	Search URL attempts and DOM extraction
Swiggy Instamart	Playwright navigation and product-card extraction
BigBasket	Playwright navigation and product-card extraction

Each platform has an independent scraping function. This separation allows platform-specific selectors and handling mechanisms to be modified independently.

The implementation uses Python's Thread Pool Executor to execute platform scrapers concurrently.

The concurrent architecture can be represented as:

$$\text{Search Query} \rightarrow \left\{ \begin{array}{l} \text{Blinkit} \\ \text{Zepto} \\ \text{Instamart} \\ \text{BigBasket} \end{array} \right. \rightarrow \text{Parallel Extraction (4)}$$

This architecture allows the system to collect available re-sults from multiple platforms without requiring each platform to complete sequentially.

Product Normalization

Product descriptions retrieved from different platforms may contain different capitalization, punctuation, pricing informa-tion, or promotional descriptions.

PriceSense applies a normalization process that:

1. Converts text to lowercase.
2. Removes currency expressions.
3. Removes non-alphanumeric characters.
4. Normalizes whitespace.

The normalized representation is then used for product comparison.

This reduces the effect of superficial textual differences between platform listings.

Package-Size Extraction And Matching

Products with identical or similar names may represent different quantities. PriceSense therefore extracts package in-formation such as:

- grams (g)
- kilograms (kg)
- millilitres (ml)
- litres (l)
- milligrams (mg)

A product with a different explicitly detected package size is not automatically considered equivalent to another product.

For example:

Chocolate 20g $\equiv$ Chocolate 100g (5) when both package sizes are explicitly available. This reduces false matches between products with similar names but different quantities.

Similarity-Based Product Matching

PriceSense combines sequence similarity and token overlap to determine whether products from different platforms are comparable.

Let A and B represent two normalized product names. The sequence similarity is calculated using a string similar-

ity approach. Token overlap is calculated as:

$$T(A, B) = \frac{|Tokens(A) \cap Tokens(B)|}{|Tokens(A) \cup Tokens(B)|}$$

The combined matching score is represented as:

$$S(A, B) = 0.55S_{seq} + 0.45T(A, B)$$

where:

- Sseq represents sequence similarity.
- T(A, B) represents token overlap.

A candidate product is considered for matching when its similarity score satisfies the implemented matching threshold. This approach allows the system to tolerate minor dif-ferences in product descriptions while reducing mismatches caused by package-size differences.

Price Processing And Reliabil

PriceSense classifies collected price values into three cate-gories.

A. Live Price

A live price is directly extracted from available platform content.

B. Estimated Price

When a product has at least two legitimate live price observations, missing platform values can be estimated using their arithmetic mean.

For n live observations:

$$P_{avg} = \frac{1}{n} \sum_{i=1}^n P_i$$

where:

$$n \geq 2$$

Estimated values are clearly labelled in the dashboard.

C. Unavailable Price

If a reliable price cannot be obtained, the corresponding platform is represented as unavailable.

This can occur when a website blocks automated access, when a product cannot be located, or when the extracted information does not satisfy the system's validation conditions.

The system therefore follows:

Missing Data → Unavailable (10)
unless sufficient legitimate live observations exist for estimation.

The system does not generate random or arbitrary prices for missing platform data.

Price Comparison And Winner Determination

The lowest-price platform is selected using only live price values.

Let the set of live prices be:

$$L = \{P_1, P_2, \dots, P_n\}$$

The minimum live price is:

$$P_{min} = \min(L)$$

The platform associated with the minimum live price is selected as the winner.

Estimated values are excluded from winner determination. If two or more platforms have the same lowest live price, the result can be represented as a tie.

This prevents an estimated value from incorrectly determining which platform is presented as the cheapest.

Dashboard And Visualization

The PriceSense dashboard provides a centralized interface for product search and price comparison. The interface contains:

- Product search field.
- Location selector.
- Delivery address or pincode input.
- Quick-search product buttons.
- Platform-wise price table.
- Best-value indicator.
- Minimum live price.
- Market average.
- Potential savings indicator.
- Interactive price comparison chart.

Chart.js is used to visualize platform-wise price information. The dashboard distinguishes live and estimated values using separate status indicators, while unavailable values are displayed explicitly.

Complete System Workflow

The current implementation has several limitations:

- Quick-commerce websites can change their DOM structure and selectors.
- Some platforms may restrict automated browser requests.
- Platform availability can vary according to geographic region.
- Browser geolocation does not guarantee that a commercial platform will apply the location to its delivery-service configuration.
- The current implementation does not guarantee explicit address or pincode entry into every platform's own delivery-location interface.
- Product matching may require additional rules for highly ambiguous product names.
- Estimated prices are statistical references and do not represent direct platform observations.
- The current implementation uses SQLite for local user persistence.

Future Scope

Several improvements can be incorporated into future versions of PriceSense.

A. Platform-Specific Location Automation

The scraping engine can be extended to explicitly set the user's delivery pincode or address inside each supported platform before performing product searches. This would provide stronger location-specific price and availability comparisons.

User → Authentication → Location → Query → Parallel Scraping → Price Analysis → Visualization

B. Machine Learning-Based Prediction

Historical observations can be used to train machine learning models for price prediction, anomaly detection, and demand-related analysis.

C. Personalized Recommendations

Future versions can use user preferences and historical searches to recommend products or platforms according to price sensitivity and shopping behavior.

D. Expanded Platform Coverage

Additional quick-commerce and e-commerce services can be incorporated through independent platform adapters.

E. Scalable Database Architecture

The SQLite implementation can be replaced with Post-greSQL or another production-grade database for multi-user deployment and larger historical datasets.

X. CONCLUSION

PriceSense provides a centralized approach to quick-commerce price comparison by integrating product search, browser automation, product matching, location context, and price analysis into a unified platform.

The current implementation uses Flask as the backend and Playwright as the browser automation framework for collecting product information from Blinkit, Zepto, Swiggy Instamart, and BigBasket. Product normalization, package-size extraction, and similarity-based matching help identify comparable products across platforms.

The updated system additionally introduces user authentication, SQLite-based persistence, delivery address and pincode input, and browser geolocation processing. These features provide a foundation for developing a more location-aware price comparison system.

The reliability-aware pricing mechanism is an important component of the platform. Directly collected values are marked as live, estimated values are generated only when sufficient legitimate observations are available, and unavailable values are not replaced with fabricated prices. Estimated values are also excluded from winner determination.

The current implementation establishes a practical foundation for future development involving platform-specific delivery-location automation, historical price tracking, machine learning-based prediction, personalized recommendations, and scalable deployment.

REFERENCES

1. Microsoft, "Playwright Documentation," Microsoft. [Online]. Available: <https://playwright.dev/>
2. Pallets Projects, "Flask Documentation." [Online]. Available: [https:// flask.palletsprojects.com/](https://flask.palletsprojects.com/)
3. Python Software Foundation, "Python Documentation." [Online]. Avail-able: <https://docs.python.org/3/>
4. Chart.js, "Chart.js Documentation." [Online]. Available: <https://www.chartjs.org/docs/>
5. SQLite, "SQLite Documentation." [Online]. Available: <https://www.sqlite.org/docs.html>
6. OpenStreetMap Foundation, "Nominatim Documentation." [Online].
7. Available: <https://nominatim.org/release-docs/latest/>
8. Pallets Projects, "Werkzeug Documentation." [Online]. Available: <https://werkzeug.palletsprojects.com/>
9. //werkzeug.palletsprojects.com/