Habib University

Dhanani School of Science and Engineering

CS/CE 353/374-L2 Software Engineering

Milestone: Hercules (Software Architecture Document)

SHOQ Mobiles

Group Members

Hammad Sajid (hs07606)

Sarim Tahir (st07112)

Muhammad Owais Waheed (ow07611)

Muhammad Khubaib Mukaddam (mk07218)

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to provide a comprehensive overview of the software architecture of the Price Comparison E-commerce Application. This document will describe the various components of the system, their interactions, and the design decisions that have been made to ensure that the system meets its functional and non-functional requirements.

1.2 Scope

The Price Comparison E-commerce Application (SHOQ Mobiles) aims to allow users to compare prices from various online listings including Daraz, OLX, and Telemart using web scraping techniques utilizing Python's Beautiful Soup library and find the best deals for the user. The application will consist of a mobile app that will be developed using the Flutter framework, and a backend server that will be developed using Node.js and MongoDB. The system will also include a web scraping module that will be responsible for extracting product information and prices from various e-commerce websites.

1.3 Terms and Definitions

- **Stakeholder**: A person, group, or organization that has an interest or concern in an organization.
- **Flutter**: Google's UI toolkit for building natively compiled applications for mobile, web, and desktop from a single codebase.
- **Node.js**: A JavaScript runtime built on Chrome's V8 JavaScript engine.
- **MongoDB**: A NoSQL database program.
- **Web Scraping**: Extracting data from websites.
- **Beautiful Soup**: A Python library for pulling data out of HTML and XML files.
- **E-commerce**: Commercial transactions conducted electronically on the Internet.
- **API**: Application Programming Interface.

1.4 References

- Flutter Documentation
- Node.js Documentation
- MongoDB Documentation
- Beautiful Soup Documentation

2 Architectural Overview

The architecture of the Price Comparison E-commerce Application is designed to be modular, scalable, and secure. It consists of two main components: the Backend Server and the Frontend Mobile App.
The architecture consists of three main components: Flutter application, a Node.js server, and a MongoDB database. The Flutter application will be responsible for providing a user interface for the application, while the Node.js server will be responsible for handling user requests, fetching data from the database, and performing web scraping. The MongoDB database will be responsible for storing user data, product information, and price data.

3 Architectural Goals and Constraints

1. **Data Aggregation and Comparison:** The system should efficiently aggregate and compare prices from various online listings and sources to provide users with accurate and up-to-date information on product prices and deals.

2. **User Experience:** The application should offer a seamless and intuitive user experience across both web and mobile platforms, with responsive design, intuitive navigation, and fast loading times to enhance user satisfaction and engagement.

3. **Scalability:** The architecture should be scalable to handle a growing number of users, products, and data sources without compromising performance or reliability, ensuring that the application remains responsive and available during peak usage periods.

4. **Reliability and Availability:** The system should be highly reliable and available, minimizing downtime and ensuring consistent access to price comparison features for users.

5. **Scraping Limitations:** The availability and reliability of fetching price data and product information from third party sources may impose constraints on system design and integration, requiring error handling mechanisms and fallback options.

6. **Maintainability:** The system should be easy to maintain and update, with clear separation of concerns and well-documented code to facilitate future development and enhancements.

7. **Performance:** The system should be performant, with fast response times and minimal latency to ensure a smooth and responsive user experience.

4 Use-Case View

Use Case	Description
Search for products	Users can search for products using the search bar in the mobile app.
View product listings	Users can view product listings with details such as product name, image, price, and source.
Compare prices and deals	Users can compare prices and deals from various sources to find the best deal.
Redirect to source	Users can click on a product listing to be redirected to the source website to view more details and make a purchase.

4.1 Architecturally-Significant Use Cases

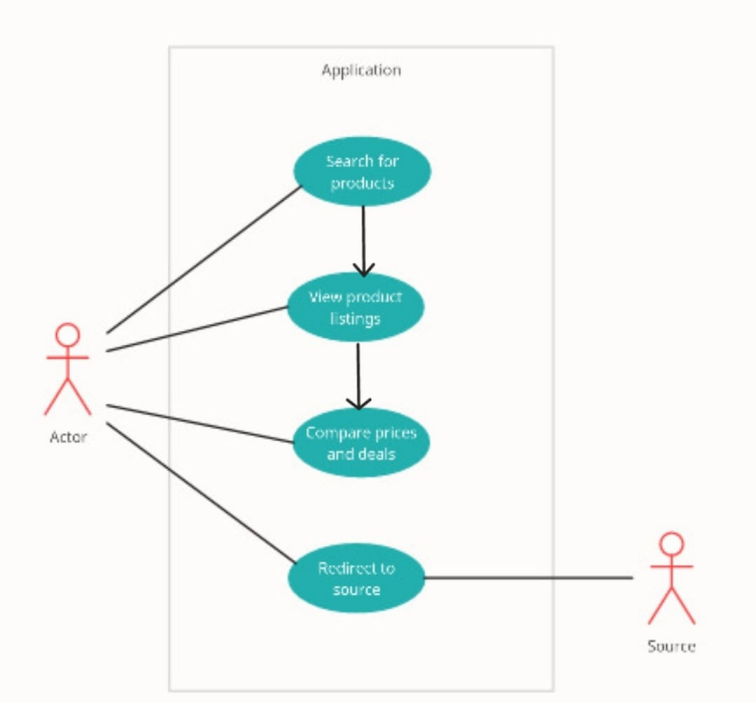

Figure 1: Use Case Diagram

- **Search for products:** Allows users to search for specific products based on keywords or filters.
- **View product listings:** Displays a list of products matching the user's search criteria, providing details such as prices, descriptions, and seller

information.

- **Compare prices and deals:** Enables users to compare prices and deals for different products, helping them make informed purchasing decisions.
- **Redirect to source:** Redirects users to the original source or seller's website for purchasing the selected product.

5 Logical View

5.1 Overview

The logical view of the system describes the high-level architecture and design of the application, focusing on the logical components, their interactions, and the flow of data and control within the system.

5.2 Logical Diagram

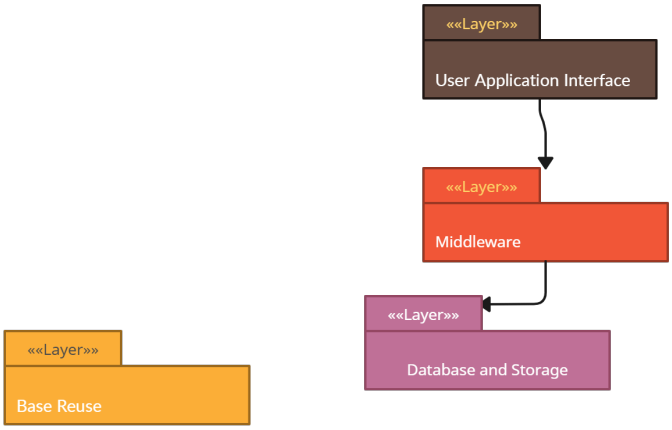

Figure 2: Class Diagram

5.2.1 User Application Interface

The user application interface is responsible for providing a user-friendly and intuitive interface for users to interact with the application, search for products, view listings, compare prices, and make purchases. It consists of various components such as search bar, product listings, price comparison results, and redirection links. The user interface is designed to be responsive, visually appealing, and easy to navigate, ensuring a seamless user experience across web and mobile platforms.

5.2.2 Middleware and Backend Server

The middleware and backend server components are responsible for processing user requests, handling business logic, and communicating with the database to fetch and store data. The backend server is designed to be scalable, reliable, and performant, ensuring that it can handle a large number of concurrent user requests and provide fast response times. It also includes a web scraping module that fetches product information and pricing data from various e-commerce websites to provide accurate and up-to-date results to users. The middleware layer provides an abstraction between the user interface and the backend server, handling user requests, data processing, and response generation.

5.2.3 Database and Data Storage

The database and data storage components are responsible for storing user data, product information, pricing data, and other relevant details required

for price comparison and response generation. The database is designed to be scalable, reliable, and secure, ensuring that it can handle a large amount of data and provide fast access times. It also includes data caching and indexing mechanisms to optimize data retrieval and storage operations. The data storage layer provides an abstraction between the backend server and the database, handling data access, storage, and retrieval operations.

5.2.4 Base Reuse

The base reuse component is responsible for providing reusable components, libraries, and modules that can be shared across different parts of the application to promote code reusability, maintainability, and consistency. It includes common utilities, helper functions, and shared components that can be used by the user interface, middleware, backend server, and other parts of the application to reduce duplication and improve development efficiency.

6 Process View

6.1 Overview

The process view of the system describes the dynamic aspects of the application, focusing on the runtime behavior, process interactions, and the flow of control and data between different components and processes.

6.2 Process Diagram

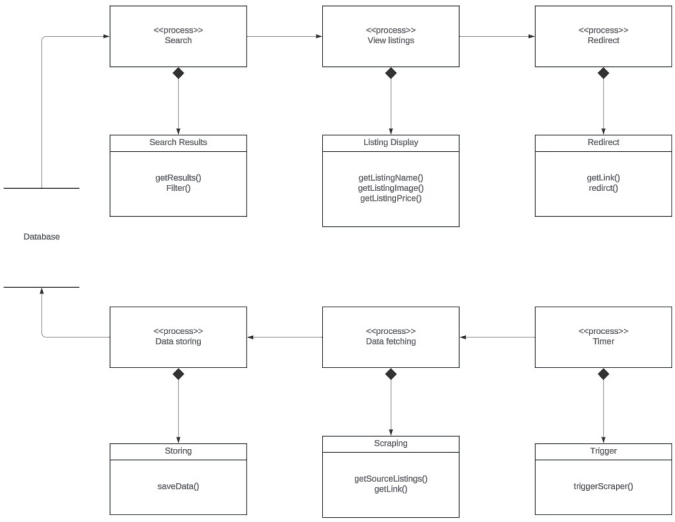

Figure 3: Process Diagram

- **Search Proces:** When a user initiates a search for a product through the user interface (UI), the application processes the search query and retrieves matching results from the database.
- **View Listings Process:** The application presents the retrieved product listings to the user through a user-friendly interface (UI), displaying relevant details such as product images, prices, and seller information for easy comparison and evaluation.
- **Redirect Process:** Upon selecting a specific listing, the user is seamlessly redirected to the seller's website or online marketplace where the product is listed, facilitating further exploration, purchase, and transaction completion.
- **Data Storing Process:** Once the data is fetched, the scraped listing data is organized and stored in the application's database, ensuring efficient retrieval and management of product information for future use.
- **Data Fetching Process:** The scraper module collects data from relevant sources i.e. online retailers' websites, by systematically scanning and extracting product details such as prices, descriptions, and availability status.
- **Timer Process:** When the predefined update time is reached, the scraper is initiated on the backend to retrieve the latest product information from various online sources.

6.3 Process Model to Design Model Dependencies

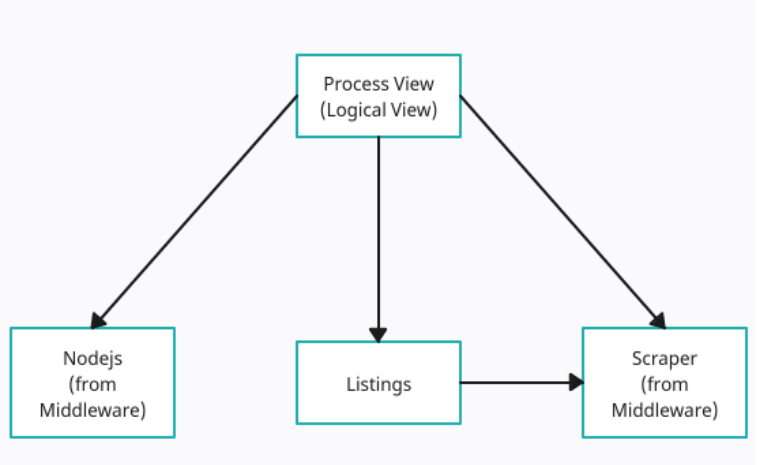

Figure 4: Process Model to Design Model Dependencies

7 Deployment View

7.1 Overview

The deployment view of the system describes the physical deployment of the application, focusing on the distribution of components, their interactions, and the infrastructure required to support the application. It includes details about the hardware, software, and network configurations necessary to run the application.The Deployment View shows the system's architecture from a higher level, focusing on how components are distributed across different nodes and how they interact with each other.

7.2 Deployment Diagram

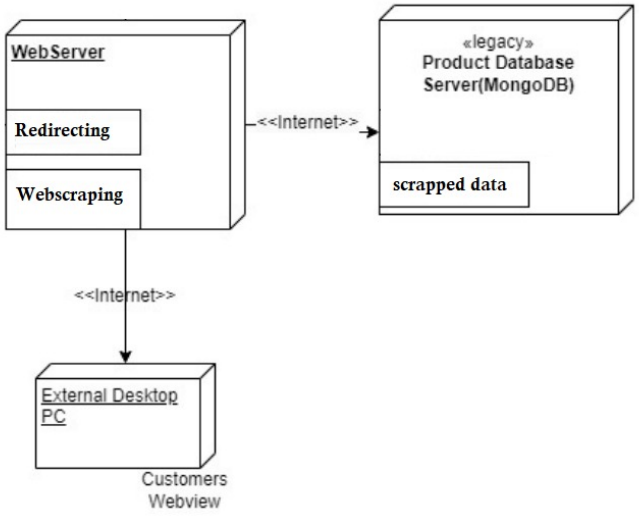

Figure 5: Deployment Diagram

7.3 External Desktop PC

The customers will be able to access the online platform using their external desktop PCs that would be connected to the internet. They will search for the product, and would be provided all the links relevant to their filters.

7.4 Web Server

Our platform's web server will host the main application. It would be handling requests from external desktop PCs, processing actions such as search requests, web scraping, and data presentation.

7.5 Product Database Server

The database server will be storing all the scrapped data that the platform obtains based on the search by the user and product information. It will be ensuring data integrity, and quick and efficient data support.

8 Size and Performance

8.1 System Size

We will initially deploy our platform on a localhost, only for a limited amount of users so that we can test it.

8.1.1 Data Storage

The system will be storing a significant amount of scrapped data in the local database, and even this will increase as we scale further.

8.1.2 Codebase Size

The codebase is initially tailored to encompass core functionalities to ensure and efficient system. However, as the testing phase evolves and user insights are gathered, the codebase will hopefully undergo expansion, embracing new functionalities and enhancements, all crafted to elevate the platform's user experience.

8.2 Performance Considerations

This part talks about how to ensure a better effeciency and how to make sure our performance would be optimal.

8.2.1 Response Times

Our primary aim with this platform is to reduce user search time and provide them with quick results for their need. For this we need to achieve quick response times in our initial testing phase, as a better response time leads to better performance.

8.2.2 Scalability

The system is designed to be scalable as we progress ahead. We will be looking into scaling after our initial deployment on localhost, so that we can have more users on the platform and a greater range of products.

8.2.3 System Maintenance and Optimization

We aim to prioritize the reliability and availability of the system, hoping to achieve uninterrupted service even during periods of high user traffic. Measures will be taken such as fault-tolerant architectures, efficient monitoring tools, and real-time performance monitoring, to ensure optimal system operations and efficiency.

8.3 Testing and Validation

In the testing phase will be strictly performing our testing to ensure optimal performance. Different testing scenarios will be designed.

9 Conclusion

The software architecture of the Price Comparison E-commerce Application is designed to be modular, scalable, and secure, with a focus on data aggregation and comparison, user experience, reliability, and availability. The system consists of two main components: the Frontend Mobile App and the Backend Server, which interact to provide users with accurate and up-to-date information on product prices and deals. The architecture is designed to be scalable, reliable, and performant, ensuring that the application remains responsive and available during peak usage periods. The logical, process, and deployment views of the system provide a comprehensive overview of the application's architecture, design, and deployment, highlighting the various components, their interactions, and the infrastructure required to support the application. The system is designed to be easy to maintain and update, with clear separation of concerns and well-documented code to facilitate future development and enhancements. The performance considerations and testing and validation strategies will ensure that the system meets its functional and non-functional requirements, providing users with a seamless and intuitive price comparison experience across web and mobile platforms.