Software Design

Document

Group 2

CompareCart

Software Engineering
Fall 2024

CS673 A1

Siming Zhao
Zichen Wang
Yifei Wang
Jingwei Ma
Sifat Singh Khalsa

1. Introduction

1.1 Purpose

The purpose of this Software Design Document is to describe the structure and design decisions made for the CompareCart project. This document will detail the architecture, data flow, major components, and interface designs to provide guidance for the development team.

1.2 Scope

This document covers the design of the CompareCart application, a web-based platform that aggregates product prices, reviews, and details from multiple e-commerce stores. Users can search, compare, and analyze products to make informed purchasing decisions.

---

2. System Architecture

2.1 Architectural Overview

The CompareCart system is designed as a web application using a **client-server architecture** with a **RESTful API**. The system consists of three primary layers:

- **Presentation Layer** (Frontend): Handles user interactions and displays data to users.
- **Application Layer** (Backend): Processes business logic and data aggregation.
- **Data Layer**: Manages data storage and retrieval.

2.2 System Components

1. **Frontend**: Built with React.js (or a similar framework) to render the user interface and make API calls to the backend.
2. **Backend**: Developed using Node.js with Express.js to handle requests, process data, and communicate with external APIs or scraping modules.
3. **Database**: MongoDB is used for data storage, managing product details, user information, and price alerts.
4. **Web Scraping and API Integration Module**: This module uses APIs or scraping techniques to gather product information from multiple online stores.

2.3 Data Flow Diagram

1. **User** enters a search query on the frontend.
2. **Frontend** sends a search request to the backend server.
3. **Backend** retrieves or updates product data by making calls to the **Web Scraping and API Integration Module**.
4. **Data** is stored or updated in the **Database**.
5. **Backend** sends product information back to the **Frontend** for display.

2.4 Backend Functional Workflow

- **Handling Frontend Requests:** The backend handles incoming requests by interpreting and processing parameters sent from the frontend. The App.py file acts as the primary API endpoint, routing requests and formatting responses.
- **Database Query and Real-time Search:** Based on parameters in the frontend requests, the backend queries the database using FirestoreClient. If no relevant results are found, the system initiates a real-time search using webSpider.py, ensuring up-to-date data is fetched and processed.
- **Sending Results to the Frontend:** After gathering and processing data, the backend organizes and sorts search results, returning them to the frontend in a structured JSON format.

2.5 Class Diagrams and Object Model

Class Diagrams:

- **Modelpack::webSpider**
  - Handles data extraction with core functions like getKeyword and getUrl.
- **Modelpack::objectInfo**
  - Stores structured data, with attributes for name (objectName), rating (objectStar), type (objectType), price (objectPrice), and comment count (objectCommentNum).
- **Modelpack::FirestoreClient**
  - Encapsulates Firebase database interactions, providing CRUD functionalities.

---

3. Data Design

3.1 Database Schema

The database structure is designed to store product information, user data, and transaction logs.

Database Tables/Collections

1. **Products Collection**
   - `product_id` (String): Unique identifier.
   - `name` (String): Product name.
   - `category` (String): Product category (e.g., electronics).
   - `price_data` (Array of Objects): Price details per store.
     - `store_name` (String): Store name.
     - `price` (Float): Product price.
     - `last_updated` (Date): Last updated timestamp.
   - `reviews` (Array): Aggregated review data.
2. **Users Collection**
   - `user_id` (String): Unique identifier.

    - `email` (String): User's email address.
    - `password` (String): Hashed password.
    - `alerts` (Array): Price alerts set by the user.
3. **Price Alerts Collection**
    - `alert_id` (String): Unique identifier.
    - `user_id` (String): Links to `Users`.
    - `product_id` (String): Links to `Products`.
    - `target_price` (Float): Price threshold for alert.
4. **Transactions Log (Optional)**
    - `transaction_id` (String): Unique identifier.
    - `user_id` (String): Links to `Users`.
    - `action` (String): Description of the action (e.g., search, alert).

3.2 Data Storage and Retrieval

- **Storage**: MongoDB for flexible and scalable document-based storage.
- **Caching**: Redis (optional) for caching popular searches or frequently accessed product data.

3.3 Real-time Search and Data Aggregation

- **Real-time Search on Query Failure:** If a database query fails to retrieve relevant results, **webSpider.py** performs data extraction from specified sources. This data is then processed and temporarily stored for immediate use, allowing the system to handle unforeseen data gaps effectively.

---

4. Component Design

4.1 Frontend Components

1. **Search Component**: Handles user input and sends search requests to the backend.
2. **Product Listing Component**: Displays search results with sorting and filtering capabilities.
3. **Comparison Table Component**: Allows users to compare selected products side-by-side.
4. **Product Details Component**: Shows detailed product information, including price history and reviews.
5. **User Profile Component**: Displays user account information, alerts, and search history.

4.2 Backend Components

1. **API Controller**: Receives and handles requests from the frontend, directing them to the correct service.
2. **Product Service**: Manages product data, handling search and comparison functions.

3. **User Service**: Manages user accounts, price alerts, and preferences.
4. **Data Aggregation Module**: Integrates with e-commerce APIs or scrapers to fetch product data.
5. **Alert Service**: Manages and triggers user-defined alerts when target prices are reached.
6. **webSpider.py:** Manages data extraction from external web pages using query parameters provided by the frontend. This module performs on-demand data gathering when database queries return no results, and it collects product details such as price, ratings, and reviews.
7. **App.py:** Acts as the primary controller, structuring extracted and database information into JSON for frontend consumption.
8. **FirestoreClient:** Handles database operations, including data retrieval, conditional queries, adding new documents, and managing indices.

4.3 Web Scraping and API Integration

- **Scraping**: A scraping engine (e.g., Puppeteer or BeautifulSoup) pulls data from sites that don't provide APIs.
- **APIs**: Use partner APIs when available for accurate and efficient data retrieval.

---

5. Front-end Architecture Design

5.1 Core Tech Stack

1. React: A JavaScript library for building user interfaces, focusing on component-based architecture and state management.
2. React Router: Facilitates single-page navigation and routing between different pages.
3. Axios: Handles communication with the backend API for data fetching and updating.
4. CSS Modules: Each component has a dedicated CSS file, reducing the risk of global style conflicts.
5. GitHub Personal Access Token: Used for version control to secure and maintain code reliability.

5.2 Key Components

1. **Navbar**: Displays navigation links (Home, Comparison, Contact Us) and provides a unified navigation experience across all pages.
2. **SearchBar**: Receives user input for searches and can be embedded on multiple pages as a reusable component.
3. **HomePage**: The main landing page that introduces core features and contains a search bar.
4. **SearchPage**: Displays search results based on user queries, with sorting and filtering options.
5. **ComparisonPage**: Allows users to view detailed comparisons of selected products.

5.3 Core Files and Functions

1. **services/api.js**: Contains all API interaction logic, using Axios to simplify data fetching within components.
2. **App.js**: Wraps global components and routing, serving as the main application component.
3. **routes.js**: Defines all routes, ensuring straightforward navigation and modularity.
4. **CSS Files**: Each component has a separate CSS file for encapsulated styling, preventing cross-component style conflicts.

5.3 State Management and Communication

1. **Data Passing Between Pages**: Uses React Router to pass search queries through the URL, which are then extracted by the SearchPage for displaying relevant results.
2. **Component State Management**: useState and useEffect manage component state, such as search input and API results.
3. **Responsive Design**: CSS Flexbox and Grid ensure layouts are responsive, with media queries adjusting layouts for various screen sizes.

5.4 Core Files and Classes

- **webSpider.py:**
  - currentDir(): Gets the current directory for managing configurations.
  - getKeyword(): Extracts keywords from frontend queries.
  - getUrl() and getInfo(): Collects URLs and gathers data into objectInfo.
- **App.py:** Consolidates data retrieved from FirestoreClient and real-time search results from webSpider.py, packaging it into a structured JSON format.
- **FirestoreClient:**
  - CRUD functions: Manages interactions with the database, including document retrieval (get_document), conditional querying (get_document_condition), and document management (add_document, delete_document).

6. Interface Design

6.1 Frontend-Backend API Design

API endpoints are defined for frontend-backend interactions, following RESTful principles.

1. **GET /api/search?query=**
   - **Description**: Searches for products based on a query.
   - **Request Parameters**: `query` (String), `filters` (Optional).
   - **Response**: List of matching products.
2. **GET /api/product/**
   - **Description**: Retrieves detailed information for a specific product.
   - **Response**: Product details, including prices, reviews, and specifications.
3. **POST /api/user/register**
   - **Description**: Registers a new user.
   - **Request Body**: User details (email, password).

    - **Response**: Registration confirmation.
4. **POST /api/alert**
    - **Description**: Sets a price alert for a product.
    - **Request Body**: Product ID, target price.
    - **Response**: Alert creation confirmation.

6.2 External Interfaces

1. **E-commerce APIs**: External APIs for product information retrieval (e.g., Amazon, Walmart).
2. **Database**: MongoDB for CRUD operations.

---

7. Security Design

7.1 Authentication and Authorization

- **User Authentication**: Use JSON Web Tokens (JWT) for secure user sessions.
- **User Roles**: Basic users and admin roles with different levels of access.

7.2 Data Protection

- **Password Encryption**: Use bcrypt for hashing passwords.
- **HTTPS**: Enforce HTTPS for all data transfers.

7.3 Rate Limiting

- **API Requests**: Implement rate limiting to avoid overloading and abuse.

---

8. Error Handling

8.1 Frontend Error Handling

- **User Notifications**: Friendly error messages for network or application issues.
- **Form Validations**: Real-time validation for user input.

8.2 Backend Error Handling

- **Logging**: Log errors with levels (INFO, WARN, ERROR) for efficient debugging.
- **Retries**: Automatic retry for failed scraping attempts.
- **Alerts**: Notify administrators if critical errors occur.