Cross-platform nested API based search engine powered by Crow C++ Framework

A CS3307 Group Project by Group 8

Maxwell Ding (jding263)
Yulun Feng (yfeng445)
Truman Huang (yhuan939)
Matthew Owen Tjhie (mtjhie)
Huiliang Xia (hxia47)

Table of Content

1. Project Summary

1.1 Overview

HOP is an innovative web application designed to transform the online shopping experience. It goes beyond traditional e-commerce platforms by allowing users to find products without needing to precisely define their search terms. Incorporating with natural language processing, HOP enables effortless product searches beyond standard text queries. It is supported by a robust MySQL database for handling large data volumes efficiently. A key feature is its keyword tagging system, powered by the OpenAI API, which tags products with relevant keywords for more intuitive and accurate search results. Also, to ensure users' privacy, all searching history would be kept locally. Overall, HOP aims to provide a seamless, fast, safe, and enjoyable shopping experience where technology intuitively understands user needs.

1.2 Features:

- Providing a user-friendly landing page, results page, and dashboard.
- Accepting user input in natural language, including both text and audio.
- Storing user input as audio file on server side for voice recognition.
- Generating product information (including name, tags, description) from an input image URL using OpenAI.
- Offering a capable dashboard for database management.
- Processing user queries using a natural language processing model to extract keywords.
- Performing thoughtful MySQL product searches when integrated with OpenAI.
- Allowing users to track their sessions and search history by storing information in local storage.
- Providing related product suggestions for a better product impression.
- Avoiding XSS and database injection using filter and sql prepared statement
- Enhancing server security with multiple techniques, including Layer 3, Layer 4 and Layer 7 protection, UFW (Uncomplicated Firewall), Cloudflare Bot Management, and Rate Limiting.

2. Key Accomplishments

2.1 Server Setup

- Establishing the server at the beginning of the project was a crucial decision. It allowed for a unified development environment despite the varying operating systems used by team members. This setup facilitated remote work by simple operations like pushing and pulling edits to the server, significantly reducing the need for complex local environment configurations. We are using following stacks to setup our server:

Domain Registrar: Spaceship.com

DNS (Domain Name System): Cloudflare.com

CDN (Content Delivery Network): Cloudflare.com

Hosting service: Zap-hosting

Hosting location: FFM / Eygelshoven, German

Operating system: Debian GNU/Linux 11 (bullseye)

Systems architecture: x86 (amd64)

Layer 3 protection: 12 Tbsp network capacity anti-DDoS on-demand protection provided by Path.net network

Layer 4 protection: UFW (Uncomplicated Firewall) running on Linux system that only allow live server port going through with rate limiting of 50 connections per ip per second

Layer 7 protection: Cloudflare Bot Management and rate limiting for anti-CC attack. Zero trust access for dashboard login.

2.2 Web Framework Utilization

- Choosing a web framework over console-based debugging proved exceptionally beneficial. It streamlined the testing and debugging processes and improved the project's overall efficiency.

  CrowCpp, also known simply as Crow, is a C++ microframework for web development, particularly for creating HTTP and Websocket web services. It's characterized by its ease of use and fast performance. Crow uses a routing system similar to Python's Flask, which contributes to its user-friendly nature. This similarity to Flask makes it an accessible choice for developers familiar with Python, allowing them to leverage their existing knowledge in a C++ environment.

2.3 Subdomain Strategy

- Creating subdomains for testing and implementing different functions was ingenious. It enabled team members to update their code independently without disrupting others' work, thus enhancing collaborative efficiency.

  It is beneficial to each group member because it saves time to setup. By using Argo Tunnel service provided by Cloudflare, avoid IP leaking, have different port map to different subdomain, so each member can run the product simultaneously.

2.4 Database Integration

- Successfully integrating the MySQL database with C++ programming, especially using the official MySQL connector package, was a significant accomplishment. This integration allowed the team to execute MySQL queries within C++ code, a critical backend system component.

2.5 Data Management Approach

- The decision to convert item information into JSON format and then store it as a long string in the database was a clever solution to data management challenges. This approach streamlined data transfer between the website and the database.

2.6 OpenAI API Implementation

- Overcoming challenges in connecting with the OpenAI API and subsequently utilizing it to generate item descriptions in a specific format was a notable achievement. Implementing OpenAI-related functions facilitated efficient data handling and enhanced the project's basic functionality. API is also used in the implementation of OpenAI Vision, which uses the same API to generate item description from uploaded images.

2.7 Efficient Search System

- Developing an effective search system, aided by the structured item and tags tables in the database, was a significant success. This system, including name and tag matching and a user-friendly search history feature, greatly enhanced the user experience.

2.8 User Safety Measures

- The decision to store search history records in a local JSON file, which clears upon closing the webpage, demonstrated a solid commitment to user safety and privacy.

2.9 Voice Input

- Capture and store the voice input from users by accessing the microphone using The MediaStream Recording API. The voice input will be processed into a blob object, then sent to the server, and stored on the server as a MP3 file, then the server will send a POST request to Replicate. After the audio file is being processed by the OpenAI Whisper API, the server will receive the transcription of what the user said.

3. Key Problem Areas

3.1 Overcoming Development Environment Incompatibilities

- Our group initially encountered a significant technical challenge in establishing a consistent development environment. We initially chose Drogon, a C++ web application library, but it presented compatibility issues. Specifically, two of our team members faced difficulties installing the necessary dependencies, and we discovered that the code wasn't cross-compatible, particularly with macOS systems. This incompatibility hindered cooperation among our team members. After a week of grappling with these issues, we opted for a more effective solution: renting a remote server to create a unified working environment for the entire team.

3.2 Optimizing Database Selection and Data Storage Format

- The next challenge involved the storage of item information in our database, a critical component for our online shopping system. It was essential to have a database capable of efficiently handling queries across numerous records. Our choice, MarineDB, struck an optimal balance in terms of query efficiency, ease of operation, and hardware requirements. With the database selected, we then faced the challenge of determining the best format for data storage. While our existing functions allowed us to run MySQL commands in C++ code and we had decided on the attributes for each item entity, the specific format for data storage remained an issue. Initially, we tried storing all data as strings, but this approach proved inefficient as it required constant conversion between strings and other variable types for data utilization. To overcome this, we turned to JSON for three key reasons: its compatibility with our HTML interface, the presence of a built-in JSON-like type called crow json in the Crow C++ library we were using, and its ability to expedite the search and grouping processes.

3.3 Navigating OpenAI API Integration Challenges

- Furthermore, we encountered a couple of significant challenges during the API set up. Firstly, integrating the OpenAI API was complex, as it required the data to be in a JSON file format. This meant we had to carefully craft and format the prompts to effectively communicate with the GPT models and receive the appropriate responses. This was vital for the extraction function to work correctly with the API.

3.4 Addressing Database Insertion Issues

- Another major hurdle was encountered when inserting product information into your database. We faced issues with the 'stmt' which is a mysql-cpp statement variable. It wasn't working as expected as we needed to delete this pointer after using it. This malfunction led to the insert function crashing, particularly when used multiple times. This problem requires a deeper understanding of database handling and the concept of smart pointer.

3.5 Debugging Voice Input Feature Complications

- The most time consuming part for implementing the voice input feature is debugging a weird issue: the Replicate will never return the result, no matter how we fetch and

re-fetch the result. Turns out the issue was because we put both the fetch and refetch function call inside one single POST request handler in router.cpp. It caused the server to keep running and try to fetch the result, without giving room for the API to access the audio file from the server in the first place. Obviously, the whole process is stalled because of it. To solve this we put the follow up refetch call in a separate action, and it gives room for Replicate to grab the audio file that it needs.

3.6 Implementing Audio File Storage and Handling

- Storing the audio file was not as straightforward as we thought. We had to record the audio first on the client side, then generate a blob object, and POST it to the server as multipart form data. After the server receives the multipart, it has to be parsed and base64 decoded into binaries. Then it can finally be stored onto the server.

4. Lessons Learned

The HOP project successfully redefined the online shopping experience by allowing intuitive product searches without precise textual queries, integrating natural language processing and OpenAI API. A key turning point was shifting from the Drogon C++ library to a remote server setup, addressing compatibility issues and enhancing collaborative efficiency. This adaptability was further displayed in choosing JSON for data management, streamlining data handling within the MySQL database.

Strategic decisions in technology selection and a user-centric design philosophy were central to the project's success. The team's flexibility in overcoming technical challenges and their focus on accessibility and privacy through features like local search history storage significantly enhanced the platform's user experience. HOP's development journey underscores the importance of adaptability, strategic planning, and a commitment to user needs in innovative technology projects.