Project Report: Query Recommender System

By :Omkar Khade

Project Github Link:
https://github.com/omkardev7/QED42-task-Query-Recommender

---

1. Introduction

This report outlines the development of a **GenAI-powered query recommender system** that generates relevant search queries and their answers based on a user's browsing history or session on a website.

For the purposes of this assignment, we have created a mock website named "**CodeBuddy 2.0**", which is designed to demonstrate how user interactions, session data, and recommendations work in a real-world scenario.Additionally, the **CodeBuddy 2.0** incorporates a **Retrieval-Augmented Generation (RAG)** pipeline to provide personalized responses to this recommended queries.

CodeBuddy 2.0 is an educational platform designed to help users enhance their skills across various fields such as web development, data structures and algorithms (DSA), full-stack development, Java programming, and software testing. The platform provides content through multiple pages such as **Blog**, **Courses**, **DSA Guide**, **Fullstack**, **Java**, **Soft Test**, and more.

1.1 Objectives of the Project:
The objectives of this project are:

1. Develop an interactive educational platform with dynamic and personalized content.
2. Track user interactions across various pages of the website, storing and processing this data to calculate page importance.
3. Leverage a RAG-based system to provide accurate and context-aware answers to user queries.
4. Enable personalized query recommendations based on user activity and engagement.

2. Technologies Used

The following technologies were used in the development of **CodeBuddy 2.0**:

- **HTML**: Used to structure the content and create the user interface for various pages on the website.
- **CSS**: Provides styling and ensures the website is visually appealing and responsive.
- **JavaScript**: Handles user interaction events such as clicks, scrolls, and page visits, allowing for real-time tracking of user activity.
- **Flask**: A Python web framework that serves the website and handles routing. It also processes user data and stores it in a database.
- **SQLite**: A lightweight relational database used for storing session tracking data (time spent, clicks, scrolls, etc.).
- **Python**: Handles back-end operations, including data processing, session management, and integration of the RAG pipeline.
- **Groq API**: Used for LLM inference.
- **LLaMA 3 LLM Model**: This language model is utilized within the RAG pipeline to generate accurate, context-aware answers based on the user's query.
- **Python-dotenv**: Manages environment variables securely, storing sensitive configuration data like API keys and database credentials.
- **RAG Pipeline:** processes website content, creates a vector store, and uses LLM model to answer questions based on the content.

3. System Architecture

The system architecture of **CodeBuddy 2.0** consists of two main components:

3.1. Front-End Design

The front-end of CodeBuddy 2.0 is built using **HTML, CSS, and JavaScript**. It consists of the following pages:

- **Home Page**: Overview of the platform and key features.
- **Blog Page**: A hub for tutorials and learning materials through blogs.
- **Course Pages**: Various courses related to technologies like web development, Java, full-stack, and DSA.
- **Other Educational Pages**: Providing detailed guides and resources in Java, Full Stack, DSA, and Software Testing.

3.2. Back-End Design

The back-end of the application is powered by **Flask** and **Python**:

- **Session Tracking**: The back-end captures user interactions like page visits, clicks, scroll depth, and time spent on pages. These metrics are logged in an **SQLite** database for processing.
- **Page Importance Calculation**: The system calculates the importance of each webpage based on user behavior.
- **Query Recommendations**: Based on the calculated importance of each page, the system recommends the most relevant Queries to users. The **RAG pipeline** integrates with the back-end to process user queries.
- **Query Response Generation:**The **RAG pipeline** integrated with the back-end to process user queries, processes website content, creates a vector store, and uses LLM model to answer questions/queries based on the content.

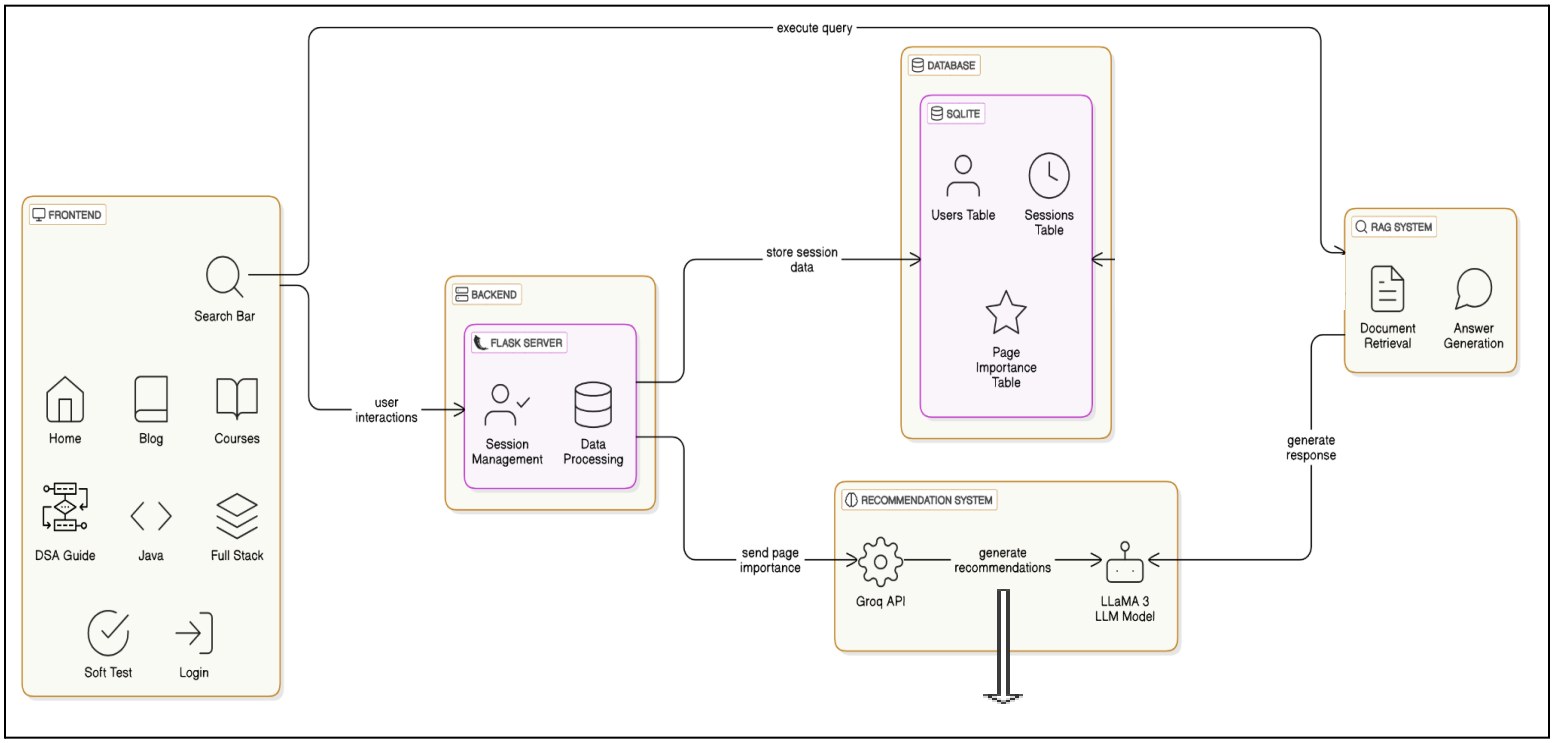

Fig. 1: System Architecture of CodeBuddy 2.0

Fig. 1: System Architecture of CodeBuddy 2.0 – This diagram illustrates the overall architecture of the CodeBuddy 2.0 platform, showcasing the integration of the front-end, back-end, session tracking, content recommendation system, and RAG pipeline. It highlights how user interactions are tracked, processed, and used to provide personalized query recommendations and search query responses using the LLaMA 3 LLM model.

4. Features and Functionality

4.1. Session Tracking System

The session tracking system is responsible for recording and processing user interactions on the website. The tracked data includes:

- **Page Visits**: Tracks which pages the user visits and in what order.
- **Click Data**: Records the number of clicks on interactive elements like buttons and links.
- **Time Spent on Each Page**: Tracks how long a user stays on a page.
- **Topics:** Tracks topics of user interacting page

All this data is sent asynchronously to the Flask back-end and stored in the **SQLite** database.

4.2. Importance Calculation and Content Recommendation

Each webpage's **importance score** is determined by the following factors:

1. **Time Spent**: Pages with higher user engagement (longer time spent) are considered more important.
2. **Clicks**: The more clicks a page receives, the more relevant it is deemed to the user.
3. **Topics:** Tracks topics of user interacting page

Importance score calculation= (time_spent * 0.7 +clicks * 0.3) / 100

Based on the calculated importance, the platform recommends pages with higher engagement to the user. This ensures users receive the most relevant content based on their interactions.

4.3. Retrieval-Augmented Generation (RAG) Search

The **RAG pipeline** is implemented to enhance the website's query response functionality. The process is as follows:

1. **Document Retrieval**: When a user submits a query, the system first retrieves documents from the database that are relevant to the query using keyword matching.

2. **Answer Generation**: After retrieving relevant documents, the **LLaMA 3 LLM Model** is used to generate an accurate, context-sensitive answer based on the retrieved content. This allows for more personalized and accurate responses than simple keyword-based search.

The **Search Your Questions** section on the website at question page is integrated with the RAG pipeline, allowing users to ask queries and receive personalized responses.

5. Task Overview

Task 1: Developing the Interactive Web Platform

- **Challenge**: Designing a dynamic and interactive educational platform
- **Solution**: The CodeBuddy 2.0 platform was developed using HTML, CSS, JavaScript, and Flask to create interactive web pages.

Task 2: Implementing Session Tracking and Recommendations

- **Challenge**: Collecting user interaction data and processing it to recommend personalized query based on importance.
- **Solution**: JavaScript event listeners were integrated for tracking user behavior such as clicks,page visit, and time spent. Tracking data was captured via JavaScript and sent to the back-end, where it was stored in an SQLite database. The importance of pages was calculated based on user interactions and formula, and a LLM-based query recommendation system was developed to provide relevant queries..

6. Output of the Project

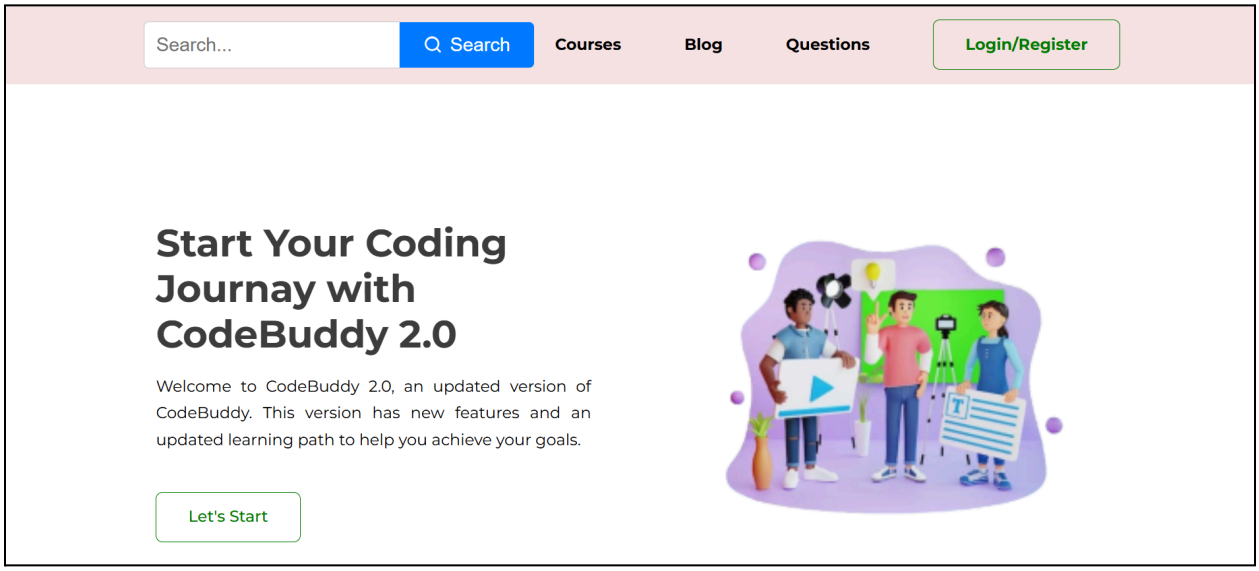

Fig. 2: Homepage of CodeBuddy 2.0 with basic content and navigation

Fig. 2: Homepage of CodeBuddy 2.0 – The main landing page featuring an overview of the platform, key features, and navigation links to various educational sections such as Blog, Courses, DSA Guide, and more, designed for easy access to different resources and content.

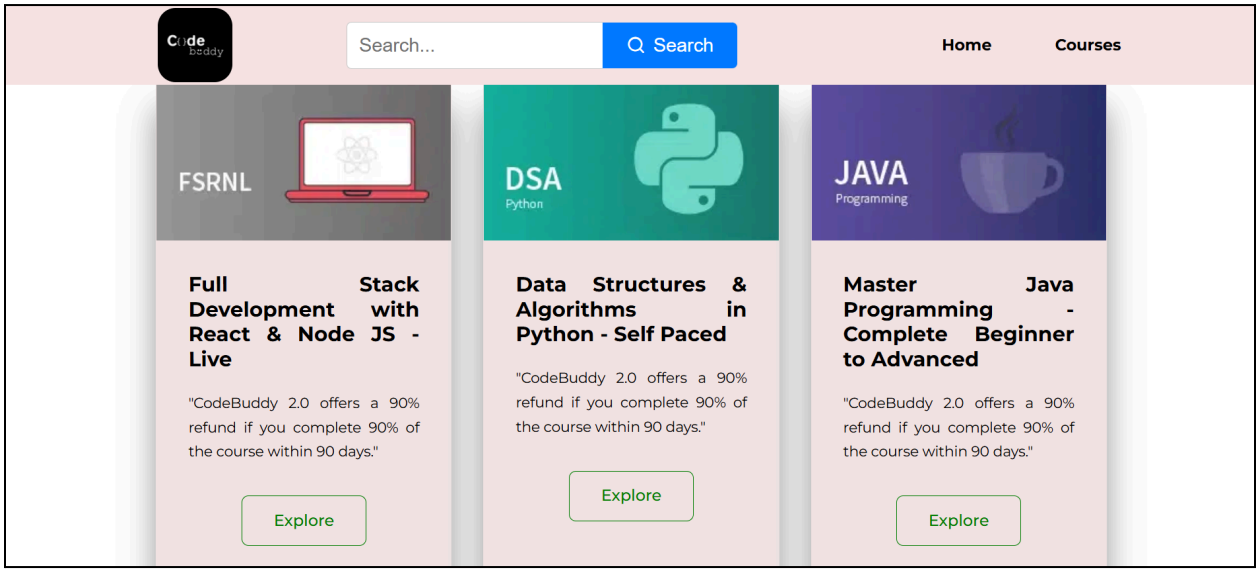

Fig. 3: Courses Page of CodeBuddy 2.0

Fig. 3: Courses Page of CodeBuddy 2.0 – This page displays a list of available courses across various subjects like Web Development, DSA, Fullstack, Java, and Software Testing, providing users with easy access to relevant learning materials and resources.

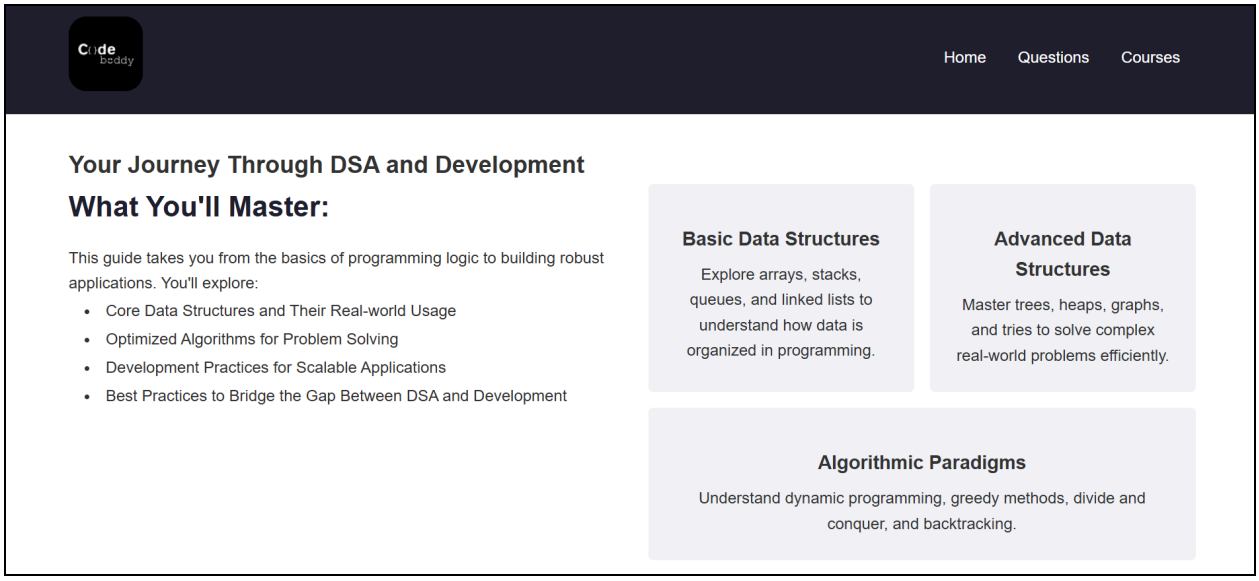

Fig. 4: DSA Page in the Courses section of CodeBuddy 2.0

Fig. 4: DSA (Data Structures and Algorithms) Page in the Courses section of CodeBuddy 2.0 – This page offers detailed content and resources related to Data Structures and Algorithms, designed to help users learn and practice DSA concepts through tutorials, examples, and exercises

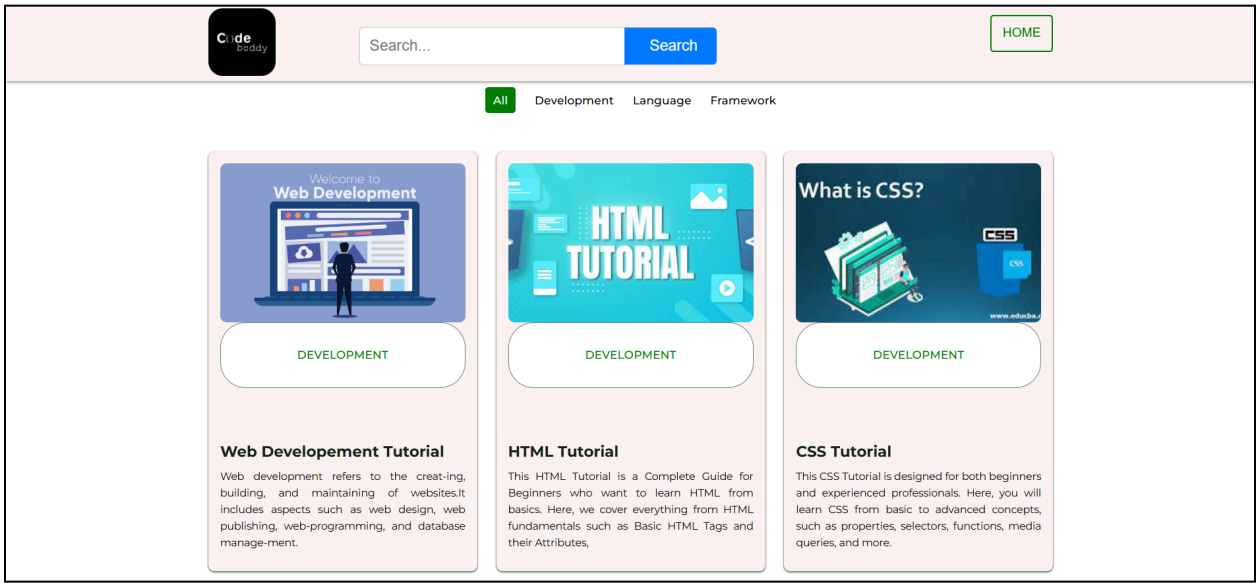

Fig. 5: Blogs Page of CodeBuddy 2.0

Fig. 5: Blogs Page of CodeBuddy 2.0 – This page features a collection of blog posts on various topics, including tutorials, tips, and industry insights, aimed at providing valuable learning material to users.

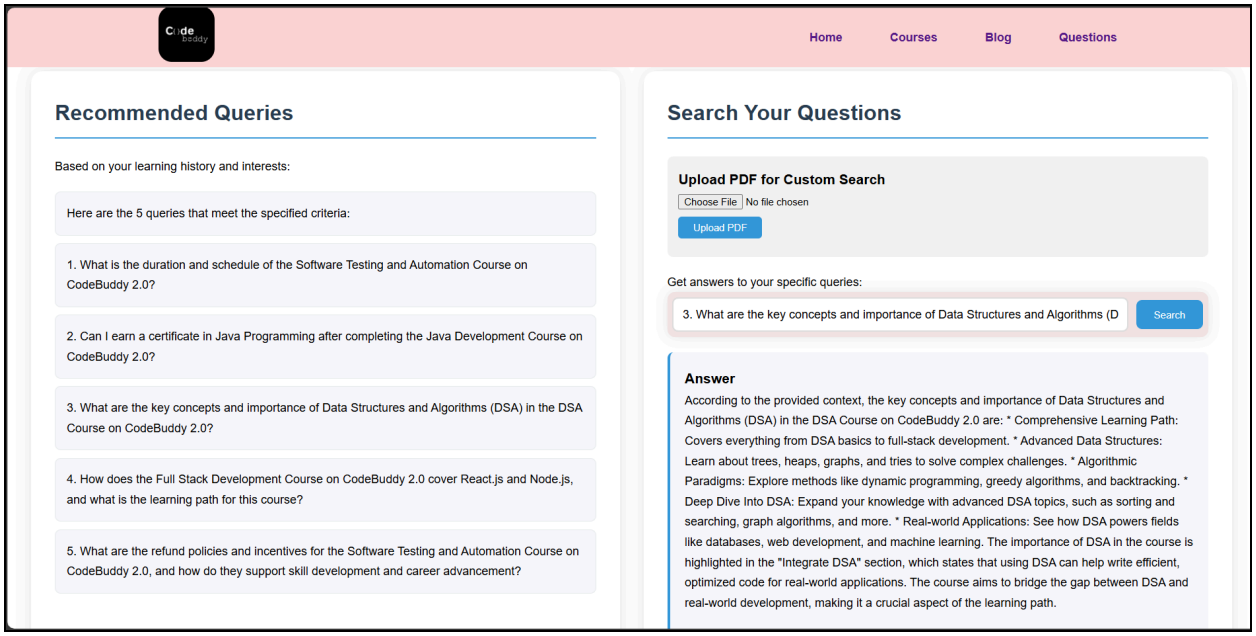

Fig. 6: Queries Recommendation Page of CodeBuddy 2.0

Fig. 6: Queries Recommendation Page of CodeBuddy 2.0 – This page dynamically suggests queries based on user interactions and the importance of web pages, providing personalized query recommendations and Rag-pipeline based **Search Your Questions** section allowing users to ask queries and receive responses.

7. Future Scope

While **CodeBuddy 2.0** is highly functional, there are many areas for further enhancement:

User Behavior Tracking Improvement:

- **Current Limitation**: The system cannot track topics, scroll depth, or specific engagement details.
- **Improvement**: Future updates will enable tracking of topics, scroll depth, etc. This will provide more accurate query recommendations.

More Accurate Prompt Template for LLM:

- **Current Limitation**: LLM responses can be generic.
- **Improvement**: Refining prompt templates to include more context from user interactions will yield more accurate, personalized responses, enhancing the relevance of generated recommendation & answers.

Live Website Content Scraping for RAG:

- **Current Limitation**: Real-time scraping of website content is not possible to feed RAG pipeline; PDFs are used instead.
- **Improvement**: Future upgrades will focus on automating PDF content extraction and indexing, improving the searchability of content. Additionally, exploring external scraping methods will help gather content dynamically for better query responses.

Use of Caching Layers:

- **Improvement**:Future upgrades will focus on Implementation of **Redis** as a caching layer to store frequently accessed user behavior data temporarily. This will reduce the load on the database, speed up read operations, and improve overall performance, especially for high-traffic users.