תמונה שמכילה צילום מסך, גרפיקה, אדום, צבעוני

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.פרויקט - מבוא למחשוב ענן

TEAM\_SLOTH

: **Link to the note-book Collab**

<https://colab.research.google.com/drive/1X1kn8J14qp9ve640WG8Hp_kRfaYPiOGB?usp=sharing>

[**https://github.com/NivCohen1/CloudComputing\_Sloth\_Project.git**](https://github.com/NivCohen1/CloudComputing_Sloth_Project.git) **:Link to GitHub**

**מהות המוצר ומרכיביו**

* **שם האתר: Huawei** <https://www.huawei.com/>

המוצר שפיתחנו הוא מנוע חיפוש, המאפשר איתור תוכן באתר Huawei בצורה מהירה, מדויקת ורלוונטית עבור המשתמשים.

* **מרכיביו:**

מנגנון חיפוש - מאפשר חיפוש באמצעות הזנת שאילתות, תוך מיקוד בתוצאות רלוונטיות ואפקטיביות.

מנגנון השלמת מילים אוטומטית - המערכת מציעה השלמות בזמן אמת במהלך הקלדת שאילתות בתיבת החיפוש.

דירוג תוצאות חיפוש - המערכת מבצעת דירוג של התוצאות, במטרה להציג למשתמש את המידע המותאם ביותר לצרכיו.

ממשק ניהול למנהל מערכת - מסך לניהול מסד הנתונים הכולל פעולות כמו מחיקה, הוספה ועדכון נתונים.

מסך סטטיסטיקות מתקדם - מציג נתונים למילות החיפוש הנפוצות ביותר.

צ'אט-בוט אינטראקטיבי - כלי תקשורת המיועד לספק מענה מיידי למשתמשים.

**פיצ'ר מעניין :** -**Auto complete השלמת חיפוש לפי תדירות החיפושים**

הרעיון הוא להציג למשתמשים בעת חיפוש את התוצאות הנפוצות ביותר (שהיו בשימוש ביותר פעמים) קודם בתור הצעה להשלמת מילה בחיפוש, וזאת על מנת למקד את החיפוש ולספק להם תוצאות רלוונטיות יותר.

הפיצ'ר מבצע סינון של התוצאות שהופיעו ביותר פעמים בחיפושים, כך שהמשתמשים רואים קודם את המונחים/תוצאות שהיו יותר פופולריות. הפיצ'ר מתבצע על ידי השגת ההיסטוריה של החיפושים ממסד הנתונים (Firebase) ומיון התוצאות לפי כמות הפעמים שחיפשו.

הפיצ'ר ממומש בשני חלקים:

1. עדכון היסטוריית החיפושים - כאשר משתמש מבצע חיפוש, נשמרת ההיסטוריה עם כמות הפעמים שנעשה שימוש ב- . Term
2. סינון התוצאות על פי תדירות החיפושים - כאשר מחפשים מונח, מתבצע מיון של התוצאות לפי תדירות החיפושים במערכת.

הפיצ'ר הזה מספק אינדיקציה למנהל על המונחים המובילים במערכת, ועוזר במיקוד המשאבים או בהבנת ההתנהגות של המשתמשים.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Microservices**

בפרויקט השתמשנו Microservices עיקריים:

1. **FirebaseService** : אחראי על תקשורת עם מסד הנתונים Firebase השירות מבצע פעולות שונות כגון: הוספת כתובת URL חדשה למונח מסוים, עדכון היסטוריית חיפושים של המשתמשים, אחזור נתונים ותוצאות חיפוש מהמסד.השירות מבודד את כל פעולות התקשורת עם מסד הנתונים, מה שמאפשר ניהול נוח, מודולריות ושדרוג קל בעתיד, בלי להשפיע על שאר רכיבי המערכת.
2. **SearchService** : אחראי על עיבוד והכנה של שאילתות חיפוש.  
   השירות מבצע: ניתוח המונח המבוקש, הכנה ואופטימיזציה של השאילתה למסד הנתונים, שילוב נתונים רלוונטיים מהיסטוריית החיפושים לצורך שיפור התוצאות.
3. **TermService :** אחראי על ניהול מונחים. השירות מטפל בניהול המונחים כגון הוספה, הסרה, וחיפוש נתונים קשורים למונחים.
4. **StatisticsService:** שולף ומנתיח מידע סטטיסטי על מונחי חיפוש והיסטוריית שימוש.
5. **ChatService :** שולף תשובות מותאמות ממאגר המדע ושומר את היסטוריית השיחות הודעות משתמש ותשובות chatbot

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**KPI-Key Performance Indicators**

**User Experience.1 :**

עיצבנו ממשק משתמש אינטואיטיבי וברור, המאפשר חיפוש קל והצגת תוצאות רלוונטיות עבור המשתמש .

**מדידה :** חוויית המשתמש נמדדת באמצעות **שביעות רצון של 90% ומעלה** ממשתמשים **בסקרי משוב.**

**2. זמן תגובה - Performance:**

שיפרנו את ביצועי המערכת באמצעות אופטימיזציה למסד הנתונים. הנתונים במילון Firebase מאורגנים לפי מילות מפתח, תוך שימוש בטכניקות כמו: Lower case, Stop words, Stemming, ואיחוד מילים מאותו השורש. שיטות אלו מאפשרות שליפת נתונים מהירה ומונעות זמני המתנה ארוכים למשתמשים.

**מדידה :**

זמן הצגת תוצאות: תוצאות יופיעו בתוך 3 שניות לכל היותר לאחר ביצוע החיפוש.

זמן תגובה ממוצע: זמן ממוצע של 90% מהחיפושים לא יעלה על 2 שניות.

**Scalability and Elasticity .3 :**

המערכת פותחה בגישה מודולרית, כך שכל פונקציה ומחלקה פועלות באופן עצמאי, מה שמאפשר גמישות גבוהה בהרחבה

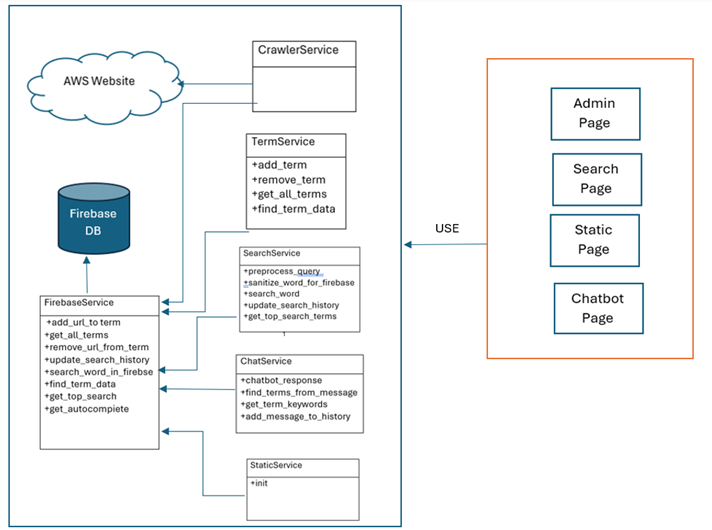
**מדידה :**

כל מחלקה ופונקציה מתפקדות באופן עצמאי, כך שניתן להוסיף תכונות חדשות ללא שינוי בקוד הקיים.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ארכיטקטורת המערכת:**

\*במערכת שלנו השתמשנו .Microservice כך שכל שירות אחראי על חלק מסוים בפעולות.



Use Case

תמונה שמכילה טקסט, תרשים, ציור, שרטוט

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**דרישות פונקציונליות**

- המערכת מאפשרת חיפוש באמצעות הזנת מילת מפתח או ביטוי לקבלת תוצאות רלוונטיות.

- המערכת מאפשרת השלמת מילים באופן אוטומטי.

- המערכת מאפשרת הצגת סטטיסטיקות עבור החיפושים הנפוצים ביותר.

- המערכת מאפשרת עריכת אינדקס, כולל מחיקה והוספה של פריטים.

- המערכת תומכת באינטראקציה עם צ'אטבוט לקבלת מידע.

**דרישות לא פונקציונליות**

**:Performance**

הפרויקט דורש זמני תגובה מהירים למערכת החיפוש, טעינת ממשק מהירה ותוצאות בזמן אמת. פתרנו את זה על ידי שימוש בשמירת מידע ב-Firebase של חיפושים פופולריים ויישום אלגוריתמים יעילים לעיבוד מחרוזות כדי לשפר את הביצועים.

**:Usability**

הפרויקט דורש ממשק ידידותי למשתמש עם תצוגות ברורות, כולל השלמה אוטומטית, גרפים וסטטיסטיקות. פתרנו את זה על ידי שימוש ב -

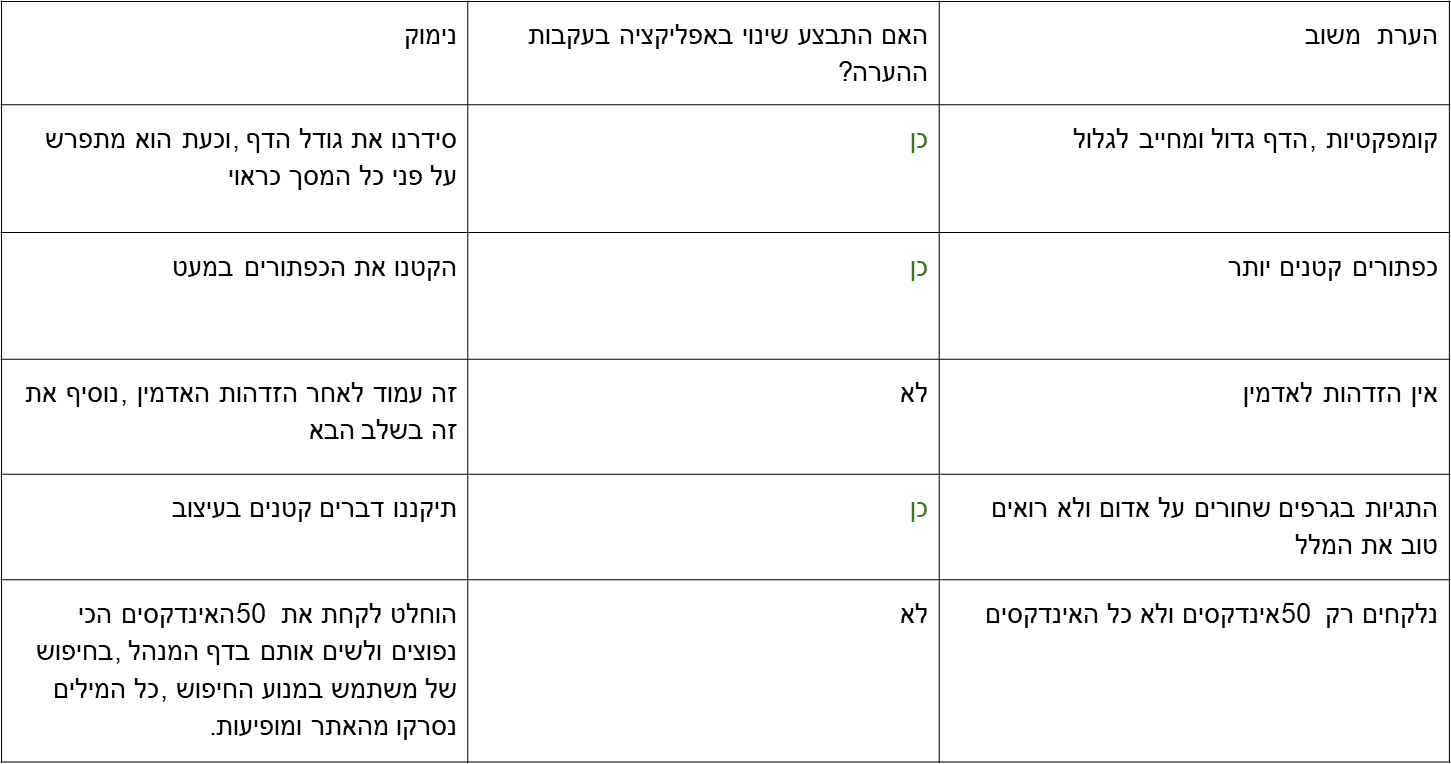
לעיצוב גמיש ו ברור. -Tailwind CSS ו-JavaScript

**:Scalability**

 המערכת תתוכנן כך שתוכל לתמוך בקלות בתוספות, שדרוגים או הרחבות עתידיות, ללא צורך בשינויים מבניים משמעותיים על ידי שימוש ב מיקרו-שירותים .

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ביקורת עמיתים ( שבוע 9)**



לאחר קבלת המשוב חישבנו את ציון ה-SUS, שהתוצאה הייתה 84 המעידה על שמישות מצוינת. המשתמשים ציינו כי המערכת אינטואיטיבית, קלה לשימוש ועקבית, והביעו שביעות רצון גבוהה מהפונקציונאליות .

**שיפורים שבוצעו:** צמצום גודל הדף, התאמת הכפתורים ושיפור ניגודיות הצבעים לקריאות טובה יותר.

**משוב שבוע 12:**

|  |  |  |
| --- | --- | --- |
| הערת משוב | האם לדעתכם יש צורך בשינוי במערכת בעקבות ההערה? | נימוק |
| צ׳טבוט דרך בינה מלאכותית ולא הארד קודד | לא | הארד קודד תמיד ייתן תשובות צפויות לפי הגדרות המערכת. במקרה שלנו מספר השאלות והתשובות מצומצם וידוע מראש. |
| הGUI בצבעים קצת לא נעימים לעין | לא | הצבעים שנבחרו לממשק המשתמש נבחרו בקפידה כדי לשקף את הזהות והמהות של האתר שלנו (huawei) |
| להוסיף יותר שקיפות אלגוריתמית | כן | הוספנו |

**Programmer Documentation**

**Introduction**

This project implements a robust search engine system, leveraging the power of Google Colab for development and Firebase as the backend for real-time data operations. Designed with modularity and scalability in mind, the system supports the following features:

**Manage Index:** Provides an interface for modifying, and deleting indexed terms and their associated URLs. This feature ensures the system remains relevant and up-to-date as the data evolves.

**Search Functionality:** Offers users the ability to execute queries with results ranked based on relevance. An integrated autocomplete feature enhances the search experience by suggesting terms as users type.

**Statistics Visualization:** Displays insights into search term trends using bar charts. This helps administrators or analysts identify popular queries and optimize the indexed content accordingly.

**Chatbot:** A conversational interface allows users to interact with the system in a human-friendly manner. The chatbot utilizes fuzzy matching to handle approximate inputs and typos, ensuring meaningful responses even with imperfect queries.

**Purpose of the System**

The goal of this project is to provide a comprehensive and interactive search platform for the site Huawei.com that combines data management, querying, visualization, and conversational interaction. This serves multiple use cases, including:

**Search Efficiency:** By preprocessing queries (e.g., stemming, stopword removal), the system ensures relevant results are returned efficiently.

**Data Insights:** Administrators can monitor search trends and optimize indexed data based on user behavior.

**User Engagement:** The chatbot interface enhances user interaction by providing answers to questions of the users. The chat bot works like hard codded.

**System Architecture**

The architecture of the system emphasizes a Model-View-Controller (**MVC**) design pattern:

**Model:** Firebase acts as the centralized database for storing terms, URLs, and search history.

**View:** The HTML/JavaScript frontend renders an intuitive user interface, ensuring seamless navigation across features.

**Controller:** Python functions, registered as callbacks in Google Colab, act as intermediaries, linking the frontend with the backend.

**Technological Highlights**

**Firebase:**

Provides real-time data management.

Handles CRUD (Create, Read, Update, Delete) operations for terms, URLs, and search history.

**Where It’s Used:**

**Manage Index Page:** CRUD operations for managing terms and their URLs.

**Search Page:** Reading term data for query processing.

**Statistics Page:** Fetching top-searched terms for data visualization.

**Chatbot Page:** Retrieving predefined responses for user queries.

**Google Colab:**

Colab supports the integration of Python with frontend components (HTML/JavaScript) via custom callbacks.

Python functions registered as callbacks can be invoked directly from JavaScript, bridging the gap between the frontend and backend.

**JavaScript,CSS and HTML:**

Power the dynamic frontend, ensuring smooth interaction with backend services.

HTML structures the content and layout of the pages.

CSS styles the interface, ensuring a clean and user-friendly design.

**JavaScript:**

Handles user interactions (e.g., button clicks, dropdown selections).

Sends requests to backend Python callbacks via Colab.

Dynamically updates content, such as tables and charts, without reloading the page.

**Where It’s Used:**

**Frontend Pages:**

**Main Page:** Navigation and UI rendering.

**Manage Index Page:** Dynamically updating term tables.

**Search Page:** Autocomplete and result display.

**Statistics Page:** Rendering charts.

**Chatbot Page:** Sending user queries and displaying responses.

**Natural Language Processing (NLP):**

Implements stemming and stopword removal to process queries effectively.

**How It Works:**

**Stemming:** Words are reduced to their root forms using the Porter Stemmer.

Example: "running" → "run"; "eaten" → "eat".

**Stopword Removal:** Common words like "the" and "is" are filtered out based on a predefined list, leaving only meaningful terms.

**Where It’s Used:**

**Search Page:**

During query preprocessing in SearchService.preprocess\_query().

**Fuzzy Matching:**

Fuzzy matching is a technique to find approximate matches between strings, useful for handling typos or variations in user input.

The system uses the FuzzyWuzzy library to calculate a similarity score between the user’s input and predefined terms.

**Example**: Input: "computng"

Stored Term: "computing"

Similarity Score: 90% (above the threshold, e.g., 80%, so it matches).

**Where It’s Used:**

**Chatbot Page**:

Matching user queries to predefined terms in ChatService.chatbot\_response().

**Chart.js:**

Chart.js processes data in JSON format and renders it into dynamic charts (e.g., bar charts).

**Where It’s Used:**

**Statistics Page:**

Visualizing the most-searched terms and their frequencies.

**System Design and Page Breakdown**

1. Main Page

The Main Page serves as the navigation hub for all system features. It includes four buttons:

**Manage Index:** To modify the indexed terms and URLs.

**Search:** To query the index dynamically.

**Statistics:** To visualize search term data.

**Chatbot:** To interact with the system in a conversational manner.

**Process Flow:**

**User Interaction:** Clicking a button triggers the showPage(pageId) function.

**Frontend Handling:** The function toggles visibility by adding/removing the hidden CSS class for respective sections.

**Result:** Users are seamlessly redirected to the requested functionality.

**Key Benefits:**

**Clarity:** Simplifies navigation for users.

**Efficiency:** Minimizes distractions by showing only relevant sections.

2. Manage Index Page

This page allows users to maintain the integrity of indexed terms and their associated URLs stored in Firebase.

**Core Features and Processes**

## 1. Adding a URL to a Term

**Objective:** Enhance the index by associating new URLs with terms.

**Step-by-Step Process:**

**User Action:** The user inputs a term and URL, then clicks "Add URL."

**Frontend Processing:** The JavaScript function add\_url\_to\_term(event) sends the term and URL to the backend via the notebook.add\_url\_to\_term callback.

**Backend Handling:** The callback calls TermService.add\_term(term, url), which interacts with FirebaseService.add\_url\_to\_term(term, url).

**Firebase Operations:** Checks if the term exists using a GET request.

If it exists:

Adds the URL or increments its frequency.

If it doesn’t exist:

Creates a new term entry with the URL. Sends a success or failure response.

**Frontend Updates:**

If successful:

The term's URL table updates dynamically.

A success message is displayed.

If failed:

An error message informs the user.

## 2. Removing a Term

**Objective:** Delete outdated or irrelevant terms and associated URLs.

**Step-by-Step Process:**

**User Action:** The user selects a term from the dropdown and clicks "Remove Term."

**Frontend Processing:** The remove\_Term(event) function sends a request to notebook.remove\_term.

**Backend Handling:** The callback invokes TermService.remove\_term(term), which uses FirebaseService.remove\_term(term) to send a DELETE request to Firebase.

**Frontend Updates:** Refreshes the dropdown list dynamically.

**Error Handling:** If Firebase fails to delete the term, an error message is displayed, ensuring transparency.

## 3. Viewing Term Details

**Objective:** Provide insight into all URLs linked to a specific term.

**Process Flow:**

**User Action:** The user selects a term from the dropdown.

**Frontend Processing:** JavaScript calls loadTermDetails(term).

**Backend Handling:** FirebaseService.find\_term\_data(term) retrieves URLs and their frequencies.

**Frontend Updates:** Displays the data in a dynamic table for easy review.

**Relevance:** Allows users to monitor and validate the integrity of the indexed data.

3. Search Page

The Search Page provides dynamic query handling with autocomplete and ranked results.

**Core Features and Processes:**

## 1. Query Preprocessing

**Objective:** Clean and standardize user input for optimal search relevance.

**Steps:**

**Stopword Removal:** Words like "the," "is," and "and" are filtered out using a predefined list, as they add little value to search accuracy.

**Stemming:** Words are reduced to their root forms using the Porter Stemmer. For instance: "Running," "runner," and "ran" → "run."This ensures that searches match variations of a word.

## 2. Autocomplete Suggestions

**Objective:** Provide term predictions as the user types.

**Process Flow:**

**User Interaction:** As the user types, the fetchAutocompleteSuggestions(query) function sends the query to notebook.get\_autocomplete\_suggestions.

**Backend Processing:** The callback queries Firebase for terms matching the prefix.Returns the results to the frontend.

**Frontend Updates:** Displays the suggestions dynamically below the search box.

## 3. Executing a Search

**Objective:** Retrieve and display relevant results, ranked by frequency.

**Process Flow:**

The user enters a query and clicks "Search." JavaScript triggers the performSearch(event) function, sending the query to notebook.search\_word.

**Backend Processing:**

The query undergoes preprocessing (stopword removal and stemming). Firebase retrieves results for the processed terms. Results are ranked by URL frequency. Search history is updated in Firebase.

**Frontend Updates:** Displays the ranked results as clickable links.

4. Statistics Page

The Statistics Page offers a visual representation of search trends.

**Core Features and Processes:**

## 1. Fetching Data

**Objective:** Retrieve the most-searched terms from Firebase.

**Process Flow:**

JavaScript calls loadTopSearchTerms(), sending a request to notebook.get\_top\_search\_terms. The callback uses SearchService.get\_top\_search\_terms(limit) to fetch data from Firebase. The results are returned to the frontend.

## 2. Visualization

**Objective:** Display data trends in an intuitive format.

**Process:**

**Frontend:** A bar chart is rendered using Chart.js, showing the most-searched terms and their counts.

5. Chatbot Page

The **Chatbot Page** provides a conversational interface, allowing users to interact with the system by typing queries in natural language. The chatbot leverages predefined responses stored in Firebase and implements fuzzy matching to handle approximate queries or typos. Additionally, the system is capable of processing multi-term queries and returning multiple relevant answers.

**Core Features and Processes**

## 1. Fuzzy Matching

**Objective:** To handle approximate queries and provide meaningful responses even when the user makes typographical errors.

**How It Works:**

The **FuzzyWuzzy** library calculates similarity scores between the user's input and predefined terms stored in Firebase.Terms with a similarity score exceeding a certain threshold (e.g., 80%) are considered matches.

## 2.Multi-Term Query Handling

**Objective:** To extract and respond to multiple terms from a single input, ensuring comprehensive coverage of user queries.

**Process:**

The user's input is split into individual components. Each component is processed separately using fuzzy matching to identify relevant terms. If multiple terms match, the corresponding responses are aggregated and returned to the user.

## 3. Generating Responses

**Objective:** To provide accurate answers or default fallback responses when no matches are found.

**Process flow:**

**User Interaction:** The user types a query and clicks "Send."

**Frontend Processing:** The sendMessage(event) function sends the query to the backend via the notebook.chatbot\_response callback.

**Backend Processing:**

**Matching Terms:**

The ChatService.chatbot\_response(user\_message) function identifies relevant terms using fuzzy matching.

Multi-term queries are processed to extract all matching terms.

**Fetching Responses:** Predefined responses for each matched term are retrieved from Firebase.

**Combining Responses:** If multiple terms are matched, their responses are concatenated.

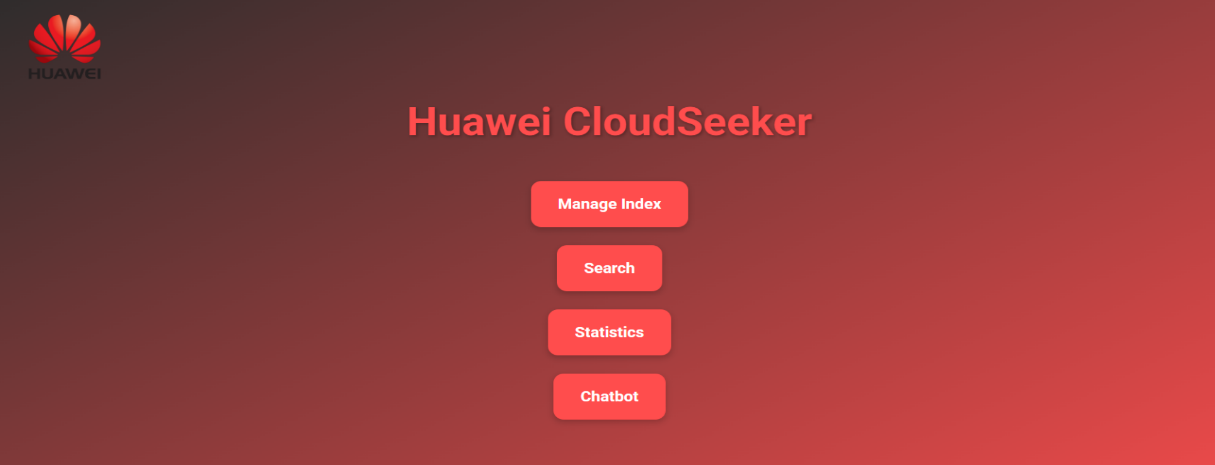
**Fallback Response:** If no matches are found, a default response is returned, such as: "I'm sorry, I don't have an answer to that question."

**Frontend Updates:** Displays the user query and chatbot response in the chat history.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**User Documents**

The Huawei CloudSeeker project is a cloud-based tool for managing, analyzing, and interacting with cloud computing data

**Screen Overview: Huawei CloudSeeker - Main Dashboard**

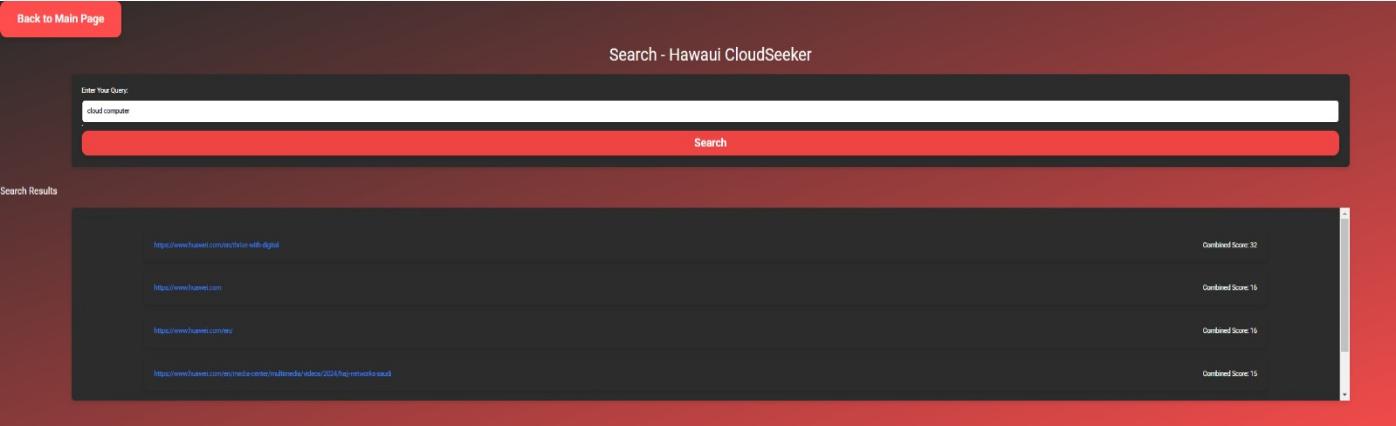
**Main Dashboard Overview-** The dashboard includes main functionalities:

**Manage Index** – The Manage Index screen allows administrators to manage terms and associated URLs efficiently.

**Search** – Opens the Search module, enabling keyword searches within indexed data.

**Statistics** – provides insights into the most searched terms through an interactive bar chart

**Chatbot** –allows users to get quick answers about Huawei services and cloud computing

**Screen Overview : Search Screen**

**Back to Main Page Button** – Redirects the user back to the main dashboard.

**Search Bar** – Users can enter queries to search relevant data, including an autocomplete feature that suggests relevant search terms.

**Search Button** – Triggers the search process and retrieves results from the database.

**Search Results** – Shown as a list of ranked URLs related to the query, with a Combined Score that represents its relevance.

**Sorting Logic** – Results are ranked based on a score calculated by the search algorithm. A higher score indicates greater relevance, determined by keyword frequency in each URL. For multi-word queries, the algorithm computes a combined score by summing the occurrences of all keywords within each URL.

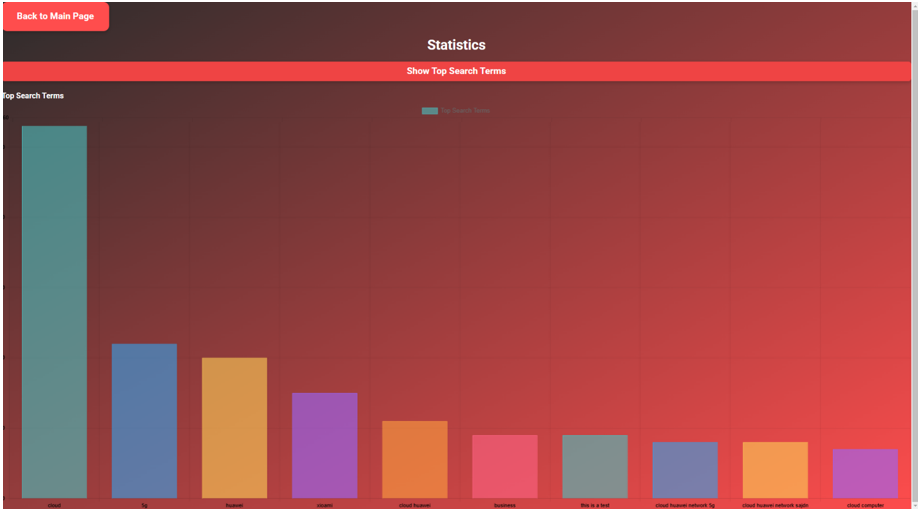
**Algorithm Transparency**

**Relevance Calculation** – The search engine retrieves data from the index and ranks results based on keyword occurrence frequency For example, if a URL contains multiple keywords from the query, its score will be higher than URLs with fewer matches.

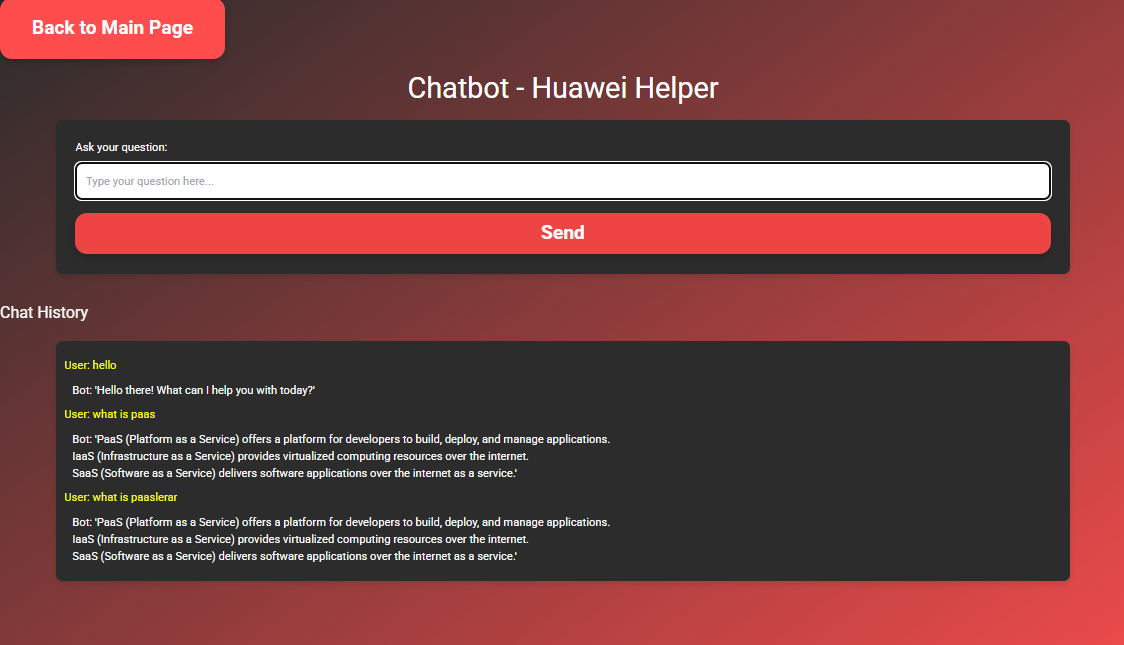
**Transparency** – The Combined Score is displayed alongside each result, providing insight into the relevance calculation. This ensures clarity in how results are prioritized and helps users trust the system.

**User Feedback** – By displaying the Combined Score, users gain an understanding of why certain results appear at the top of the list, fostering trust in the decision-making process behind search rankings.

**Screen Overview: Statistics - Huawei CloudSeeker**



**Show Top Search Terms Button** – Displays a ranked list of search terms with their frequency of use. Before clicking, users see a prompt to view data. Upon selection, a bar chart dynamically represents the most searched terms, where bar length indicates search frequency.

**Screen Overview: Chatbot - Huawei Helper**

**Ask Your Question Field** – Users enter questions, with automatic spelling correction ensuring accurate responses.

**Send Button** – Submits the question to the chatbot.

**Chat History** – Displays past conversations, showing user queries and chatbot responses.

תמונה שמכילה טקסט, צילום מסך, עיצוב

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.**Screen Overview: Manage Index - Admin Panel**

**Remove Term** – Select a term from a dropdown list and remove it from the index.

**Add URL** – Choose an existing term and associate it with a new URL via a text input field.

**Term Table** – Displays indexed terms with their associated URLs and frequency counts, with an option to remove individual URLs.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**שקיפות אלגוריתמית**

אלגוריתם מנוע החיפוש שלנו אינו נגיש למשתמשים, אך הסברנו כיצד נבחרו תוצאות החיפוש. כאשר מתבצע חיפוש במנוע, התוצאות המוחזרות מהאינדקס מסודרות לפי סדר עדיפויות המבוסס על מספר המופעים של המילה בכל URL

במידה והשאילתה מורכבת ממספר מילים, האלגוריתם מחשב את סכום המופעים של כל מילה בביטוי עבור כל דף ומסדר את הדפים בהתאם לעדיפות זו.

בזמן הצגת תוצאות החיפוש למשתמש, אנו מציגים את הציון של כל דף ומוצג למשתמש כ: Total Appearances

ציון זה מאפשר למשתמש להבין כיצד התבצעה בחירת הדפים בתוצאות החיפוש.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**אתגרים שעלו במהלך העבודה וכיצד התמודדנו**

1. חוסר תיאום בין חברי הצוות:
   * קבענו מפגשים קבועים, חילקנו משימות ברורות וביצענו סקירות התקדמות שבועיות כדי לשמור על תיאום בין כל חברי הצוות.
2. בעיות בתיאום של רכיבים שונים:
   * תכננו שלבי אינטגרציה הדרגתיים עם בדיקות בכל שלב ויצרנו ממשקים ברורים ומסודרים בין הרכיבים.
3. מגבלת מכסת נתונים ב-DB החינמי:
   * ביצענו אופטימיזציה באמצעות stop words, stemming, המרת מילים ל-lowercase ומניעת כפילויות. בנוסף, צמצמנו את העומס על ה-crawler על ידי הגבלת מספר האתרים הסרוקים.

באמצעות אופטימיזציה ותיאום טוב יותר, הצלחנו להתגבר על האתגרים ולהשיג תוצאות טובות יותר.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**מקורות מידע**

1. חומרי קורסים אקדמיים:

נעשה שימוש בחומרי ההרצאות של קורס מחשוב ענן ו-קורס Web במסגרת הלימודים האקדמיים.

ההפניות לתוכן זה מבוססות על סיכומי הרצאות, מצגות שהועברו בקורסים, ותכנים נוספים שנלמדו.

1. שימוש בבינה מלאכותית (ChatGPT):

נעזרנו ב ChatGPT -לצורך קבלת הבהרות על מושגים מסוימים, שיפור הניסוח, וקבלת רעיונות לארגון המידע.