מתכנת

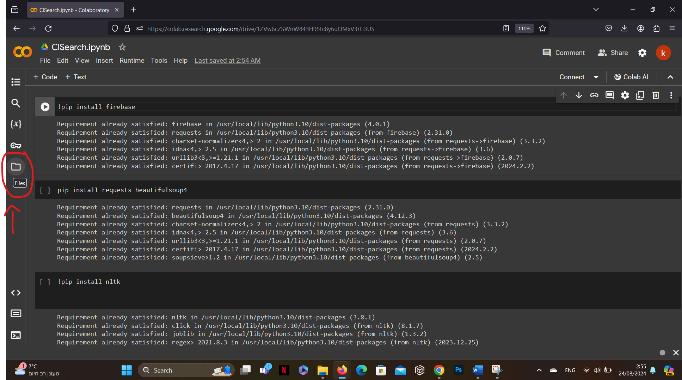
Introduction:

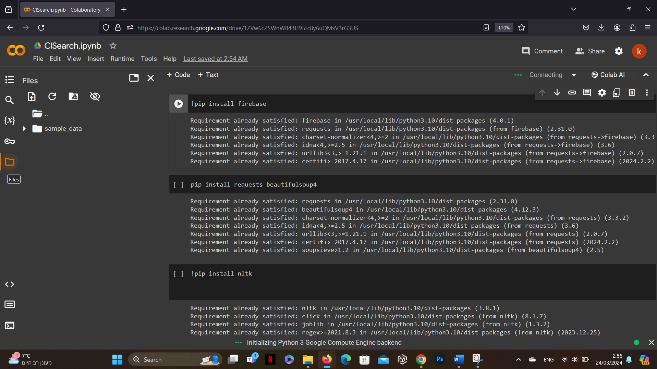
ClSearch is a search engine application designed to help users, with an intuitive and friendly user interface and robust features the application streamlines task organization.

The primary purpose of ClSearch is to provide users with a search engine tool that is accurate, reliable, and easy to use.

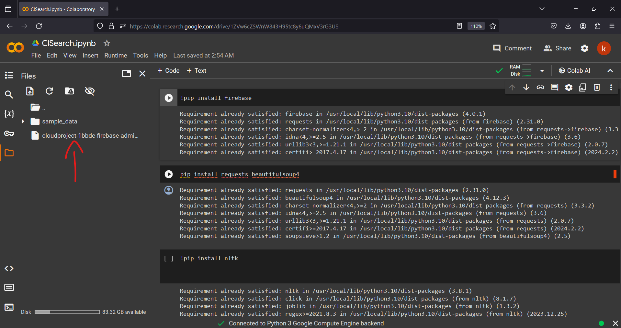
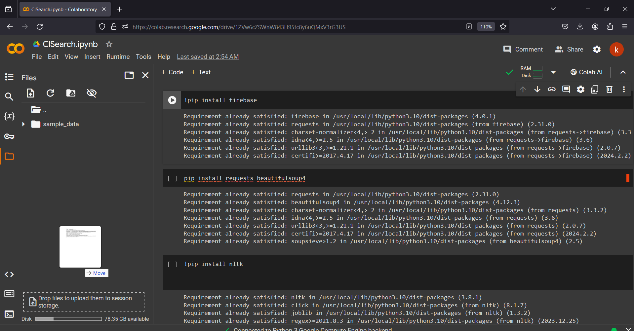
Main Files:

* ClSearch.ipynb: this file contains the main code, all the functions of the search engine, all the designs, all the backends, all the frontends. This file is the search engine, we can say it’s the entry point of the application.
* DB.ipynb: this file build the Database, gathers the information to start the application.
* cloudproject-1bbde-firebase-adminsdk-849hb-47666b58c8.json: without this file nothing works. It’s likely a service account key file for Firebase authentication. This JSON file contains credentials that allows access to Firebase service from a server or other trusted environment. (This file MUST be added locally to Google Colab notebook in order to run the code for both files: ClSearch.ipynb and DB.ipynb). Here is a brief explanation:





Now drag cloudproject-1bbde-firebase-adminsdk-849hb-47666b58c8.json file and drop it below sample\_data (not in the same directory!)

Now we can run the code peacefully. 😊

Main Functions (ClSearch):

* query\_word: this function retrieves information related to specific word from a database collection. It queries “inverted\_index” collection, searching for documents matching the given word, and returns the corresponding data if found.
  + Parameters:
    - word: A string representing the word to be queried.
* create\_index: this function initializes an index with a predefined set of words (important words). This initialization process helps in reducing the time complexity of searching by providing a starting point for the index.
  + Parameters:
    - index: The index to be initialized or updated with the predefined words.
* update\_index: this function is responsible for updating the index with new words and their relevant links. If a user types a word that is not present in the index, this function adds the word along with its associated links.
  + Parameters:
    - words\_list: A list of words to be checked and added to the index if necessary.
    - index: The index to be updated with new words and links.
* login: this function validates the provided username and password against stored user data in a Firestore database. It checks if the user exists and if the provided password matches the stored password. Additionally, it manages user login state and updates relevant information upon successful login.
  + Parameters:
    - username: The username entered by the user.
    - password: The password entered by the user.
* logout: this function handles the process of logging out users from the system. It updates the status of logged-in users to indicate that they have logged out. Additionally, it records the logout time and calculates the total session duration for each user.
  + Parameters: None
* signup: this function facilitates the signup process for new users. It handles the creation of user accounts, validation of input data, and saving user information to a Firestore database. Additionally, it initializes user session data for tracking user activity.
  + Parameters:
    - username: The username entered by the user.
    - password: The password entered by the user.
    - security\_question: The security question chosen by the user.
* calculate\_rank: this function computes the rank for each page based on the occurrences counts of a particular word. It applies a modified logarithmic transformation to the occurrence counts and then calculates the rank using transformed values.
  + Parameters:
    - occurrences: A list containing the occurrence counts of particular word for each page.
* count\_query\_occurrences: this function receives the content of a given link and counts the occurrences of words from a user query within that content. It returns the maximum count of occurrences among the words from the user query.
  + Parameters:
    - link: The URL of the page from which content will be retrieved.
    - user\_query: The user’s query string containing words to search for in the page content.
* get\_documents\_id: this function retrieves a list of documents IDs from a specified collection in Firestore database.
  + Parameters:
    - collection\_name: The name of the collection from which document IDs will be retrieved.
* display\_user\_times: this function retrieves and displays user time data from Firestore database for the last 7 days. It plots the user’s time spent over the last week and also plots link counts for better visualization.
  + Parameters:
    - user\_id: The ID of the user for whom the time data will be displayed.
* count\_links: this function retrieves documents from the “index” collection in Firestore database and counts the occurrences of each link present in those documents.
* get\_user\_id: this function retrieves the user ID of the user who is currently logged in by querying the Firestore database for users with ‘isLoggedIn’ set to 1.
* calculate\_total\_senconds: this function calculates the total time spent by a user between login and logout events and updates the Firestore database with the aggregated time data.
  + Parameters:
    - login\_times: A list containing login timestamps.
    - logout\_times: A list containing logout timestamps.
    - user\_id: The ID of the user for whom the total time is being calculated.
* get\_logout\_times: this function retrieves logout times for a specific user from the Firestore database.
  + Parameters:
    - user\_id: The ID of the user for whom logout times are being retrieved.
* get\_login\_times: this function retrieves login times for a specific user from the Firestore database.
  + Parameters:
    - user\_id: The ID of the user for whom logout times are being retrieved.
* generate\_user\_id: this function generates a unique user ID for new users in the system.
* create\_line\_plot\_with\_makers: this function generates a line plot with markers representing the time spent by a user on different dates.
  + Parameters:
    - user\_id: The ID of the user for whom the line plot is generated.
* display\_statistics: this function generates a pie chart displaying the occurrences of links indexed.
  + Parameters:
    - index\_data: Optional parameter. If provided, contains the index data. If not provided, the function fetches index data from Firestore.
* handle\_new\_link: this function manages the addition of a new link to the index associated with a selected word.
  + Parameters:
    - selected\_words: The word selected by the user.
    - new\_link: The new link to be added.
* handle\_new\_item: this function manages the addition of a new word and its associated link to index.
  + Parameters:
    - new\_word: The new word to be added.
    - new\_link: The link associated with the new word.
* remove\_word\_from\_firestore: this function deletes a document corresponding to a given word from the Firestore database.
  + Parameters:
    - word: The word for which the corresponding document needs to be deleted from Firestore.
* extract\_index\_data\_from\_firestore: this function retrieves index data from firestore, extracting words and their associated links.
* generate\_index\_table\_rows: this function creates HTML table rows for each word in the index along with its associated unique links.
  + Parameters:
    - index\_data: A dictionary containing index data with words as keys and associated links as values.
* upload\_index\_to\_firestore: this function uploads or updates the index data in Firestore.
  + Parameters:
    - index: The index to upload or update, represented as a dictionary where keys are words and values are arrays of links.
* extract\_link\_title: this function extracts the title of a page from the provided link.
  + Parameters:
    - link: The URL of the page from which to extract the title.
* apply\_stemming: this function applies stemming to a word using the Porter stemming algorithm.
  + Parameters:
    - word: The word to be stemmed.
* remove\_stop\_words: this function removes common stop words from a given text index.
  + Parameters:
    - index: The text index from which stop words are to be removed.

Main Functions (DB):

* fetch\_page: this function retrieves the HTML content and links from a specified URL.
  + Parameters:
    - url: The URL of the page to fetch.
* extract\_data\_from\_url: this function extracts text data from the HTML content of a web page.
  + Parameters:
    - soup: The BeautifulSoup object representing the parsed HTML content of the web page.
* create\_json\_object: this function creates or updates a JSON object with data associated with a given URL.
  + json\_object: The JSON object to be created or updated.
  + url: The URL to be used as a key in the JSON object.
  + data: The data to be associated with the URL.
* create\_index: this function generates an index of textual data extracted from web pages linked to a given URL.
  + Parameters:
    - url: The URL of the web page from which to start the indexing process.
    - jsonData: The JSON object to store the indexed data.
* remove\_stop\_words: this function eliminates stop words from a given query, enhancing the relevance and precision of search results.
  + Parameters:
    - query: The input query containing text to be processed.
* clean\_special\_characters: this function removes special characters and non-ASCII characters from a given text string, ensuring compatibility and cleanliness for further processing or analysis.
  + Parameters:
    - text: The input text string containing special characters to be cleaned.
* is\_valid\_url: this function checks whether a given URL is valid and conforms is valid and conforms to standard URL protocols (HTTP and HTTPS), ensuring it is safe for use in web applications and prevents potential security vulnerabilities.
  + Parameters:
    - url: The input URL string to be validated.
* fetch\_page\_iterative: this function retrieves web page links iteratively, up to specified depth, while avoiding cyclic dependencies and ensuring each link is valid and safe to visit.
  + Parameters:
    - url: The initial URL string from which to begin fetching links.
    - depth: The maximum depth of link retrieval, controlling the number of levels to traverse.
    - visited: A set containing URLs that have already been visited to prevent revisiting and cyclic dependencies.
* save\_to\_json\_file: the function enables the storage of Python data structures, such as dictionaries or lists, into a JSON file on disk. This functionality is valuable for persisting structured data in a universally compatible format for future retrieval and processing.
  + Parameters:
    - data: The Python data structure to be saved to the JSON file.
    - file\_path: The path to the JSON file where the data will be stored.
* build\_index: this function constructs an inverted index from a collection of web pages. It treats uppercase and lowercase words as equivalent and excludes non-word characters to generate a normalized index.
  + Parameters:
    - urls: A list of URLs representing the web pages to index.
* upload\_index\_to\_firestore: this function is designed to upload an inverted index to Firestore, a NoSQL document database provided by Google Cloud.
  + Parameters:
    - Inverted\_index: The inverted index to upload to Firestore. It is a dictionary where keys are terms (words) and values are lists of URLs where each term appears.

Instructions for running the code:

Our project is divided into two main files: ClSearch.ipynb which is the project and the system, includes backends, frontends, logics, etc. DB.ipynb is for building the database, this file is executed only once, where we gather information to build the database, and after that there is no need to run the file.

* DB.ipynb: as mentioned above this file is for building the database, we can define any depth we want to go further if we changed a variable called “depth” in the code. Due to it’s huge search for data, it might take somewhile to finish the code, for that, if you want to run the code (which there is no need cause the database is already built), we suggest you to stop the code manually.
* ClSearch.ipynb: as mentioned above, this file is the whole system, we can navigate through the file as we want, it displays results, statistics, etc.

In both files, when executing, you may get this error: The default Firebase app does not exist, Make sure to initialize the SDK by calling initialize\_app().

This error typically occurs when the Firebase SDK has not been properly initialized in the application. Firebase SDK, or Software Development Kit, is a collection of libraries and tools provided by Google to help developers integrate Firebase services into their web, mobile, and server-side applications.

To solve this error: In ZIP file that we submitted, there is a file called: cloudproject-1bbde-firebase-adminsdk-849hb-47666b58c8.json, which is a service account key file for accessing Firebase service. So first of all we have to add it manually and locally to Google Colab as showed before, in our code you can find two lines:

cred = credentials.Certificate(‘cloudproject-1bbde-firebase-adminsdk-849hb-47666b58c8.json’)

# firebase\_admin.initialize\_app(cred)

In order to solve the error you facing, just remove the ‘#’ before firebase\_admin.initialize\_app(cred) – hence, to convert it to regular code line, and not a comment line, this will initialize the Firebase integration and solve the problem.

After doing that you can run the code with no problems.