Faculty of Engineering

CMPN306 – Advanced Programming Techniques

Cairo University

Credit Hours System

Project Algorithms

Submitted by: Team (6)

Aliaa Khalifa	1170315
Shorouk Mohamed Roshdy	1170110
Samah Mohamed Ahmed	1170160
Habiba Mahmoud El-Helaly	1170347

1. The Web Crawler

➔ The Database

- In Seeds collection

Each URL has 4 fields in the database

- **URL** : string (The url string)
- **Visited** : boolean (Whether this url was previously visited or not)
- **Assigned** : boolean (Whether it's assigned to a thread or not)

- In Robot collection

Each Host URL has 4 fields in the database

- **Host**: string (The url String)
- **Allows**: Array of allows paths
- **Disallows**: Array of disallows paths

➔ The Main Loop

- While visited seeds < 5000:

  - Get url from database that is not visited before or assigned to another thread
  - Connect and get document and hyperlinks from the URL
  - Get robot.txt allows and disallows for the url
  - Loop on extracted hyperlinks

    - If not empty and not in disallows urls

    - Insert it into the database with variables (Visited, assigned) set to false

➔ **Multithreading**

- Each thread assigned a url if (not visited and not assigned before to any thread)

- If a thread doesn't find any url to get and number of visited urls < 5000 → it waits (sleeps) untill a new url is inserted into the database

- Each time a thread insert a new url into the database → it notifies other threads so if there's a waiting thread, it wakes up and continue

- RobotLock → prevents two threads from accessing Robot Collection at the same time

- Database Object Lock → prevents two threads from reading a new url or inserting a new one at the same time

2. The Indexer

The indexer is implemented using multi-threading.

For each thread, it iterates on given documents and does the following:

- Parses the document by getting sentences in all tags in the <*head*> and <*body*> tags.
- It splits these sentences to get each word separately, then adds all the words to a list.
- The indexer then iterates on the list of words, and removes the stop words, then performs stemming on each word.
- The indexer then iterates on the new list of words (Stemmed, and without the stop words), and counts the occurrence for each word in the current document, then accesses the database (using synchronization between

threads), and check if this word already exists in the database, if yes, it updates its fields (TF, IDF, URLs and their titles, and tags), if the word doesn't exist, it inserts a new document in the db. for this word.

➔ Stop words are stored in an ArrayList, read from a .txt file.

➔ Number of the threads is sent from the Crawler to the Indexer

➔ For each thread, it iterates on the length of document (5000), and adds its number on the current document (its ID) in each iteration.

➔ Special characters are removed from the word, and then converted to lowercase

Ex.: Ne?ed → need

3. The Query Processor

- It's implemented using SpringBoot, where 2 local API endpoints are implemented, one for retrieving search suggestions from the database and sending it back as a response to the GUI, and another for retrieving the results of the searched word for the GUI to display.

- It performs stemming, converts the word to lowercase, and removes special characters, before retrieving the results from the database using that word.

4. GUI

The GUI is implemented by android java using the android studio and the connection between it and the query processor is through Api's interfacing by using the Retrofit library that use the concept of threading as a background process in order not to freeze the App

➔ That layout and UI

Is implemented through the XML lines by using horizontal and vertical guidelines with percentage ratio to make sure of how compatible is the UI , and the UI is divided in to 3 screens

1- the screen where our search occur
2- the results screen
3- the webview screen

➔ The logic of each screen

1- The screen where our search occurs "the Main activity"

- Implemented the AutoView to view suggestion and linked with an arraylist that has string and it is linked and updated with our suggestion Api it starts working when the user inserts a letter and is sync with user input
- The Voice search is working in the apk run on the android device not emulator
- And a button that movies me to the next screen

2- The results screen

-A recycle View is used to display the results of the search word and page limiting is implemented manually their logic as each page has 10 links

-each item is used in the recycle View is clickable and moves to the third screen which is the web page that displays the url

3- The webview screen

- When clicked on a certain url or title it opens in our app a webview that shows this Url

➔ Projects files

Layout : activity_main , activity_results ,activity_website

➔ Java files :

1- JsonPlaceHolderApi

- To specify the routes of the api and function will be caled when such an api is called

2-MainActivity

// getting the views from the activity
//getting the speech to text dialog
//going to next indent and passing the query
//API parsing for Api
///function for the mic button
///for action bar search

3-ResultsActivity
/////Getting query
////recycleView
//// get api
////pages handling
//Navigating through the pages

4-UrlAdapter

// to handle the recycleView and its elements

5-UrlItem

// the object we receive from the query apI

6-WebsiteActivity

//getting the url

//displaying the Url