CODE FUNCTIONALITIES

This section covers major files used in Django and small descriptions of each file with their functionalities.

- Views.py - Backend logic is being written in this file in multiple functions
- urls.py - This is the start point of the project and contains the routes to which the web browser has to be routed once the user enters the URL
- models.py - we currently do not have any database usage in the project and this file contains the section which can be used for the database setup
- templates/ - This folder contains the html templates which have to be rendered with respect to each function's functionality.
- /media/ - This contains the documents that are temporarily uploaded to process before being deleted.
- Scrapy Tool functionality:
    - Initially the scrapy project needs to be created. Once we have the project in place, a spider needs to be created. A spider is a chunk of python code which determines the content of the web page that needs to be scrapped. The generation of the scrapy project and related files are taken care of by our software.
    - Some of the important files which will be created by running scrapy.
    - Spider Folder: /Amazon_Comments_Scrapper/amazon_reviews_scraping/amazon_reviews_scraping/spiders
    - The above directory will have all the python classes spider/crawlers.The scrapy.py will have the xpath for the reviews and comments of the amazon web page. These data are saved into the json format. These data are passed for sentiment analysis.
- /authapp/ - This contains the templates and functionalities for authentication functionality.
    - /template/ - This folder contains the html pages for login and register
    - urls.py - This file contains the paths to login page and register page
    - views.py - This file provides the logic for login as a current user and register as a new user.

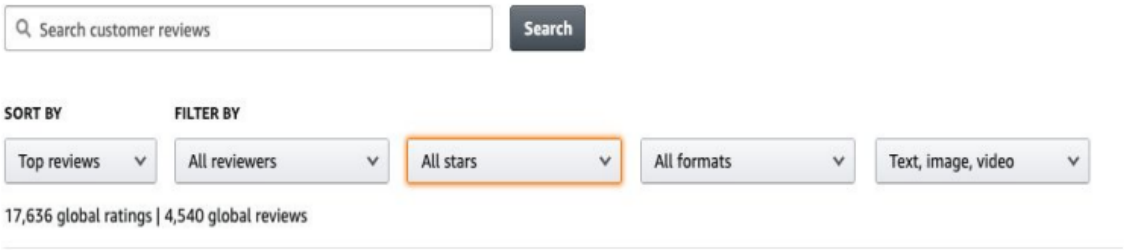
In this software code documentation, we majorly concentrate on the view.py file as it contains the major part of the development work.

Now let us see each function listed in view.py along with the functionalities

- def analysis(request): This renders the homepage of the web url that is localhost:8000
- def input(request): This function renders the home.html page. This html page receives the input of the document to be analysed. Here please remember that the files to be inputted has to be of the following format only.
    - pdf - For Document Analysis
    - .txt - For Document Analysis
    - .wav - For Audio Analysis. Once you upload the document, the documents initially get stored in the media/ folder and are converted into text. Here we have a variable “value” that stores the text in a list of strings as an array. “Value” serves as the input for the sentiment analysis analyser.

- def productanalysis(request):
  This function receives the input as the link of the product reviews that has to be analysed. Here the link to be given in the url input point is specific and is as follows :
    - Go to https://www.amazon.com
    - Select the product that needs to be reviewed.
    - Click on the ratings

- Make sure that all star options is selected.



- Please provide the following http link.

- def textanalysis(request): This function receives the input in textual format from the user and is used for raw text analysis. The text is converted into list of strings and stored in the variable “value”.

- def get_clean_text(text):
  Cleans the text by removing hyperlinks, emojis, special characters, punctuations and Extra white spaces. This is done using regular expression re package. This method also removes stop-words from the text which do not contribute to the meaning of the text.

- def detailed_analysis(result):
  This function takes a list of strings as input and performs cleaning and sentiment analysis using Vader to generate a sentiment dictionary. Then the scores for the strings within the list are added up and divided by the

total score.

- def sentiment_scores(text):
  This function is responsible for leveraging SentimentIntensityAnalyzer() which is a component of vaderSentiment, the core tool used for text sentiment analysis in our application.

- def detailed_analysis_sentence(text):
  This function takes in a paragraph in string format and uses SentimentIntensityAnalyzer to get the polarity score of the complete sentence. Since we are using the complete sentence for analysis and not individual words separately we get a more accurate result for the emotions behind the review or post.

- def scrapNews(topicName, nums, jsonOutput):
  This function searches google news with the query ‘topicName’ and finds the most relevant news articles. The number of articles to fetch is specified by ‘nums’. The news articles are then parsed and their summary is returned as a list. If the ‘jsonOutput’ flag is true then this list of summaries is dumped into the ‘news.json’ file.

- def getNewsResults(query, nums):
  This function searches google news for the given query for the most relevant news articles. The number of articles to fetch is specified by ‘nums’. The ‘urls’ of these news articles are then returned as a list.

FUNCTIONAL TESTING :

For functional testing of the code, a separate testing script is written to validate the output generated by the CELT.
The scripts are written in :
SE_Project1/sentimental_analysis/realworld/functional_tesing.py

The original code has been copied into the functional_testing.py and scrapy the environment has been recreated with another file amazon_test.py. All the HTML
and CSS dependencies have been removed in order to have an independent execution of the functional test. Initially there are three test cases that have been used for validation of the software.

Test case 1: Here, a pre configured string has been parsed to the text analysis function which outputs the sentimental analysis values in the dictionary format which is further verified with our pre-configured values.

Test case 2: The amazon product feedback link that has to be analysed is initially stored in the productanalysis.txt file. The path of the productanalysis.txt file is parsed into the product analysis function which invokes the scrapy execution and provides us with a review.json file. From review.json file we collect the comments of all the feedback and pass it to the CELT. The output from the CELT is compared with the precomputed values and verified.

Test case 3: We initially store the file that has to be analysed in a particular folder and copy that path and parse it into the input function. This then checks for the extension name of the file and processes it to CELT irrespective of it being a PDF, txt or wav file. The output from the CELT is again verified with the pre-computed values.

We have added additional unit test cases increasing the coverage of individual utility functions, application authentication flow and new analyzer with the updated scraper.

We focus not only on adding new functionality to the application but detecting and improving the test coverage of the application in the previous version. The development should have test cases that cover all possible scenarios as an isolated code block but also needs to have many cases integrating with the existing system.

For functional testing for news scraping, we put the test file under /sentimental_analysis/realworld/_test_/test_newsScraper.py. There are in all 20 test cases which includes the following situations:

1. Test if getNewsResults is able to search a single query on google news.
2. Test if getNewsResults is able to parse the url to the article for one result.
3. Test if the single query url parsed by getNewsResults is valid.
4. Verify that the url is pointing to a relevant article as per the search query given to getNewsResults .
5. Test if getNewsResults is able to search for hundred queries on google news.
6. For multiple queries given to getNewsResults verify that all urls returned by getNewsResults are valid.
7. Check if the urls returned by getNewsResults lead to articles relevant to the search query.
8. Test if the argument requirements to the method getNewsResults are meet
9. Test if scrapeNews is able to search google news for article summary for a single query.
10. Test if scrapeNews is able to parse the article text from the website for one result.
11. Test if scrapeNews is able to parse the article text from the website for hundred results.
12. Test if the argument requirements to the method scrapeNews are meet
13. Verify that the summary generated is valid
14. Test if the summaries generated for multiple queries doesn't have default web-scraping blocked messages and contain actual information.
15. Test that the summaries contain information relevant to the search query.
16. Test if the summaries returned from scrapeNews match the provided search count.
17. Test if the summaries are getting dumped into 'news.json' file
18. Test if the summaries are getting correctly loaded from the 'news.json' file

19. Test that the number of summaries parsed from ‘news.json’ files match the count provided to scrapeNews.
20. Test if the summaries retrieved from ‘news.json’ contain information relevant to the search query.

For functional testing for authentication including login and registration, we put the test file under /sentimental_analysis/realworld/_test_/tests_authentications. There are in all 20 test cases which includes the following situations:

1. Test if the registration page is accessible.
2. Test if the login page is accessible.
3. Test if the register page uses the correct template.
4. Test if the login page uses the correct template.
5. Test if the user can successfully register.
6. Test if the user can't register with the wrong password.
7. Test if registration fails with a short password.
8. Test if registration fails with a password similar to username.
9. Test if registration fails with an existing username.
10. Test if registration fails with an empty username.
11. Test if registration fails with an empty password.
12. Test if a user can log in with valid credentials.
13. Test if login fails with an incorrect password.
14. Test if login fails with an incorrect username.
15. Test if login fails with an empty username.
16. Test if login fails with an empty password.
17. Test if a logged-in user can log out.
18. Test if login is case-sensitive for username.
19. Test if a logged-out user is redirected to the login page.
20. Test if the homepage is not accessible if not logged in.