Developer’s Manual

NLP Recommendation System

Professor Carlo Lipizzi
Kunal Batra
Hojat Behrooz
Arya Guddemane Vishwakumar

Table of Contents

How to start the project

- To start the project, download the project from Github and save it in a new folder.
- Open a terminal at this folder location and install all required libraries listed in requirements.txt

```
pip install -r requirements.txt
```

- Open a terminal at this folder and type the command:

```
python app.py
```

- This starts the project along with a localhost link displayed on the terminal. Copy and paste that link on to your browser. If this doesn't work, try changing the port number to available port in your system. The link looks like below

```
http://localhost:8000/
```

MongoDB needs to be installed before starting the project. Edit the connection string in the file **app.py** at line number 48 and 52.

To first create a user, login as admin using **admin** as username and **admin** as password.

Create a new user and logout from the admin's page.

Login using new user credentials.

Directory

The structure of the project directory is as follows:

```
main_folder
        |______ Carlo_ngrams_tool
        |               |_______________ chunking_bforce_plus_HB_v2.py
        |               |_______________ chunking_bforce_plus_HB.py
        |               |_______________ chunking_bforce_plus_space_add.py
        |               |_______________ chunking_bforce_plus.py
        |               |_______________ stopwords_en.txt
        |
        |______ clustering
        |               |_______________ stores cluster ranks of uploaded documents (.CSV)
        |
        |______ components
        |
        |______ static
        |           |______ css
        |           |           |__________ index.css
        |           |           |__________ main.css
        |           |           |__________ normalize.css
        |           |           |__________ styles.css
        |           |           |__________ template.css
        |.          |______ images
        |           |______ js
        |           |           |__________ script.js
        |           |______ user_files(files and folders are created when files are uploaded/processed)
        |
        |______ templates (expanded in next page)
        |
        |______ app.py
        |______ clustering.py
        |______ models.py
        |______ pdf2text_tika.py
        |______ Recommendation_system_v11.py
        |______ visualization.py
        |______ word2vec_v1.model
        |______ stopwords_en.txt
```

Templates folder (expanded)

```
./templates/
        |____ adminbase.html
        |____ adminlogin.html
        |____ base.html
        |____ changepassword.html
        |____ chpassbyadmin.html
        |____ error.html
        |____ home.html
        |____ login.html
        |____ loginbase.html
        |____ output.html
        |____ showfiles.html
        |____ signup.html
        |____ template.html
        |____ userlist.html
```

Terminologies

Session: A session is created when either an admin or a user logs in. Session holds information across the website from login to log out. The session is cleared when the user/admin logs out. A session is a dictionary, and the keys are

- **'csrf_token':** Alpha numeric value
- **'curr_file':** List of list – contains folder name (string of digits) and name of files that were recently uploaded
- **'files_saved':** List – contains filenames that are saved while uploading
- **'group':** Dictionary – key: folder name, value: unique id
- **'logged_in':** Boolen value (True if logged in, else false)
- **'processed':** List – contains file names that are processed
- **'results_saved':** List – contains names of files who's results are saved
- **'save_file':** Boolen value (True if user checks the option "Save File" else False)
- **'save_result':** Boolen value (True if user checks the option "Save Results" else False)
- **'user':** Dictionary – contains currently logged in information of the user
- **'userfolder'**: string – name of the user folder created at login. All files and folders are created at this location (see project directory)

Page Tab ID: This is a counter to track the number of tabs opened for this website. It is initialized to 1 and should always be maintained at 1. When the user copies and pastes the link in another tab, this count increments which results in "restricted access" webpage. This prevents the user from using multiple tabs when the user is logged in. Multiple tabs with different users logged are allowed.

Tool and Technologies

- Python
- PyMongo, MongoDB
- Flask
- HTML
- CSS
- JavaScript

User Interface

Template HTML files are present which are being used by other files. These template files are as follows:

- adminbase.html
- base.html
- loginbase.html
- template.html

Login Page

The username and password are used to check if the user exists. If not, then a message is displayed. Admin can create a user from the admin's page.

The forms can be found in **login.html** (for user), **adminlogin.html** (for admin). The submitted forms are handled by **script.js** and is searched in the database by flask functions.

Sign Up

The admin can create a new user from admin's page. To go to the admin's page, a link is provided in the login page.

Homepage

The HTML file associated with homepage is **home.html**

The homepage allows users to upload individual or multiple files. The dropzone allows a user to Drag 'n' Drop or browse the files from local storage.

The options "Save Files", and "Save Results", are checked by default. The "save_file" and "save_result" keys in **session** are assigned to **True** when the user clicks on upload button. If unchecked, it is assigned to **False.** The purpose of this is to determine if the browsed file/s and result/s to be saved to the database.

The **Upload** button is only used to save the file to the database or to the local storage. By default, the files are saved in local storage temporarily until the user signs out.

The **Submit** button takes all the files that were uploaded just before clicking this button. **Session** stores the file information in ‘cur_file’ and hence can be used to retrieve those files from local storage for processing.

Once the results are out, they are displayed in **Results** section in a form of a table. This is called **DataTables**. Column names of the table are Filename, View Result, Download. A group is created based on how the files were uploaded (individual/multiple). The **group** in session holds this value to create groups based on unique ids created.

Values and links in each cell

There are multiple scenarios where the user choses to check or uncheck **Save Files** and **Save Results** options before uploading the files. To make it user-friendly, a flexible option to upload files and results are provided from the DataTable.

Case 1:

☑ Save Files ☑ Save Results

No action required from DataTable

Case 2:

☑ Save Files ☐ Save Results

The file is already saved to the database and if the file is processed, a link “upload” is provided to save the result to the database.

Case 3:

☐ Save Files ☑ Save Results

The file is not saved but only the result is saved, then this becomes unavailable to the user as the original file is not found in the database.

Case 4:

☐ Save Files ☐ Save Results

If both these options are unchecked, then a link is provided to upload the file and result to the database. First the file needs to be uploaded and then the result. The link to upload results is disabled unless the original file is uploaded to the database.

Case 5:

If the user uploads the documents but does not process it (this happens when the user refreshes the page after uploading the documents, i.e., before clicking “Submit” button), a “Process File” link is provided in the “View Results” column against the name of the file. This takes the file from the local storage and processes it. The results will be available to view, but it requires to be uploaded to the database using the link.

Results

The files are processed and shown in result section. If multiple files are uploaded, then they are displayed in groups in the datatable. If individual file is uploaded, then that group contains a single file.

For example:

Multiple files: File1.pdf, File2.pdf, File3.pdf

These three files are considered as a group and the group number is assigned accordingly.

Individual file: File1.pdf

This file belongs to another group.

Clicking on **view results** in the datatable of a group that has multiple files results in 2 images and an embedded pdf view. The first image is a 3D chart of "Benchmarks clusters similarity to input documents", the second image is the 2D vertical chart of class similarity of each file. The embedded pdf view is the highlighted document of the original file.

Clicking on different "View Result" link of the same group results in changing the second image and the embedded pdf view.

Clicking on different "View Result" link of different group results in changing all images and the pdf view.

The class (in second image) can be changed from the dropdown menu by clicking on "Class" button.

These results can be downloaded in the ZIP format. This zip folder contains "Benchmark clusters similarity to input documents", multiple images of class similarity and multiple highlighted pdf documents (if a group contains multiple files) or one highlighted pdf document (if a group has a single file).

The result in the page "showfiles.html" works the same as it does in "home.html" except that the files are retrieved from the database. (Further explained)

The result section can be cleared by clicking on "Clear Session" or "Sign out" in the homepage. Clicking these will delete all previously uploaded files from the local storage.

Change Password

Clicking on **Change Password** link opens another page to change the password for the user. The file associated with this is **changepassword.html.** The form is handled by script.js

Clear Session

Clicking on **Clear Session** link deletes all the values in **session.**

Files on User Page

The user can view all files previously uploaded in a different page. To navigate into this page, a “Show my files” link is provided in the homepage, just below the dropzone.

The files here are retrieved from the database. Please check the code in “showfile.html” file. Additional options are “Delete File” and “Archive”. Delete file removes that file from the database. Archive makes the file invisible to the user, but can be unarchived by the admin.

Admin’s Page

The admin has all rights over user. When the admin creates a new user, the details are stored in database and used when the user tries to login. Another link “Show Users” to view all the users is provided. The users are displayed in the datatable with fields such as Name, Email, Files, Password, and Remove User.

The columns Files let the admin view all files of a particular user. The password column lets the admin to change the password of the user and remove user column lets the admin delete the user from the database.

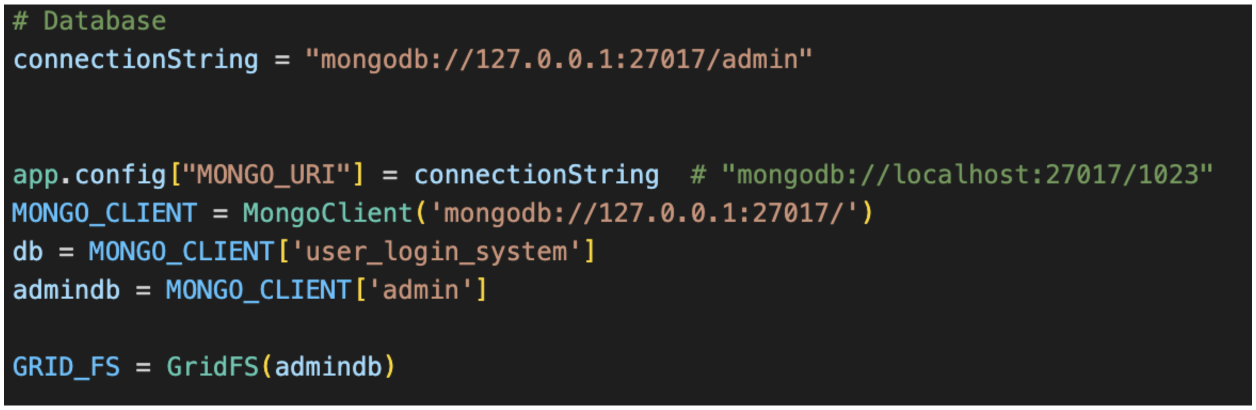
Where and how are users created and stored?

An user can be created by the admin and the details required are Name, Email and Password.

These details are stored in “users” collection in “user_login_system” document in MongoDB. The database name that is assigned in the code can be found in “app.py” file.

Two databases are created, one to store user details, another one to store files and results.

Below image shows the code in “app.py”



```
# Database
connectionString = "mongodb://127.0.0.1:27017/admin"


app.config["MONGO_URI"] = connectionString  # "mongodb://localhost:27017/1023"
MONGO_CLIENT = MongoClient('mongodb://127.0.0.1:27017/')
db = MONGO_CLIENT['user_login_system']
admindb = MONGO_CLIENT['admin']

GRID_FS = GridFS(admindb)
```

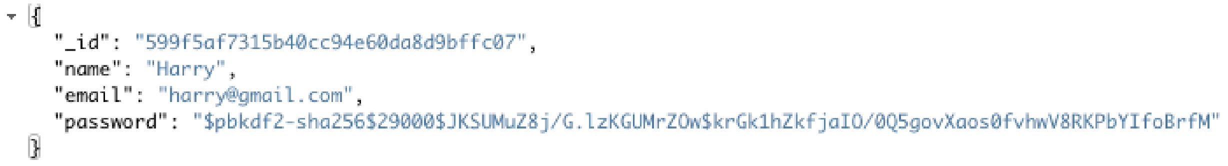
The database “db” holds user information and “admindb” holds files and results

The collection present in each database is as follows:

“db” = users

“admin” = “fs.files”, “fs.results”

The user information is stored as follows:



```
{
  "_id": "599f5af7315b40cc94e60da8d9bffc07",
  "name": "Harry",
  "email": "harry@gmail.com",
  "password": "$pbkdf2-sha256$29000$JKSUMuZ8j/G.lzKGUMrZOw$krGk1hZkfjaIO/0Q5govXaos0fvhwV8RKPbYIfoBrfM"
}
```

(screenshot from MongoDB compass)

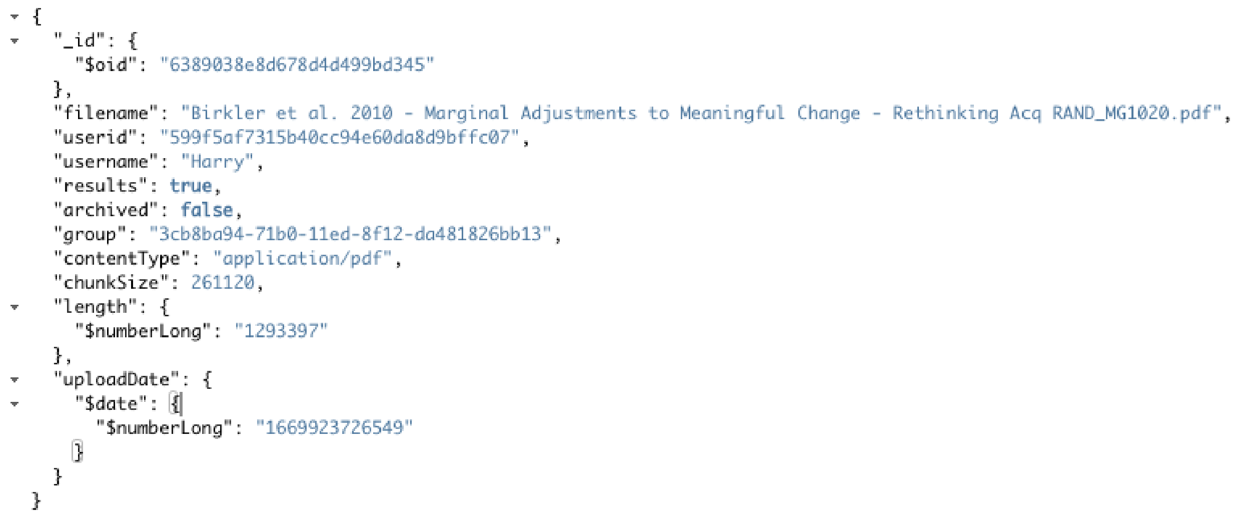
The information is stored in a JSON format which is easy to retrieve. The user here is “Harry”, email is harry@gmail.com, and password is encrypted.

To get information of the user, it is important to use appropriate database i.e., “db”.

Where and how files and results stored and retrieved?

The files are stored in “admindb” database. The way MongoDB saves is different than other storage systems. The files information is stored in “fs.files” collection and contents of that file is stored in “fs.chunks” collection. “fs.chunks” contains multiple objects that is associated with a single file, whereas “fs.files” contain a single object for a file.

The below screenshot explains it:

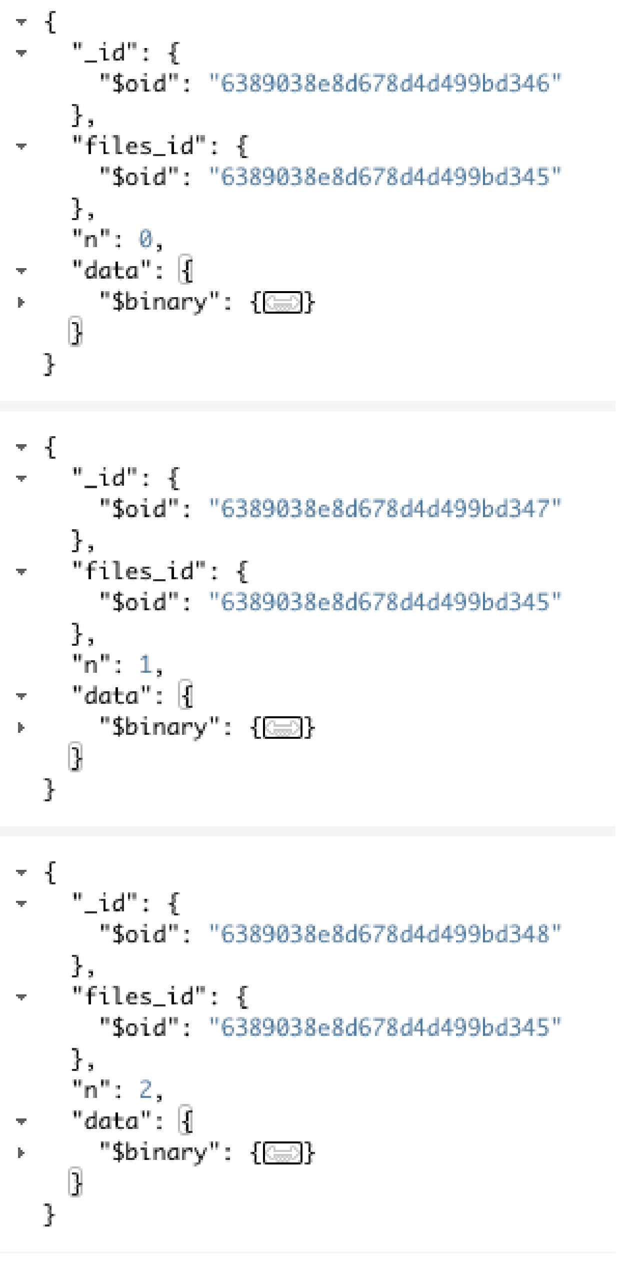


```
{
  "_id": {
    "$oid": "6389038e8d678d4d499bd345"
  },
  "filename": "Birkler et al. 2010 - Marginal Adjustments to Meaningful Change - Rethinking Acq RAND_MG1020.pdf",
  "userid": "599f5af7315b40cc94e60da8d9bffc07",
  "username": "Harry",
  "results": true,
  "archived": false,
  "group": "3cb8ba94-71b0-11ed-8f12-da481826bb13",
  "contentType": "application/pdf",
  "chunkSize": 261120,
  "length": {
    "$numberLong": "1293397"
  },
  "uploadDate": {
    "$date": {
      "$numberLong": "1669923726549"
    }
  }
}
```

(screenshot from “fs.files”)

“_id” is a unique id created when a file is uploaded to the database. This id is the “files.id” in “fs.chunks”.

“userid” is the unique id of the user that is created when the admin creates a new user. (Refer the user information screenshot)
“group” is a unique id that is created when the user clicks “Upload” button on homepage. This ID is same to all files that are uploaded in a group.



```
{
  "_id": {
    "$oid": "6389038e8d678d4d499bd346"
  },
  "files_id": {
    "$oid": "6389038e8d678d4d499bd345"
  },
  "n": 0,
  "data": {
    "$binary": {}
  }
}
```

```
{
  "_id": {
    "$oid": "6389038e8d678d4d499bd347"
  },
  "files_id": {
    "$oid": "6389038e8d678d4d499bd345"
  },
  "n": 1,
  "data": {
    "$binary": {}
  }
}
```

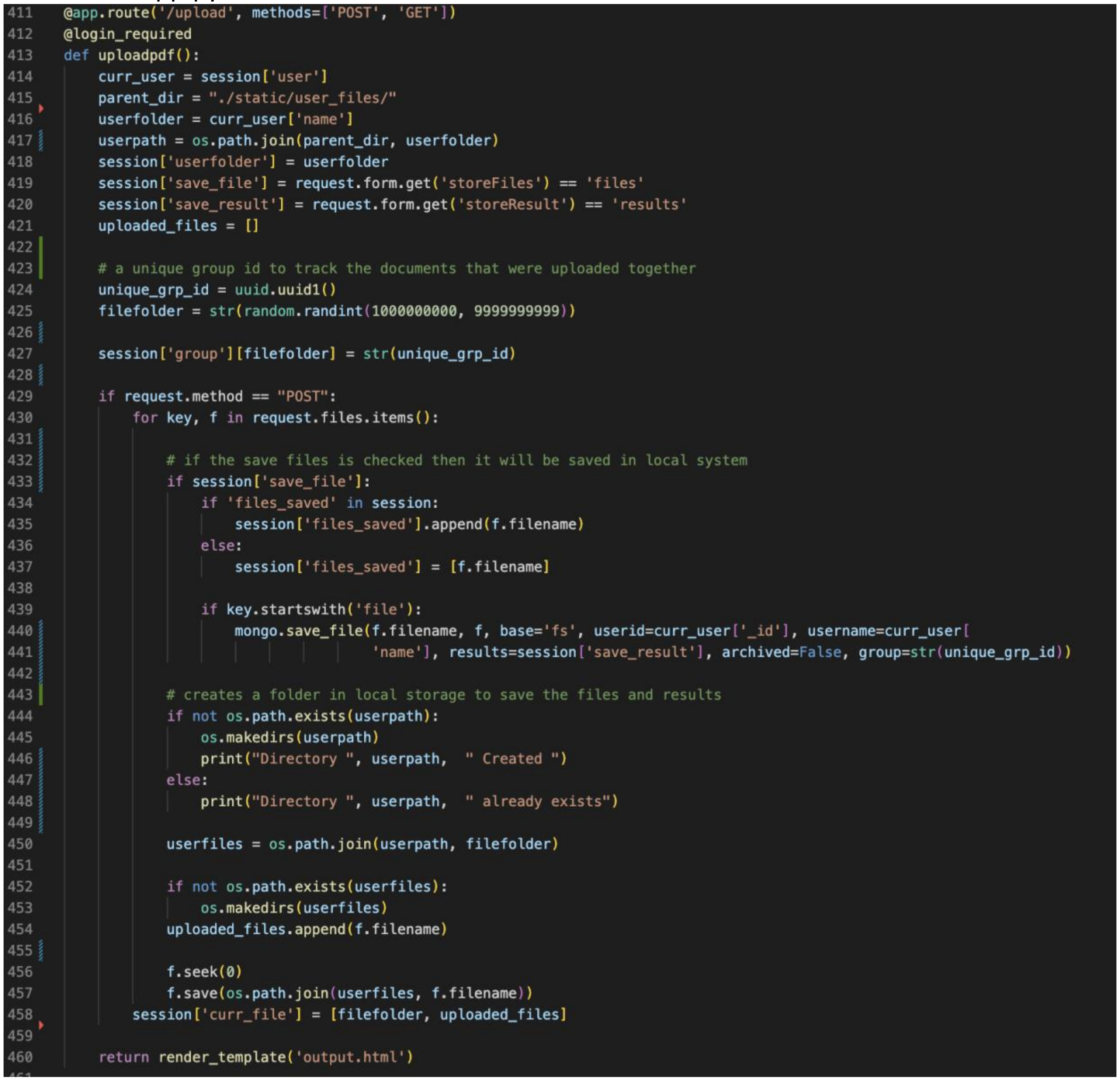
```
{
  "_id": {
    "$oid": "6389038e8d678d4d499bd348"
  },
  "files_id": {
    "$oid": "6389038e8d678d4d499bd345"
  },
  "n": 2,
  "data": {
    "$binary": {}
  }
}
```

(screenshot from “fs.chunks”)

“files_id” in “fs.chunks” is same as “_id” in “fs.files”. The above image shows that one file is divided into 3 chunks which can be determined by field “n”. The data is stored in binary format.

Code to upload documents

The documents are uploaded to local storage and MongoDB simultaneously. The decision to save files to database is determined from the “save_result” and “save_files” from **session.** See line 433 in “app.py”



```
411  @app.route('/upload', methods=['POST', 'GET'])
412  @login_required
413  def uploadpdf():
414      curr_user = session['user']
415      parent_dir = "./static/user_files/"
416      userfolder = curr_user['name']
417      userpath = os.path.join(parent_dir, userfolder)
418      session['userfolder'] = userfolder
419      session['save_file'] = request.form.get('storeFiles') == 'files'
420      session['save_result'] = request.form.get('storeResult') == 'results'
421      uploaded_files = []
422
423      # a unique group id to track the documents that were uploaded together
424      unique_grp_id = uuid.uuid1()
425      filefolder = str(random.randint(1000000000, 9999999999))
426
427      session['group'][filefolder] = str(unique_grp_id)
428
429      if request.method == "POST":
430          for key, f in request.files.items():
431
432              # if the save files is checked then it will be saved in local system
433              if session['save_file']:
434                  if 'files_saved' in session:
435                      session['files_saved'].append(f.filename)
436                  else:
437                      session['files_saved'] = [f.filename]
438
439                  if key.startswith('file'):
440                      mongo.save_file(f.filename, f, base='fs', userid=curr_user['_id'], username=curr_user[
441                                      'name'], results=session['save_result'], archived=False, group=str(unique_grp_id))
442
443              # creates a folder in local storage to save the files and results
444              if not os.path.exists(userpath):
445                  os.makedirs(userpath)
446                  print("Directory ", userpath,  " Created ")
447              else:
448                  print("Directory ", userpath,  " already exists")
449
450              userfiles = os.path.join(userpath, filefolder)
451
452              if not os.path.exists(userfiles):
453                  os.makedirs(userfiles)
454              uploaded_files.append(f.filename)
455
456              f.seek(0)
457              f.save(os.path.join(userfiles, f.filename))
458          session['curr_file'] = [filefolder, uploaded_files]
459
460      return render_template('output.html')
```

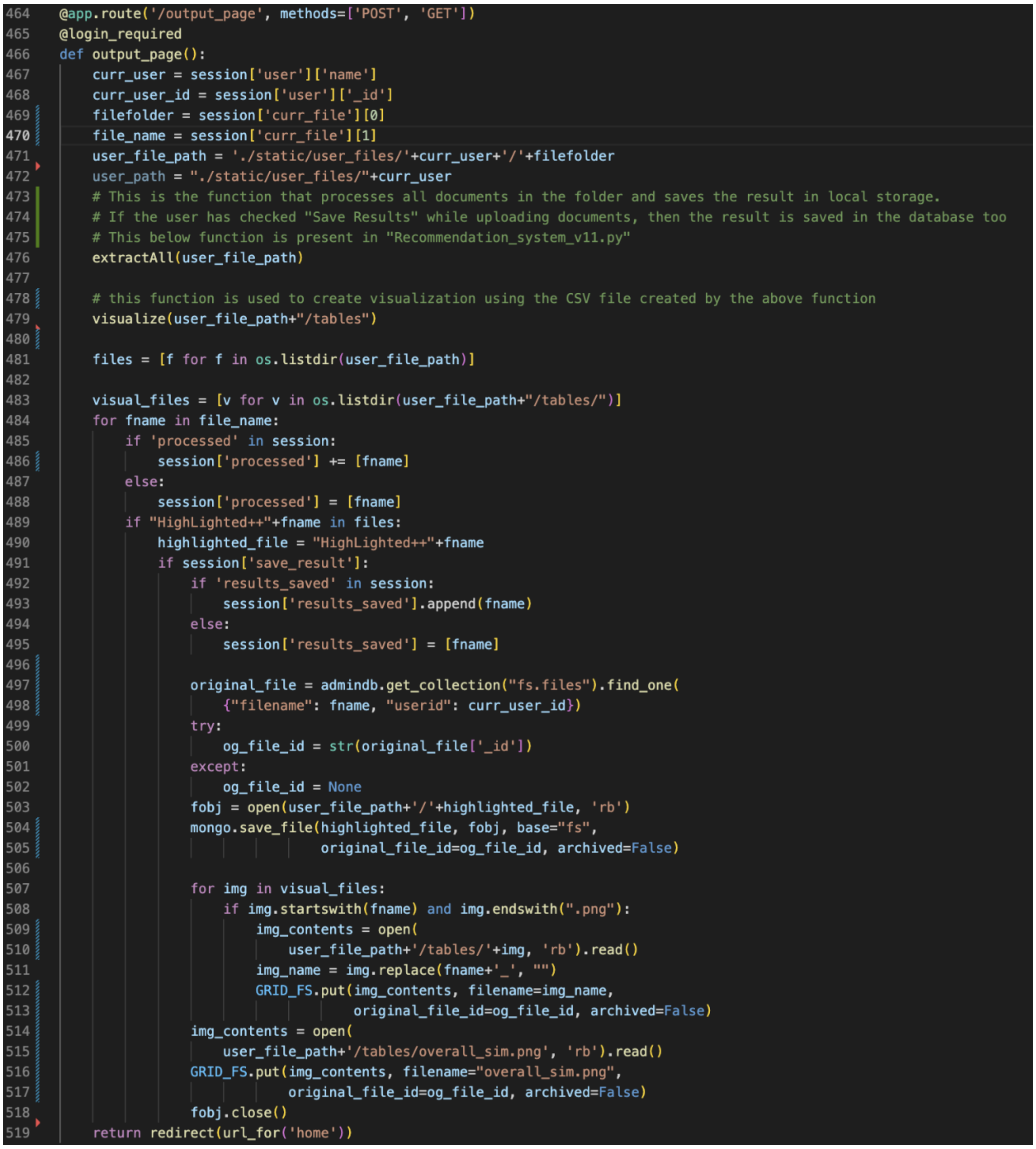
The file is uploaded using the following: (see line 440)

```
    mongo.save_file(f.filename, f, base='fs', userid=curr_user['_id'], username=curr_user['name'], results=session['save_result'], archived=False, group=str(unique_grp_id))
```

Line 444 onwards: Checks if the folder is created, if not it creates a folder and uploads all documents and results into that folder.

Starting point to process document

Once the document is uploaded, the user clicks on “Submit” button on homepage. Upon submitting, a function “output_page()” in the app.py is called.



```python
@app.route('/output_page', methods=['POST', 'GET'])
@login_required
def output_page():
    curr_user = session['user']['name']
    curr_user_id = session['user']['_id']
    filefolder = session['curr_file'][0]
    file_name = session['curr_file'][1]
    user_file_path = './static/user_files/'+curr_user+'/'+filefolder
    user_path = "./static/user_files/"+curr_user
    # This is the function that processes all documents in the folder and saves the result in local storage.
    # If the user has checked "Save Results" while uploading documents, then the result is saved in the database too
    # This below function is present in "Recommendation_system_v11.py"
    extractAll(user_file_path)

    # this function is used to create visualization using the CSV file created by the above function
    visualize(user_file_path+"/tables")

    files = [f for f in os.listdir(user_file_path)]

    visual_files = [v for v in os.listdir(user_file_path+"/tables/")]
    for fname in file_name:
        if 'processed' in session:
            session['processed'] += [fname]
        else:
            session['processed'] = [fname]
        if "HighLighted++"+fname in files:
            highlighted_file = "HighLighted++"+fname
            if session['save_result']:
                if 'results_saved' in session:
                    session['results_saved'].append(fname)
                else:
                    session['results_saved'] = [fname]

                original_file = admindb.get_collection("fs.files").find_one(
                    {"filename": fname, "userid": curr_user_id})
                try:
                    og_file_id = str(original_file['_id'])
                except:
                    og_file_id = None
                fobj = open(user_file_path+'/'+highlighted_file, 'rb')
                mongo.save_file(highlighted_file, fobj, base="fs",
                                original_file_id=og_file_id, archived=False)

                for img in visual_files:
                    if img.startswith(fname) and img.endswith(".png"):
                        img_contents = open(
                            user_file_path+'/tables/'+img, 'rb').read()
                        img_name = img.replace(fname+'_', "")
                        GRID_FS.put(img_contents, filename=img_name,
                                    original_file_id=og_file_id, archived=False)
                img_contents = open(
                    user_file_path+'/tables/overall_sim.png', 'rb').read()
                GRID_FS.put(img_contents, filename="overall_sim.png",
                            original_file_id=og_file_id, archived=False)
                fobj.close()
    return redirect(url_for('home'))
```

The above function is called from “home.html”. The function “extractAll(user_file_path”)” is called from “Recommendation_system_v11.py”. The parameter is the path of the original file located in the local storage. “Recommendation_system_v11.py” processes the document and saves the highlighted files in the local storage. Furthermore, “visualize()” function is called to create visualization of the results. The data that is used to create visualization is stored in local storage in CSV format. These results are stored in database based on the value of “save_file” from **session**. Line 491.

The “og_file_id” is the original file id that is used to determine the resultant file associated with the original file.

The images that are generated are also stored into the database.

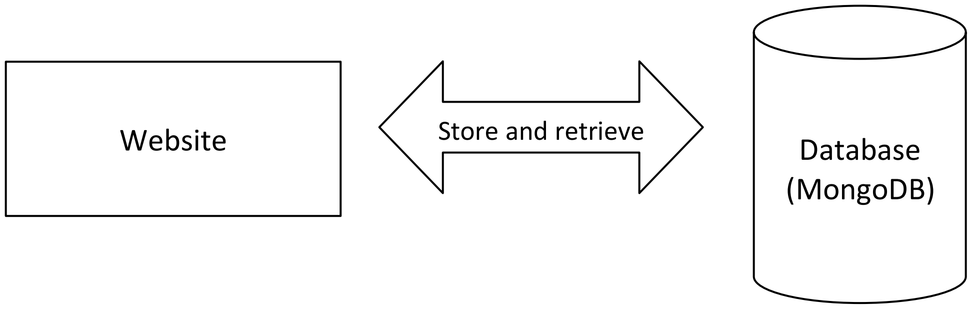
The retrieving of each file uses the “_id”, “filename”, “username” and “user_id” .
The retrieving of result files is determined by using “original_file_id” which is same as “_id” of the original file.

Functions such as

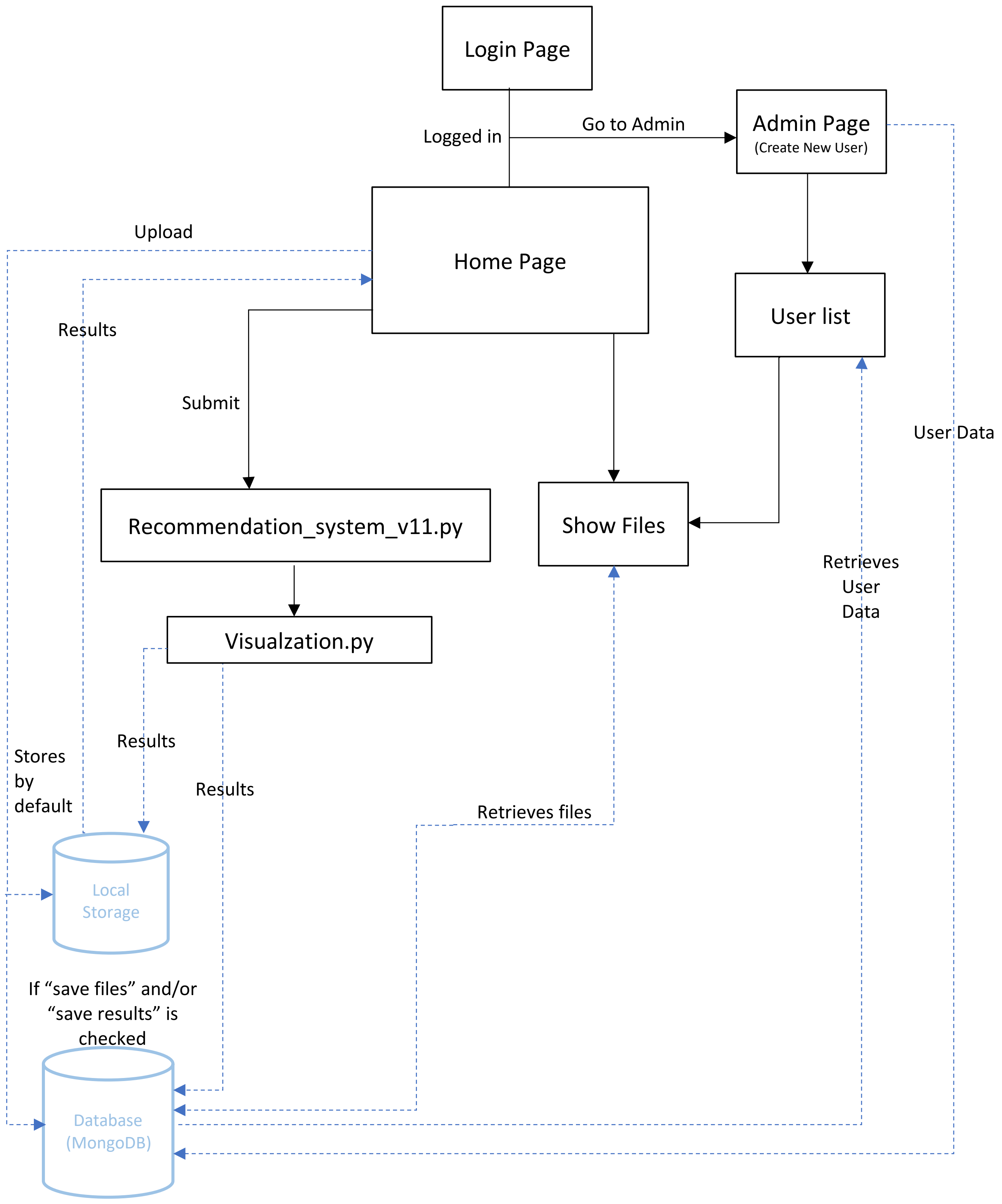
- start_session()
- start_admin_session()
- password_check()
- signup()
- signout()
- login()
- changepassword()
- removeUser()
- deletefile()
- showFiles()
- showFilesbyAdmin()
- adminlogin()
- userlist()
- getallfiles()

are present in **models.py** file, which are all called from **app.py.** The models.py file creates model for each functionality such as getting all files, remove the user, deleting a file etc.

A simple illustration of the follow of job



System Diagram


Login Page
Logged in
Go to Admin
Admin Page
(Create New User)
Upload
Home Page
Results
User list
Submit
User Data
Recommendation_system_v11.py
Show Files
Retrieves
User
Data
Visualzation.py
Results
Stores
by
default
Results
Retrieves files
Local
Storage
If "save files" and/or
"save results" is
checked
Database
(MongoDB)