Website.py

Contains all the functions needed to render the website.

Download as .pdf!

Functions:

run

function **run (**kwargs,): -->**

It's called by main.py to start the website.

Parameters:

****kwargs** : *dict*

The arguments are directly passed to Flask when he runs the website.

homepage

function **homepage (): -->**

The function renders the homepage in which the project is explained briefly.

documentation

function **documentation (*file*,): -->**

Return the webpages with the documentation of the project.

Every webpage of the docs has it's name added after "/documentation/"
This means that you can add all the webpages that you want and you won't need to write a single line of code, just add the html files to "templates/documentation/" and the .json files with the documentation to the folder containing all the .json written in "settings.py".
It uses getDocumentation() from "mediator.py" to get the documentation from the .json files.

Any eventual OSError exception is purposefully caught here and not in mediator.py to show the error as a popup with flash().

Parameters:

file : *str*

Is the name of the documentation webpage. For example if file="settings" then the url will be "documentation/setting".

functions

function **functions (): -->**

Returns the webpage which lets you select the operation you want to do with the datasets.

download

function **download (): --> download**

Allows to download the table computed by an operation as tsv file.

Steps:
Step 1) Get "name_file" from the page that requested the download.
Step 2) Check if "name_file" is None. If it is it means that the previous page did not return any value, or does not have any button named "name_file".
Step 3) If "name_file" is not None, it computes the complete path by joining the current path of execution of the program, thus the main directory of the program, and "name_file". Then it checks if it's a file and if it's exists. If it does it means the previous page requested a file to download, and it downloads it, otherwise it means "name_file" is the key of the table in the cache and also the name that will have the table once it'll be converted to .tsv.
Step 3.1) In the latter case, it requests the table from the cache. If the data retrieved is None it means that there was not any table in the cache, thus it redirects to the previous page and tells the user through a notification that he needs to reload the page as the table probably expired from the cache.
Step 4) Extract from the dicitionary "data_to_save" the rows and the labels of the table.
Step 5) A csv.writer is instantiated. It needs StringIO module to instantiate a file-object.
Step 6) Write as the first row the labels of the columns, then write all the rows
Step 7) Make a response which allows the .tsv file to be downloaded
Step 8) Set some information of the file that will be downloaded like its name and filetype.

```
# Step 4)
rows = data_to_save['rows']
labels = [data_to_save['labels']]

# Step 5)
si = StringIO()
cw = csv.writer(si, delimiter='\t')

# Step 6)
cw.writerows(labels)
cw.writerows(rows)

# Step 7)
output = make_response(si.getvalue())

# Step 8)
output.headers["Content-Disposition"] = f"attachment; filename={name_file}.tsv"
output.headers["Content-type"] = "text/tsv"
return output
```

To know more about the concept behind, please refer to "Download a table" section in the Project Overview.

browseGeneDataset

function **browseGeneDataset (): -->**

Renders the webpage which lets you go through gene dataset. To do the pagination it uses Pagination() from flask-paginate.

browseDiseasesDataset

function **browseDiseasesDataset (): -->**

Renders the webpage which lets you go through gene dataset. To do the pagination it uses Pagination() from flask-paginate.

info

function **info (): -->**

Renders the webpage which presents all the information about the datasets and a preview of heads and tails.
It uses getInfo() from "mediator.py" to get the results.

distinctGenes

function **distinctGenes (): -->**

Renders the webpage which presents all the unique distinct genes in the gene dataset.
It uses getDistinctGenes() from "mediator.py" to get the results.

distinctDiseases

function **distinctDiseases (): -->**

Renders the webpage which presents all the unique distinct diseases in the disease dataset.
It uses getDistinctDisease() from "mediator.py" to get the results.

geneEvidences

function **geneEvidences (): -->**

The first time the user access "geneEvidences" it is requested with 'GET' method. Then it returns a webpage which lets the user input a geneSymbol or a geneID. It is then submitted back to "geneEvidences" but with 'POST' method. Now it returns a webpage which lists all the evidences in literature of the relation between the gene and COVID-19.
It uses getGeneEvidences() from "mediator.py" to get the results.

diseaseEvidences

function **diseaseEvidences (): -->**

The first time the user access "diseaseEvidences" it is requested with 'GET' method. Then it returns a webpage which lets the user input a diseaseID or a diseaseName. It is then submitted back to "diseaseEvidences" but with 'POST' method. Now it returns a webpage which lists all the evidences in literature of the relation between the disease and COVID-19.
It uses getDiseaseEvidences() from "mediator.py" to get the results.

correlation

function **correlation (): -->**

The webpage lists the correlations between genes and diseases.

It allows th user to customize the results, he can decide the number of correlations to show ("rows") and the minimum number of occurrences a correlation needs to have to be shown ("occurrences").

If "occurrence" is given and the user hasn't written anything in "rows" then it sets "rows" to 0 which means that all rows will be returned. Also, if the user wants 50 rows, but the rows which meet the occurrence requirement are 30, only 30 rows will be returned.
It uses getCorrelation() from "mediator.py" to get the results.

diseasesRelatedToGene

function **diseasesRelatedToGene (): -->**

The first time the user access "diseasesRelatedToGene" it is requested with 'GET' method. Then it returns a webpage which lets the user input a geneSymbol or a geneID. It is then submitted back to "diseasesRelatedToGene" but with 'POST' method. Now it returns a webpage which lists all the diseases related to the gene found in literature.
It uses getDiseasesRelatedToGene() from "mediator.py" to get the results.

genesRelatedToDisease

function **genesRelatedToDisease (): -->**

The first time the user access "genesRelatedToDisease" it is requested with 'GET' method. Then it returns a webpage which lets the user input a diseaseName or a diseaseID. It is then submitted back to "genesRelatedToDisease" but with 'POST' method. Now it returns a webpage which lists all the genes related to the disease found in literature.
It uses getGenesRelatedToDisease() from "mediator.py" to get the results.