Exercise 08 - Front-End Part 2: Adding Functionality

Deadline

December 21, 2020 - 12:00/noon (UTC+1:00)

Submission Guidelines

1. Clone your repository to your local workstation/computer if you have not done so
2. Structure and work on your submission.
   - Create a directory (folder) for the exercise sheet, e.g. name the directory ”exercise08” for Exercise 08.
   - Place all relevant files/scripts in this newly made directory
   - Do not include the `.git` folder!
3. Commit your work to the git repository.
4. Create a git tag.
   - Tag name must be equivalent to ”SubmissionExerciseSheet08”.
   - Tag can be made via the command line or using the GitLab GUI

Grading (10 pts):

Task	0	1	2	3	4	5
Points	1	1	3	1	1 (+1)	3

Using Previous Homework

This exercise requires you to build on the work done for Exercise07. If you did not do or finish Exercise07, or do not wish to use your work, you may use the provided PLAB Package and/or Web Application in the sample_files folder.

Requirements

- Run `pip install flask` (it is recommended that you work in a virtual environment!)

Disclaimer

In this exercise, there will be fewer HTML examples given and you will be expected to find solutions in your own way. There will be recommendations in certain tasks for how to accomplish it, but you are allowed to be as creative as you wish in getting your web application to do what is required below. Style and aesthetics are not important so long as your application contains the functionalities listed in each task.

Tasks

Task 0 - *Upgrade* (1 pt)

- Change all of your `<h4>` tags to `<h3>`
- Add the following to your imports in `run.py`:
  - `from flask import session`
- Add `app.secret_key = "someSecretKey"` directly beneath `app = Flask(__name__)` in `run.py`

Task 1 - *Burn It All Down* (1 pt)

It may occur that someone uploads a file by mistake and now it is in the `uploads` folder with no way of removing it. In this task, you will add a feature that will allow the user to delete all of the files they have uploaded.

1. Create a "Clear Contents" button in the "Upload" (`/upload`) page (0.5 pts)
    - In the `upload.html` template, add an additional `<input>` tag to your form with the value "Clear Contents"
        - Have it use the "reset" type
        - At the moment, your "Upload" button probably does not have a `name=` attribute, give it a name and give the same name to your new "Clear Contents" input tag
2. Alter the `upload_file` method so that it checks which button was pressed on the `upload.html` page (0.5 pts)
    - If the "Upload" button is clicked, then it should save the file just like it did before and return the user to the home page
    - If the "Clear Contents" button is pressed:
        - All files in the `UPLOAD_FOLDER` should be deleted
        - It should render the `/upload` page again with a message stating "Uploaded files successfully removed."

Task 2 - *PPI or Node/Edge Lists* (3 pts)

Since we have been working with both PPI files and node/edge lists throughout this semester, it is time to include the ability to choose which type of file the user wants to use in your web application. We will keep this feature relatively simple for the sake of the exercise. Ultimately, you are going to create a feature that allows the user to import specific file(s) and remember this decision for the duration of the **session**.

1. Create a new section at the top of your home page for choosing whether to import an uploaded PPI file or uploaded node/edge lists (1 pt)
    - Create an `<h3>` tag with the header "Choose Your Network"
    - Create 2 radio buttons and a "Submit" button directly under this header
        - You can use another `<form>` tag with `<input>` tags inside of it to create this. The radio options should be of type "radio" and the "Submit" button should be of type "submit"
        - One radio option should be labeled "PPI File" and the other should be "Node/Edge Lists"
2. Design it so that if the user selects the "PPI File" radio option and clicks submit, the file path to the uploaded PPI file is stored in the `session` object, likewise if the "Node/Edge Lists" radio option is selected and they click submit then the uploaded node/edge lists are stored in the `session` object (2 pts)
    - *Note*: The `session` object can be used to store any information you may need. You may also choose to have it store whether "PPI" or "Node/Edge" was selected. Once saved to `session`, this object (and its information) can be accessed by any method
    - For selecting file paths, you can assume (for the purposes of this exercise) that if a file path ends with ".csv" that it is a PPI file and if it ends with ".tsv" it is either a node list or edge list. Alternatively, you may wish to build additional folders for storing and accessing these different types of files
    - Make sure to include checks as to how many file paths are stored by on the file type. If "PPI File" is selected, only one file path should be stored, and only two for "Node/Edge Lists"
        - If the user has uploaded more than one PPI file or more than 2 node/edge lists, have it display a message warning the user of the issue and suggest they clear the contents of the upload folder and upload only **one** PPI file and/or **one** node list / **one** edge list
    - *Hint*: It may be useful to set the radio options to have the same `name=` and different `value=` so you can collect specific information that was `POST`ed
    - If everything works correctly and the correct number of file paths are stored in the `session` object, have it display a message on the homepage (under the "Submit" button) saying "File(s) imported successfully"

Task 3 - *Update Shortest Paths* (1 pt)

- Update your "Shortest Path Tool" to use the imported file path(s) stored in the `session` object for determining the shortest path between 2 given HGNC symbols

Task 4 - *Show Me the Stats!* (1 pt [+1])

Once a file or files are selected for import, we can have the web application display the summary statistics at the top of the page.

1. Modify your web application so that if a PPI file or node/edge lists are imported and saved in the session that a table of the imported graph's summary statistics is displayed underneath the "Choose Your Network" section
    - *Note*: For this web application, disable the "Average Degree Connectivity" statistic in your `network.py` module (may require you to reinstall your package)

2. **BONUS (+1)**: Make it so the table by default is collapsed (i.e. not visible) and users can expand it and see the table if desired

Task 5 - *Create a Graph Image* (3 pts)

If you have made it this far, congratulations, your hard work is to be rewarded.

1. Add the following method to your `run.py` : (1 pt)

```
@app.route('/plot.png', methods=['GET', 'POST'])
def plot_png():
    # Add code here for creating the graph image using your package's code
    plt = # This needs to be the matplotlib.pyplot object created in your code
    img = io.BytesIO()
    plt.savefig(img, format='png')
    plt.close()
    img.seek(0)
    plot_url = base64.b64encode(img.getvalue()).decode('utf8')
    return render_template('plot.html', plot_url=plot_url)
```

- *Note*: You can modify this function however you wish, or build your own if desired so long as the right graph image is made

1. Create a new HTML template file called `plot.html` which consists of the following HTML:

```
<!DOCTYPE html>
<img src=" {{ plot_url }}">
```

3. Add a button (maybe using a `<form>`) with the value "View Graph" that when clicked, will create an image of the graph based on the imported file(s). Make sure that the `action=` is set to use the **route** defined in part 1 of this task.
    - *HINT*: Make this button visible only if the summary statistics table is generated