Exercise 07 - Front-End Part 1: Introduction to Flask and Jinja

Deadline

December 14, 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 ”exercise07” for Exercise 07.
    - 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 ”SubmissionExerciseSheet07”.
    - Tag can be made via the command line or using the GitLab GUI

Grading (10 pts):

Task	1	2	3	4	5	6
Points	0.5	1.5	2	3	3	(+1)

Using Previous Homework

This exercise requires you to build on the work done for Exercise06. If you did not do or finish Exercise06, or do not wish to use your work, you may use the provided PLAB Package in the sample_files folder. This script will be made available on December 09, 2020.

Tutorial Source

The webpage template we are using today is a modified version of the Jinja Template Tutorial. For a more in depth explanation of what is discussed here and how it is used, please see their website.

Requirements

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

Tasks

Task 1 - *Gather Everything* (0.5 pts)

This is an organizational step before you begin building your webpage.

- If you haven't done so already, create your `exercise07` folder in your repository directory

- Copy your package you created in exercise06 to the `exercise07` folder. If you do not have a working/completed package, then copy the `sample_plab2_pkg/` folder from the `sample_files/` folder in this exercise directory
- Copy the `flask_template/` folder in this exercise directory to your `exercise07` folder
- Your `exercise07/` folder show look something like this:

```
exercise07/
├── flask_template
│   ├── run.py
│   └── templates
│       ├── layout.html
│       ├── macros.html
│       └── template.html
└── plab2_package
    ├── plab2
    │   ├── cli.py
    │   ├── __init__.py
    │   ├── network.py
    │   ├── startup.py
    │   └── utils.py
    ├── README.md
    ├── setup.py
    └── tests
        ├── __init__.py
        ├── test_cli.py
        └── test_network.py
```

- If you are working in a new virtual environment, make sure to install flask and your package from exercise06

Test Start

Once you have everything organized in your folder properly, it is time to start your server and try running the base application (webpage). The basic elements for compiling and rendering the web pages are found in the `flask_template/run.py` file. To run the template application simply execute `python run.py` in your terminal.

- By default, the server runs the application at `http://127.0.0.1:5000/`, while python is running `run.py`, you can copy and paste this web address into any browser to see how the web pages are being rendered

Task 2 - *Make It Your Own* (1.5 pts)

Here you will add some links and customize the web page so that it is unique to you.

1. Change the header of the main webpage (where it reads "Flask Super Example") to read "{your name}'s Network Analyzer". For an example of how it should look, see Figure 1 below (0.5 pts)
    - Because Jinja allows HTML templates to be inherited, you will need to find where the headers/footers are first defined and edit them there
2. Add a link to your GitLab profile page in the footer of the webpage. The link text should see "{your name}'s GitLab Profile". (0.5 pts)
3. Remove the following from the layout as they are not needed (0.5 pts):
    - The search bar and it's submit button
    - The "Link" button in the top right
    - The drop down menu in the top right (labeled "Dropdown")

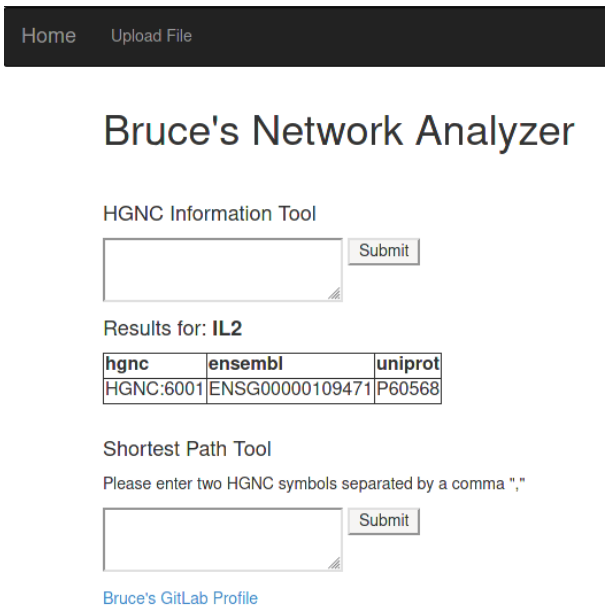

Figure 1: Example of the homepage and HGNC Information Tool output.

Task 3 - *Create an Upload Page* (2 pts)

We want users to be able to upload a PPI file that can be used for the other (soon-to-be-added) website features. In this task, you will create a separate web page whose sole purpose is for uploading a PPI file. In terms of handling data and executing code, all of your methods/modifications for such tasks should be made in `run.py`. Web page design and information display will primarily be done in the corresponding HTML files. Several components have already been created for you including methods for handling an imported file.

1. Define where the uploaded files will be saved (0.5 pts)
    - Import your cache data directory path from your plab2 package
        - If it is not saved as a variable, then define it in `run.py`
    - Have `run.py` create a new folder in your cache data directory called `uploads/`
        - i.e. Create `{gitlab handle}/data/uploads/`
    - Set the `UPLOAD_FOLDER` variable to the folder path for this new `uploads/` folder
    - Set the `ALLOWED_EXTENSIONS` variable to include any file extension that is acceptable for a PPI file
2. Add a navigation link to the upload web page in the header (0.5 pts)
    - Create a navigation link with the text "Upload" in the header of your application that will take you to the upload web page. When clicked on, this nav link should redirect to `http://127.0.0.1:5000/upload`
    - *NOTE*: In `run.py`, there is already a method defined for rendering the new Upload Page called `upload()`. When this method is called, it will render the `upload.html` file in the `templates/` folder that you will create in the next step
3. Build a basic web page for uploading files (1 pt)

- In the `templates/` folder, create a new file called `upload.html`
- Edit this file so that users can upload a PPI file
    - This web page should also contain the same header and footer as your root page (think inheritance like in `template.html`)
    - You can use this basic form template:

```
<form method="POST" action="/uploader" enctype="multipart/form-data">
  <p><input type="file" name="file"></p>
  <p><input type="submit" value="Upload"></p>
</form>
```

- *NOTE*: Notice `action="/uploader"`? This is set to the same `@app.route()` as the `upload_file()` method in `run.py`. This way, when a user presses the "Upload" button in this form, it knows to execute `upload_file()`

- Once the file is selected and the user clicks the "Upload" button in the form, it should redirect the user back to the home page
- *TIP*: To see if it working correctly, check the `uploads` folder created in part 1 and see if a PPI file is there

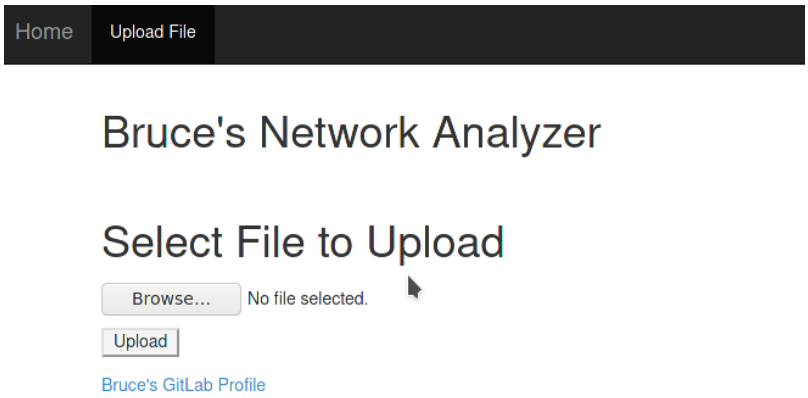

Figure 2: Example of the upload page.

Task 4 - *HGNC Info Tool* (3 pts)

Now that your code contains methods for deriving identifier information for a given HGNC symbol, it would be useful to include this feature in your web application so that user's can quickly look up information without having to leave your website.

1. Modify the `template.html` file so your index/homepage displays a form (1 pt)

- Have the section include an `h4` header tag that reads "HGNC Information Tool"
- You can use this basic form template:

```
<form method="POST" action="">
    <textarea name="textbox"></textarea>
    <button type="submit" name="submit">Submit</button>
</form>
```

- *NOTE*: The `action=` parameter in the HTML `form` tag was left blank on purpose, this will need to be filled in with the proper `@app.route()` created in the next part

1. In `run.py`, create a method that will (2 pts):
    - Gather the identifier information for the given gene symbol entered in the **textbox** created in part 1 of this task
        - This should use your package's methods for obtaining this information
    - Render the index/homepage with the results
        - If information for the entered HGNC symbol is found, there should be a table of identifier information created (have it display the HGNC ID, Ensembl Gene ID, and the **first** UniProt ID)
        - If no information is found (e.g. the user input random nonsense), have it display "No information found for {symbol}" where "symbol" is what was extracted from the **textbox**

Task 5 - *Add Shortest Paths Tool* (3 pts)

Finally, you will add the shortest path tool you developed to your web application. For this, you will need to able to access a PPI file uploaded by the user (Task 3).

1. Similar to what you did for Task 4, modify the `template.html` file so your index/homepage displays a form for entering HGNC symbols (1 pt)

    - To keep it simple, only have one **textbox** and in the next part, you can split the entry by a delimiter
    - Have the section for this tool include an `h4` header tag that reads "Shortest Path Tool"
    - Beneath this header, enter instructions specifying the number of HGNC symbols to enter and how they should be separated
2. In `run.py`, create a method that will (2 pts):
    - Gather the HGNC symbols entered in the **textbox** created in part 1 of this task
    - Split the HGNC symbols by your specified delimiter
    - Finds the shortest paths in a network generated from the first PPI file found in the `UPLOAD_FOLDER`
        - The first HGNC symbol should be considered the "source" or starting protein node and the second symbol should be considered the "target" or ending protein node
    - Render the index/homepage with the results
        - If no PPI is found, a message should appear that says "No PPI File was uploaded."
        - If no shortest paths are found, a message should appear that says "No paths found between {source} and {target}" where "source" is the first HGNC symbol given and "target" is the second HGNC symbol given
        - If there are shortest paths, display **only the first** shortest path as a bulleted list in which first HGNC symbol given is at the top, the second HGNC symbol provided is at the bottom, and the path node symbols are in between (in the correct order)
            - For this part, it may be useful to use the statement delimiter from Jinja i.e. `{% ... %}`
            - For testing:
                - If you used an undirected graph: source = "CREBBP", target = "TRA2B"
                - If you used a direct graph: source = "SYMPK", target = "RAG1"

BONUS Task 6 - *Spruce It Up* (+1 pt)

Use bootstrap or add CSS to your web application so that you change font family of your headers. You will only get points if:

- the CSS/bootstrap is clearly defined (either within the HTML files or in a separate CSS file)
- the changes are properly applied in your homepage