INFO3180 Lab 5 - (20 Marks)

Due: March 2, 2020 at 11:55pm

The goal of this lab is to implement a working login system using Flask-Login. The authentication and authorization will be against a PostgreSQL database and all routes should be protected to prevent viewing unless a user is logged in. You will also create database migrations using Flask-Migrate.

It is recommended that you read the Flask-Login documentation at https://flask-login.readthedocs.org/ and Flask-Migrate documentation at https://flask-migrate.readthedocs.io .

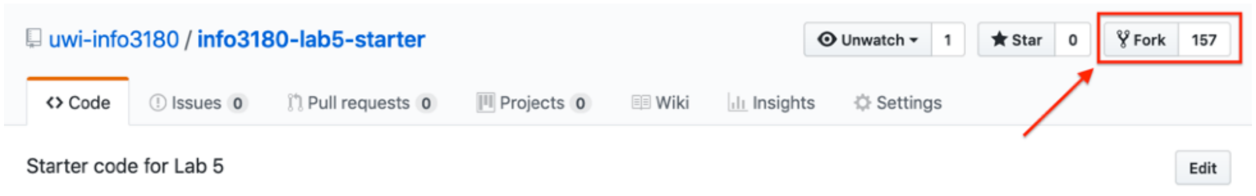
Exercise 1 - Using Flask Login to Manage User Logins

For reference you can look at the example from the lecture located at: https://github.com/uwi-info3180/flask-demo-user

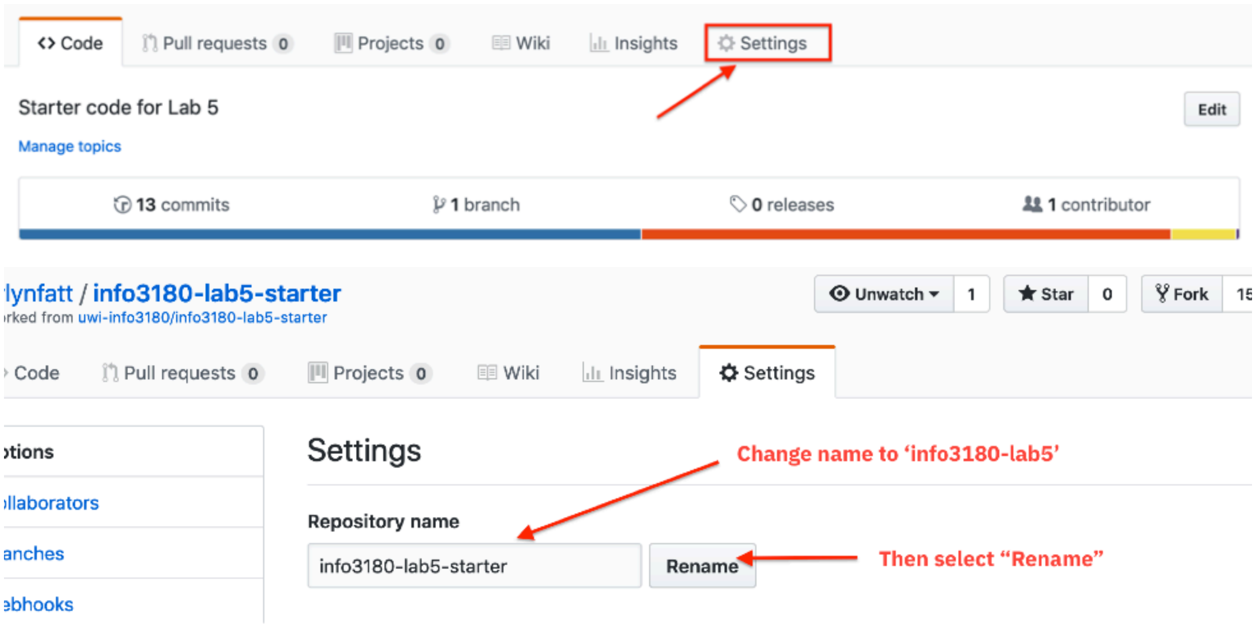
Step 1 - Start with the info3180-lab5-starter starter app

Start with the example at: https://github.com/uwi-info3180/info3180-lab5-starter

Fork the repository to your own account



In github.com rename it by going to settings and changing the name to **`info3180-lab5.`**



Starter code for Lab 5

lynfatt / info3180-lab5-starter
rked from uwi-info3180/info3180-lab5-starter

ptions
llaborators
anches
ebhooks

Settings

Repository name: info3180-lab5-starter

To start working on your code, clone it from your newly forked repository for example:

If you are working in Cloud9 or if you are working locally on your own computer (ie. NOT in Cloud9) :

```
git clone https://github.com/{yourusername}/info3180-lab5
```

Step 2 - Install the dependencies (Flask-Login and Flask-Migrate)

Important

Remember to create and activate your **`virtual environment`** BEFORE doing any package installation with pip or running python!

Ensure that **`Flask-Login`** , **`Flask-Migrate`** and **`Flask-Script`** are listed in your **`requirements.txt`** and then use:

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

Note:
Pay attention to the flask migration script (**`flask-migrate.py`**). You will find it useful in this lab and in later projects. For more information on using this script see Tutorial 5.

This script will be important as you make database changes. By running the following commands, changes to your models will be reflected in your database. We will use these later in this lab.

```
python flask-migrate.py db migrate
python flask-migrate.py db upgrade
```

Step 3 - Setting up your PostgreSQL database

Installing PostgreSQL on Cloud9 / Ubuntu Linux

On Cloud9 or Ubuntu Linux you can install Postgres using the following command:

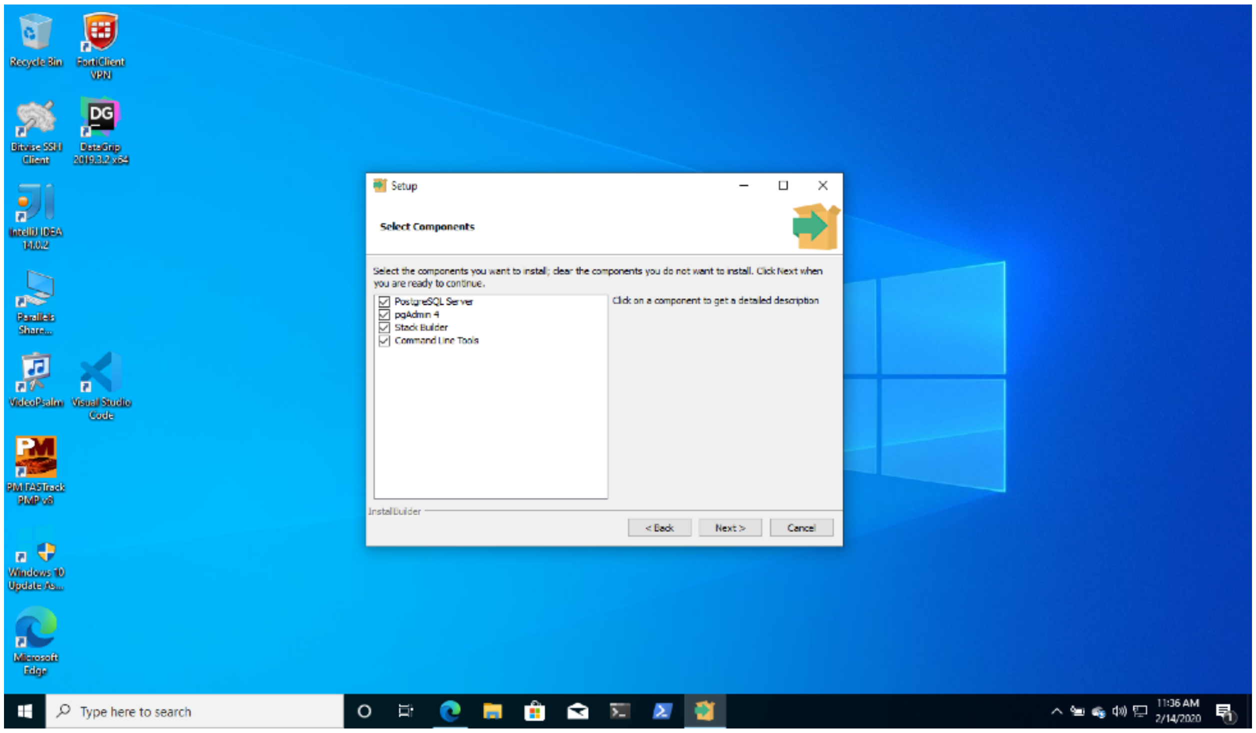
```
sudo apt install postgresql
```

Then connect to PostgreSQL command line interface by running in your Terminal:

```
sudo -u postgres psql
```

Installing PostgreSQL on Windows

1. Download the installer from https://www.postgresql.org/download/windows and then open the installation file you downloaded. Follow the various steps to complete the installation.



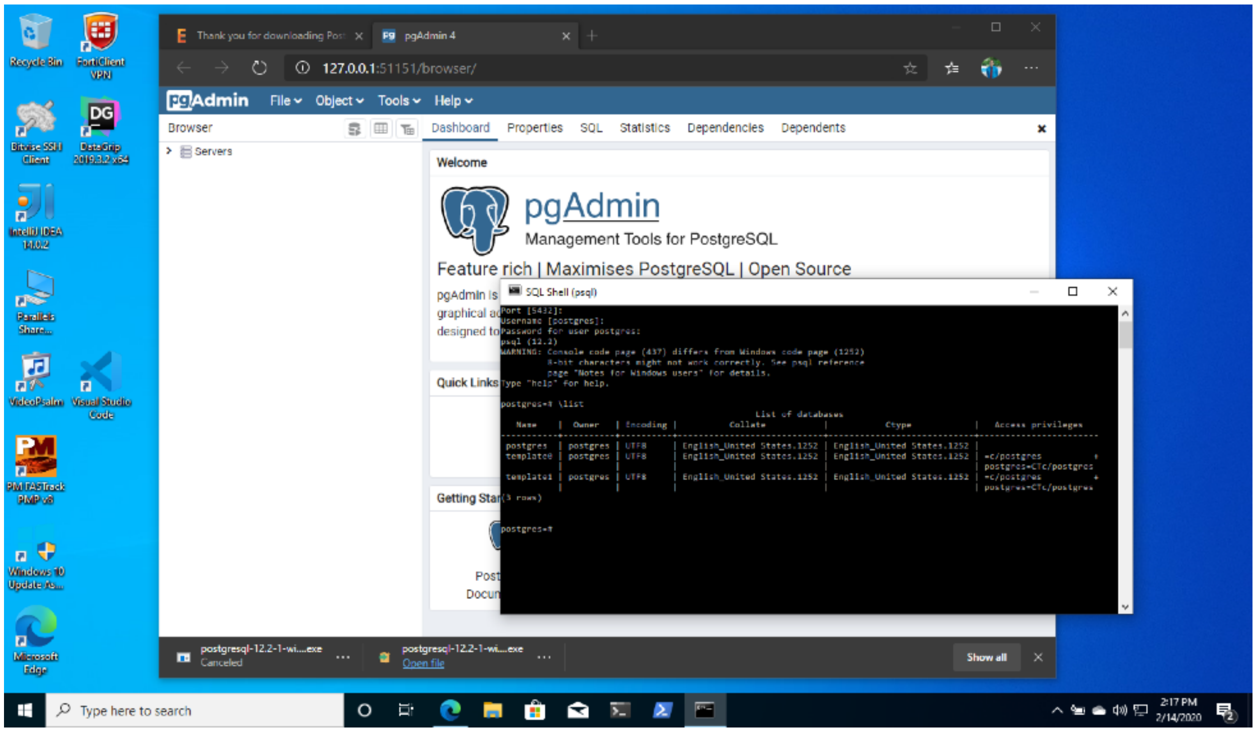
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- [✓] PostgreSQL Server
- [✓] pgAdmin 4
- [✓] Stack Builder
- [✓] Command Line Tools

Click on a component to get a detailed description

2. Once the installation is complete, look for and open "**SQL Shell (psql)**" in the Start menu or via the Search box on Windows to access the command line interface for Postgres. Optionally you can try to use the pgAdmin GUI tool (**Note**: the instructions for adding a user and creating a database do no apply to the GUI, but only to the command line

interface).



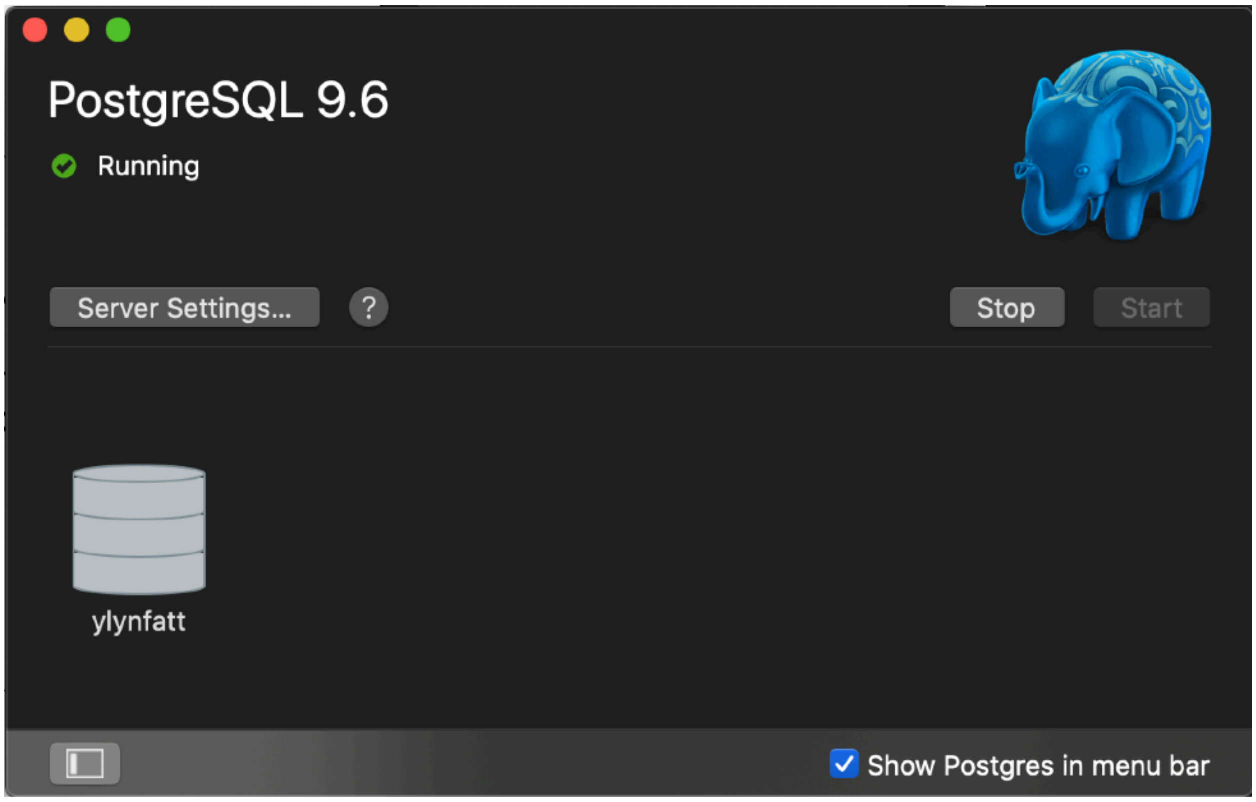
```
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (12.2)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=# \list
                                                 List of databases
   Name    |  Owner   | Encoding |          Collate           |           Ctype            |   Access privileges
-----------+----------+----------+----------------------------+----------------------------+-----------------------
 postgres  | postgres | UTF8     | English_United States.1252 | English_United States.1252 |
 template0 | postgres | UTF8     | English_United States.1252 | English_United States.1252 | =c/postgres          +
           |          |          |                            |                            | postgres=CTc/postgres
 template1 | postgres | UTF8     | English_United States.1252 | English_United States.1252 | =c/postgres          +
           |          |          |                            |                            | postgres=CTc/postgres
(3 rows)


postgres=#
```

Installing PostgreSQL on MacOS

1. Download Postgres.app from https://postgresapp.com and follow the instructions.
2. Open the Postgres.app application, Start the database server by clicking the "Start" button and then Double-click on one of the databases listed. It should open the command line interface for you.



Setting up a user and creating a database

Once you have installed PostgreSQL and are connected to the PostgreSQL command line interface, create a user and a database and set a password for the user and make them the owner of the lab5 database by doing the following steps:

```
create user "lab5";
create database "lab5";
\password lab5
alter database lab5 owner to lab5;
```

You can now quit the PostgreSQL command prompt by using:

```
\q
```

Step 4 - PostgreSQL connection string and setup your Flask-Login Manager

Look in the **`app/__init__.py`** file. You'll need to change the **`SQLALCHEMY_DATABASE_URI`** connection string so that SQLAlchemy connects properly to your **`lab5`** database with the **`lab5`** user and password you created.

Step 5 - Define a Flask Login User model and create your first migration

Take a look at your **`models.py`** file. You will notice that you have been provided an initial **`UserProfile`** class. Pay attention to the properties that have been defined. These represent the columns that will be mapped to a table in your database. Also pay attention to the **`is_authenticated()`**, **`is_active()`**, **`is_anonymous()`**, and **`get_id()`** methods. These are needed by Flask-Login.

Now create your first migration by running the following commands:

```
python flask-migrate.py db init
python flask-migrate.py db migrate
python flask-migrate.py db upgrade
```

You should now see a **`migrations`** folder in the root directory of your application. And if you check your database (by connecting back to the

Postgres command prompt), you should also see two tables **`alembic_version`** and **`user_profile`**.

Commit your code and push to your Github repository.

Step 6 - Add a password field, Create a new migration and update your database

In order to manage local logins, you'll need to add a password field to the **`UserProfile`** class in your **`models.py`** file. Ensure the column is of the **String** type that can take up to **255** characters. Also you will want to ensure that you hash this password before it is stored in the database. To do this ensure you import the **`generate_password_hash()`** function from the **`werkzeug.security`** library and create an **`__init__()`** method within your UserProfile class. You can view this example as a reference: https://github.com/uwi-info3180/flask-demo-user/blob/master/app/models.py

You have now added a new password field to your model, however, you need to create a new migration to represent that change and also upgrade your database to reflect that change.

```
python flask-migrate.py db migrate
python flask-migrate.py db upgrade
```

Commit your code and push to your Github repository.

Step 7 - Add a user to your database so you can login.

Now we will need to add a user so that we can test our login. Launch a python prompt from the command line and then you can use the following commands to quickly add a user.

```
$ python
>>> from app import db
>>> from app.models import UserProfile
>>> user = UserProfile(first_name="Your name",
last_name="Your last name", username="someusername",
password="somepassword")
>>> db.session.add(user)
>>> db.session.commit()
>>> quit()
```

Note: You could also have connected to your PostgreSQL database and written the appropriate SQL insert statement. Or created a form within your web application and have it add the user when the form is successfully submitted (you'll do this in your project). Also under normal circumstances you would not store user passwords in plain text. Instead you should hash the passwords.

Step 8 - Update the /login route

Look at https://github.com/uwi-info3180/flask-demo-user/blob/master/app/views.py#L39 for reference.

Now let us take a look at your **`views.py`** file. You will see that a **`login`** route, view function and **`login.html`** template has been defined, however it is incomplete. It needs to be able to do the following:

- Validate that a username and password was entered on your login form. Change the if statement to check **`if form.validate_on_submit()`**.
- Check that the username and password entered matches that of the user you created in your database in Step 7. You'll need to actually make a query to the database using the **`UserProfile`** model and utilize the **`query.filter()`** method.
- You will also need to ensure that when checking that the password matches, that you utilize the **`check_password_hash()`** function from the **`werkzeug.security`** library (this will need to be imported).
- Redirect the user to the **`/secure-page`** route and display a **`flash`** message to the user letting them know that they have successful logged in. You will create this route in the next step.

Commit your code and push to your Github repository.

Step 9 - Create the /secure-page route and secure_page() view function

3. Create the **`/secure-page`** route and **`secure_page()`** view function along with a template file called **`'secure_page.html'`** that is to be rendered. This page should have an **`<h1>`** heading that says "***Secure Page***" and a simple paragraph of your choosing.
4. Ensure that you add the **`@login_required`** decorator to the route so that it is only accessible when a user is logged in.

5. Test to ensure that when the user is not logged in and they visit the **/secure-page**, they get sent to the **/login** page. Then test that when the user is logged in, the are able to view the **/secure-page** route.
6. Open your **'header.html'** file an ensure that you add a link to your navigation for the **/secure-page** route.

Commit your code and push to your Github repository.

Step 10 - Create a logout route

1. Create a **logout** route and ensure you use the Flask-Login, **logout_user()** method to logout a user.
2. Ensure you also **flash** a message to the user and **redirect** them to the **home** route.
3. Also, update your **header.html** template file and ensure that you switch the login link to display logout link only when the **current_user is_authenticated.**

Commit your code and push to your Github repository.

Submission

Submit your code via the "Lab 5 Submission" link on OurVLE. You should submit the following link:

1. Your Github repository URL for your Flask app e.g. https://github.com/{yourusername}/info3180-lab5

Grading

1. You should have 2 migration files and when migrate command is run it should recreate the database. (3 marks)
2. Your models.py file the `UserProfile` class should have a password field. (1 mark)
3. Your UserProfile model should have a constructor (ie __init__() method) defined with instance variables and ensure your password is hashed before storing it in the database.
4. For the login route you should get the username and password submitted from login form (e.g. `request.form['username']` or `form.username.data`). (1 mark)
5. For the login route you should also validate the user input using form.validate_on_submit() or form.validate(). (1 mark)
6. You should query the database for the username and password and ensure the login_user() method is used to login that user. (2 marks)
7. You should redirect user to `/secure-page` route and display an appropriate flash message. (2 marks)
8. You should create the secure page route and view function and restrict access to that route with the @login_required decorator. (2 marks)
9. The secure page should display a <h1> heading and a paragraph. (1 mark)
10. A link should be added to the navigation bar for the secure page. (1 mark)
11. Create a logout route and logout() view function which uses the logout_user() method. (2 marks)
12. Add a logout and login link in your header.html template which should change when user logged in/out. (2 marks)