CS-1520 Final Project

1 Introduction

In lieu of a writing final exam, you will create a Flask API simulating a Web-based Chat Application. By developing it, you will use the following concepts learned on this course:

- HTML
- CSS
- Responsive Web Design (optional)
- JavaScript
- DOM
- Python
- Flask
- Jinja2
- Templates
- Static files (CSS and JavaScript files)
- Models
- Database (SQLAlchemy)
- AJAX
- Polling

2 General overview of the Chat windows

When you first go to the chat URL, (http://127.0.0.1:5000/), your application should redirect you to the login page, as shown in Figure 1.

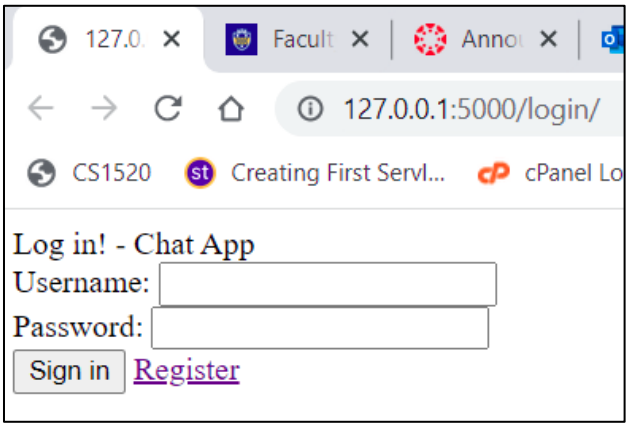

Figure 1: The login page

In this page, the user can either sign in with an registered username or register for an account. If the user click on Register link, your application should show the registration page, as shown in Figure 2.

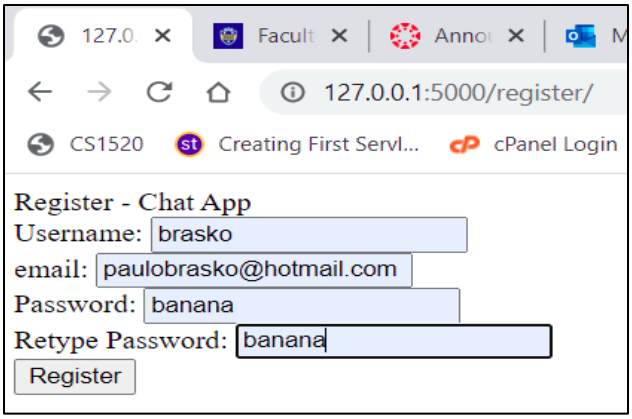

Figure 2: The registration page

After entering the user information, when clicking on the Register button, your application should verify that the passwords entered match. If not, just go back to this page to the user reenter their information again. No need to generate an error message in the webpage. A simple printout in the terminal window is fine.

If the information is ok, then your application should save the user information in a database and display the page shown in Figure 3.

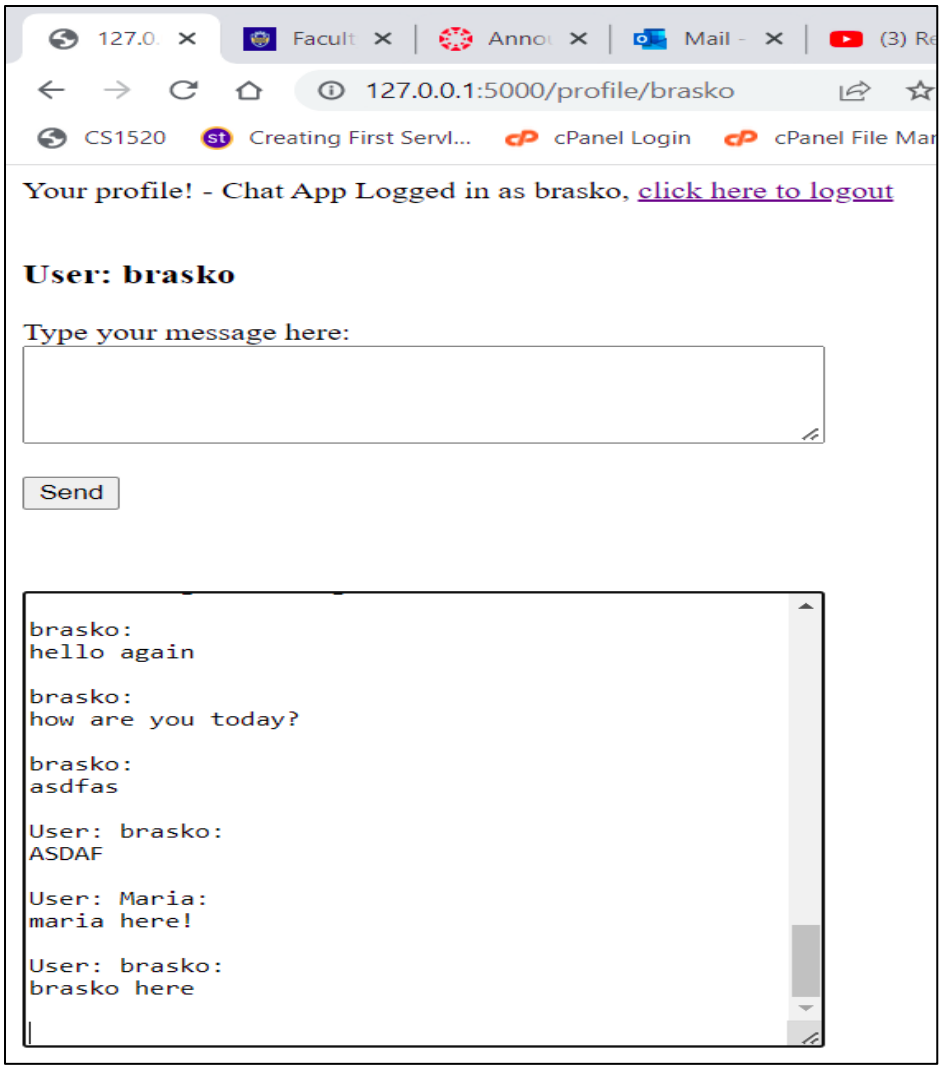

Figure 3: The chat page

The same window from Figure 3 should be presented if an already registered user signs in as shown in Figure 4.

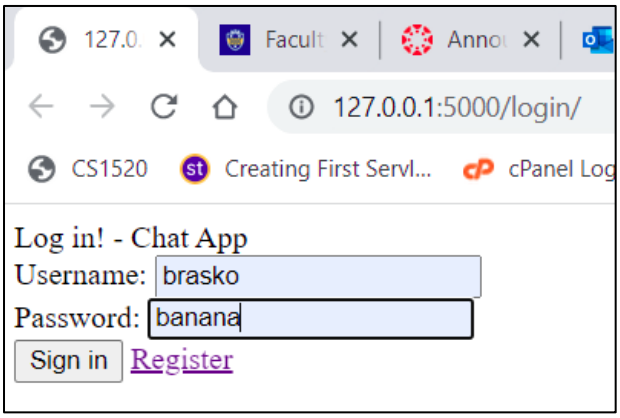

Figure 4: Signing in

If the user tries to sign in, but they are not registered yet, your application should just redirect the user to this page again. No need for a flash message here, just a printout in the terminal window will be sufficient for this project.

Your application should provide the following functionality for the chat page:

- As soon as this page is displayed, your application shall fetch all the messages saved in the database table that contains these messages and present them in the lower text area of this page.
- The registered user can type a message in the first text area and by clicking on the SEND button, the message should send to the server, which in turn will save this new message (and author) into the database table. After doing so, the new message text area should be clear out and the text area displaying all the messages should be updated to contain your new message.
- Your application shall update the messages text area on a pre-defined interval, let's say every 15 seconds.

Try to open a second browser window, and login as another user (you need to register him/her first, of course). Then try to write in one browser page and see the effects on the other browser page, as shown in Figure 5. In the browser that the new message was typed, the messages text area should be updated right away, while in the other browser, the text area should take few seconds to get updated (base on the polling time interval).

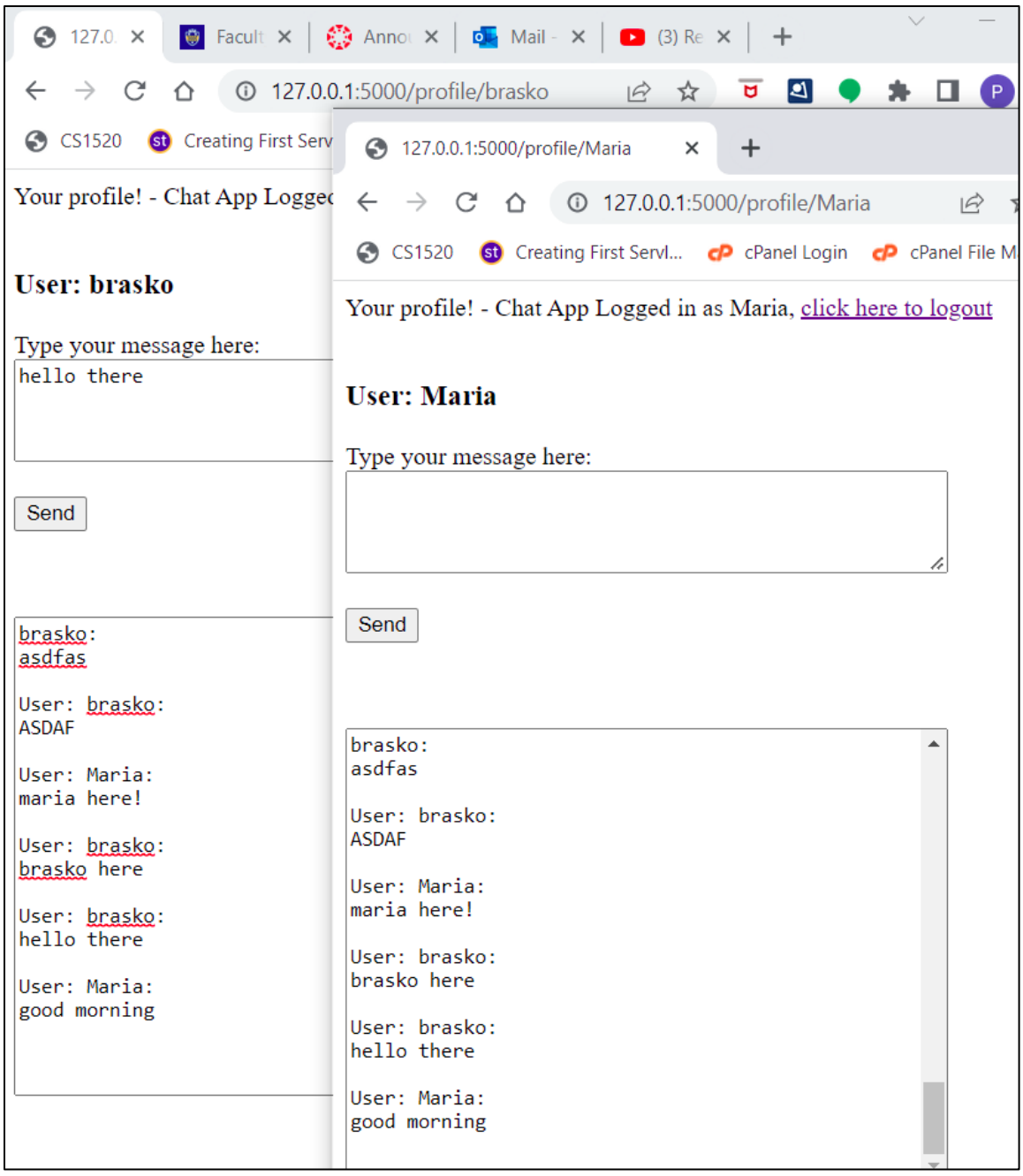

Figure 5: Two users chatting in your application

3 General Application Structures

Your application should have the structure shown in Figure 6, which shows the html page names, a javascript file, a python file (your flask application), and two databases: app.db and chat.db. The app.db I have used to save registered users, while the chat.db I have used to store the chat messages/authors. I decided to use two completely independent tables (no foreign keys linking them, but it is up to you).

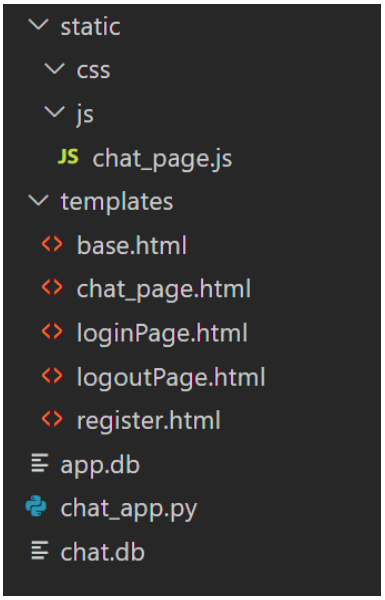

Figure 6: Application structure

I strongly recommend you look at (and adapt the existing code for) the login application that we have discussed in class and also our last couple of lectures where we discussed AJAX. Use the codes developed in class as basis for your application.

A word of advice, remove all the code related to sessions. If you keep them, you will not be able to reproduce the functionality shown in Figure 5 (two windows with two different users, in the same computer).

4 General Hints

In this section I provide few hints about implementing the features needed in your application.

4.1 The database

To have two unrelated tables in the same application, you have to create one and the second one you need to bind to the first one (confusing, but that is the approach that I used). The code below shown how to do that:

```
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///app.db'
app.config['SQLALCHEMY_BINDS'] = {'chat': 'sqlite:///chat.db'}
```

Similar to what has been done in class, you will need to create 2 classes/models: a User class and a Chat class. When creating the mode for the chat.db, you need to include a __bind_key__ as shown below.

```
class Chat(db.Model):
    __bind_key__ = 'chat'
```

4.2 The routes

I would expect that you implement the following routes in your application:

```
@app.route("/")
def default():

@app.route("/login/", methods=["GET", "POST"])
def login_controller():

@app.route("/register/", methods=["GET", "POST"])
def register_controller():

@app.route("/profile/<username>")
def profile(username=None):

@app.route("/logout/")
def unlogger():

@app.route("/new_message/", methods=["POST"])
def new_message():

@app.route("/messages/")
def messages():
```

Each of the routes associated functions shall be implemented accordingly. Please check the codes discussed in class. Most of your code can be extracted from them.

4.3 The JavaScript file

For this project, there will be a single JavaScript file, the chat_page.js, which will be linked to the chat_page.html.

4.3.1 Fetching data from the database tables

Your application shall use AJAX to send the new message into the server api. The first portion of a possible fetch command is shown below for your convenience. You need to implement the ".then" and ".catch" portions.

```
fetch("/new_message/", {
      method: "post",
      headers: { "Content-type": "application/x-www-form-urlencoded; charset=UTF-8" },
      body: `username=${author}&message=${message}`
   })
```

The process of retrieving and manipulating the chat messages from the server database tables is a little bit more convoluted. To help you out, I am providing a same code of how you should do it.

```
fetch("/messages/")
    .then((response) => {
        return response.json();
    })
    .then((results) => {
        let chat_window = document.getElementById("chat_window");

        let messages = "";
        for (let index in results) {
            current_set = results[index];
            for (let key in current_set) {
                author = key;
                message = current_set[key];
                messages += `${author}:\n${message}\n\n`;
            }
        }
        chat_window.value = messages;
    })
    .catch(() => {
        chat_window.value = "error retrieving messages from server";
    });
```

4.4 The html files

Look at the sample codes shown in the past recent Flask lecture and the login project to have a feeling of what you need to place in the html files.

5 Final thought

I will create a video showing the application running soon. But don't wait for it, the instructions herein should be enough for you to get up and running.

DO NOT WAIT UNTIL THE LAST COUPLE OF DAYS TO DO THIS PROJECT! Start today!

I have no problem with you discussing it with other class members. Just try to understand every simple piece of code that you have in your project.

Good luck

Paulo