INFO3180 Lab 3 (20 marks)

Due Date: **21 February, 2021 at 11:55 PM**

One of the most basic functions in a web application is the ability to send emails to your users.

The **Flask-Mail** extension provides a simple interface to set up SMTP with your Flask application and to send messages from your views and scripts. In this lab we will also use **Flask-WTF** to create a Contact form and then use Flask-Mail to send a test email to a Fake SMTP server.

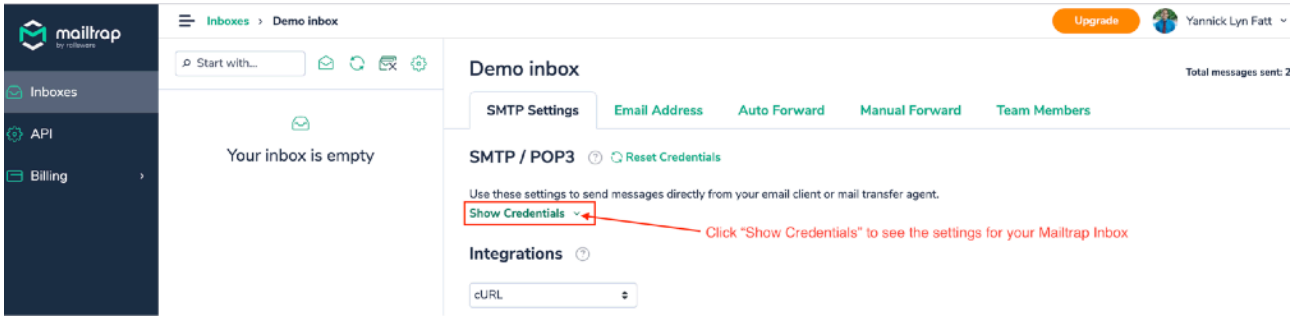
Here are some helpful resources:
https://pythonhosted.org/Flask-Mail/
https://flask-wtf.readthedocs.io/en/stable/

Exercise 1 - Setup a Fake SMTP Server

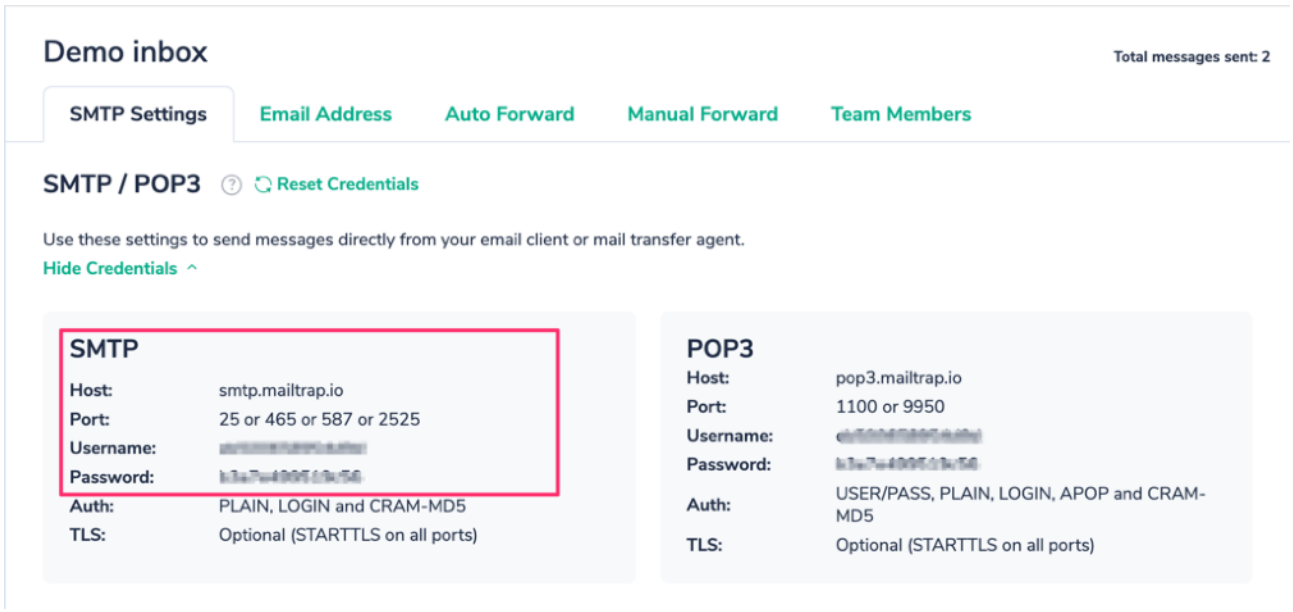
Typically you would have a proper SMTP server setup or you could use an existing email account like your Gmail or Outlook account. However, in today's lab we will use a fake SMTP server called Mailtrap. Mailtrap allows development teams to test, view and share emails sent from the development and staging environments without spamming real users.

Sign up for a FREE account at https://mailtrap.io/register/signup.

Once you have signed up for an account, you should see your inboxes and there should be one called 'Demo inbox'.



Click 'Demo inbox' and you should see an Empty inbox. Then click on the "Show Credentials" link under SMTP Settings so that you can see the necessary credentials to connect to this Fake SMTP server.



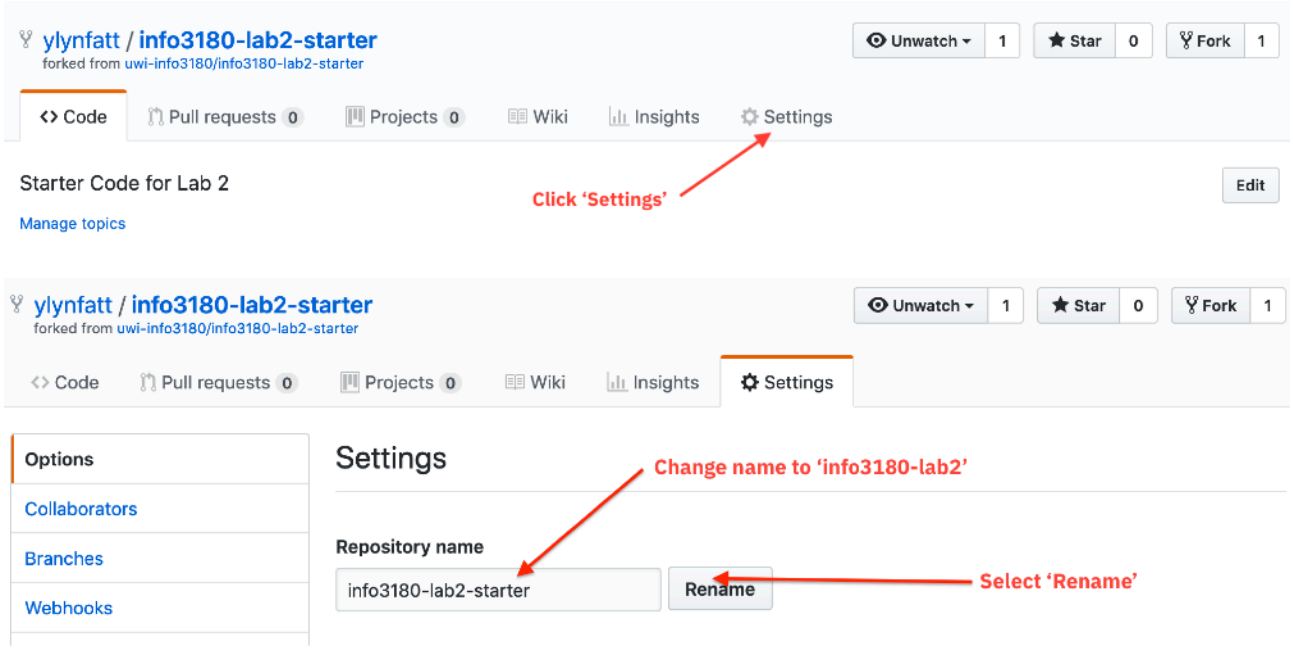
NOTE: Ensure you note the Host, Port (try to use 465 or 2525) and the Username and Password. We will need these in the next exercise.

Exercise 2 - Creating an Email Sending Form

Step 1

Start with the starter code at: https://github.com/uwi-info3180/info3180-lab3-starter

Fork the repository to your own account as you have done in previous labs. Then, in github.com rename it by going to Settings and changing the name to **info3180-lab3**



To start working on your code clone it from your newly forked repository.

```
git clone https://github.com/{yourusername}/info3180-lab3 info3180-lab3
```

Note: Ensure you change **{yourusername}** to your actual Github username.

Step 2 - Setup a Python virtual environment and install the necessary libraries.

1. Create a new virtual environment by running **`python -m venv venv`** at the command prompt.

2. Activate your virtual environment by running **`source venv/bin/activate`**. (or **`.\venv\Scripts\activate`** if using Windows)
3. Next, open your **`requirements.txt`** file and add the **email_validator**, **`flask_mail`** and **flask_wtf** packages to it and then save the changes to the file.
4. Now run **`pip install -r requirements.txt`** at the command prompt. This should install all the necessary packages for this lab.
5. Next, open your **`app/config.py`** file and update it with the following configuration for Flask_Mail:

```
class Config(object):
    """Base Config Object"""
    DEBUG = False
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'Som3$ec5etK*y'
    MAIL_SERVER = os.environ.get('MAIL_SERVER') or 'localhost'
    MAIL_PORT = os.environ.get('MAIL_PORT') or '25'
    MAIL_USERNAME = os.environ.get('MAIL_USERNAME')
    MAIL_PASSWORD = os.environ.get('MAIL_PASSWORD')
```

6. Typically you don't want to accidentally store any sensitive information like usernames, passwords or secret keys directly in your Git repository as such for this lab we will opt to use environment variables and then retrieve their values in our code similar to what you did the *config.py* file in Step 5 above.

   We will set our environment variables at the command line and you will need to do these one at a time at the command line for this lab. Enter the appropriate information that you got earlier from your Mailtrap SMTP Settings page at the command prompt/terminal like this:

```
export MAIL_SERVER="smtp.mailtrap.io"
export MAIL_PORT=465
export MAIL_USERNAME="your mailtrap SMTP username"
export MAIL_PASSWORD="your mailtrap SMTP password"
export SECRET_KEY="some random set of characters"
```

Note: on Windows you will use "**set**" instead of "**export**". For example **set MAIL_SERVER="**smtp.mailtrap.io**"**

Note 2: If you happen to close your command prompt/terminal or open a new terminal you may need to re-create your environment variables again.

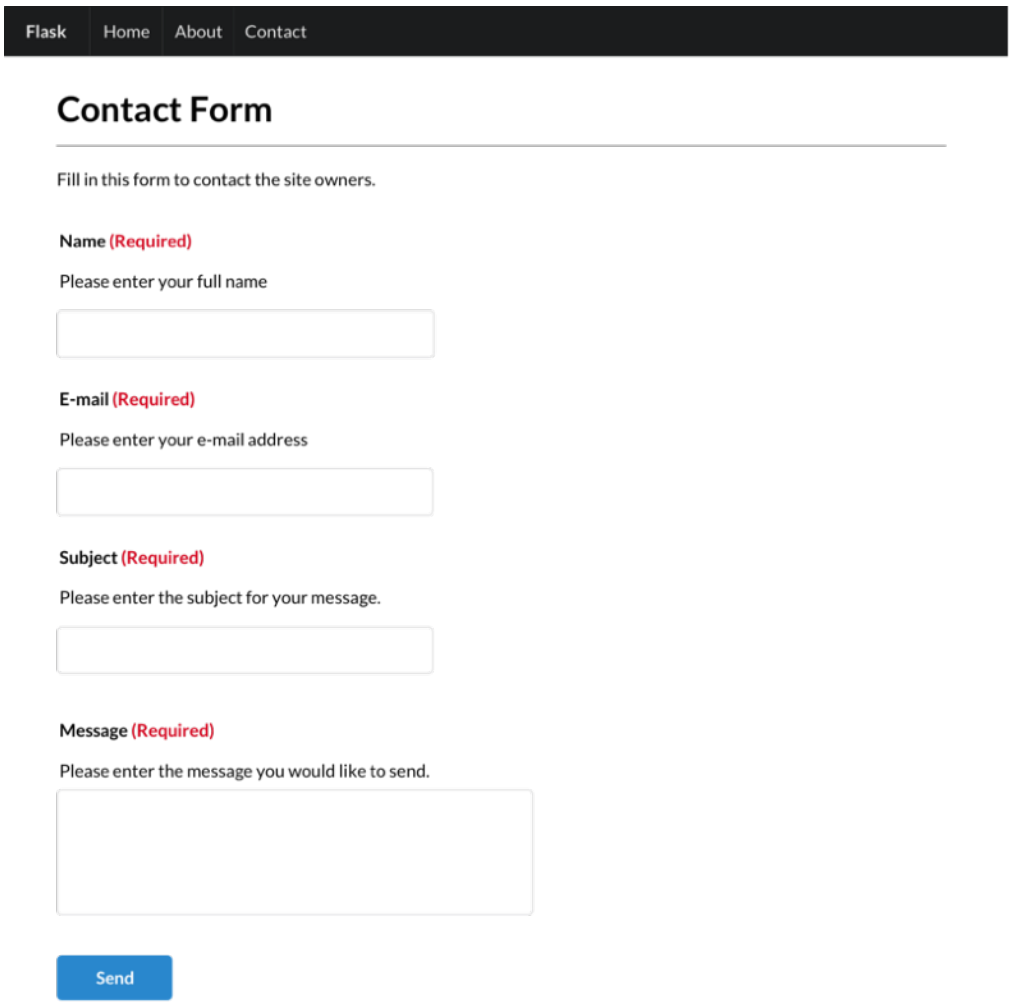
7. Open your **app/__init__.py** file and update it to import your Config class and add the configuration values to your Flask **app.config** and also import and initialize Flask_Mail. It will look like the following:

```
from flask import Flask
from flask_mail import Mail
from .config import Config

app = Flask(__name__)
app.config.from_object(Config)

mail = Mail(app)
from app import views
```

Step 3 - Create Contact form

Feel free to check your lecture slides or the Flask-WTF documentation on how to create forms, output them in your template files and validate user input.

1. Create a file called **`forms.py`** in your **`app/`** directory.
2. In **`forms.py`** create a Python class called **`ContactForm`** to represent your form using the various field types and validators available to you in Flask-WTF. Your form should have a text field for **`name`**, **`email`**, **`subject`** and **`a text area for a message`**. Also remember to protect your form with a **CSRF token**.
3. Now, in your **`views.py`** file, make your contact form available at the route **`"/contact"`** and name the associated view function **`"contact()"`**. You should create and name your template where you will output your form, **`"contact.html".`** It should look something like this:



Note: Remember that with Flask-WTF you output your form fields in your template files using the format

```
{{ form.field_name.label }}
{{ form.field_name }}
```

Also remember that you need to output a field for your CSRF token to secure your form. You do this by printing the following in your

template file (within the **<form></form>** tags).

```
{{ form.csrf_token }}
```

If you forget to do this then your form may give validation errors.

4. Open your **templates/header.html** file and add a link to your newly created **Contact** form in the navigation.

Step 4 - Process the contact form submission, send the email and redirect the user

1. First let us ensure we import the **Message** class from Flask-Mail and the **mail** variable we instantiated in Step 2. Do this by adding the following near the top of your **views.py** file.

```
from app import mail
from flask_mail import Message
```

2. Next, update your **contact()** view function and ensure you allow for **POST** requests by setting the **methods** property on the route definition.

   Ensure you check that *if* a *POST* request is made and that you validate the user input on submit. Otherwise simply display the contact form.

   https://flask-wtf.readthedocs.io/en/stable/quickstart.html#validating-forms

If the validation checks pass successfully, send an email by doing something similar to the following:

```
msg = Message("Your Subject", sender=("Senders Name", "from@example.com"),recipients=["to@example.com"])
msg.body = 'This is the body of the message'
mail.send(msg)
```

This might be helpful: https://pythonhosted.org/Flask-Mail/#sending-messages

Of course you will replace the items above with the values submitted from your form fields. You can access the values from your form fields by using the global request object, for example **`request.form['field_name']`** or by using the Flask-WTF way, for example **`formname.field_name.data`**.

Hint: An example of how to use the **`request`** object to get form data can be found at the following link: https://flask.palletsprojects.com/en/1.1.x/quickstart/#the-request-object

3. After you send the message, ensure you **redirect** the user to your home page (ie. your "/" route) and display a **flash** message letting them know the user know their email was successfully sent.

Hint: An example of how to use the **`flash()`** and **`redirect()`** methods

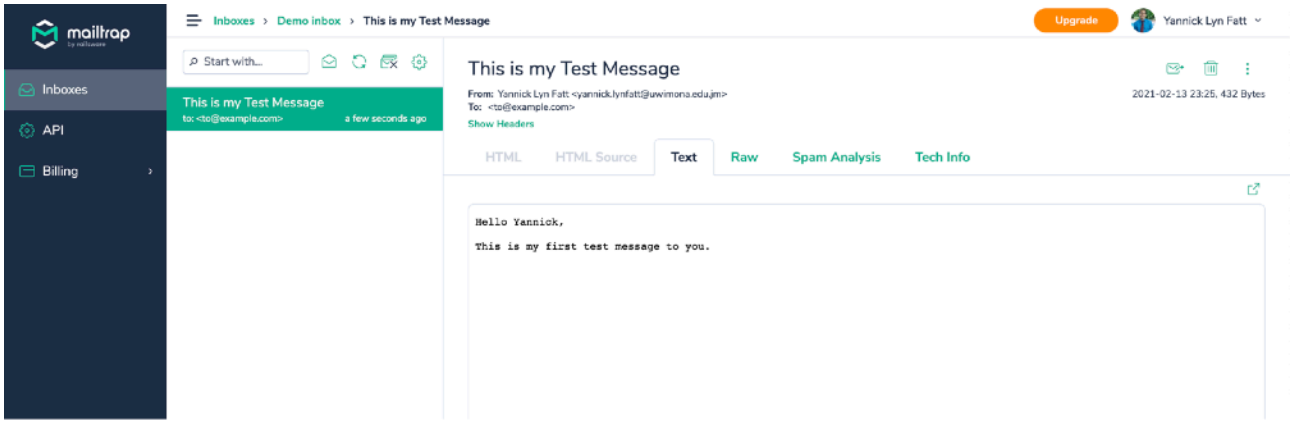
can be found at the following link: https://flask.palletsprojects.com/en/1.1.x/patterns/flashing/

You will also need to put code like the following in your ***base.html*** template, just above your *main block* in order to print the flash message in your template:

```
{% with messages =
get_flashed_messages(with_categories=true) %}
  {% if messages %}
    <ul class="flashes">
    {% for category, message in messages %}
      <li class="{{ category }}">{{ message }}</li>
    {% endfor %}
    </ul>
  {% endif %}
{% endwith %}
```

Feel free to adjust and style this message using CSS so it stands out and users can notice the message.

4. Now ensure the Flask development server is running and browse to your **"/contact"** route. Fill out the form and hit the Submit button for your form. If all went well, you should be redirected to your homepage and a message should be displayed showing you that you successfully submitted the contact form. You should also see a new message in your Mailtrap Demo inbox. See example below:



Submission

Submit your code via the "Lab 3 Submission" link on OurVLE. You are NOT required to push your code to Heroku. You should submit the following links:

1. Your Github repository URL for your Flask Exercise e.g. https://github.com/{yourusername}/info3180-lab3

Grading

1. Create contact route and associated view function (1 mark)
2. Create *forms.py* file with a class that defines fields for *name*, *email address*, *subject* and a text area for a *message* using flask-wtf (5 marks)
3. Create template file that renders the form when */contact* route is visited with appropriate form fields as shown in screenshot. (2 marks)
4. When contact form is submitted, check if it's a `POST` request and if the data validates with *validate_on_submit()* method. (2 marks)
5. Data from the form should be retrieved using *request.form['field_name']* or *form.field_name.data* and stored in variables and passed to the Message object. (3 marks)

6. The Message object should be instantiated with appropriate parameters *[subject, from, to, body]* and then *mail.send()* method called to send email. (2 marks)
7. The User should be redirected to home page (using *redirect()* and *url_for()* methods) after successful sending of email. (2 marks)
8. A flash message (using the *flash()* function) should be set letting the user know their email was successfully sent and this message should be displayed on the home page when redirected. (2 marks)
9. The Email should successfully be sent and appear in the Mailtrap inbox. (1 mark)