Tuesday: Image Upload and Sending Email

Uploading Images

We want our users to be able to upload their profile pictures to their profile. Flask-uploads is a very good extension that allows us to upload files to our flask application

```
(virtual)$ pip install flask-uploads
```

We first install the extension then we can set it up.

app/__init__.py

```
from flask_uploads import UploadSet,configure_uploads,IMAGES

#.............
photos = UploadSet('photos',IMAGES)
def create_app(config_name):
#...................
```

We first import the `UploadSet` class that defines what type of file we are uploading. We pass in a name and the Type of file to upload which is an Image.

app/__init__.py

```
def create_app(config_name):
    app = Flask(__name__)
    #........

    # configure UploadSet
    configure_uploads(app,photos)

    #.......
```

We then use the `configure_uploads` function and pass in the `app` instance and the `UploadSet` instance.

config.py

```
class Config:
# ....
    UPLOADED_PHOTOS_DEST ='app/static/photos'
```

We then create a new configuration `UPLOADED_PHOTOS_DEST` which specifies the destination to where we want to store our Images. We set the destination to be a photos folder inside our static folder.

It is not advisable to store files inside the database. Instead we store the files inside our application and we store the path to the files in our database.

app/templates/profile/profile.html

```
........
  <div class="col-md-4">
    {% if user.profile_pic_path %}

    <img src="{{url_for('static',filename=user.profile_pic_path)}}" alt="" class="img-thumbna
il">

    {%else%}
    <p>No profile picture</p>

    {% endif %}

        {% if user == current_user %}
    <form  action="{{url_for('main.update_pic',uname=user.username)}}" method="post" enctype
= "multipart/form-data">

      <input type="file"  name="photo" class="form-control" required>
      <input type="submit">
    </form>
      {% endif %}
  </div>
 .........
```

We create a file upload form. To upload a file we need to use the `enctype = "multipart/form-data"` attribute on form tag.

app/main/views.py

```
......
from .. import db,photos
......
@main.route('/user/<uname>/update/pic',methods= ['POST'])
@login_required
def update_pic(uname):
    user = User.query.filter_by(username = uname).first()
    if 'photo' in request.files:
        filename = photos.save(request.files['photo'])
        path = f'photos/{filename}'
        user.profile_pic_path = path
        db.session.commit()
    return redirect(url_for('main.profile',uname=uname))
```

We first import the `photos` instance from our application factory module. We then create a route that will process our form submission request. This route only accepts post requests.

We then query the database to pick a user with the same username we passed in. We then use the flask `request` function to check if any parameter with the name `photo` has been passed into the request.

We use the `save` method to save the file into our application. We then create a path variable to where the file is stored. We then update the `profile_pic_path` property in our user table and store the path to the file.

We finally commit the changes to the database and redirect the user back to the profile page.

Sending an Email

We want to thank our users and welcome them when they register for our application. Flask uses the Flask-Mail extension to send emails to users.

```
(virtual)$ pip install flask-mail
```

The extension connects to a Simple Mail Transfer Protocol server. It passes the messages we want to send to the SMTP server and it delivers the server.

Connecting To Gmail

We need our application to send emails through a google account. We need to setup some configuration files to connect to the gmail smtp server.

config.py

```
#  email configurations
    MAIL_SERVER = 'smtp.googlemail.com'
    MAIL_PORT = 587
    MAIL_USE_TLS = True
    MAIL_USERNAME = os.environ.get("MAIL_USERNAME")
    MAIL_PASSWORD = os.environ.get("MAIL_PASSWORD")
```

We setup the SMTP server and configure the port to use the gmail SMTP server port. Then we set `MAIL_USE_TLS` configuration to true which enables a transport layer security to secure the emails when sending the emails.

`MAIL_USERNAME` and `MAIL_PASSWORD` are our email address and password to authenticate to the gmail SMTP server. We set them as environment variables.

start.sh

```
#.....
export MAIL_USERNAME=<Your Email Address>
export MAIL_PASSWORD=<Your Email Password>
python3.6 manage.py server
```

Here we add the export code for the `MAIL_USERNAME` and `MAIL_PASSWORD` to our start script. This will export the variables to the environment every time we run our start script.

We then can initialize the extension.

app__init__.py

```
#........
from flask_mail import Mail

#........
mail = Mail()

def create_app(config_name):
    app = Flask(__name__)
    #........
    mail.init_app(app)
#........
```

We import the the `Mail` class from the flask_mail extension and Initialize it.

We now can create a module that will handle the email logic. We create the _email.py_ module inside the app directory.

app/email.py

```
from flask_mail import Message
from flask import render_template
from . import mail
```

We first import the the `Message` class from flask_mail extension. We also import the `mail` instance from the application factory module.

app/email.py

```
def mail_message(subject,template,to,**kwargs):
    sender_email = <Your Email address>

    email = Message(subject, sender=sender_email, recipients=[to])
    email.body= render_template(template + ".txt",**kwargs)
    email.html = render_template(template + ".html",**kwargs)
    mail.send(email)
```

We first create **sender_email** where we store our email address.

We then create a `mail_message` function that takes in 4 parameters which are the **subject** of the email, the **template** which is where we create the message body, We pass in the template without an extension because we need to create the text version and a HTML version, the **recipient** and any keyword arguments.

We then create a `Message` instance and pass in the `subject`, the `sender_email` and the `recipient`.

We then set up the email body and HTML. We use the `send` method of the `mail`instance and pass in the email instance.

Sending Welcome Email

app/auth/views.py

```
from ..email import mail_message
```

We import the `mail_message` function in our auth blueprint _views.py_ file.

app/auth/views.py

```
@auth.route('/register',methods = ["GET","POST"])
def register():
    form = RegistrationForm()
    if form.validate_on_submit():
        user = User(email = form.email.data, username = form.username.data,password = form.pa
ssword.data)
        db.session.add(user)
        db.session.commit()

        mail_message("Welcome to watchlist","email/welcome_user",user.email,user=user)

        return redirect(url_for('auth.login'))
```

```
        title = "New Account"
    return render_template('auth/register.html',registration_form = form)
```

We then call the function inside our `register` view function. We pass in the the subject and template file where our message body will be stored. We then pass in the new user's email address which we get from the registration form. We then pass in a user as a keyword argument.

We can then create the email body and html templates. Create 2 new files `welcome_user.html` and `welcome_user.txt` inside an email subfolder in our template folder.

app/templates/email/welcome_user.txt

```
Hello {{user.username}}
Welcome to Watchlist
We have just seen you have signed up for our application and want to welcome you to the famil
y. Please enjoy the reviews of all your  favourite films.
```

Now each time a user registers with our application they receive a welcome email from our application. You can find the application at this point here https://github.com/mbuthiya/watchlist/tree/28-sending-email