DEVELOPER DOCUMENTATION:

This developer documentation tells us about the code from the developer's perspective i.e how the code works according to the inputs from the user. It gives us information about the flow of data through the various lines of the code. It also sheds light upon the uses of various functions that have been used (for authentication, image upload etc) and explains their internal workings.

userUI :

- When any user registers himself into the application, his email address and password is stored in the database. The password is stored as the hashed value in the database.

```
70
71  @app.route('/register',methods=['GET', 'POST'])
72  def register():
73      """Method for user registration """
74      form = RegistrationForm(request.form)
75      if request.method == 'POST' and form.validate():
76          hash = pbkdf2_sha256.encrypt(form.password.data, rounds=200000, salt_size=16)
77          user = User(form.email.data, hash)
78          db.session.add(user)
79          db.session.commit()
80          login_user(user, remember=True)
81          flash('User Registered! ')
82          return redirect(url_for('home'))
83      return render_template('register.html',form=form)
84
```

- Authentication process takes place when a returning user enters his credentials. If the values match in the database, user is allowed to access the application.

```
53  @app.route('/login', methods=['GET', 'POST'])
54  def login():
55      """ Method to login into the webpage. Here user authentication will take place"""
56      form = LoginForm(request.form)
57      if request.method == 'POST' and form.validate():
58          next = request.args.get('next')
59          # is_safe_url should check if the url is safe for redirects.
60          if not is_safe_url(next):
61              return abort(400)
62          user = User.query.get(form.email.data)
63          if user:
64              user.authenticated = True
65              db.session.add(user)
66              db.session.commit()
67              login_user(user,remember=True)
68          return redirect(next or url_for('home'))
69      return render_template('login.html',form=form,email=request.cookies.get('email'))
```

- When the user clicks on the 'Upload Photo' tab, the application redirects the user to a form. Here the user selects and uploads image. If the file selected by the user doesn't satisfy certain extensions , then it will throw an error and give a pop up message saying '"Incorrect file extension. Please choose a png, jpg, jpeg or gif image." The uploaded file then gets saved in the s3 bucket.

```
def upload_file(req_file, cur_user):
    """Transform and upload the files on s3 and save the filename in the database"""

    # check if the post request has the file part
    if 'file' not in req_file:
        flash('No file part')
        return False
    file = req_file['file']
    # if user does not select file, browser also submit a empty part without filename
    if file.filename == '':
        flash('No selected file')
        return redirect(request.url)
    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        f_name = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        # save the file on s3
        file.save(f_name)
        s3.meta.client.upload_file(f_name, 'ece1779xdz', filename)

        # returns an array of all the names after transformation
        f_trans = transform(filename, f_name)
        flash('Photo Uploaded!')

        # saves all the file names in the db
        img = Img(filename, cur_user.email, f_trans[0], f_trans[1], f_trans[2])
        db.session.add(img)
        db.session.commit()
        return [True, filename]
    return False
```

- Once the image is uploaded by the user, its transformation takes place and all the three transformation gets saved into the s3 bucket.

```
def transform(filename, f_name):
    """
    Performs three transformations and save the transformed images on s3
    :param filename: String.
    :param f_name: String.
    :return: [string]. An array of transformed file names.
    """
    img = Image(filename=f_name)

    flopped = img.clone()
    flopped.flop()
    name_flopped = filename.split('.', 1)[0] + '_flopped.'+filename.split('.', 1)[1]
    f_name_flopped = os.path.join(app.config['UPLOAD_FOLDER'], name_flopped)
    flopped.save(filename=f_name_flopped)
    s3.meta.client.upload_file(f_name_flopped, 'ece1779xdz', filename)

    rotated = img.clone()
    rotated.rotate(45)
    name_rotated = filename.split('.', 1)[0] + '_rotated.'+filename.split('.', 1)[1]
    f_name_rotated = os.path.join(app.config['UPLOAD_FOLDER'], name_rotated)
    rotated.save(filename=f_name_rotated)
    s3.meta.client.upload_file(f_name_rotated, 'ece1779xdz', filename)

    enhanced = img.clone()
    enhanced.evaluate(operator='rightshift', value=1, channel='blue')
    enhanced.evaluate(operator='leftshift', value=1, channel='red')
    name_enhanced = filename.split('.', 1)[0] + '_enhanced.'+filename.split('.', 1)[1]
    f_name_enhanced = os.path.join(app.config['UPLOAD_FOLDER'], name_enhanced)
    enhanced.save(filename=f_name_enhanced)
    s3.meta.client.upload_file(f_name_enhanced, 'ece1779xdz', filename)

    return [name_flopped, name_rotated, name_enhanced]
```

- When the user selects 'Gallery' from the menu, all the uploaded images will be displayed. When any image is clicked a thumbnail of all the transformations of that image gets displayed. This is done using the ImageMagick.

- After the program runs successfully, you can test it on your local machine. After that you can connect to the instance by performing the following commands on the terminal.

  - Type ssh -i .ssh/ece1779.pem ubuntu@PublicIP-Of-Instance.
  - This will connect to the instance.

managerUI

The managerUI has the following functionalities :

- The following method calculates the CPU utilization for all the workers.

```
def calculate_total_cpu():
    """
    Calculate the total CPU utilization for all workers.
    :return: [integer, integer] for total CPU and number of running instances.
    """
    instances = ec2.instances.filter(
        Filters=[{'Name': 'instance-state-name', 'Values': ['running']}])
    inst_num = 0
    cpu_sum = 0
    for instance in instances:
        cpu = client.get_metric_statistics(
            Namespace='AWS/EC2',
            MetricName='CPUUtilization',
            dimensions={'InstanceId': [instance.id]},
            Period=300,
            StartTime=datetime.utcnow() - timedelta(seconds=300),
            EndTime=datetime.utcnow(),
            Statistics='Average'
        )
        for data_point in cpu['Datapoints']:
            cpu_sum += data_point['Average']
        inst_num += 1
    return [cpu_sum, inst_num]
```

- To shrink the worker pool by deleting an instance and to grow the worker pool by creating an instance thus managing the load.

```
def create_instance():
    """
    Grow the worker pool by creating a new instance.
    :return: Null
    """
    instance = ec2.create_instances(
        ImageId='ami-e3f432f5',
        InstanceType='t2.micro',
        KeyName='ece1779',
        MaxCount=1,
        MinCount=1,
        Monitoring={'Enabled': True},
        SecurityGroupIds=['sg-751f1e06'],
        DryRun=True
    )
    lb.register_instances_with_load_balancer(
        LoadBalancerName='ece1779lb',
        Instances=[{'InstanceId': instance[0].id}]
    )

def terminate_instance():
    """
    Shrink the worker pool by terminating a new instance.
    :return: Null
    """
    instances = ec2.instances.filter(
        Filters=[{'Name': 'instance-state-name', 'Values': ['running']}])
    inst_id = instances[0].id
    lb.deregister_instances_from_load_balancer(
        LoadBalancerName='ece1779lb',
        Instances=[{'InstanceId': inst_id}])
    ec2.instances.filter(InstanceIds=[inst_id]).terminate()
```

- To calculate the CPU utilization and resize the worker pool and to check if it meets the upper and lower threshold the following method is defined.

```
def resize_pool(upper_threshold, lower_threshold):
    """
    Resize the worker pool by calculating the CPU utilization and checks if it meets the upper and lower
    threshold.
    :param upper_threshold: float
    :param lower_threshold: float
    :return: int the number of running instances
    """
    cpu_sum, inst_num = calculate_total_cpu()
    while cpu_sum > upper_threshold:
        terminate_instance()
        cpu_sum, inst_num = calculate_total_cpu()

    while cpu_sum < lower_threshold:
        create_instance()
        cpu_sum, inst_num = calculate_total_cpu()
    return inst_num
```

- To check every fifteen minutes whether resizing is needed or not, the following while loop gets executed.

```
# checks every fifteen seconds if resizing is needed
while True:
    time.sleep(15)
    client = boto3.client('cloudwatch')

    # extract the parameters from the db
    cur.execute("SELECT * FROM auto_scale")
    row = cur.fetchone()
    if row:
        upper_threshold = row[0]
        lower_threshold = row[1]
        grow_ratio = row[2]
        shrink_ratio = row[3]
    else:
        print("No data in the auto_scale table.\n")
        continue

    # resize pool based on the current thresholds
    instance_num = resize_pool(upper_threshold, lower_threshold)

    # determine how many new instances need to be added or subtracted
    add_instance_num = grow_ratio * instance_num - instance_num
    subtract_instance_num = instance_num - instance_num / shrink_ratio
    delta_instance_num = abs(add_instance_num - subtract_instance_num)

    # create or terminate the instances and resize the pool
    if add_instance_num > subtract_instance_num:
        for i in range(delta_instance_num):
            create_instance()
    elif add_instance_num < subtract_instance_num:
        for i in range(delta_instance_num):
            terminate_instance()
    resize_pool(upper_threshold, lower_threshold)

    # reset the growing and shrinking ratios in the db
    set_auto_scale(upper_threshold, lower_threshold, 1, 1)
```

DEPLOYMENT INSTRUCTIONS

After connecting to the managerUI instance (you can connect by executing "ssh -i .ssh/ece1779.pem ubuntu@PublicIP-Of-Instance"), site can start by giving the following commands:

1. First you need to set the virtual environment. This can be done by the following command:
- **cd Desktop/ece1779/aws/venv/bin**
- **source activate**

2. After this going back to the ece1779 folder, go to the managerUI folder by : **cd a2/managerUI** and if starting from the root execute the command **cd Desktop/ece1779/a2/managerUI**
3. Then running the server.py file by : **python server.py**
4. After this using the instance's publicIP and the port for the managerUI that is 5000 (i.e publicIP:5000) , you can access the application.

5. Then you can go to the AWS account. A user instance(userUI) will be created. Then using the instance's publicIP and port 5001, the application will run automatically.
6. An instance, user_backup, has been created. This is the userUI whose images are being launched automatically. If you wish you can run that by connecting to the instance and then setting the virtual environment by "cd Desktop/ece1779/aws/venv/bin" then "source activate" and then executing the command (starting from the root) 'cd Desktop/ece1779/a2/userUI' and then 'server.py'. Then the instance's public IP and the port 5001.
7. If the managerUI throws an error, then go to managerUI's Security groups in AWS and change inbound permissions(HTTP, SSH, Custom TCP Rule) to MyIP from Custom. Also change the userUI's Security group's inbound rules to MyIP.

DATABASE SCHEME

We have the following two tables in the database ece1779 that our application shall be using:

1. Table Img : It has the following structure. Contains the following fields : 1) img_name : the name of the image uploaded by the user. 2) user_email : name of the user who have uploaded it. 3) img_trans1: the name of the first transformed image. 4) img_trans2 : the name of the second transformed image. 5) img_trans3 : the name of the third transformed image.

```
mysql> describe Img;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| img_name   | varchar(80) | YES  |     | NULL    |       |
| user_email | varchar(80) | YES  |     | NULL    |       |
| img_trans1 | varchar(80) | YES  |     | NULL    |       |
| img_trans2 | varchar(80) | YES  |     | NULL    |       |
| img_trans3 | varchar(80) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

2. Table Users: It has the user details. When anyone registers, it saves the email and the password in this table.

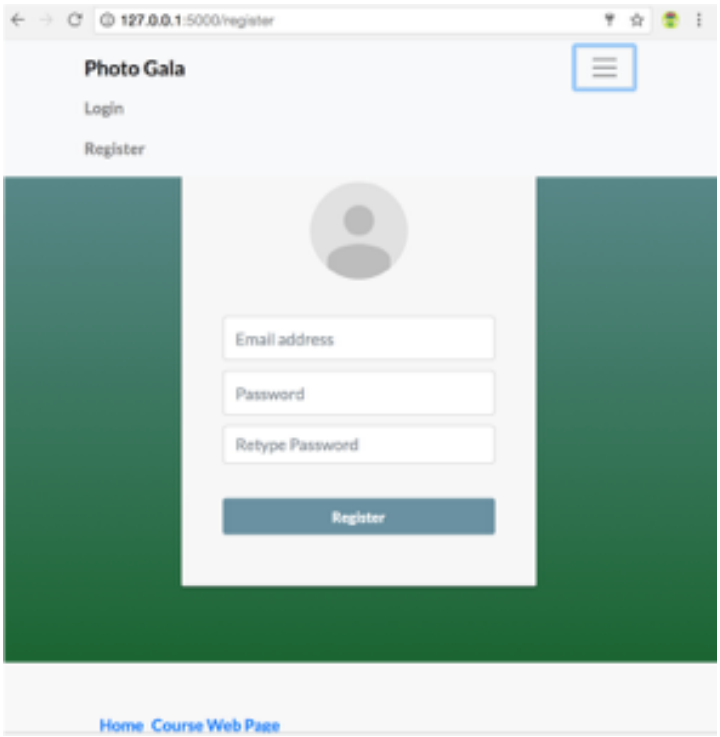
```
mysql> describe Users;
+---------------+--------------+------+-----+---------+-------+
| Field         | Type         | Null | Key | Default | Extra |
+---------------+--------------+------+-----+---------+-------+
| email         | varchar(40)  | YES  |     | NULL    |       |
| password      | varchar(120) | YES  |     | NULL    |       |
| authenticated | tinyint(1)   | YES  |     | NULL    |       |
+---------------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

USER DOCUMENTATION

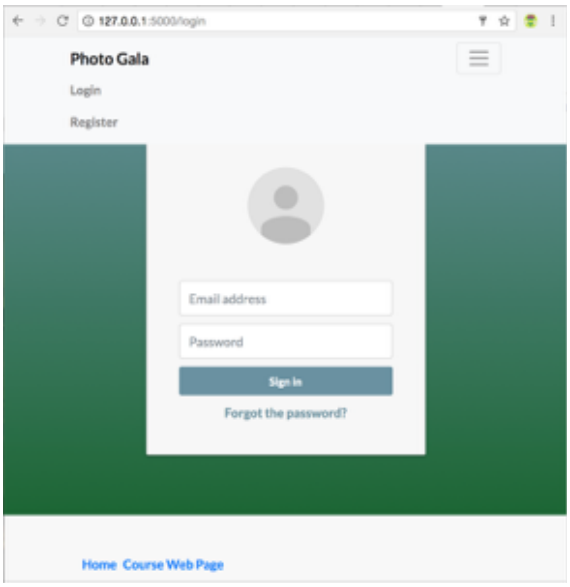
UserUI

PhotoGala is an ultimate solution for your photo storage and transformation needs. Photographs have become an integrated part of our day to day lives and we often need to edit those photos. Photo gala provides a hassle free, convenient and time-saving tool to do the same. Let's have a look at how it works.

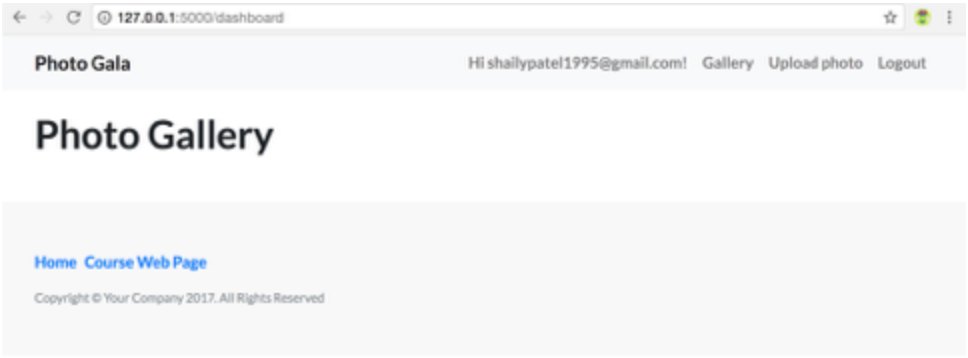
1) If you are using the application for the first time you will have to register yourself. You can do that by clicking on the menu and then selecting Register. You will see the following screen. Enter your email address and set a password.



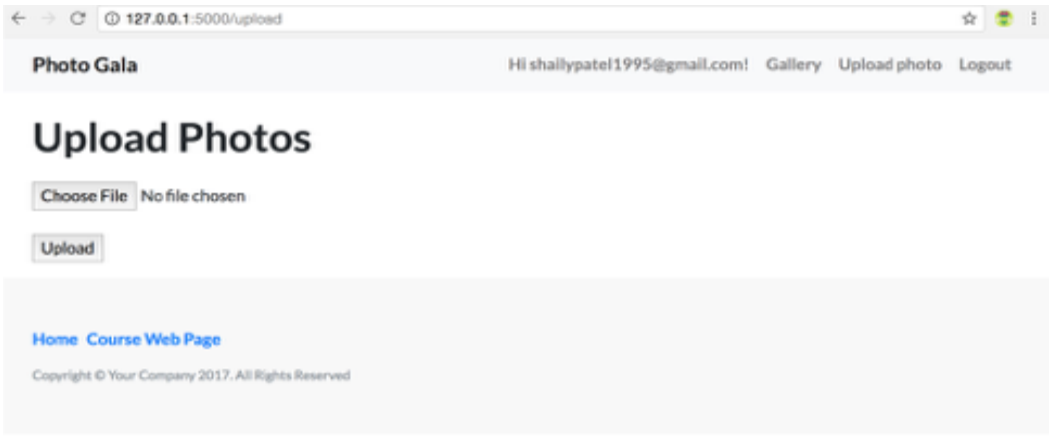
2) After registering yourself you will have to login, by entering your email address and your password, into the application.



3) After the user is authenticated, you will be able to see the dashboard. From here you can upload your photo, view your gallery and logout.



4) For uploading the image, select the Upload photos.



5) To view the uploaded photos and their respective transformations go to gallery. Select the photo and swipe right/left to view the transformation.

6) There's a link provided to the Course Web Page. If you are done with your work, logout from the application.

ManagerUI

When you connect to the managerUI, you can use functionalities like - list workers, auto-scaling, change worker pool size and delete data.

1.) List Worker : Lists the current workers.

List workers

Worker name	CPU utilization (10 min avg)
i-0ae3fb600f0007826	10.0
i-02abc8982ae4992d0	9.83

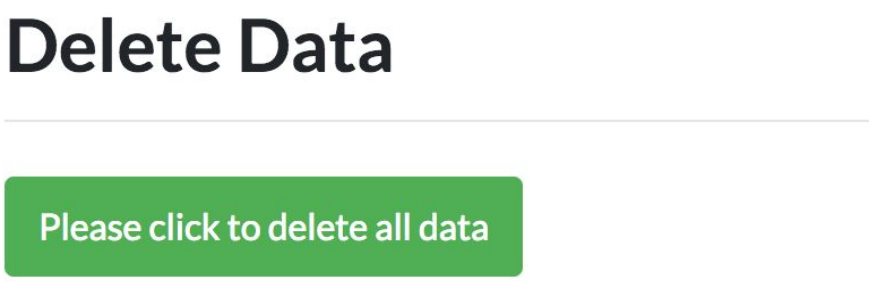
2.) Change worker pool size : It allows you to increase/decrease the worker pool by 1. That is you can increase or decrease an instance.

Change Worker Pool Size

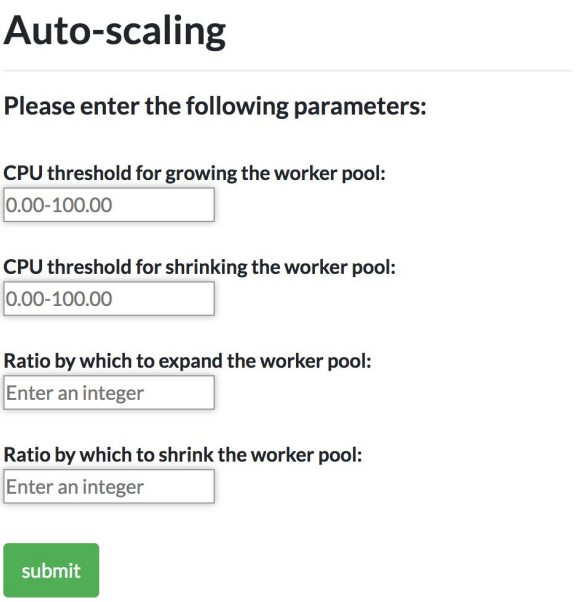
- Grow worker pool by 1
- Shrink worker pool by 1

submit

3.) Delete data : It lets you delete all the data.



4.) Auto-scaling : It allows you to set a CPU threshold for growing/shrinking the worker pool and the ratio by which we want to expand or shrink the worker pool.



NOTES:

We have used lightbox-plus-jquery.js file that includes jQuery v2.x and supports IE9+ Therefore you must use Version 9+ if you use IE.