Assignment -2

Python Programming

Assignment Date	29 September 2022
Student Name	Saravanan S
Role	Team Member 1
Student Roll Number	130719104069

Question-1:

Create User table with user with email, username, roll number, password.

Solution:

```
CREATE TABLE user(email,username,rollno,pw);
INSERT INTO user (email,username,rollno,pw) VALUES ('gohul@gmail.com', 'gohul','01','123');
INSERT INTO user (email,username,rollno,pw) VALUES ('raju@gmail.com', 'raju','02','456');
INSERT INTO user (email,username,rollno,pw) VALUES ('raman@gmail.com', 'raman','03','789');
INSERT INTO user (email,username,rollno,pw) VALUES ('sam@gmail.com', 'sam','04','101');
```

Question-2:

Perform UPDATE,DELETE Queries with user table

Solution:

```
UPDATE user SET username = 'gohul' WHERE rollno = '01';
SELECT* FROM user

DELETE FROM user WHERE username = 'raman';
SELECT* FROM user
```

Question-3:

Connect python code to db2.

Solution:

```
pip install ibm_db
from ibm_db import connect
# Careful with the punctuation here - we have 3 arguments.
# The first is a big string with semicolons in it.
# (Strings separated by only whitespace, newlines included,
#  are automatically joined together, in case you didn't know.)
# The last two are emptry strings.
connection = connect('DATABASE=<database name>;'
                     'HOSTNAME=<database ip>;'  # 127.0.0.1 or localhost works if it's local
                     'PORT=<database port>;'
                     'PROTOCOL=TCPIP;'
                     'UID=<database username>;'
                     'PWD=<username password>;', '', '')
def results(command):

    from ibm_db import fetch_assoc

    ret = []
    result = fetch_assoc(command)
    while result:
        # This builds a list in memory. Theoretically, if there's a lot of rows,
        # we could run out of memory. In practice, I've never had that happen.
        # If it's ever a problem, you could use
        #     yield result
        # Then this function would become a generator. You lose the ability to access
        # results by index or slice them or whatever, but you retain
        # the ability to iterate on them.
        ret.append(result)
        result = fetch_assoc(command)
    return ret  # Ditch this line if you choose to use a generator.
from ibm_db import tables

t = results(tables(connection))
from ibm_db import exec_immediate

sql = 'LIST * FROM ' + t[170]['TABLE_NAME']
rows = results(exec_immediate(connection, sql))
```

Question-4:

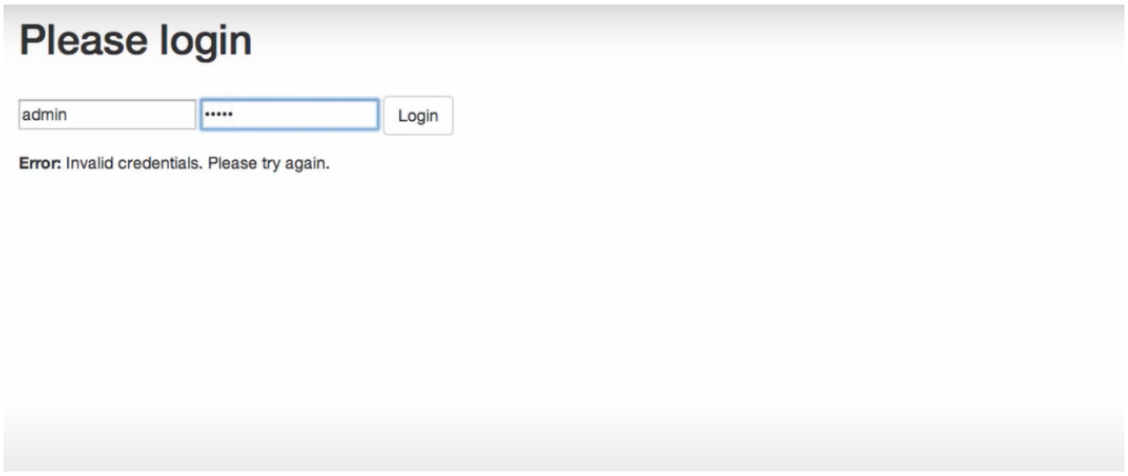
Create a flask app with registration page, login page and welcome page. By default load the registration page once the user enters all the fields store the data in database and navigate to login page authenticate user username and password. If the user is valid show the welcome page

Solution:

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    error = None
    if request.method == 'POST':
        if request.form['username'] != 'admin' or request.form['password'] != 'admin':
            error = 'Invalid Credentials. Please try again.'
        else:
            return redirect(url_for('home'))
    return render_template('login.html', error=error)
```

html code:

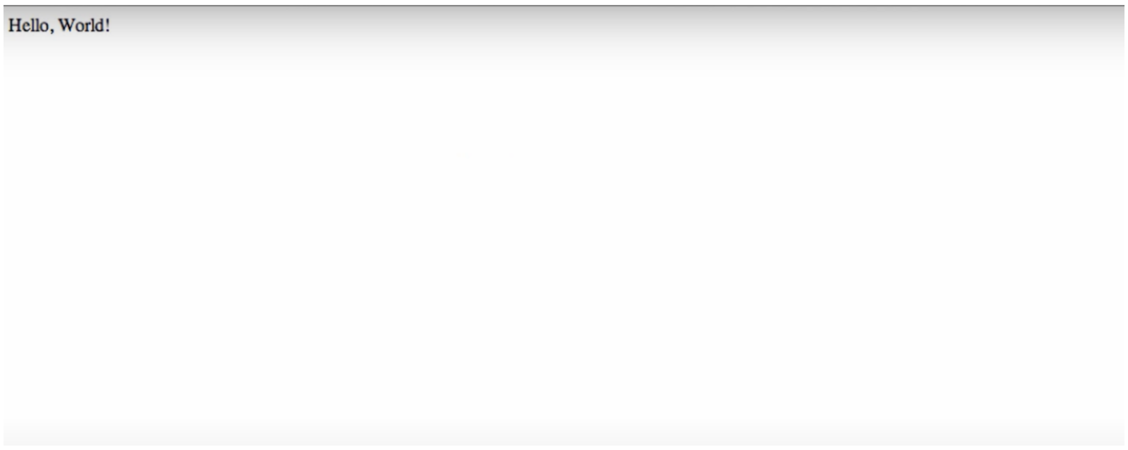
```
<html>
  <head>
    <title>Flask Intro - login page</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="static/bootstrap.min.css" rel="stylesheet" media="screen">
  </head>
  <body>
    <div class="container">
      <h1>Please login</h1>
      <br>
      <form action="" method="post">
        <input type="text" placeholder="Username" name="username" value="{{
          request.form.username }}">
         <input type="password" placeholder="Password" name="password" value="{{
          request.form.password }}">
        <input class="btn btn-default" type="submit" value="Login">
      </form>
      {% if error %}
        <p class="error"><strong>Error:</strong> {{ error }}
      {% endif %}
    </div>
  </body>
</html>
```



Please login

admin ••••• Login

Error: Invalid credentials. Please try again.



Hello, World!