App Development Project
Week 8, Lesson 2

👀 Looking Forward 👀

At the end of this lesson, you will be able to:

- Create a **retrieveUsers.html** and **retrieveCustomers.html** template and add an entry into Flask **route()** to point to it.
- Retrieve **User** and **Customer** objects persisted (stored) in **shelve** and display their data accordingly in the **Retrieve Users** and **Retrieve Customers** page.
- Add an Update User and Update Customer function to your SimpleWebApplication using **WTForms**.
- Create an **updateUser.html** and **updateCustomer.html** template and add an entry into Flask **route()** to point to it.
- Use **shelve** to persist (store) data.

1. Activity 1: Retrieving Stored Data in a Project

1.1 Create a New Retrieve Users Function for Your Flask Web Application

For the **Retrieve Users Function**, you will first write the codes in **__init__.py** before creating the Retrieve Users **template**. This way, the server-side scripts that you will be creating for your template will make more sense.

1.2 Add retrieveUsers.html to the Flask route()

Step 1: Add in a new route for **/retrieveUsers** to the Flask **route()** in **__init__.py** that points to **retrieveUsers.html**.

__init__.py

```python
@app.route('/retrieveUsers')
def retrieve_users():

    return render_template('retrieveUsers.html')

```

Step 2: Retrieve the **users_dict** object from **shelve** using the **'Users'** key.

__init__.py

```python
@app.route('/retrieveUsers')
def retrieve_users():
    users_dict = {}
    db = shelve.open('user.db', 'r')
    users_dict = db['Users']
    db.close()

    return render_template('retrieveUsers.html')

```

Step 3: Retrieve all **user** objects from the **users_dict** dictionary and store them in the **users_list** list.

__init__.py

```python
@app.route('/retrieveUsers')
def retrieve_users():
    users_dict = {}
    db = shelve.open('user.db', 'r')
    users_dict = db['Users']
    db.close()

    users_list = []
    for key in users_dict:
        user = users_dict.get(key)
        users_list.append(user)

    return render_template('retrieveUsers.html')
```

```
80
```

Step 4: Define and send the **count** variable and **users_list** list to the **Retrieve Users** template so that they can be used there.

__init__.py

```python
@app.route('/retrieveUsers')
def retrieve_users():
    users_dict = {}
    db = shelve.open('user.db', 'r')
    users_dict = db['Users']
    db.close()

    users_list = []
    for key in users_dict:
        user = users_dict.get(key)
        users_list.append(user)

    return render_template('retrieveUsers.html', count=len(users_list), users_list=users_list)

```

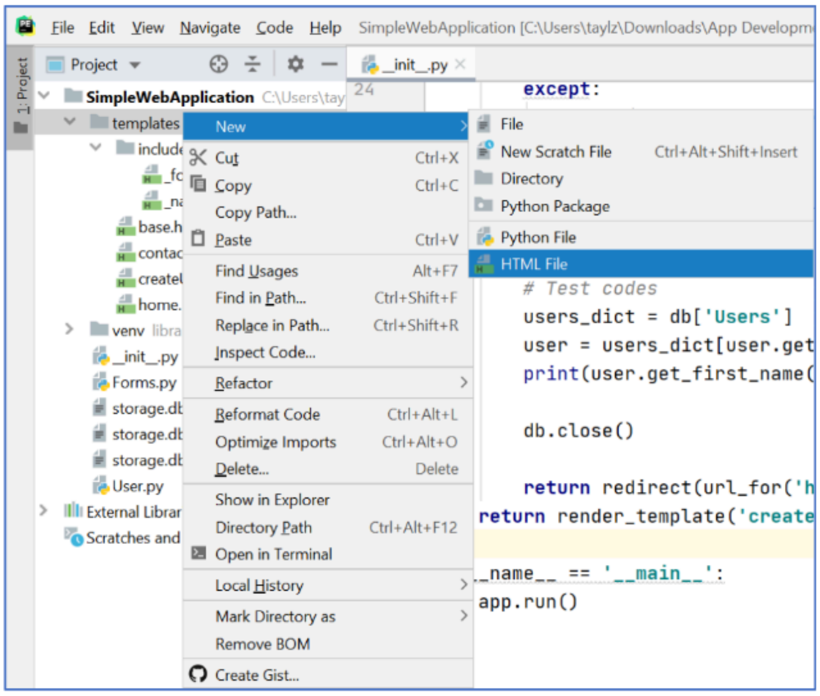
🐬 LET IT SINK IN 🐬

users_list
– The users_list list will be used in the **Retrieve Users** template to retrieve and display the details of all the **user** objects that were stored in the **users_dict** dictionary that was stored in **shelve**.

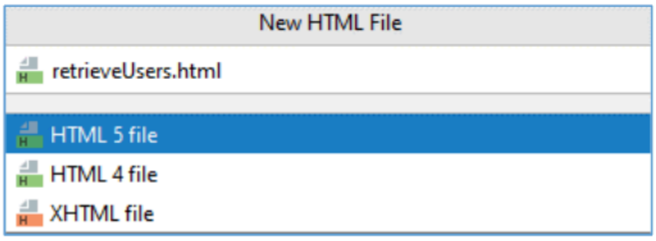
count
– The count variable will be used in the **Retrieve Users** template to give the **total number** of **users** that were stored in the **users_dict** dictionary that was stored in **shelve**.

1.2.1 Create a New Template for your Retrieve Users page

Step 1: Right-click on the **templates** folder and select **New > HTML File** to create a new HTML template called **retrieveUsers.html**.



```
        except:
            # Test codes
            users_dict = db['Users']
            user = users_dict[user.get
            print(user.get_first_name(

            db.close()

            return redirect(url_for('h
    return render_template('create

_name__ == '__main__':
    app.run()
```



retrieveUsers.html

Step 2: Delete all the auto-generated HTML codes in your **retrieveUsers.html** template and add the following codes to it.

retrieveUsers.html

```
{% extends "base.html" %}
{% block title %}Library Loan System - Retrieve Users{% endblock %}

{% block content %}
<h1 class="display-4">Retrieve Users</h1>
<div>
 <table class="table table-striped">
  <thead>
   <tr>
    <th>User ID</th>
    <th>First Name</th>
    <th>Last Name</th>
    <th>Gender</th>
    <th>Membership</th>
    <th>Remarks</th>
```

```
        <th></th>
        <th></th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td></td>
        <td></td>
        <td></td>
        <td></td>
        <td>Fellow</td>
        <td></td>
        <td><a href="#" class="btn btn-warning">Update</a></td>
        <td>
          <form action="" method="POST">
            <input type="submit" value="Delete" class="btn btn-danger">
          </form>
        </td>
      </tr>
    </tbody>
  </table>
</div>
{% endblock %}

```

Step 3: Add in a new <div></div> block to display the number of users stored in **shelve**.

retrieveUsers.html

```
{% extends "base.html" %}
{% block title %}Library Loan System - Retrieve Users{% endblock %}

{% block content %}
<h1 class="display-4">Retrieve Users</h1>
<div>
  {% if count == 0 %}
  <p>There are no users.</p>
  {% elif count == 1 %}
  <p>There is 1 user.</p>
  {% else %}
  <p>There are {{ count }} users.</p>
  {% endif %}
</div>
<div>
  <table class="table table-striped">
    <thead>
      <tr>
        <th>User ID</th>
        <th>First Name</th>
        <th>Last Name</th>
        <th>Gender</th>
        <th>Membership</th>
        <th>Remarks</th>
        <th></th>
        <th></th>
      </tr>
    </thead>
    <tbody>
      <tr>
```

```
        <td></td>
        <td></td>
        <td></td>
        <td></td>
        <td>Fellow</td>
        <td></td>
        <td><a href="#" class="btn btn-warning">Update</a></td>
        <td>
          <form action="" method="POST">
            <input type="submit" value="Delete" class="btn btn-danger">
          </form>
        </td>
      </tr>
    </tbody>
  </table>
</div>
{% endblock %}


```

The appropriate <p></p> block will be shown depending on the conditions specified by the **if…elif…else** statements within the **{% %}** Jinja2 **server-side** script tags.

‼ IMPORTANT ‼

Note that the **Jinja2** code used between the **{% %} server-side** script tags are very similar to Python, but they are **not** Python code.

DID YOU KNOW

The {{ }} Jinja **server-side** script tags are used to **print out** variables or expressions.

There are a few kinds of Jinja **server-side** script tags:

- {% ... %} for **Statements**
- {{ ... }} for **Expressions** to print to the template output
- {# ... #} for **Comments** not included in the template output
- # ... ## for **Line Statements**

More information can be found here: https://jinja.palletsprojects.com/en/2.11.x/templates/

Step 4: Add in a **for** loop to display the details of all the **user** objects stored in the **users_list** list within the <tbody></tbody> tags.

retrieveUsers.html

```
{% extends "base.html" %}
{% block title %}Library Loan System - Retrieve Users{% endblock %}

{% block content %}
<h1 class="display-4">Retrieve Users</h1>
<div>
```

```
  {% if count == 0 %}
  <p>There are no users.</p>
  {% elif count == 1 %}
  <p>There is 1 user.</p>
  {% else %}
  <p>There are {{ count }} users.</p>
  {% endif %}
</div>
<div>
  <table class="table table-striped">
    <thead>
      <tr>
        <th>User ID</th>
        <th>First Name</th>
        <th>Last Name</th>
        <th>Gender</th>
        <th>Membership</th>
        <th>Remarks</th>
        <th></th>
        <th></th>
      </tr>
    </thead>
    <tbody>
    {% for user in users_list %}
      <tr>
        <td>{{ user.get_user_id() }}</td>
        <td>{{ user.get_first_name() }}</td>
        <td>{{ user.get_last_name() }}</td>
        <td>{{ user.get_gender() }}</td>
        {% if user.get_membership() == "F" %}
        <td>Fellow</td>
        {% elif user.get_membership() == "S" %}
        <td>Senior</td>
        {% elif user.get_membership() == "P" %}
        <td>Professional</td>
        {% endif %}
        <td>{{ user.get_remarks() }}</td>
        <td><a href="#" class="btn btn-warning">Update</a></td>
        <td>
          <form action="" method="POST">
            <input type="submit" value="Delete" class="btn btn-danger">
          </form>
        </td>
      </tr>
    {% endfor %}
    </tbody>
  </table>
</div>
{% endblock %}

```

1.2.2 Modify the Navigation Bar and Run Your SimpleWebApplication

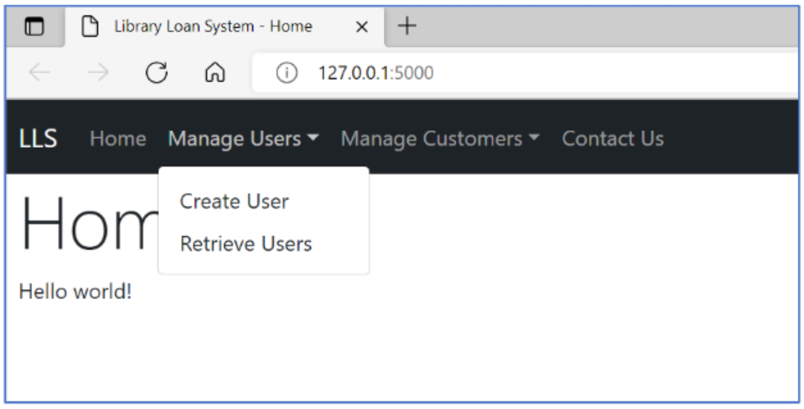
Step 1: Modify the **Retrieve Users** link in **_navbar.html** to point to **/retrieveUsers**.

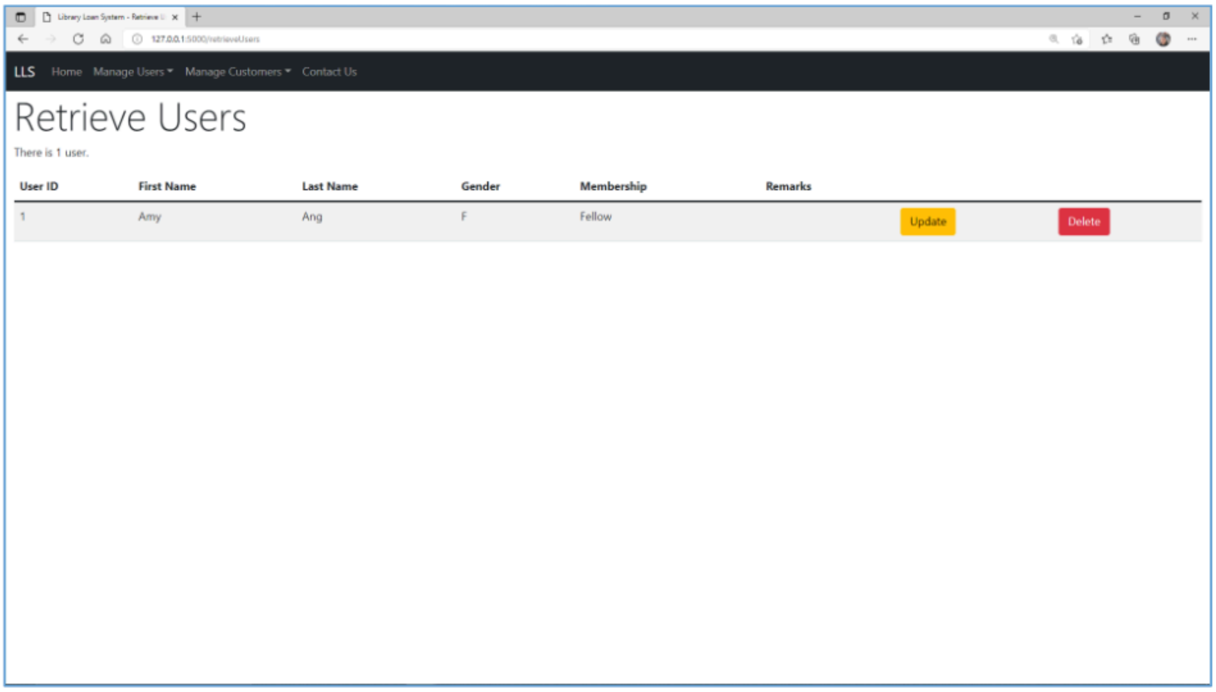
_navbar.html

```
<li class="nav-item dropdown">
  <a class="nav-link dropdown-toggle" href="#" id="navbardrop" data-toggle="dropdown">
```

```
   Manage Users
  </a>
  <ul class="dropdown-menu">
    <li><a class="dropdown-item" href="/createUser">Create User</a></li>
    <li><a class="dropdown-item" href="/retrieveUsers">Retrieve Users</a></li>
  </ul>
</li>
```

Step 2: Run your **SimpleWebApplication** and click on http://127.0.0.1:5000/. Then click on **Manage Users > Retrieve Users** from the **navbar**.




Library Loan System - Retrieve U
127.0.0.1:5000/retrieveUsers
LLS
Home
Manage Users
Manage Customers
Contact Us

Retrieve Users

There is 1 user.

User ID	First Name	Last Name	Gender	Membership	Remarks		
1	Amy	Ang	F	Fellow		Update	Delete

😲 **THINK ABOUT IT** 😲

Where did the nicely formatted, **table** and **Update** and **Delete** buttons come from? As you have imported Bootstrap into your **base.html** template and extended it in your **retrieveUsers.html** template, the **table** and **Update** and **Delete** buttons are formatted nicely by Bootstrap.

If you look at the **HTML** codes for the **table** and **Update** and **Delete** buttons, you will notice that their **class** attributes are filled up using Bootstrap-defined **class names** which correspond to the format specified by Bootstrap **CSS**.

1.2.3 Modify the create_user() Method to Redirect to /retrieveUsers

Step 1: Remove the test codes for **create_user()**.

__init__.py

```
@app.route('/createUser', methods=['GET', 'POST'])
def create_user():
    create_user_form = CreateUserForm(request.form)
    if request.method == 'POST' and create_user_form.validate():
        users_dict = {}
        db = shelve.open('user.db', 'c')

        try:
            users_dict = db['Users']
        except:
            print("Error in retrieving Users from user.db.")

        user = User.User(create_user_form.first_name.data, create_user_form.last_name.data,
create_user_form.gender.data, create_user_form.membership.data, create_user_form.remarks.data)
        users_dict[user.get_user_id()] = user
        db['Users'] = users_dict

        ~~# Test codes~~
~~        users_dict = db['Users']~~
~~        user = users_dict[user.get_user_id()]~~
~~        print(user.get_first_name(), user.get_last_name(), "was stored in user.db successfully with user_id ==",~~
~~user.get_user_id())~~

        db.close()

        return redirect(url_for('home'))
    return render_template('createUser.html', form=create_user_form)
```

Step 2: Redirect the user to the url_for('retrieve_users') instead of url_for('home').

Now the user will be redirected to the **Retrieve Users** page immediately after a new user is created. You no longer need to rely on test codes to check if a new user was added to **shelve**.

__init__.py

```
@app.route('/createUser', methods=['GET', 'POST'])
def create_user():
    create_user_form = CreateUserForm(request.form)
    if request.method == 'POST' and create_user_form.validate():
        users_dict = {}
        db = shelve.open('user.db', 'c')

        try:
            users_dict = db['Users']
        except:
```

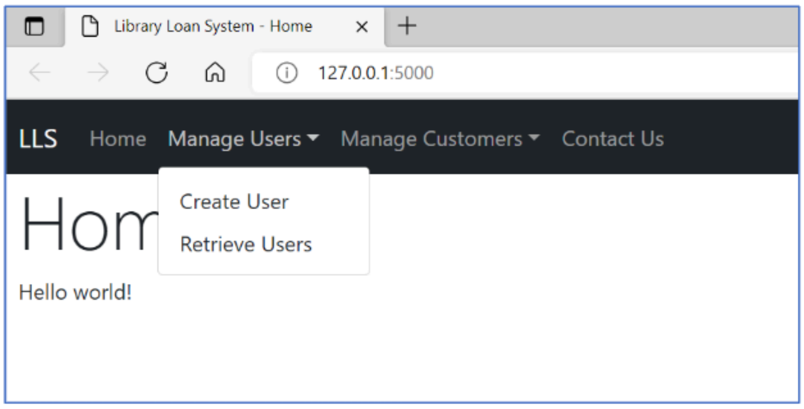
```
        print("Error in retrieving Users from user.db.")

    user = User.User(create_user_form.first_name.data, create_user_form.last_name.data,
create_user_form.gender.data, create_user_form.membership.data, create_user_form.remarks.data)
    users_dict[user.get_user_id()] = user
    db['Users'] = users_dict

    db.close()

    return redirect(url_for('retrieve_users'))
  return render_template('createUser.html', form=create_user_form)


```

LET IT SINK IN

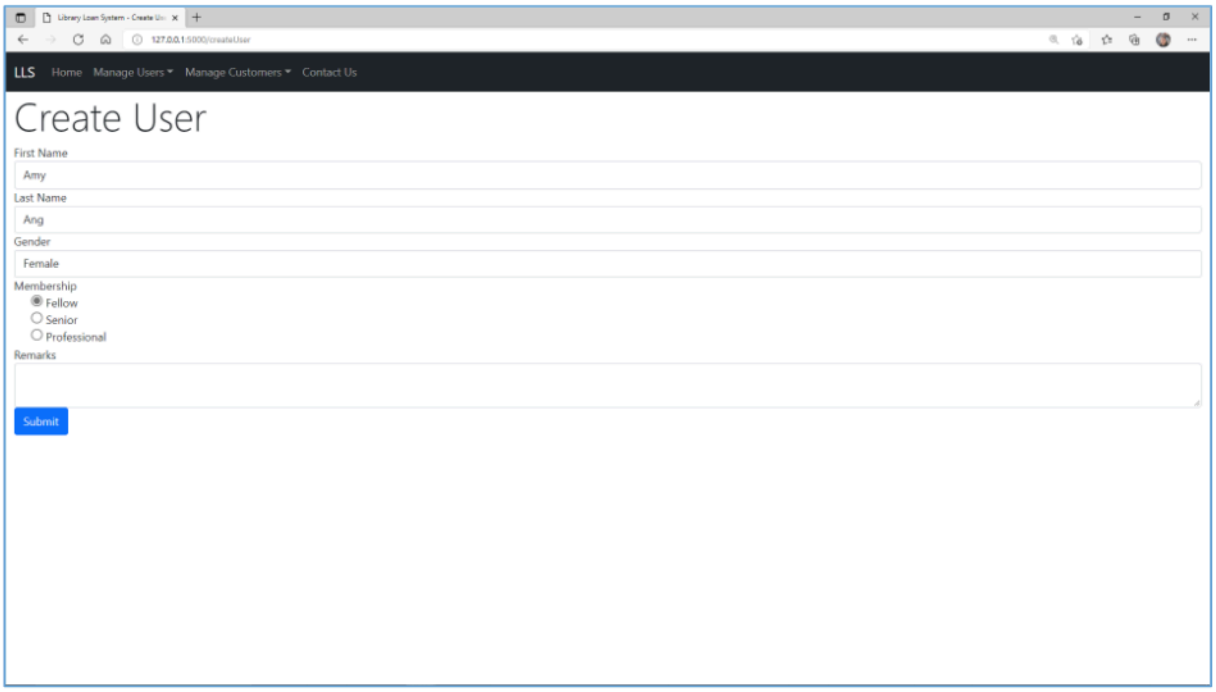
Why is the parameter for url_for() **'retrieve_users'** and not **'/retrieveUsers'**?

You will notice that the url_for() Flask function returns the relative URL **'/retrieveUsers'** when you provide the Python function name of **'retrieve_users'** as the parameter. The URL is defined in the its Flask **route()** decorator @app.route(**'/retrieveUsers'**).

Step 3: Click on the button in the Run window to **rerun** your **SimpleWebApplication** and click on http://127.0.0.1:5000/. Then click on **Manage Users > Create User** from the **navbar**.



Step 4: Fill up all required fields and click **Submit**. If the form validation succeeds, you will be redirected to **/retrieveUsers**.



Create User

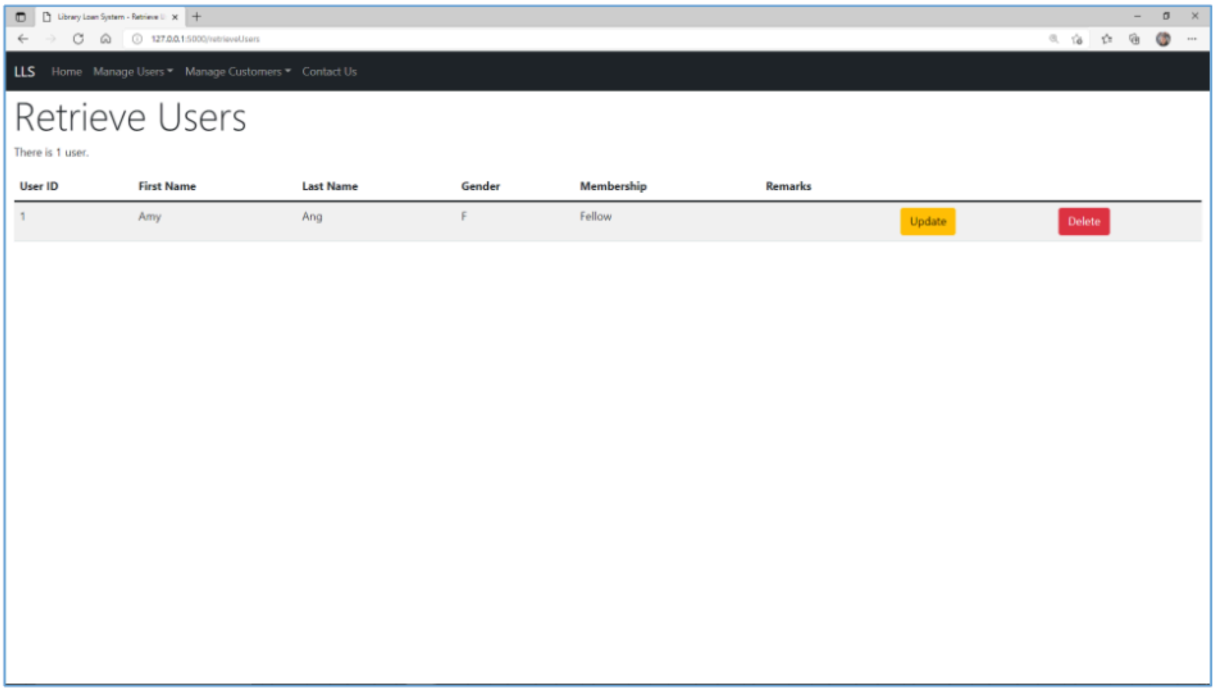
First Name: Amy

Last Name: Ang

Gender: Female

Membership
(● Fellow) ○ Senior ○ Professional

Remarks:


Library Loan System - Retrieve U
127.0.0.1:5000/retrieveUsers
LLS
Home
Manage Users
Manage Customers
Contact Us

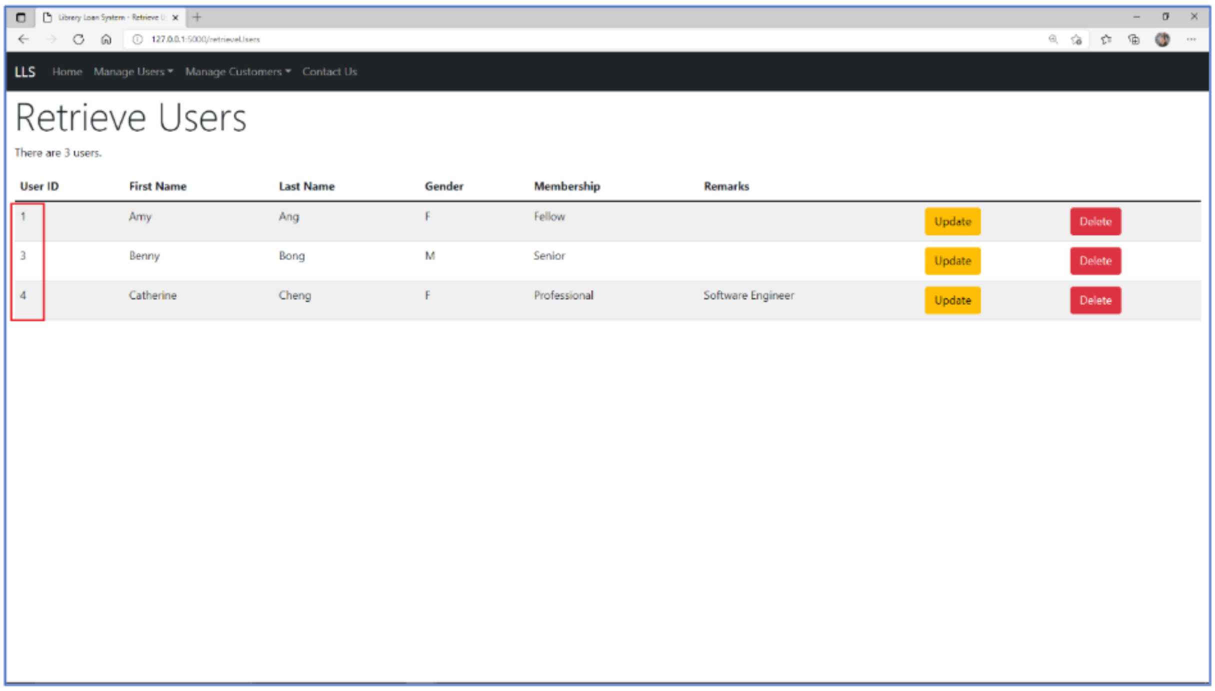
Retrieve Users

There is 1 user.

User ID	First Name	Last Name	Gender	Membership	Remarks		
1	Amy	Ang	F	Fellow		Update	Delete

Step 5: Create **2 more** users with the following details:

First Name	Last Name	Gender	Membership	Remarks
Benny	Bong	M	Senior	
Catherine	Cheng	F	Professional	Software Engineer


Library Loan System - Retrieve U
127.0.0.1:5000/retrieveUsers
LLS
Home
Manage Users
Manage Customers
Contact Us
Update
Delete

Retrieve Users

There are 3 users.

User ID	First Name	Last Name	Gender	Membership	Remarks
1	Amy	Ang	F	Fellow	
3	Benny	Bong	M	Senior	
4	Catherine	Cheng	F	Professional	Software Engineer

THINK ABOUT IT

Notice the **User ID** sequence. Why do you think **User ID 2** is missing?

Remember when we created a new **Customer**, a new **User** is also created because **Customer** inherits from **User**, thus incrementing the **User's** class attribute **count_id** in the process. If we retrieve the **Customer** data from the persistent storage, we should find **User ID 2** there (please refer to page 16).

1.3 Add retrieveCustomers.html to the Flask route()

Add in a new route for **/retrieveCustomers** to the Flask **route()** in **__init__.py** that points to **retrieveCustomers.html**.

__init__.py

```
@app.route('/retrieveCustomers')
def retrieve_customers():
    customers_dict = {}
    db = shelve.open('customer.db', 'r')
    customers_dict = db['Customers']
    db.close()

```

```
customers_list = []
for key in customers_dict:
    customer = customers_dict.get(key)
    customers_list.append(customer)

return render_template('retrieveCustomers.html', count=len(customers_list), customers_list=customers_list)

```

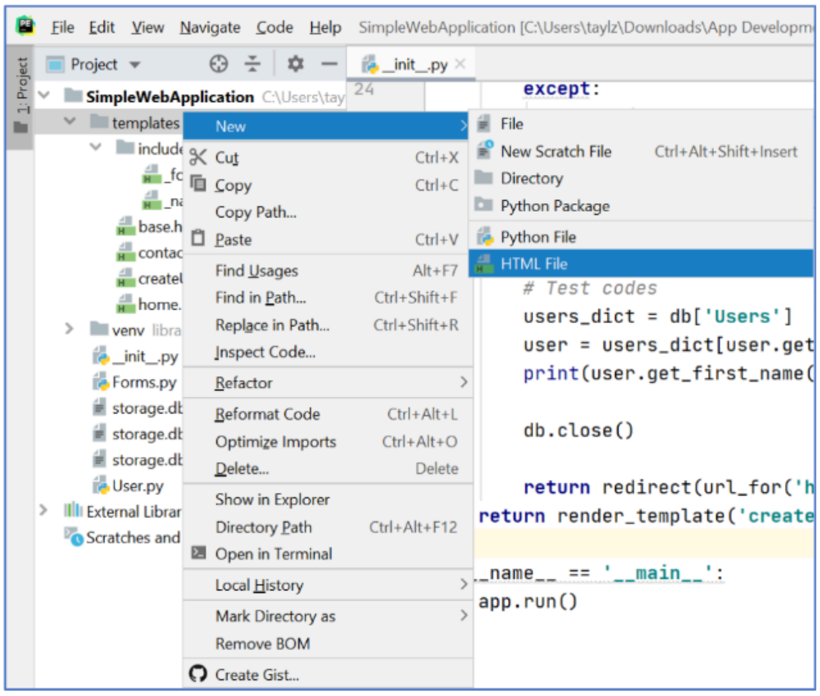
LET IT SINK IN

customers_list
– The customers_list list will be used in the **Retrieve Customers** template to retrieve and display the details of all the **customer** objects that were stored in the **customers_dict** dictionary that was stored in **shelve**.

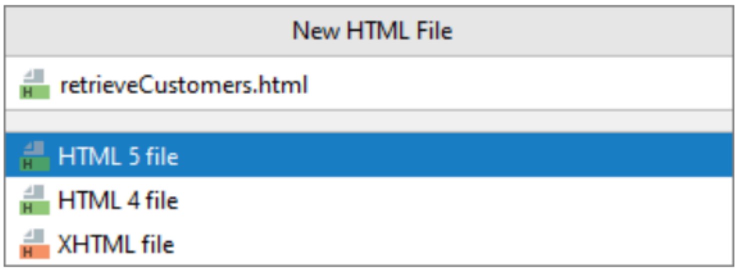
count
– The count variable will be used in the **Retrieve Customers** template to give the **total number** of **customers** that were stored in the **customers_dict** dictionary that was stored in **shelve**.

1.3.1 Create a New Template for your Retrieve Customers page

Step 1: Right-click on the **templates** folder and select **New > HTML File** to create a new HTML template called **retrieveCustomers.html**.



```
    except:
        # Test codes
        users_dict = db['Users']
        user = users_dict[user.get
        print(user.get_first_name(

        db.close()

        return redirect(url_for('h
    return render_template('create

_name__ == '__main__':
app.run()
```



Step 2: Delete all the auto-generated HTML codes in your **retrieveCustomers.html** template and add the following codes to it.

retrieveCustomers.html

```
{% extends "base.html" %}
{% block title %}Library Loan System - Retrieve Customers{% endblock %}

{% block content %}
<h1 class="display-4">Retrieve Customers</h1>
<div>
 {% if count == 0 %}
 <p>There are no customers.</p>
 {% elif count == 1 %}
 <p>There is 1 customer.</p>
 {% else %}
 <p>There are {{ count }} customers.</p>
 {% endif %}
</div>
<div>
 <table class="table table-striped">
   <thead>
     <tr>
       <th>User ID</th>
       <th>Customer ID</th>
       <th>First Name</th>
       <th>Last Name</th>
       <th>Gender</th>
       <th>Email</th>
       <th>Date Joined</th>
       <th>Address</th>
       <th>Membership</th>
       <th>Remarks</th>
       <th></th>
       <th></th>
     </tr>
   </thead>
   <tbody>
     {% for customer in customers_list %}
       <tr>
         <td>{{ customer.get_user_id() }}</td>
         <td>{{ customer.get_customer_id() }}</td>
         <td>{{ customer.get_first_name() }}</td>
         <td>{{ customer.get_last_name() }}</td>
         <td>{{ customer.get_gender() }}</td>
         <td>{{ customer.get_email() }}</td>
         <td>{{ customer.get_date_joined() }}</td>
         <td>{{ customer.get_address() }}</td>
         {% if customer.get_membership() == "F" %}
```

```
          <td>Fellow</td>
        {% elif customer.get_membership() == "S" %}
          <td>Senior</td>
        {% elif customer.get_membership() == "P" %}
          <td>Professional</td>
        {% endif %}
        <td>{{ customer.get_remarks() }}</td>
        <td><a href="#" class="btn btn-warning">Update</a></td>
        <td>
          <form action="" method="POST">
            <input type="submit" value="Delete" class="btn btn-danger">
          </form>
        </td>
      </tr>
    {% endfor %}
    </tbody>
  </table>
</div>
{% endblock %}

```

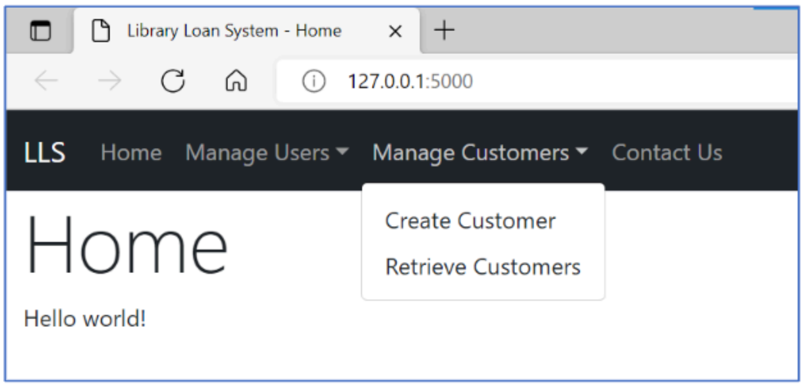
1.3.2 Modify the Navigation Bar and Run Your SimpleWebApplication

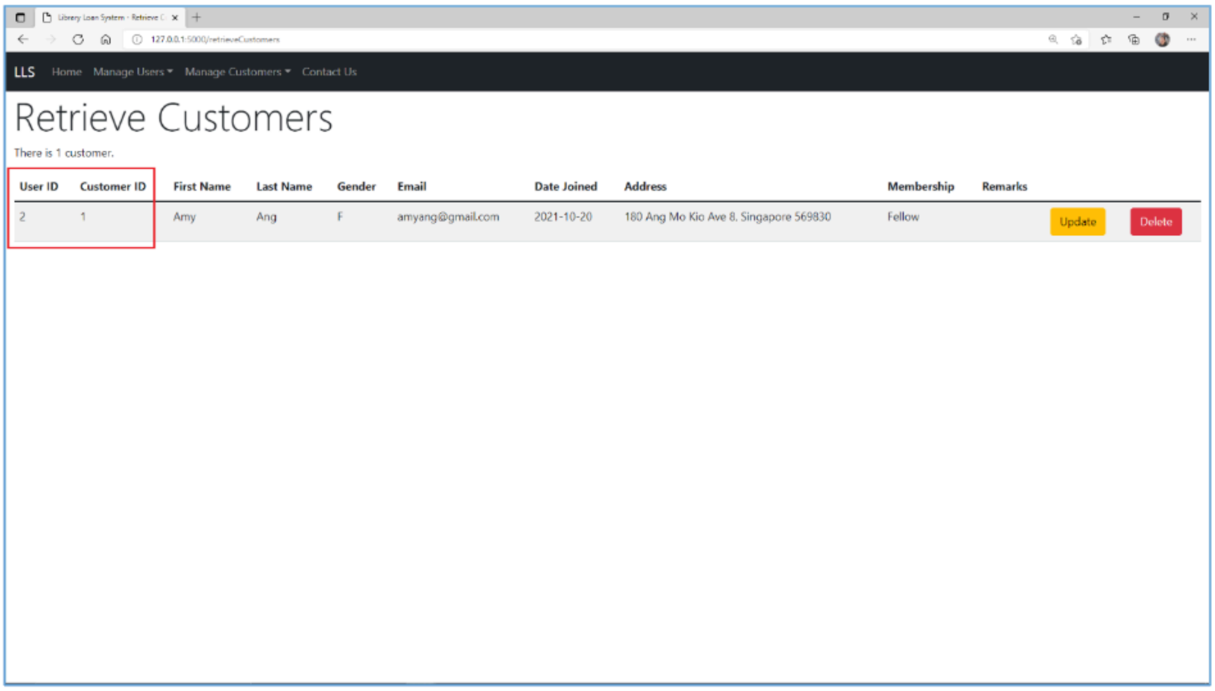
Step 1: Modify the **Retrieve Customers** link in **_navbar.html** to point to **/retrieveCustomers**.

_navbar.html

```
<li class="nav-item dropdown">
  <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown">Manage
Customers</a>
  <ul class="dropdown-menu">
    <li><a class="dropdown-item" href="/createCustomer">Create Customer</a></li>
    <li><a class="dropdown-item" href="/retrieveCustomers">Retrieve Customers</a></li>
  </ul>
</li>


```

Step 2: Run your **SimpleWebApplication** and click on http://127.0.0.1:5000/. Then click on **Manage Customers > Retrieve Customers** from the **navbar**.




Library Loan System - Retrieve C
127.0.0.1:5000/retrieveCustomers
LLS
Home
Manage Users
Manage Customers
Contact Us

Retrieve Customers

There is 1 customer.

User ID	Customer ID	First Name	Last Name	Gender	Email	Date Joined	Address	Membership	Remarks		
2	1	Amy	Ang	F	amyang@gmail.com	2021-10-20	180 Ang Mo Kio Ave 8, Singapore 569830	Fellow		Update	Delete

LET IT SINK IN

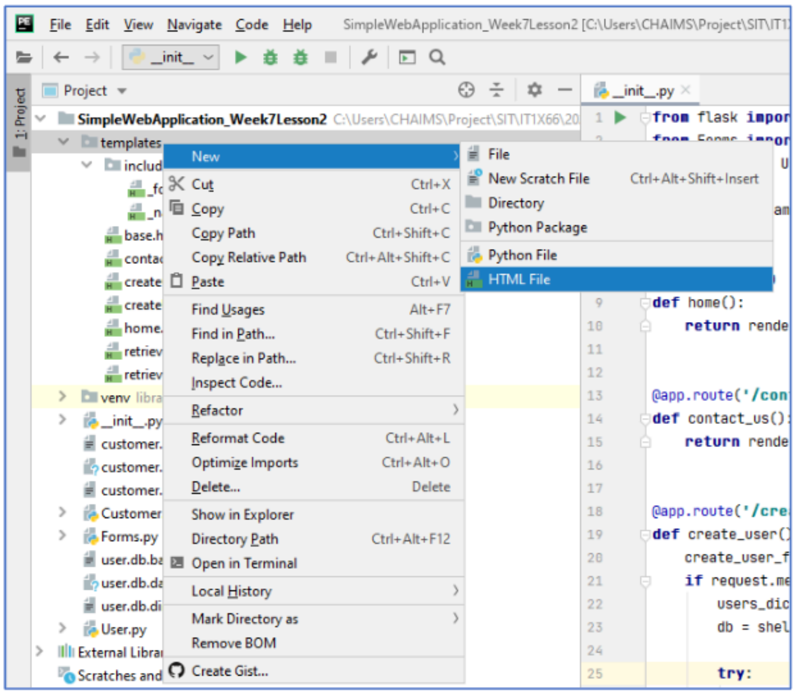
Notice the difference in the User ID and the Customer ID. While there may be only one customer, the total number of users may be more than one during inheritance.

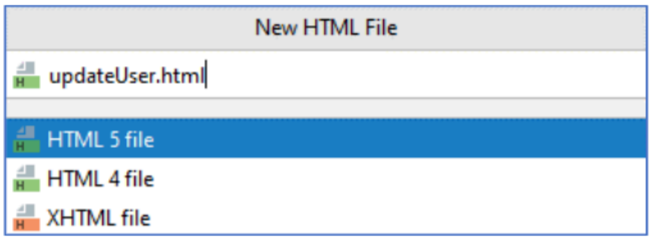
2. Activity 2: Updating Stored Data in a Project

2.1 Add an Update User Function to Your SimpleWebApplication

2.1.1 Create a New Template for your Update User page

Step 1: Right-click on the **templates** folder and select **New > HTML File** to create a new HTML template called **updateUser.html**.





Step 2: Delete all the auto-generated codes in your **updateUser.html** template and add the following codes to it.

Alternatively, you could copy all the codes from **createUser.html** to **updateUser.html** and modify them to match the following codes.

updateUser.html

```
{% extends "base.html" %}
{% block title %}Library Loan System - Update User{% endblock %}

{% block content %}
{% from "includes/_formHelper.html" import render_field %}

```

```
<h1 class="display-4">Update User</h1>

<form method="POST" action="">
  <div class="form-group">
    {{ render_field(form.first_name, class="form-control") }}
  </div>
  <div class="form-group">
    {{ render_field(form.last_name, class="form-control") }}
  </div>
  <div class="form-group">
    {{ render_field(form.gender, class="form-control") }}
  </div>
  <div class="form-group">
    {{ render_field(form.membership, class="form-check", style="list-style-type:none") }}
  </div>
  <div id="prof">
    <div class="form-group">
      {{ render_field(form.remarks, class="form-control") }}
    </div>
  </div>
  <input type="submit" value="Submit" class="btn btn-primary"/>
</form>
{% endblock %}


```

2.1.2 Add updateUser.html to the Flask route()

Step 1: Add in a new route for **/updateUser** to the Flask **route()** in **__init__.py** that points to **updateUser.html**.

__init__.py

```
@app.route('/updateUser/<int:id>/', methods=['GET', 'POST'])
def update_user(id):

    return render_template('updateUser.html')

```

LET IT SINK IN

Adding **/<int:id>/** to the **route()** decorator allows the **Update User** page to take in id as an `int` parameter through its **URL**, e.g. http://127.0.0.1:5000/updateUser/**1**/ where **1** is the user ID. The id is received as a **parameter** through the updateUser(**id**) function as an integer called **id**.

Step 2: Create a new **update_user_form** object from the **CreateUserForm** class and provide **request.form** as a parameter. Include it as **form=update_user_form** for rendering of the **updateUser.html** template.

__init__.py

```
@app.route('/updateUser/<int:id>/', methods=['GET', 'POST'])
def update_user(id):


    update_user_form = CreateUserForm(request.form)

    return render_template('updateUser.html', form=update_user_form)
```

Step 3: Add in an **if** statement to handle a validated **Update User** form **submission** and an **else** statement to handle when the **Update User** page is **first requested** before any update is submitted.

__init__.py

```
@app.route('/updateUser/<int:id>/', methods=['GET', 'POST'])
def update_user(id):
    update_user_form = CreateUserForm(request.form)
    if request.method == 'POST' and update_user_form.validate():
        return redirect(url_for('retrieve_users'))
    else:
        return render_template('updateUser.html', form=update_user_form)


```

Step 4: Under the **else** statement, add in the following codes to handle when the **Update User** page is **first requested**.

__init__.py

```
@app.route('/updateUser/<int:id>/', methods=['GET', 'POST'])
def update_user(id):
    update_user_form = CreateUserForm(request.form)
    if request.method == 'POST' and update_user_form.validate():
        return redirect(url_for('retrieve_users'))
    else:
        users_dict = {}
        db = shelve.open('user.db', 'r')
        users_dict = db['Users']
        db.close()

        user = users_dict.get(id)
        update_user_form.first_name.data = user.get_first_name()
        update_user_form.last_name.data = user.get_last_name()
        update_user_form.gender.data = user.get_gender()
        update_user_form.membership.data = user.get_membership()
        update_user_form.remarks.data = user.get_remarks()

        return render_template('updateUser.html', form=update_user_form)

```

THINK ABOUT IT

The codes retrieve the **users_dict** from **shelve** and then retrieves the **user** object of the selected user from the **users_dict** using the **User ID** of the user as the **key**.

The details of the **user** object are then used to populate the **update_user_form** that is used to display the corresponding user's **current** details on the **Update User** page before any update occurs.

Hold on, but where does the **id** in user = users_dict.get(**id**) come from?

It came from the **id** parameter of the **update_user()** function defined for the **Update User** function in **__init__.py**. Its Flask **route()** decorator was defined using **/update_user/<int:id>/** which takes in **id** as an **int** parameter through its **URL**.

Step 5: Under the **if** statement, add in the following codes to handle a validated **update_user_form** submission from the **Update User** page.

__init__.py

```
@app.route('/updateUser/<int:id>/', methods=['GET', 'POST'])
def update_user(id):
    update_user_form = CreateUserForm(request.form)
    if request.method == 'POST' and update_user_form.validate():
        users_dict = {}
        db = shelve.open('user.db', 'w')
        users_dict = db['Users']

        user = users_dict.get(id)
        user.set_first_name(update_user_form.first_name.data)
        user.set_last_name(update_user_form.last_name.data)
        user.set_gender(update_user_form.gender.data)
        user.set_membership(update_user_form.membership.data)
        user.set_remarks(update_user_form.remarks.data)

        db['Users'] = users_dict
        db.close()

        return redirect(url_for('retrieve_users'))
    else:
        users_dict = {}
        db = shelve.open('user.db', 'r')
        users_dict = db['Users']
        db.close()

        user = users_dict.get(id)
        update_user_form.first_name.data = user.get_first_name()
        update_user_form.last_name.data = user.get_last_name()
        update_user_form.gender.data = user.get_gender()
        update_user_form.membership.data = user.get_membership()
        update_user_form.remarks.data = user.get_remarks()

        return render_template('updateUser.html', form=update_user_form)
```

🐬 LET IT SINK IN 🐬

The corresponding **user** object is retrieved from the **users_dict** which was retrieved from **shelve**. The **user** object is then updated using the new values retrieved from the fields of the **update_user_form** from the **Update User** page. Finally, the **users_dict** that now has the updated **user** object is stored into **shelve** again.

2.1.3 Modify the Retrieve Users Page and Run Your SimpleWebApplication

Step 1: Modify the **Update** link in the **retrieveUsers.html** template to point to **/updateUser/{{user.get_user_id()}}**.

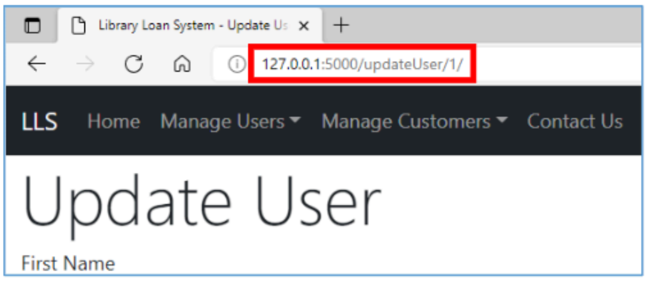
The modified link points to the **Update User** page and provides the corresponding **user_id** in the **URL** for each user that is displayed on the **Retrieve Users** page.

retrieveUsers.html

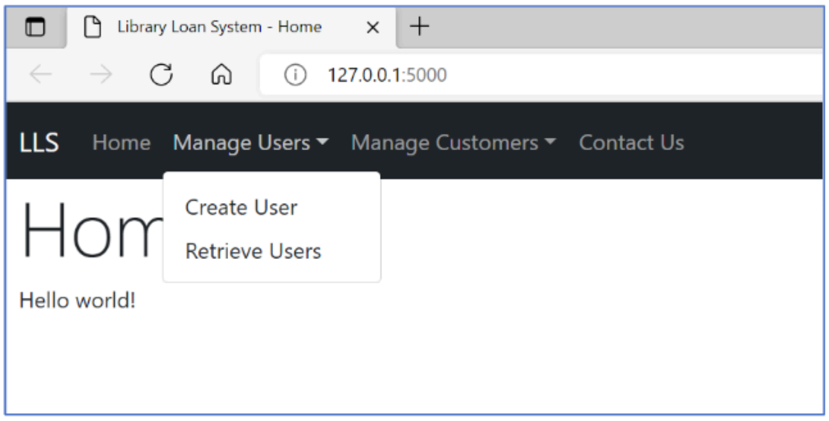
```
      <tbody>
  {% for user in users_list %}
   <tr>
    <td>{{ user.get_user_id() }}</td>
    <td>{{ user.get_first_name() }}</td>
    <td>{{ user.get_last_name() }}</td>
    <td>{{ user.get_gender() }}</td>
    {% if user.get_membership() == 'F' %}
    <td>Fellow</td>
    {% elif user.get_membership() == 'S' %}
    <td>Senior</td>
    {% elif user.get_membership() == 'P' %}
    <td>Professional</td>
    {% endif %}
    <td>{{ user.get_remarks() }}</td>
    <td><a href="/updateUser/{{user.get_user_id()}}" class="btn btn-warning">Update</a></td>
    <td>
     <form action="" method="POST">
      <input type="submit" value="Delete" class="btn btn-danger">
     </form>
    </td>
   </tr>
  {% endfor %}
  </tbody>
```

THINK ABOUT IT

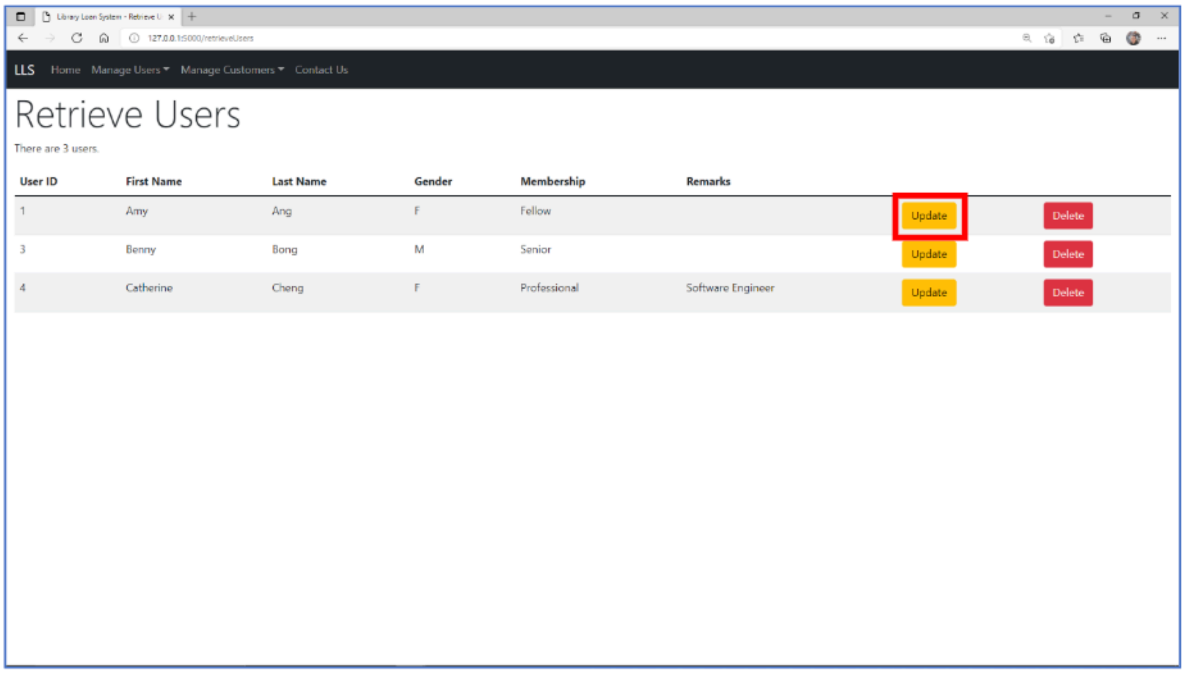
Note that the user's **user_id** will be sent through the **URL** once the **Update** button is clicked and received by the **update_user(id)** function in **__init__.py** as **id** on the server-side.



Step 2: Run your **SimpleWebApplication** and click on http://127.0.0.1:5000/. Then click on **Manage Users > Retrieve Users** from the **navbar**.



Step 3: Click on the **Update** button for **Amy Ang**.


Library Loan System - Retrieve U
127.0.0.1:5000/retrieveUsers
LLS
Home
Manage Users
Manage Customers
Contact Us

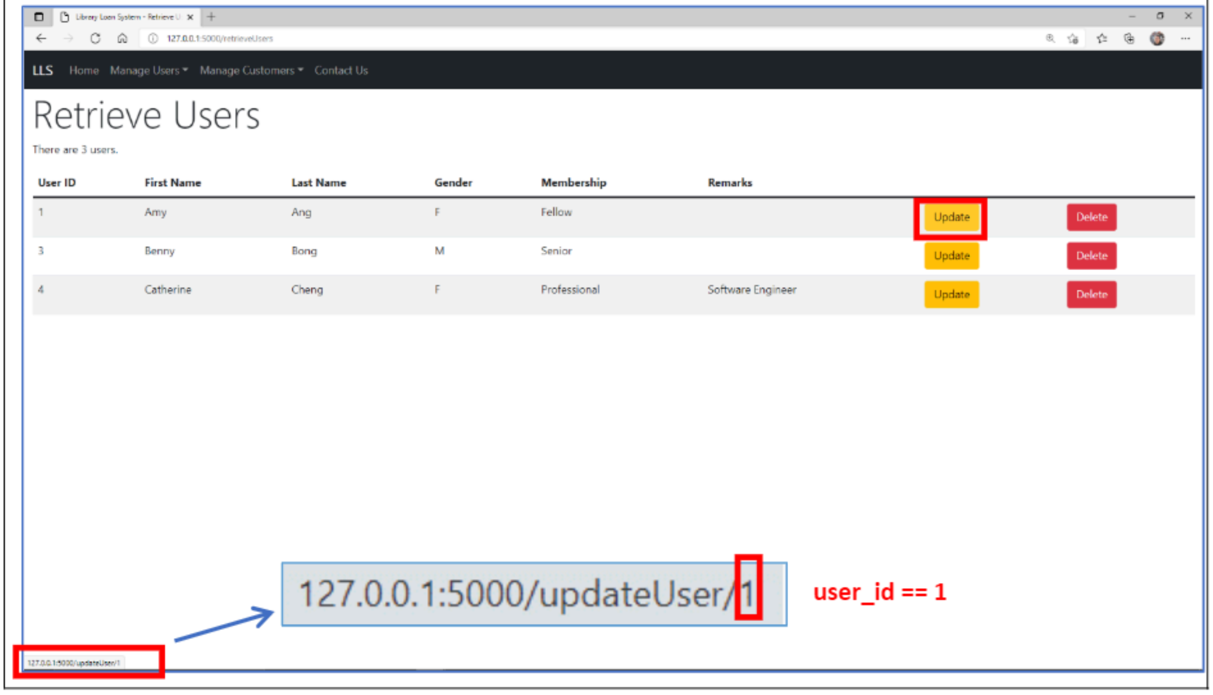
Retrieve Users

There are 3 users.

User ID	First Name	Last Name	Gender	Membership	Remarks		
1	Amy	Ang	F	Fellow		Update	Delete
3	Benny	Bong	M	Senior		Update	Delete
4	Catherine	Cheng	F	Professional	Software Engineer	Update	Delete

DID YOU KNOW

Notice that the link changes as you move your mouse cursor over each user's **Update** button. you will notice that the corresponding user's **User ID** is displayed as part of the **URL**.


Library Loan System - Retrieve U
127.0.0.1:5000/retrieveUsers
LLS
Home
Manage Users
Manage Customers
Contact Us

Retrieve Users

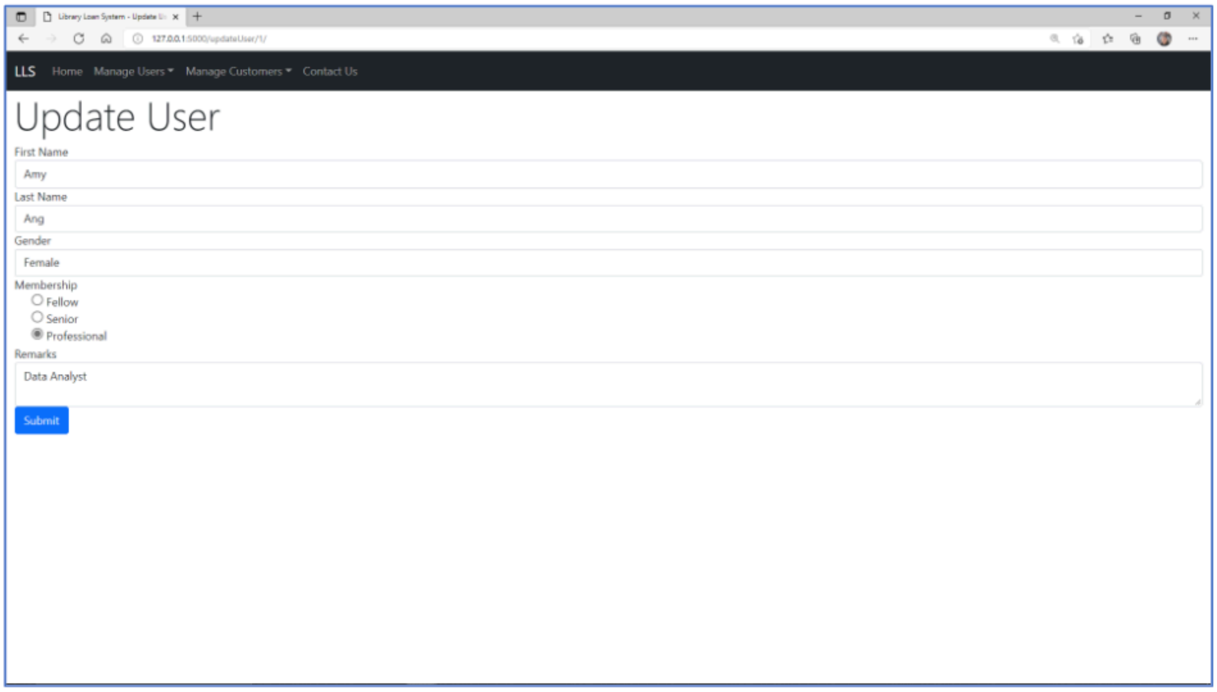
There are 3 users.

User ID	First Name	Last Name	Gender	Membership	Remarks		
1	Amy	Ang	F	Fellow		Update	Delete
3	Benny	Bong	M	Senior		Update	Delete
4	Catherine	Cheng	F	Professional	Software Engineer	Update	Delete

127.0.0.1:5000/updateUser/1

user_id == 1

Step 4: Change **Amy's** Membership to **Professional** and under **Remarks** type **Data Analyst**. Then click **Submit**.



Update User

First Name: Amy

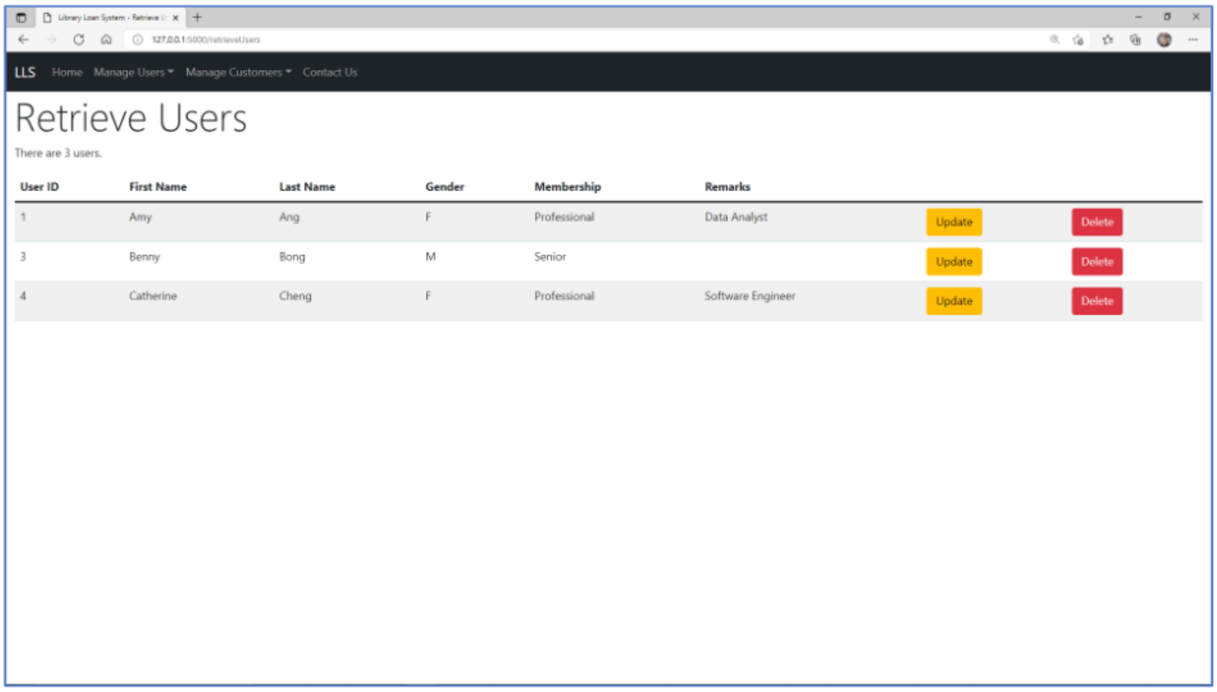
Last Name: Ang

Gender: Female

Membership: ○ Fellow ○ Senior [✓ Professional]

Remarks: Data Analyst

Step 5: Check to see that **Amy's** Membership has been changed to **Professional** and under **Remarks** it reads **Data Analyst**.


Library Loan System - Retrieve U
127.0.0.1:5000/retrieveUsers
LLS
Home
Manage Users
Manage Customers
Contact Us

Retrieve Users

There are 3 users.

User ID	First Name	Last Name	Gender	Membership	Remarks		
1	Amy	Ang	F	Professional	Data Analyst	Update	Delete
3	Benny	Bong	M	Senior		Update	Delete
4	Catherine	Cheng	F	Professional	Software Engineer	Update	Delete

2.2 Add an Update Customer Function to Your SimpleWebApplication

Repeat steps defined in section 2.1 for Customer.

3. Activity 3: Project Idea

Continue to work on your project idea (group and individual) for your Project Proposal Presentation.

~ End ~