App Development Project
Week 8, Lesson 1

Looking Forward

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

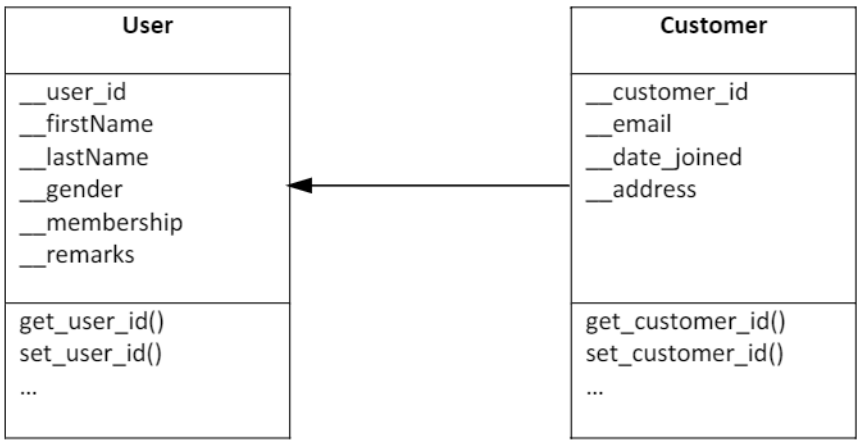
- Explain what and why inheritance and polymorphism is needed.
- Create a **Customer class** (model) to aid in storage, retrieval and usage of Customer **data**.
- Add a **Create Customer** function to your SimpleWebApplication using **WTForms**.
- Create a **createCustomer.html** template and add an entry into Flask **route()** to point to it.
- Use **WTForms** to specify **constraints** for fields and **validate** them.
- Use **shelve** to persist (store) data.

1. Activity 1: Inheritance

🤔 THINK ABOUT IT 🤔

Let us recall the **User** class we created from Week 3, Lesson 1. What if we want to create another class so that we can create objects that reuse attributes and methods from this **User** class? Wouldn't it be nice that we do not have to duplicate attributes and methods in several classes? This can be achieved through inheritance, whereby a subclass inherits the attributes and methods from a superclass, and at the same time has its own attributes and methods.

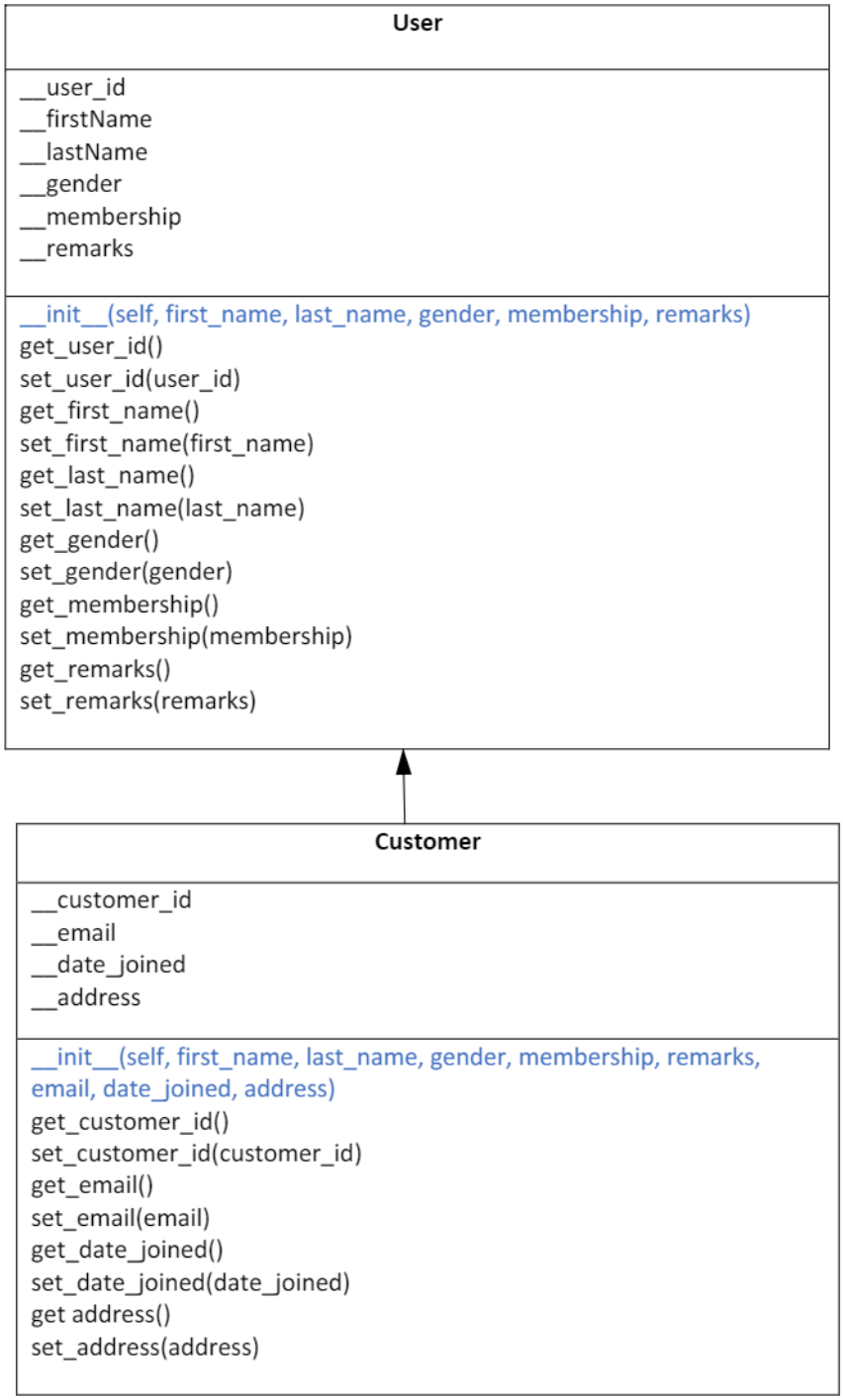
Let us consider a customer, which is essentially a user with attributes like ID, first name, last name, gender etc. However, it also has other attributes like email, date joined and address. We can then create a **Customer** subclass that inherits from the **User** superclass.



2. Activity 2: Polymorphism

THINK ABOUT IT

Let us recall how an object is created from the **User** class. To create an object in Python, a class may use an initializer **__init__** which is a method that receives input parameters required to initialize that object.



To create a **User** object, you would call its initializer method below and pass in the required 5 data attributes as defined.

__init__(self, first_name, last_name, gender, membership, remarks)

Because the **Customer** class inherits from the **User** class, you may call **User's** initializer method to create a **Customer** object. But what if you want to pass in and initialize more data attributes that is unique to the **Customer** during initialization, and at the same time maintain the same method name? You need to then override **User's** initializer method and define a new initializer method below in the **Customer** class.

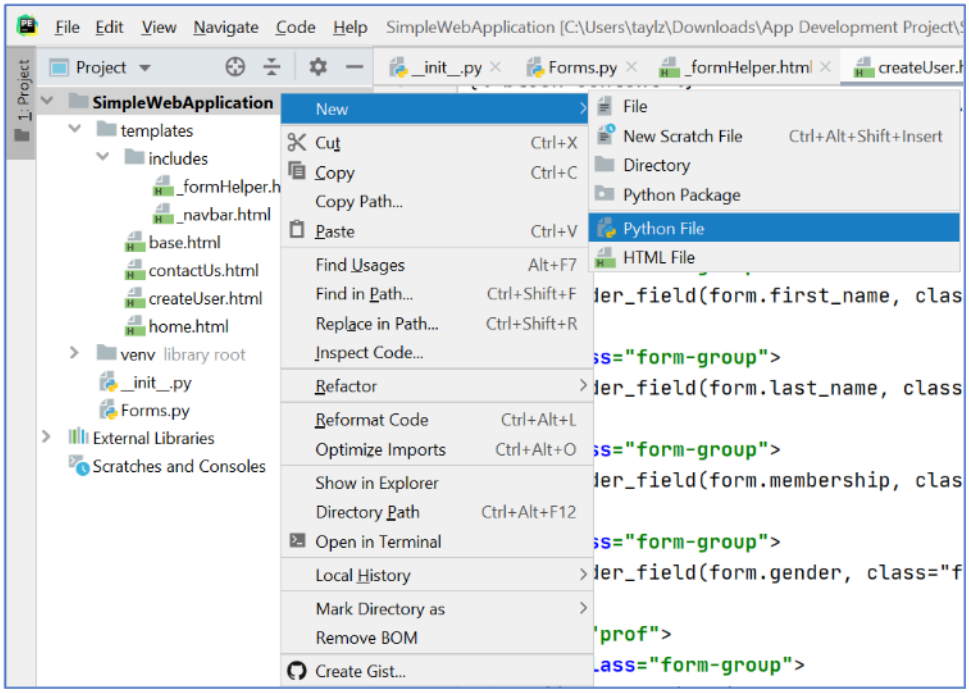
__init__(self, first_name, last_name, gender, membership, remarks, email, date_joined, address)

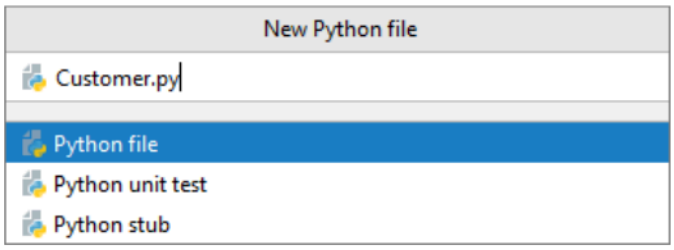
3. Activity 3: Continue the Project.

3.1 Create Classes for Your SimpleWebApplication

3.1.1 Create a New Customer Class

Step 1: Right-click on the **SimpleWebApplication** folder and select **New > Python File** to create a new Python File called **Customer.py**.





Step 2: Import the **User** module and create a **Customer** class with the following **private** attributes:

1. __email
2. __date_joined
3. __address

The **initializer** needs to take in **parameters** required by the **User** class as well as each of the private attributes except **__customer_id**. Note how the **User** attributes are initialized using **super()**.

Customer.py

```
import User


class Customer(User.User):

    def __init__(self, first_name, last_name, gender, membership, remarks, email, date_joined, address):
        super().__init__(first_name, last_name, gender, membership, remarks)
        self.__customer_id = ""
        self.__email = email
        self.__date_joined = date_joined
        self.__address = address

```

Step 3: Create a **class attribute** called **count_id** to be used as a counter. Increment the **count_id** class attribute and use it to initialize the **__customer_id** data attribute.

Customer.py

```
import User


class Customer(User.User):
    count_id = 0

    def __init__(self, first_name, last_name, gender, membership, remarks, email, date_joined, address):
        super().__init__(first_name, last_name, gender, membership, remarks)
        Customer.count_id += 1
        self.__customer_id = Customer.count_id
        self.__email = email
        self.__date_joined = date_joined
        self.__address = address

```

IMPORTANT

THINK ABOUT IT

Notice that the Customer class' initializer method does not require **customer_id** be taken in as a parameter. This is because, **count_id** will be used to create a **primitive** form of **auto-increment** for the **__customer_id**.

One obvious limitation it has is that every time you **restart** the web application, the **count_id resets to 0**. Once the **count_id** resets, the next newly created Customer's **__customer_id** will start from **1** again and **overwrite** any Customer that previously had **__customer_id == 1**.

Can you think of a better alternative?

Step 4: Create the **accessor** and **mutator methods** for each of the private attributes.

Customer.py

```
15      def get_customer_id(self):
16          return self.__customer_id
17
18      def get_email(self):
19          return self.__email
20
21      def get_date_joined(self):
22          return self.__date_joined
23
24      def get_address(self):
25          return self.__address
```

Customer.py

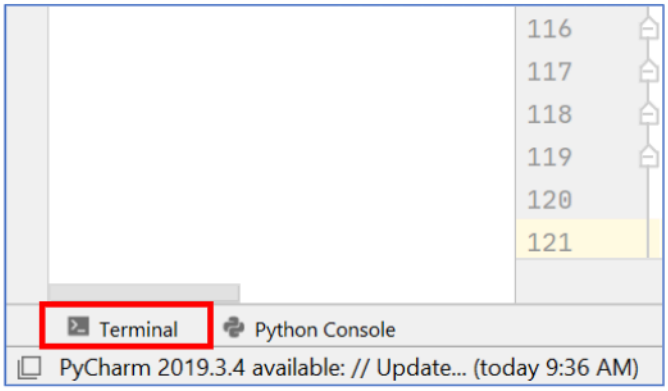
```
26
27      def set_customer_id(self, customer_id):
28          self.__customer_id = customer_id
29
30      def set_email(self, email):
31          self.__email = email
32
33      def set_date_joined(self, date_joined):
34          self.__date_joined = date_joined
35
36      def set_address(self, address):
37          self.__address = address
```

3.2 Add a Create Customer Function to Your SimpleWebApplication

3.2.1 Update Your Form Definitions File

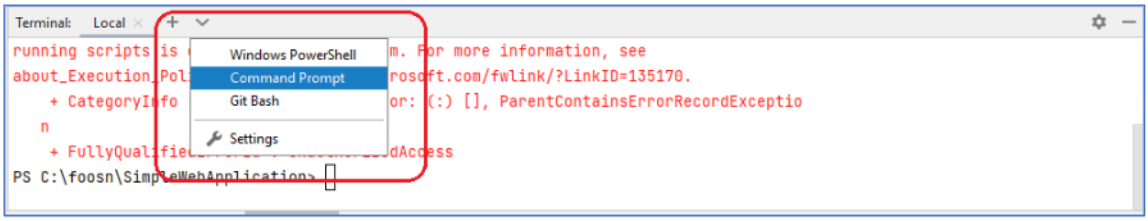
3.2.2 Install email_validator into Your SimpleWebApplication Project

Step 1: Click on **Terminal** at the **bottom left** hand corner of the PyCharm application screen.

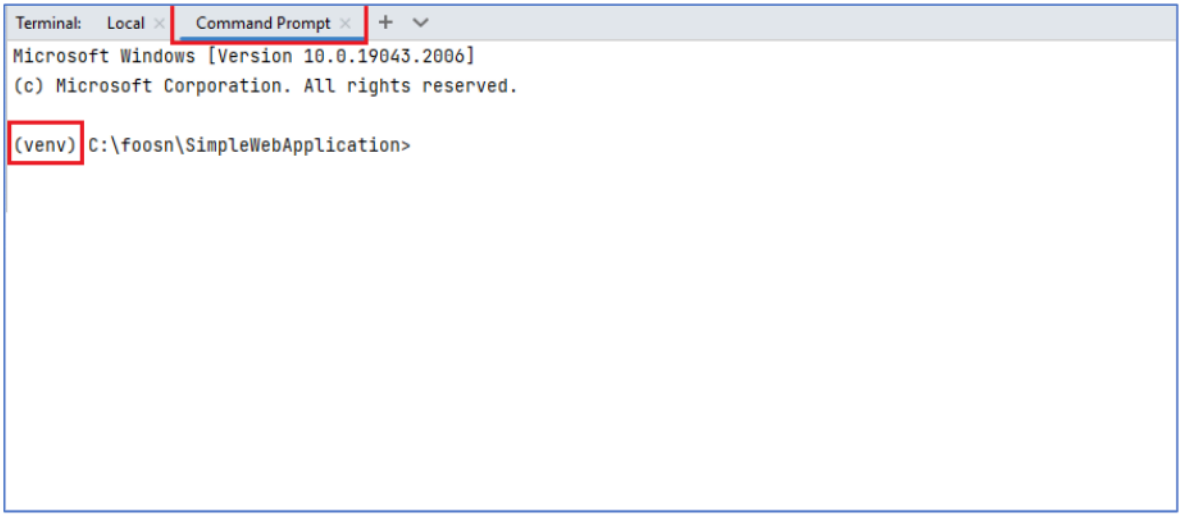


REMINDER

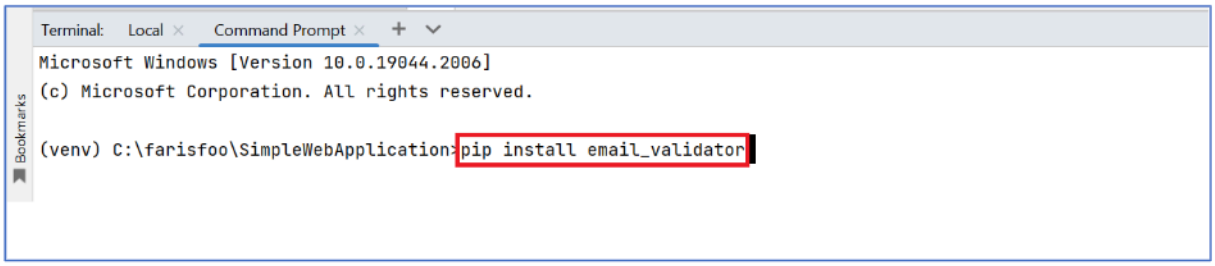
In the Terminal window, select Command Prompt from the drop down (as indicated below)



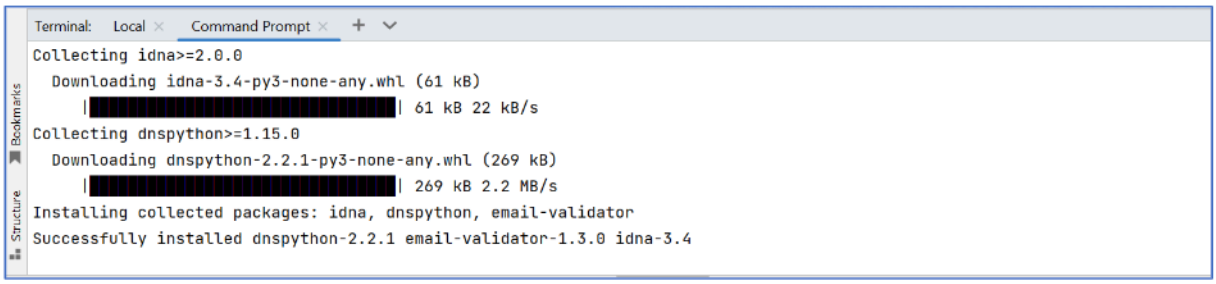
Ensure that your Terminal window has Command Prompt selected. You can verify by ensuring that the **venv** is displayed as the active directory.



Step 2: In the **Terminal** window (Command Prompt), type **pip install email_validator** and press **Enter**.



Step 3: You have successfully installed **email_validator** into Your **SimpleWebApplication** project.



Step 4: Import the WTForms **objects** that you will need by typing the following in **Forms.py**:

Forms.py

```
1  from wtforms import Form, StringField, RadioField, SelectField, TextAreaField, validators
   from wtforms.fields import EmailField, DateField
2
```

Step 5: Create a new **class** for **CreateCustomerForm()**.

Forms.py

```
1  from wtforms import Form, StringField, RadioField, SelectField, TextAreaField, validators
   from wtforms.fields import EmailField, DateField
2
3  class CreateUserForm(Form):
4      first_name = StringField('First Name', [validators.Length(min=1, max=150), validators.DataRequired()])
5      last_name = StringField('Last Name', [validators.Length(min=1, max=150), validators.DataRequired()])
       gender = SelectField('Gender', [validators.DataRequired()], choices=[('', 'Select'), ('F', 'Female'), ('M', 'Male')],
6  default='')
       membership = RadioField('Membership', choices=[('F', 'Fellow'), ('S', 'Senior'), ('P', 'Professional')], default='F')
7      remarks = TextAreaField('Remarks', [validators.Optional()])

8  class CreateCustomerForm(Form):
       first_name = StringField('First Name', [validators.Length(min=1, max=150), validators.DataRequired()])
9      last_name = StringField('Last Name', [validators.Length(min=1, max=150), validators.DataRequired()])
10     gender = SelectField('Gender', [validators.DataRequired()], choices=[('', 'Select'), ('F', 'Female'), ('M', 'Male')],
11 default='')
12     email = EmailField('Email', [validators.Email(), validators.DataRequired()])
       date_joined = DateField('Date Joined', format='%Y-%m-%d')
13     address = TextAreaField('Mailing Address', [validators.length(max=200), validators.DataRequired()])
       membership = RadioField('Membership', choices=[('F', 'Fellow'), ('S', 'Senior'), ('P', 'Professional')], default='F')
14     remarks = TextAreaField('Remarks', [validators.Optional()])
```

🤔 THINK ABOUT IT 🤔

EmailField object
– Used to create an HTML **email** input.

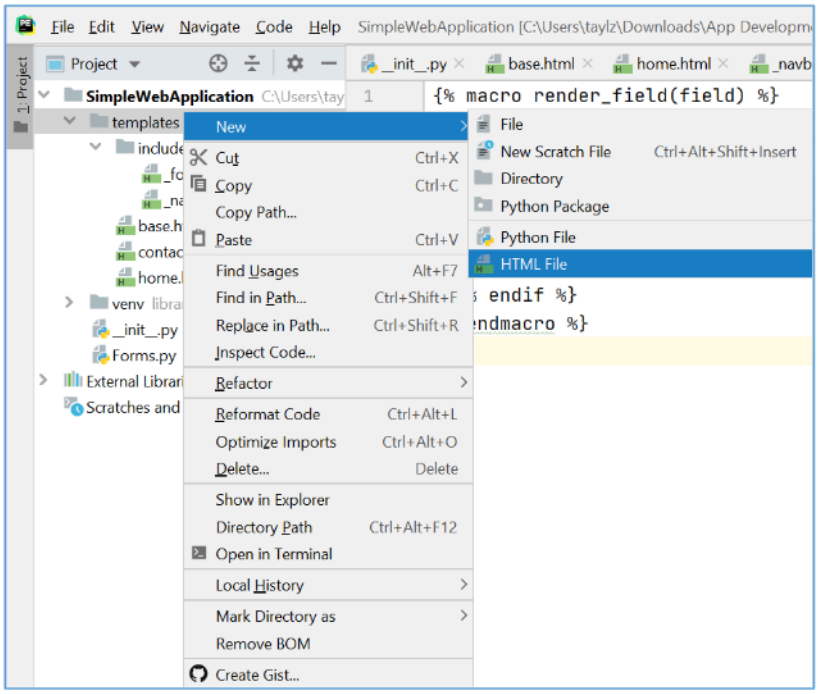
DateField object
– Used to create an HTML **date calendar** input.

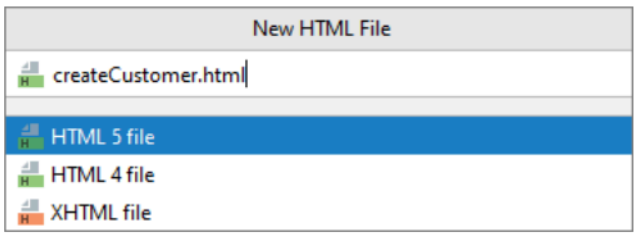
You may visit the following pages to find out more:

- https://wtforms.readthedocs.io/en/2.3.x/
- https://wtforms.readthedocs.io/en/2.3.x/crash_course/

3.2.3 Create a New Template for Your Create Customer Page

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





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

createCustomer.html

```
1  {% extends "base.html" %}
2  {% block title %}Library Loan System - Create Customer{% endblock %}
3
4  {% block content %}
5  {% from "includes/_formHelper.html" import render_field %}
6
7  <h1 class="display-4">Create Customer</h1>
8
```

```
<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.email, class="form-control") }}
  </div>
  <div class="form-group">
    {{ render_field(form.date_joined, class="form-control datepicker") }}
  </div>
  <div class="form-group">
    {{ render_field(form.address, 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 %}
```

3.2.4 Add createCustomer.html to the Flask route()

Step 1: Import the required **objects** that you will need by typing the following in **__init__.py**:

__init__.py

```
from flask import Flask, render_template, request, redirect, url_for
from Forms import CreateUserForm, CreateCustomerForm
import shelve, User, Customer

```

Step 2: Add in a new route for **/createCustomer** to the Flask **route()** in **__init__.py** that points to **createCustomer.html** and add the **creatCustomer()** method to open **shelve** and **store** the newly created Customer data.

__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)


@app.route('/createCustomer', methods=['GET', 'POST'])
def create_customer():
    create_customer_form = CreateCustomerForm(request.form)
    if request.method == 'POST' and create_customer_form.validate():
        customers_dict = {}
        db = shelve.open('customer.db', 'c')

        try:
            customers_dict = db['Customers']
        except:
            print("Error in retrieving Customers from customer.db.")

        customer = Customer.Customer(create_customer_form.first_name.data,
create_customer_form.last_name.data,
                                     create_customer_form.gender.data, create_customer_form.membership.data,
                                     create_customer_form.remarks.data, create_customer_form.email.data,
                                     create_customer_form.date_joined.data, create_customer_form.address.data)
        customers_dict[customer.get_user_id()] = customer
        db['Customers'] = customers_dict

        db.close()

        return redirect(url_for('home'))
    return render_template('createCustomer.html', form=create_customer_form)
```

IMPORTANT

Always add new Flask routes above the **if** __name__ == **'__main__'**: statement. Anything that comes after that **if** statement will **not be in effect** when your SimpleWebApplication is run.

THINK ABOUT IT

create_customer_form = CreateCustomerForm(request.form)
– receives the **form** posted by the **createCustomer.html** template. This is done by using the posted form from the **request** object request.form.

createCustomerForm.validate()
– returns **True** if validation is successful. Returns **False** if validation fails.

render_template('**createCustomer.html**', form=create_customer_form)
– sets the **form** used for **createCustomer.html** to **create_customer_form**.

3.2.5 Modify the Navigation Bar and Run Your SimpleWebApplication

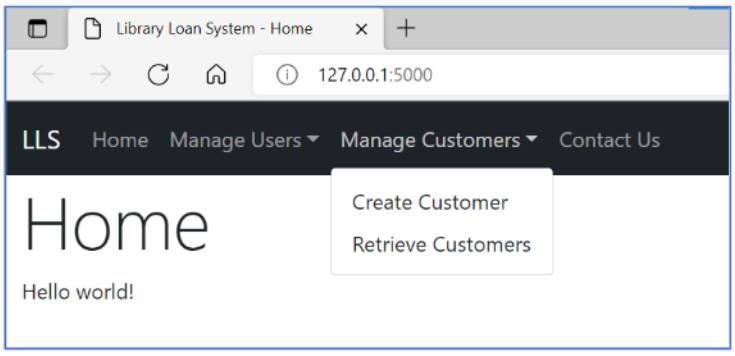
Step 1: Add a Manage Customers dropdown in **_navbar.html** with the **Create Customer** link to point to **/createCustomer**.

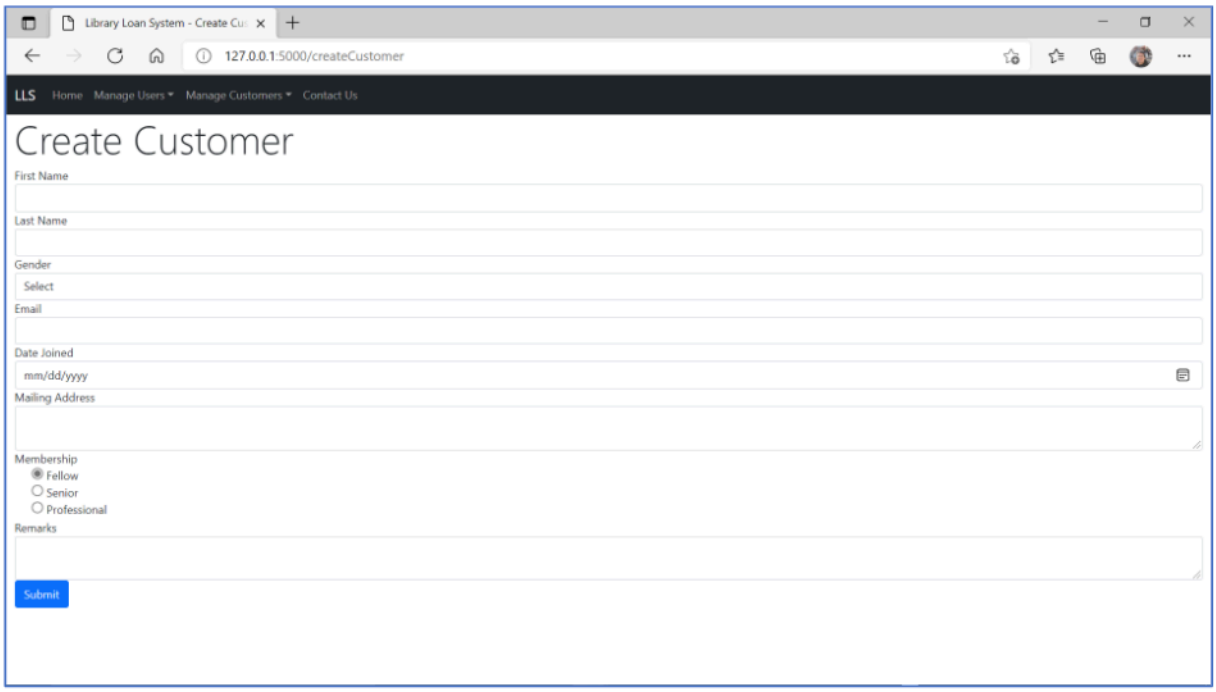
_navbar.html

```
19 <li class="nav-item dropdown">
20   <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown">Manage
   Customers</a>
21   <ul class="dropdown-menu">
22     <li><a class="dropdown-item" href="/createCustomer">Create Customer</a></li>
       <li><a class="dropdown-item" href="#">Retrieve Customers</a></li>
23   </ul>
24 </li>
25
26
```

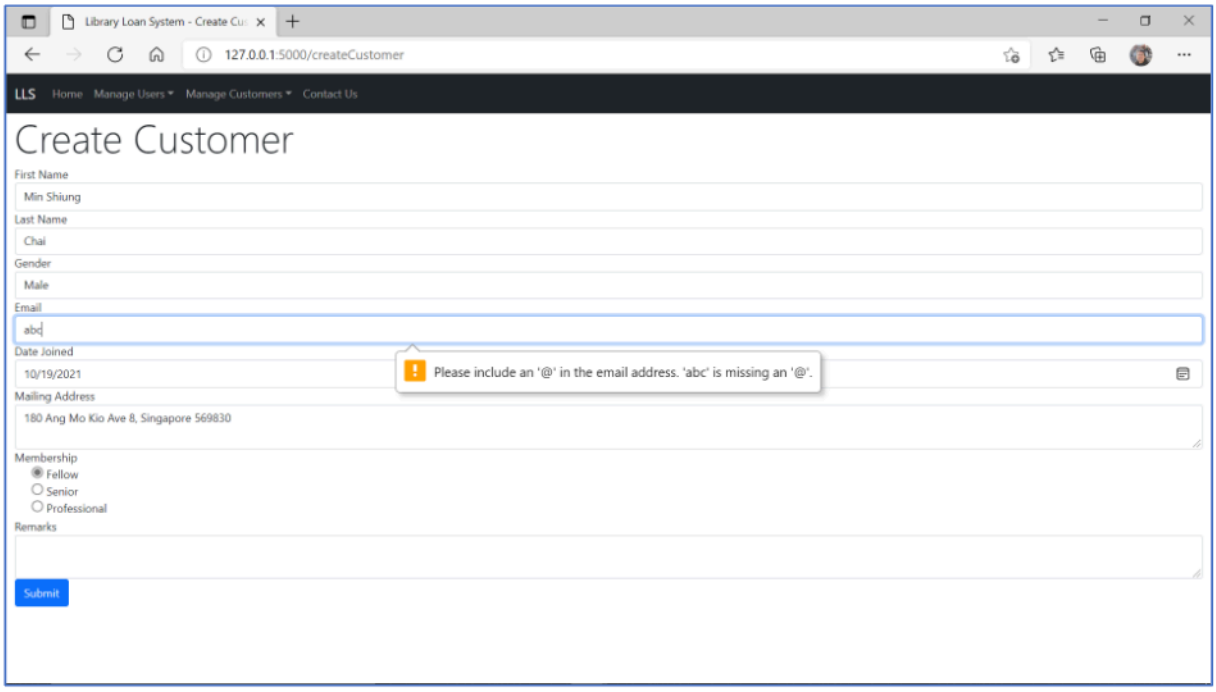
Step 2: Run your **SimpleWebApplication** by right-clicking on **__init__.py** and selecting **Run '__init__'**. Then click on the http://127.0.0.1:5000/ **link** in PyCharm application.

Click on **Manage Customers > Create Customer** from the **navbar**.

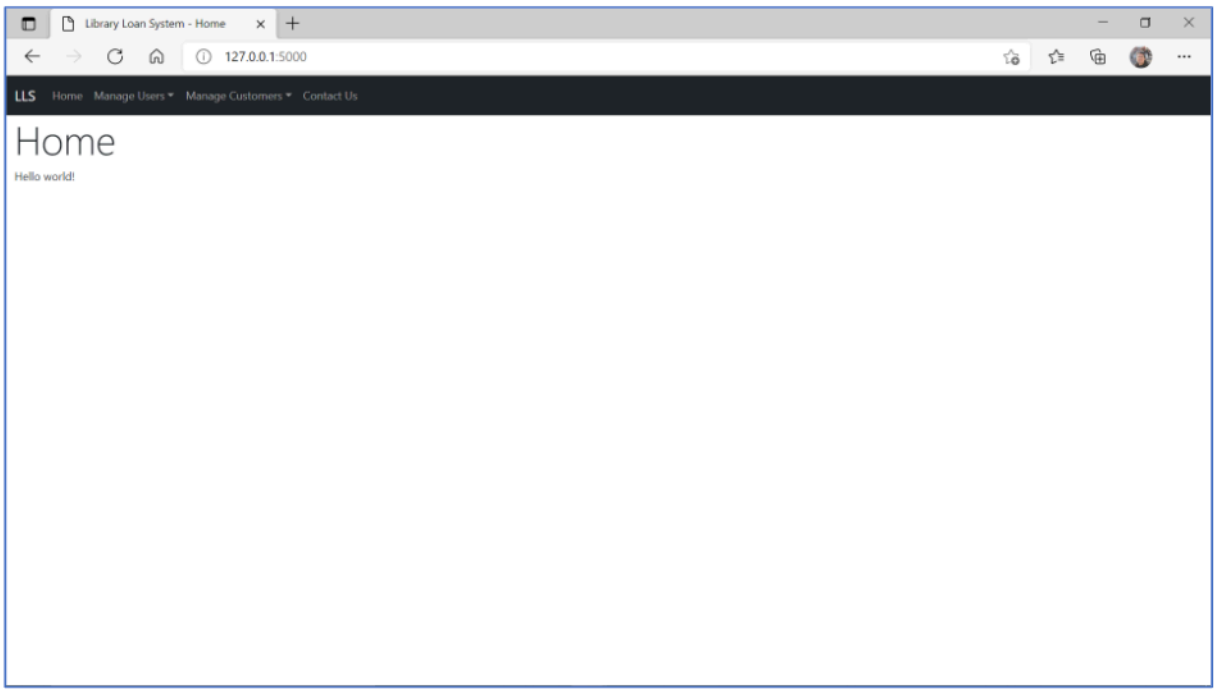




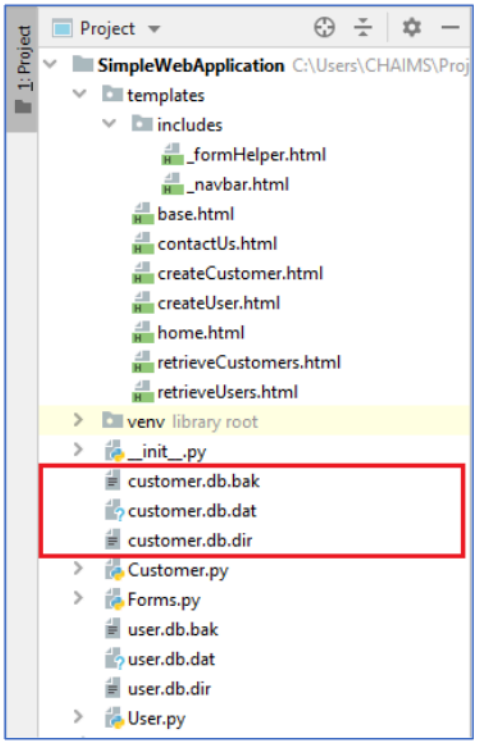
Step 3: Test out the form **validation** by entering some of the fields **incorrectly** and clicking on **Submit**. The **validation fails** so you are not allowed to proceed until you correct the **validation errors**.



Step 4: Now fill up all the required fields (**Remarks** is optional) and click **Submit**. If the form validation succeeds, you will be redirected to the **Home** page.



Notice that **3 new files** were created by **shelve** after a new **Customer object** was added to it. **customer.db.dir** is moved to **customer.db.bak** as new entries are committed and are finally stored in **customer.db.dat**.



For **MacOS** users, there should be only 1 file.

~ End ~