Quick Intro to Flask Part II

Version 1.01

This is the second installment of an introduction to web application programming using Flask. The coverage includes basic security and transaction management.

Run Flask

If you haven't done so yet, run Flask from Terminal (Mac OS users) or from the Command Line/Anaconda Prompt (Windows users).

As we haven't yet covered how to make environment variables more permanent, we will have to go through the next few steps again:

If you are on Mac:
```
$ export FLASK_APP=app.py
$ export FLASK_ENV=development
```

If you are on Windows Anaconda Prompt:
```
C:\path\to\app>set FLASK_APP=app.py
C:\path\to\app>set FLASK_ENV=development
```

Make sure you are in the directory where your app.py file is located.

To run Flask:
```
$ flask run
```

Login:

For this second installment, we shall assume that users are already registered in the system. Since we haven't covered database management yet, we do not have full-blown registration and login modules. As such, we will be hardcoding username and password information in our mock database.

Exercise: Implement user login and navigation

1. Edit **database.py** and add this hardcoded dictionary entry. NOTE: Unlike the previous session, you will be adding code blocks instead of copying and pasting whole file contents, so please be careful in updating your files.

   ```
   users = {
       "chums@example.com":{"password":"Ch@ng3m3!",
                            "first_name":"Matthew",
                            "last_name":"Uy"},
       "joben@example.com":{"password":"Ch@ng3m3!",
                            "first_name":"Joben",
                            "last_name":"Ilagan"},
       "bong@example.com":{"password":"Ch@ng3m3!",
                           "first_name":"Bong",
                           "last_name":"Olpoc"},
       "joaqs@example.com":{"password":"Ch@ng3m3!",
                            "first_name":"Joaqs",
                            "last_name":"Gonzales"},
       "gihoe@example.com":{"password":"Ch@ng3m3!",
                            "first_name":"Gio",
                            "last_name":"Hernandez"},
       "vic@example.com":{"password":"Ch@ng3m3!",
                          "first_name":"Vic",
                          "last_name":"Reventar"},
       "joe@example.com":{"password":"Ch@ng3m3!",
                          "first_name":"Joe",
                          "last_name":"Ilagan"},
   }
   ```

2. Still in **database.py**, add the following function:

   ```
   def get_user(username):
       try:
          return users[username]
       except KeyError:
          return None
   ```

   At this point, to fully appreciate the code snippet above, you may want to review our previous discussions on Python error handling.

3. Create a new file **authentication.py**. Copy and paste the code below as contents of the new file:

```
import database as db

def login(username, password):
    is_valid_login = False
    user=None
    temp_user = db.get_user(username)
    if(temp_user != None):
        if(temp_user["password"]==password):
            is_valid_login=True
            user={"username":username,
                  "first_name":temp_user["first_name"],
                  "last_name":temp_user["last_name"]}

    return is_valid_login, user
```

4. A webform, web form or HTML form on a web page allows a user to enter data that is sent to a server for processing. Forms can resemble paper or database forms because web users fill out the forms using checkboxes, radio buttons, or text fields. For example, forms can be used to enter shipping or credit card data to order a product, or can be used to retrieve search results from a search engine.[1]

   Create a new file under the **templates** folder named **login.html** with the following contents:

```
{% include "header.html" %}
<h1>Login</h1>
<form action="/auth" method="POST">
   <div>
     Username or email address
   </div>
   <input type="text" name="username"></input>
   <div>
     Password
   </div>
   <input type="password" name="password"></input>
   <div>
   <input type="submit" value="Login"/>
   </div>
</form>
{% include "footer.html" %}
```

[1] https://en.wikipedia.org/wiki/Form_(HTML)

There are a few noteworthy things with the code above:

- The **`action`** parameter of form specifies the url the request will go to after submission.
- The **`method`** parameter indicates the HTTP request verb to use. By default, even for forms, the method is **GET**, but in practice, this is usually specified as **POST**. The main difference is that GET parameters are passed as part of the URL request while POST parameters are passed in the request body (not visible in the URL request string itself).

5. Edit **app.py**. First, add these new imports:

```
import authentication
import logging
```

6. Edit the line importing Flask as follows:

```
from flask import Flask,redirect
```

7. Next, add two lines of code right below

```
app = Flask(__name__)
```

like so:

```
logging.basicConfig(level=logging.DEBUG)
app.logger.setLevel(logging.INFO)
```

The first few lines of **app.py** should now look similar to this:

```
from flask import Flask,redirect
from flask import render_template
from flask import request
import database as db
import authentication
import logging

app = Flask(__name__)

logging.basicConfig(level=logging.DEBUG)
app.logger.setLevel(logging.INFO)
```

8. Still on **app.py**, add these new routes:

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    return render_template('login.html')

@app.route('/auth', methods = ['POST'])
def auth():
    username = request.form.get('username')
    password = request.form.get('password')

    is_successful, user = authentication.login(username,
password)
    app.logger.info('%s', is_successful)
    if(is_successful):
        return redirect('/')
    else:
        return redirect('/login')
```

Recall that **`/auth`** is the URL we specified in the **`action`** attribute of the HTML form we used in **login.html** earlier. When you click on the Login button of the HTML form, the username and password are sent to the web application server and the route handling **`/auth`** processes these form parameters. Meanwhile, **`/login`** is the URL that brings us to the Login form itself.

Save your changes.

What is URL redirection?

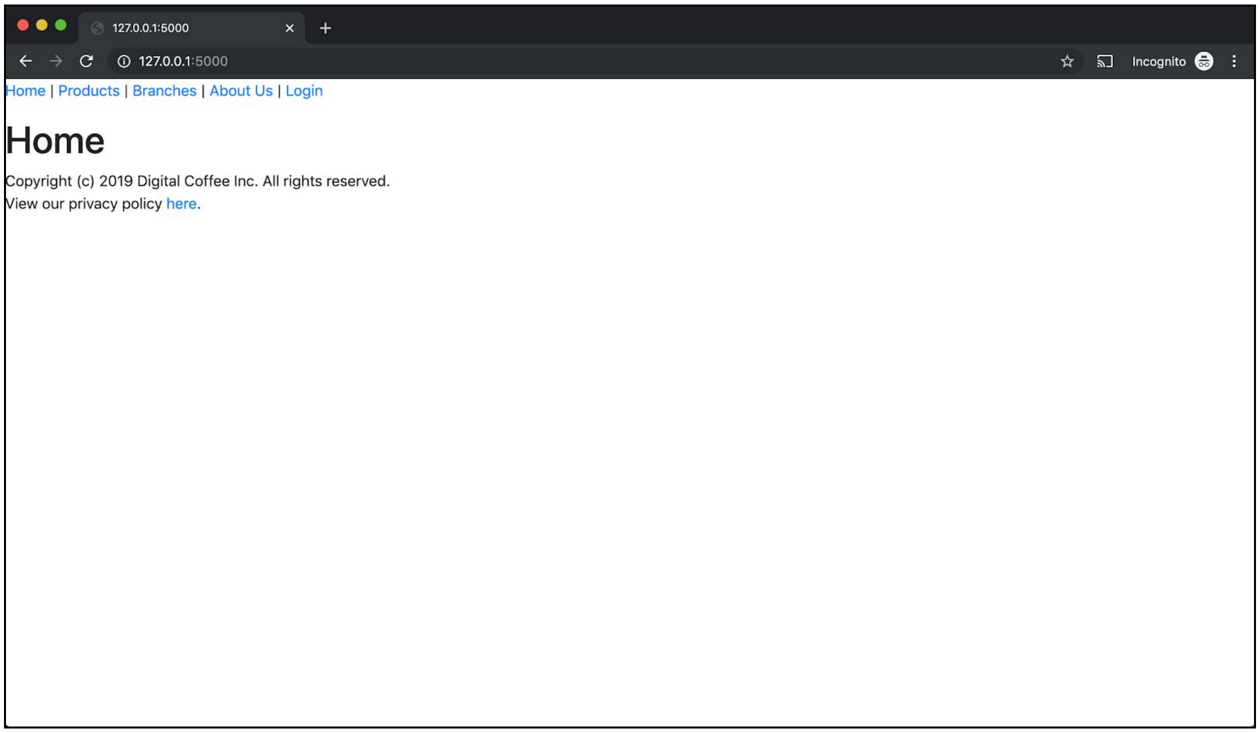
URL redirection, also known as URL forwarding, is a technique to give a page, a form or a whole Web application, more than one URL address. HTTP provides a special kind of responses, HTTP redirects, to perform this operation used for numerous goals: temporary redirection while site maintenance is ongoing, permanent redirection to keep external links working after a change of the site's architecture, progress pages when uploading a file, and so on.

Source: MDN Web Docs (https://developer.mozilla.org/en-US/docs/Web/HTTP/Redirections)

9. Add a link to the Login screen in the navbar. Edit **header.html** located in the **templates** folder. Depending on how far you've gone from the last take-home quiz, your navbar may look something like this:

```
        <a href='/'>Home</a> | <a href='/products'>Products</a>
|
        <a href='/branches'>Branches</a> | <a
href='/aboutus'>About Us</a> |
        <a href='/login'>Login</a>
```

10. Test our new Login module. Reload the home page on your web browser. You should be able to see your new navigation bar like so:

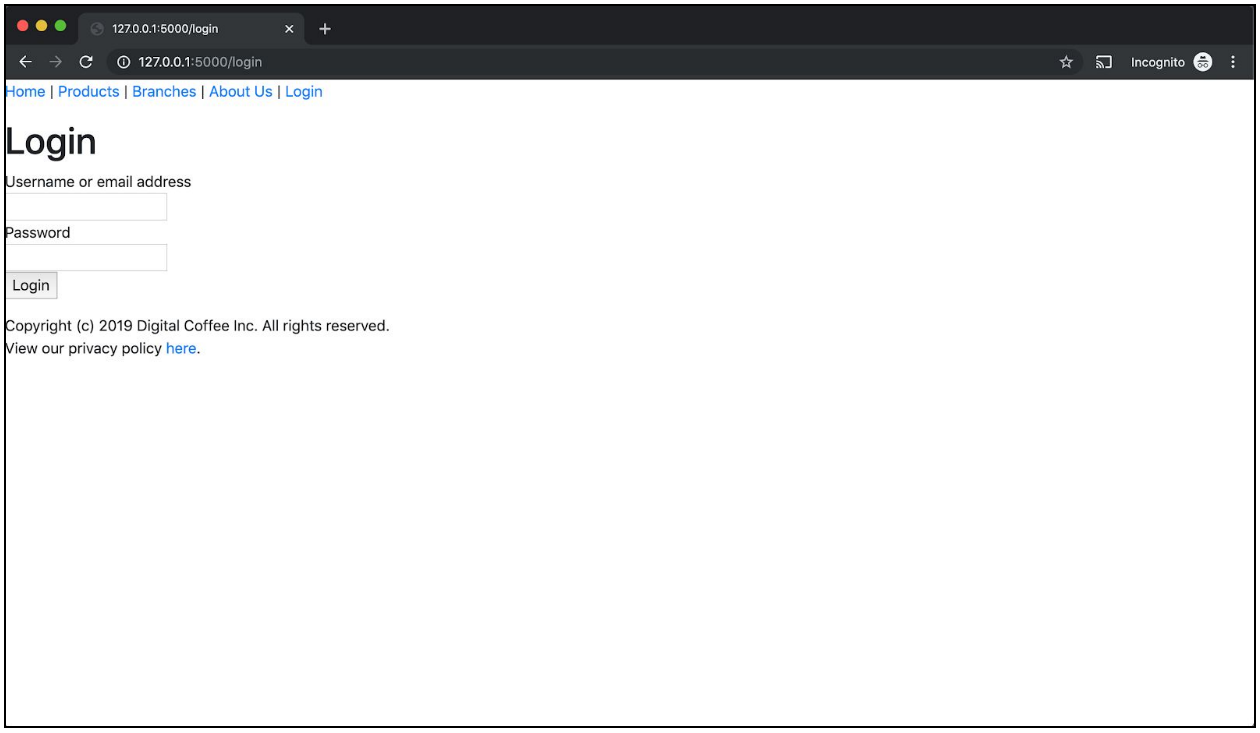


Home | Products | Branches | About Us | Login

Home

Copyright (c) 2019 Digital Coffee Inc. All rights reserved.
View our privacy policy here.

Click on the **Login** link. On the screen shown below, try different combinations of usernames and passwords following it.



127.0.0.1:5000/login

Home | Products | Branches | About Us | Login

Login

Username or email address

Password

Login

Copyright (c) 2019 Digital Coffee Inc. All rights reserved.
View our privacy policy here.

a) Login using the following username and password:

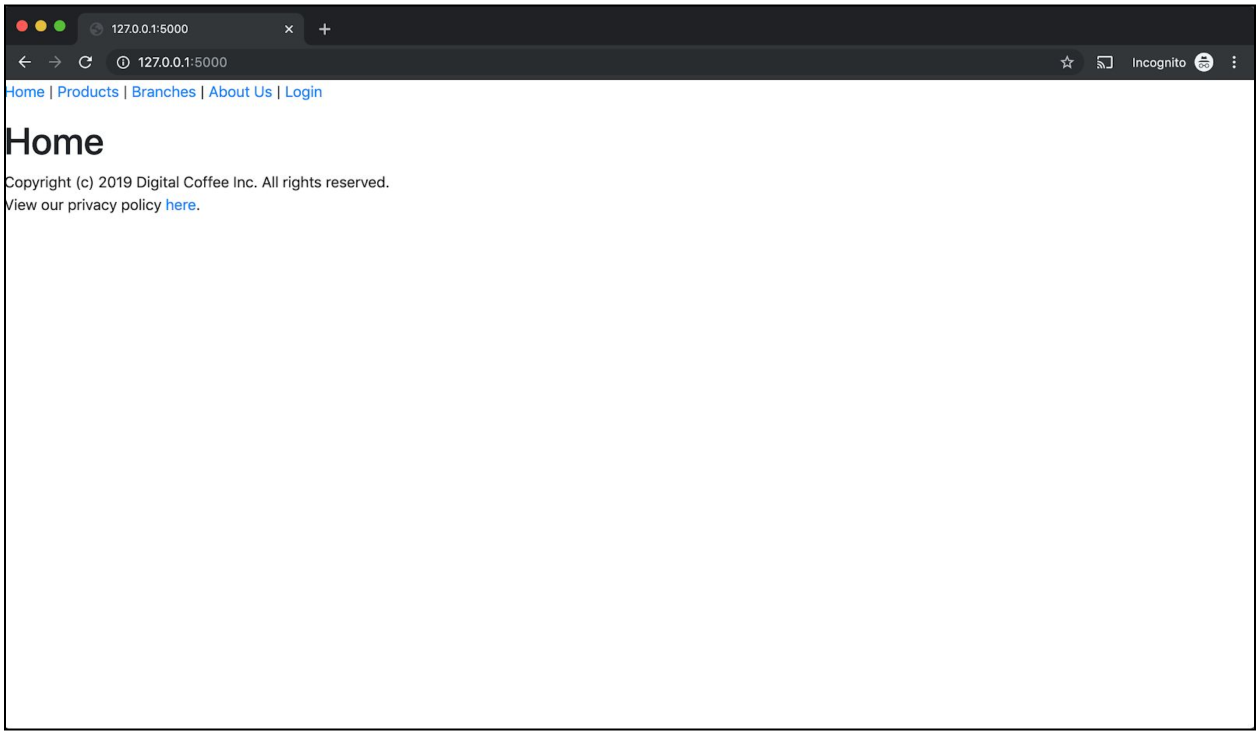
   Username or email address: chums@example.com
   Password: welcome

   Note that this goes back to the Login page.

b) Try logging in again using the following:

   Username or email address: chums@example.com
   Password: Ch@ng3m3!

   Note that the login was successful. You will know if you are redirected to the home page. There's nothing else, however, that indicates that you've been able to log in successfully. This does not good from a user experience standpoint. We shall address this in future steps.



Home | Products | Branches | About Us | Login

Home

Copyright (c) 2019 Digital Coffee Inc. All rights reserved.
View our privacy policy here.

At this point, you've been introduced to the mechanism in HTTP to send data through forms. In particular, we've been able to use forms to implement one of the most basic use cases of any transactional web application--logging in.

One flaw which we haven't been able to address yet is how to keep contextual information across several HTTP requests. In particular, how do we pass the login user's information such as the first name throughout page requests?

There are many other issues related to User Experience that we are not able to address at the moment. For instance, we don't exactly know whether our login attempt was successful or not, or even if we've logged in successfully, nothing on the screens tell us that we are currently logged in.

One way to fix these problems is to keep on passing this set of information as HTTP request parameters, but this can get messy and difficult to manage. The better way would be to keep this information in some area in memory on the server called a "session"; however, HTTP wasn't originally designed to handle session information.

Fortunately, modern web application frameworks have implemented workarounds to session management in spite of limitations involving the underlying stateless nature of HTTP.

Session and State Management:

In the beginning, HTTP was designed as a stateless, request-response oriented protocol that made no special provisions for stateful sessions spanning across several logically related request and response exchanges.

As HTTP protocol grew in popularity and adoption more and more systems began to use it for applications it was never intended for (such as e-commerce, online marketplaces, or social media applications). Thus, the support for state management became a necessity.

Cookie

An HTTP cookie is a ***token*** or short packet of state information that the HTTP agent and the target server can exchange to maintain a session.[2]

Cookies are stored on the browser and are sent back to the server that owns them for every request made.

Session

Unlike a Cookie, **Session** data is kept on the server. Conceptually, a “session” is the time interval when a client logs into a server and logs out of it. The session data, which is needed to be held across requests throughout this session, is stored temporarily on the server.

A session with each user is assigned a **Session ID**. The session data is identified via this ID and is stored in the cookie. For our protection, the server encrypts them. Note that encryption typically requires a cryptographic key for signing. In our case, the Flask application will need this key to be assigned to a variable called **SECRET_KEY**.

The Session variable in Flask is a dictionary object containing *key-value* pairs of session variables and associated values.

For example, to set a **username** session variable, use the following statement:

```
session['username'] = 'joben@example.com'

# new shopping cart example
cart = []
cart.append({"code":100,"qty":1})
cart.append({"code":200,"qty":2})
session["cart"] = cart
```

[2] https://hc.apache.org/httpcomponents-client-4.5.x/tutorial/html/statemgmt.html

To retrieve the contents of a session variable and use it to perform some action, do something like this:

```
username = session['username']

# sample action on the username
greetings = "Hello,"+username

#...

# add to shopping cart
item = {"code":300,"qty":1}
session["cart"].append(item)
```

To release a session variable, use the **pop()** method.

```
session.pop("username", None)
```

This is useful in these common examples:

- where you need to empty or abandon a shopping cart (although there are other ways of doing this)
- if you wish to log a user out of a session

Exercise. **Enable Flask Session Management**

1. Import session from flask:
   ```
   from flask import session
   ```

2. Set our app's secret key for our sessions. Edit **app.py** and add the following line under

   ```
   app = Flask(__name__)
   ```

   like so:

   ```
   # Set the secret key to some random bytes.
   # Keep this really secret!
   app.secret_key = b's@g@d@c0ff33!'
   ```

The first few lines of **app.py** should look similar to the following:

```
from flask import Flask,redirect
from flask import render_template
from flask import request
from flask import session
import database as db
import authentication
import logging

app = Flask(__name__)

# Set the secret key to some random bytes.
# Keep this really secret!
app.secret_key = b's@g@d@c0ff33!'


logging.basicConfig(level=logging.DEBUG)
app.logger.setLevel(logging.INFO)
```

Save your changes.

3. Still at **app.py**, replace the route **/auth** as follows:

```
@app.route('/auth', methods = ['GET', 'POST'])
def auth():
    username = request.form.get('username')
    password = request.form.get('password')

    is_successful, user = authentication.login(username,
password)
    app.logger.info('%s', is_successful)
    if(is_successful):
        session["user"] = user
        return redirect('/')
    else:
        return redirect('/login')
```

4. Still at **app.py**, add a new route **/logout** as follows:

```
@app.route('/logout')
def logout():
    session.pop("user",None)
    return redirect('/')
```

5. Edit **header.html** in the **templates** folder. Replace the nav bar as follows:

```
        <a href='/'>Home</a> | <a href='/products'>Products</a>
|
        <a href='/branches'>Branches</a> | <a
href='/aboutus'>About Us</a> |
        {% if session["user"] is defined %}
        Welcome, {{ session["user"]["first_name"] }}.
        <a href='/logout'>Logout</a>
        {% else %}
        <a href='/login'>Login</a>
        {% endif %}
        <p/>
```

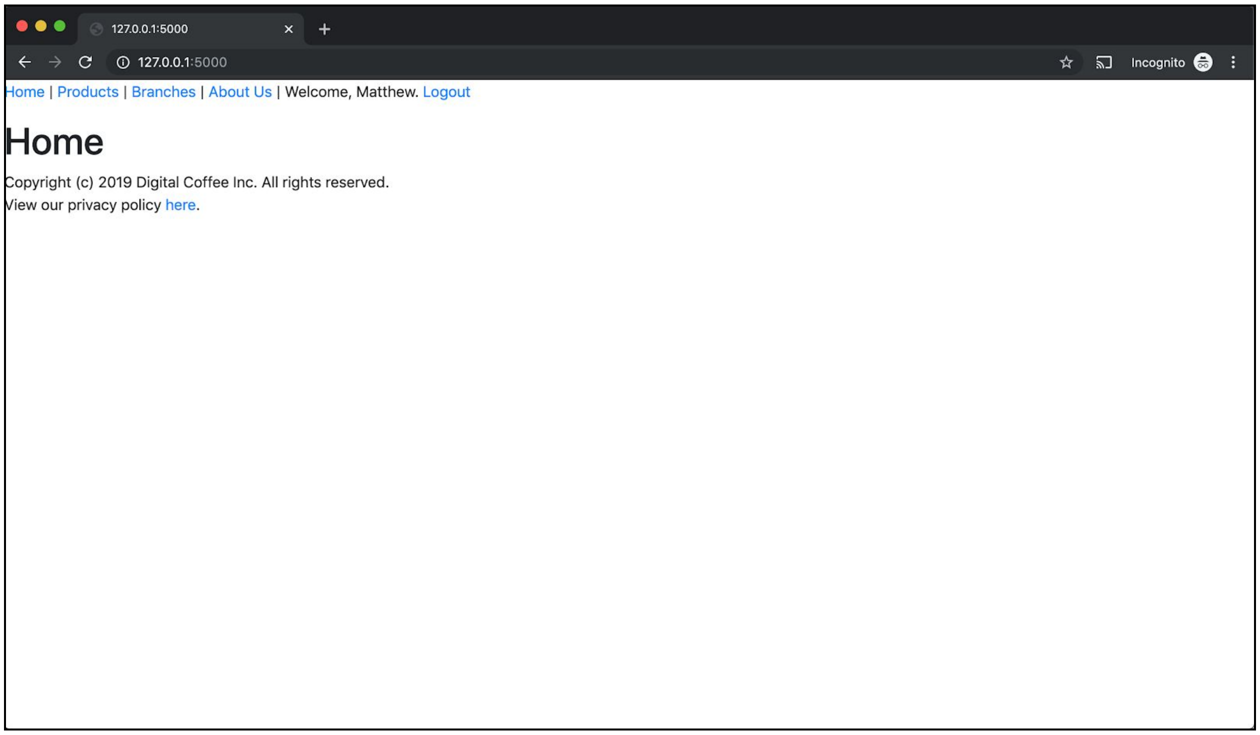
   Notes:
   - We introduce a new construct within Jinja2 templates where we can have conditional branching via if-else directives. In this case, we want to check if our session contains user information. If so, then we display a greeting to the logged in user and a link to opt to logout; else, we simply display the link to log in.
   - The session variable can be accessed within the template itself. This is not necessarily the best location to do so (and as much as possible, it's best to delegate logic to the Python app), but it's used here to illustrate Jinja's capabilities.

   Save your changes.

6. Test your updated app. Reload the home page on your web browser. Login again using a valid user:

   Username or email address: chums@example.com
   Password: Ch@ng3m3!

   You should see a screen similar to the following. Note the greeting to the user and a link to log out:

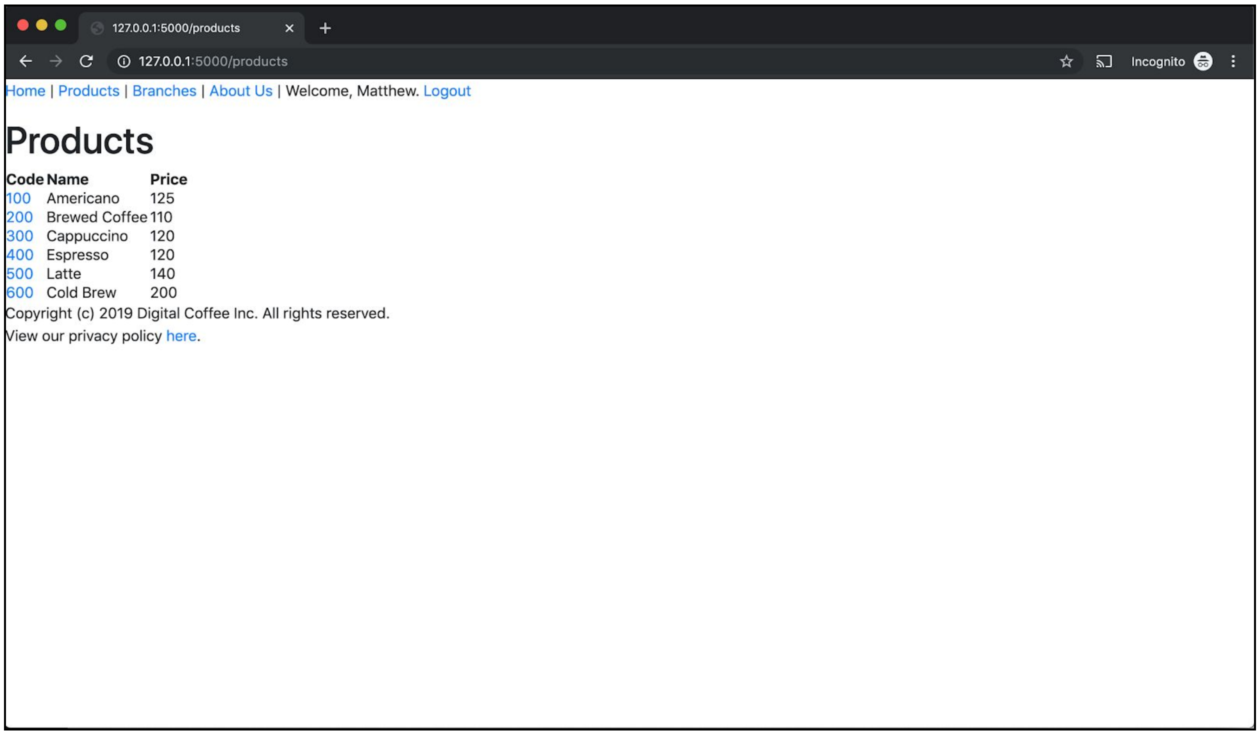


Home | Products | Branches | About Us | Welcome, Matthew. Logout

Home

Copyright (c) 2019 Digital Coffee Inc. All rights reserved.
View our privacy policy here.

Now go to another page, say **Products**. Notice the nav bar. You should still see the same greeting to the user as well as the **Logout** link.



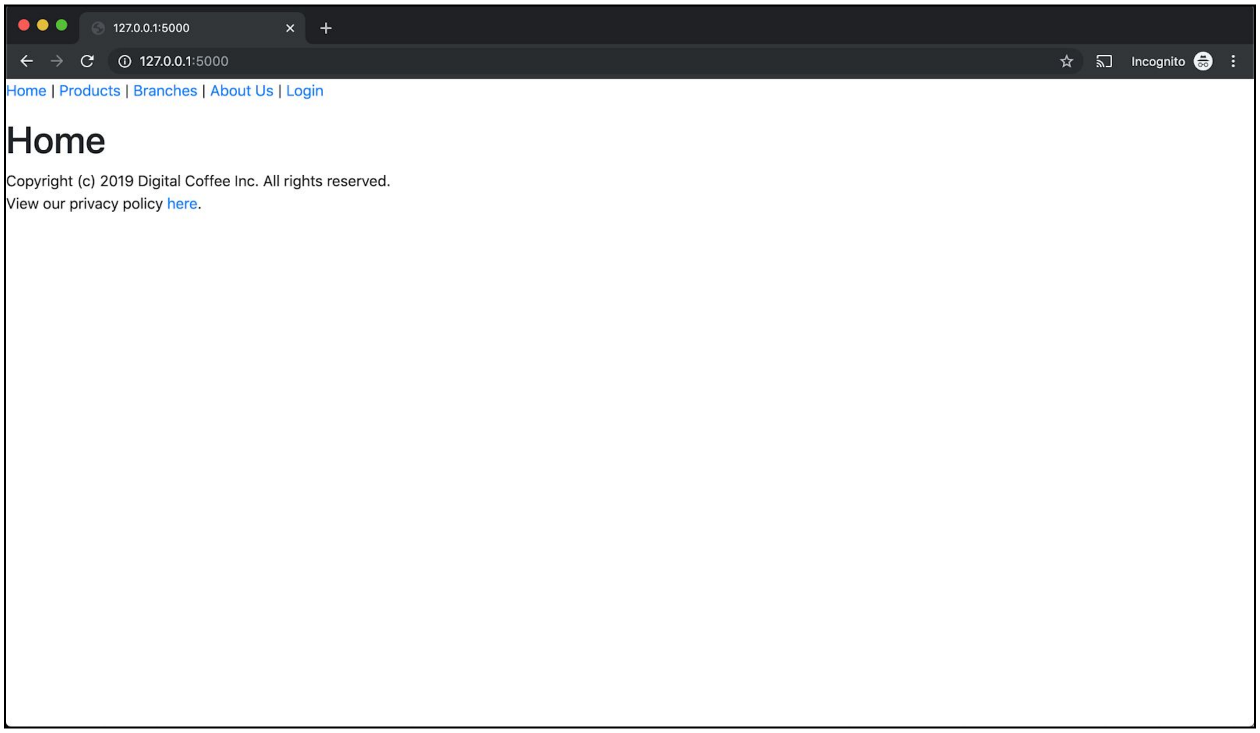
Home | Products | Branches | About Us | Welcome, Matthew. Logout

Products

Code	Name	Price
100	Americano	125
200	Brewed Coffee	110
300	Cappuccino	120
400	Espresso	120
500	Latte	140
600	Cold Brew	200

Copyright (c) 2019 Digital Coffee Inc. All rights reserved.
View our privacy policy here.

Click on the **Logout** link. You should be redirected to the Home Page, which should look something like this:



Home | Products | Branches | About Us | Login

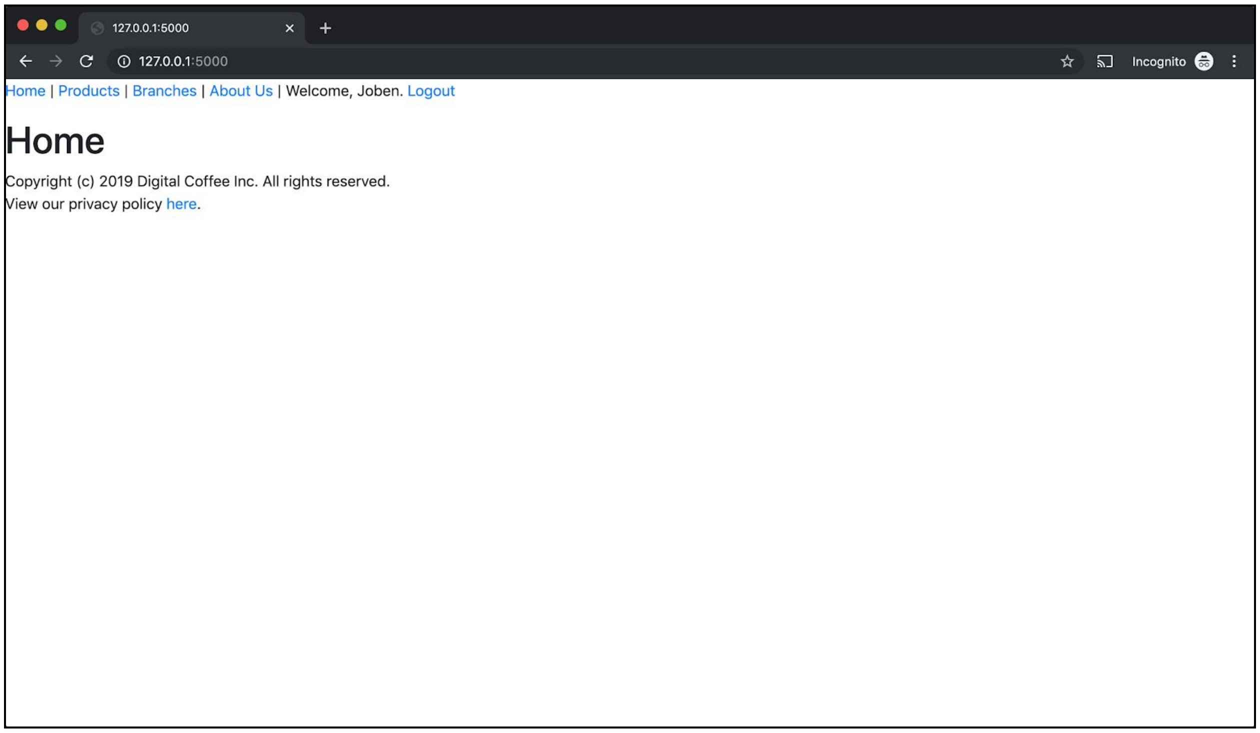
Home

Copyright (c) 2019 Digital Coffee Inc. All rights reserved.
View our privacy policy here.

Click on **Login**. Use another valid user to log in:

Username or email address: joben@example.com
Password: Ch@ng3m3!

You should see the nav bar change, this time with a greeting to the new user:



Home | Products | Branches | About Us | Welcome, Joben. Logout

Home

Copyright (c) 2019 Digital Coffee Inc. All rights reserved.
View our privacy policy here.

Navigate through the different pages and notice that the nav bar has the same contents. Logout as you please.

Introduction to Order Management and Basic E-Commerce:

LIMITATIONS:

- Since we don't have a proper database yet for this system, we have no way of saving the shopping cart contents more permanently for future use. For now, we shall impose the rule that all shopping cart contents that are not checked out will be deleted once the user logs out.
- A user needs to be logged in before placing products into the shopping cart.

Exercise: Make Products Page interactive.

1. Update **productdetails.html**. Add a link to be able to place the item into the shopping cart. Note that we need to check first if a login session exists before displaying the link.

```
{% include "header.html" %}
    <h1>Product Details</h1>
    <table>
        <tr><td>Code</td><td>{{ code }}</td></tr>
        <tr><td>Description</td><td>{{
product["name"]}}</td></tr>
        <tr><td>Price</td><td>{{ product["price"]}}</td></tr>
        {% if session["user"] is defined %}
          <tr><td colspan=2><a href="/addtocart?code={{ code
}}">Add to Cart</a></td></tr>
        {% endif %}
    </table>
    <a href="/products">Back</a>
{% include "footer.html" %}
```

Notes:

- We needed to set the **colspan** attribute of <td> to 2 for the Add to Cart link.

2. Edit **app.py** and add the new route **/addtocart**:

```
@app.route('/addtocart')
def addtocart():
    code = request.args.get('code', '')
    product = db.get_product(int(code))
    item=dict()
    # A click to add a product translates to a
    # quantity of 1 for now

    item["qty"] = 1
    item["name"] = product["name"]
    item["subtotal"] = product["price"]*item["qty"]

    if(session.get("cart") is None):
        session["cart"]={}

    cart = session["cart"]
    cart[code]=item
    session["cart"]=cart
    return redirect('/cart')
```

Note the redirect to a new route **/cart**, which we shall create in the next step.

3. Still in **app.py**, add a new route **/cart** as follows:

```
@app.route('/cart')
```

```
def cart():
    return render_template('cart.html')
```

4. Still in **app.py**, modify the route **/logout** as follows:

```
@app.route('/logout')
def logout():
    session.pop("user",None)
    session.pop("cart",None)
    return redirect('/')
```

5. Create a new file **cart.html** under the **templates** folder with the following lines of code:

```
{% include "header.html" %}
        <h1>Cart</h1>

            {% if session["cart"] is defined %}
            <table>

<tr><th>Name</th><th>Quantity</th><th>Subtotal</th></tr>
            {% for item in session["cart"].values() %}
                        <tr><td>{{ item["name"] }}</td><td>{{
item["qty"] }}</td><td>{{ item["subtotal"] }}</td></tr>
            {% endfor %}
            <tr><td colspan=2><b>Total</b></td><td><b>{{
session["cart"].values()|sum(attribute="subtotal")
}}</b></td></tr>
            </table>
            {% else %}
            <div>Your cart is empty</div>
            {% endif %}

{% include "footer.html" %}
```

Save your changes.

Implementation Notes:

- The computation of Totals of the Shopping Cart is a fairly advanced topic involving Jinja2 Templating. If you want to know more about how it works, here is the link: https://jinja.palletsprojects.com/en/master/templates/#sum ; however, for our purposes in this lab session, just assume that it works.

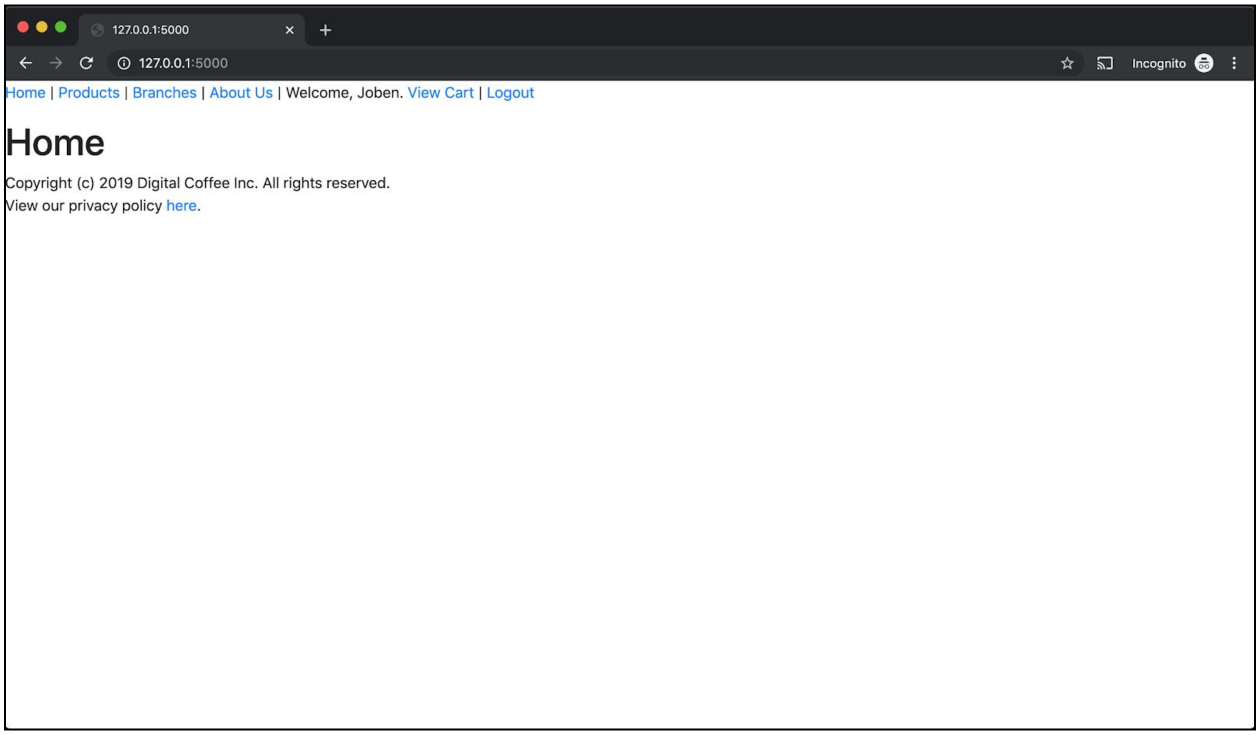
6. Update **header.html** to include a link to the Cart from the nav bar. Only show the link to Cart when a user is logged in.

```
  <a href='/'>Home</a> | <a href='/products'>Products</a> |
        <a href='/branches'>Branches</a> | <a
href='/aboutus'>About Us</a> |
        {% if session["user"] is defined %}
        Welcome, {{ session["user"]["first_name"] }}.
        <a href='/cart'>View Cart</a> |
        <a href='/logout'>Logout</a>
        {% else %}
        <a href='/login'>Login</a>
        {% endif %}
        <p/>
```

7. Let's test our new web app. Reload our Home Page. Make sure we are logged off first. Login using a valid user:

   Username or email address: joben@example.com
   Password: Ch@ng3m3!

   You should see the Home screen with the new nav bar containing a link to **View Cart** similar to the screenshot below:

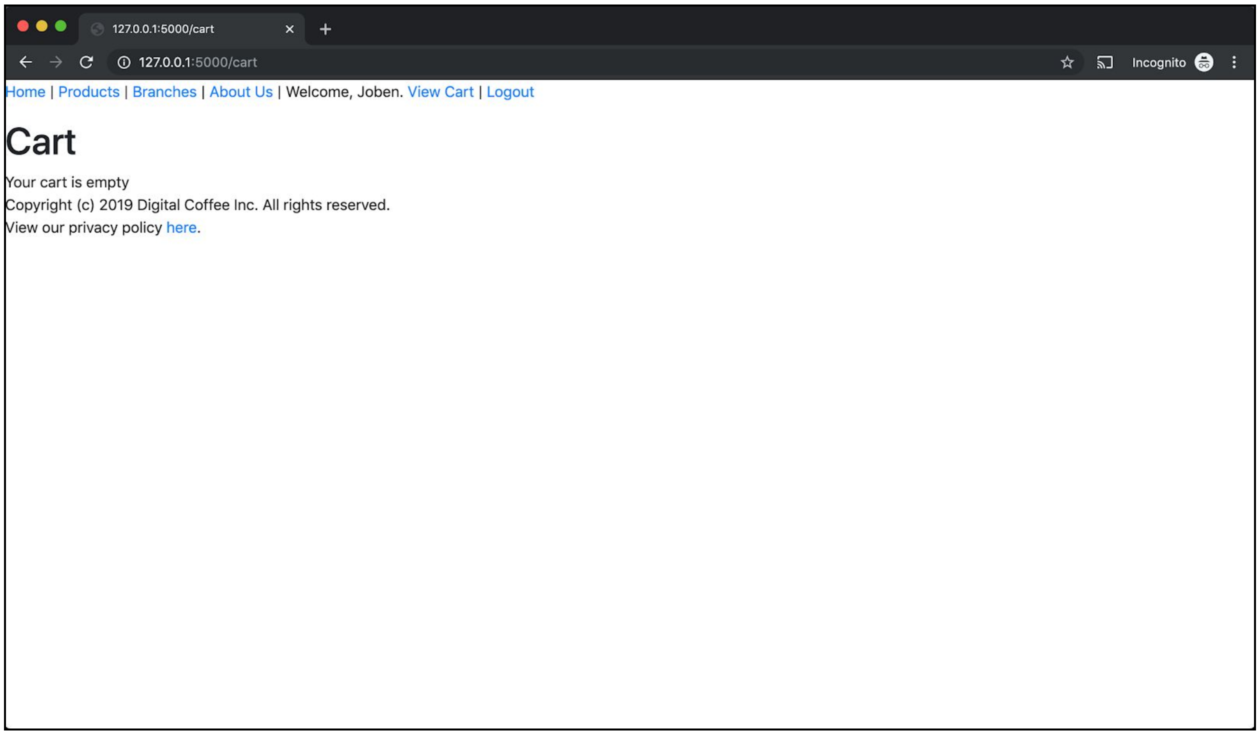

127.0.0.1:5000

Home | Products | Branches | About Us | Welcome, Joben. View Cart | Logout

Home

Copyright (c) 2019 Digital Coffee Inc. All rights reserved.
View our privacy policy here.

Click on the **View Cart** link. You should see the Cart page saying that the Shopping Cart is empty. We shall be adding products to our shopping cart next.

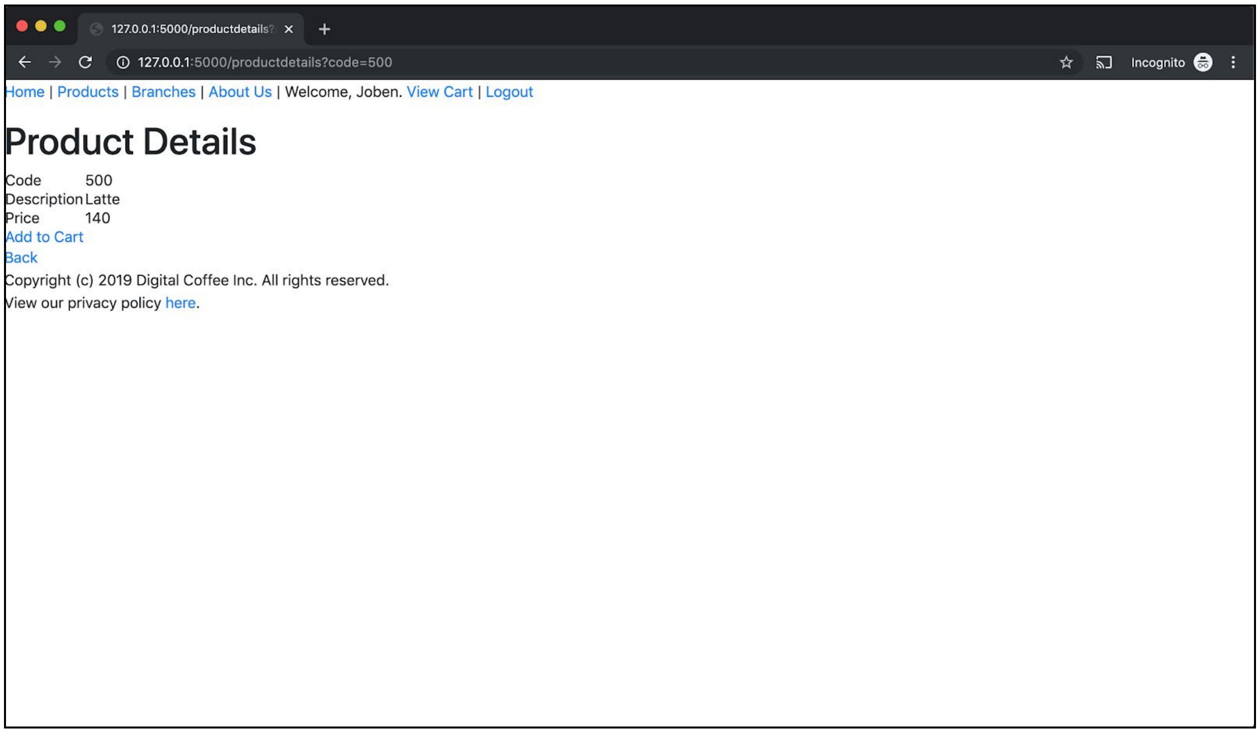

127.0.0.1:5000/cart

Home | Products | Branches | About Us | Welcome, Joben. View Cart | Logout

Cart

Your cart is empty
Copyright (c) 2019 Digital Coffee Inc. All rights reserved.
View our privacy policy here.

Click on **Products**. Click on Latte (500). You should see the Product Details screen with a link to **Add to Cart**, Hover your mouse cursor over the Add to Cart link. You should see a URL pointing to **/addtocart** with a parameter to code=500.


127.0.0.1:5000/productdetails?
127.0.0.1:5000/productdetails?code=500
Incognito

Home | Products | Branches | About Us | Welcome, Joben. View Cart | Logout

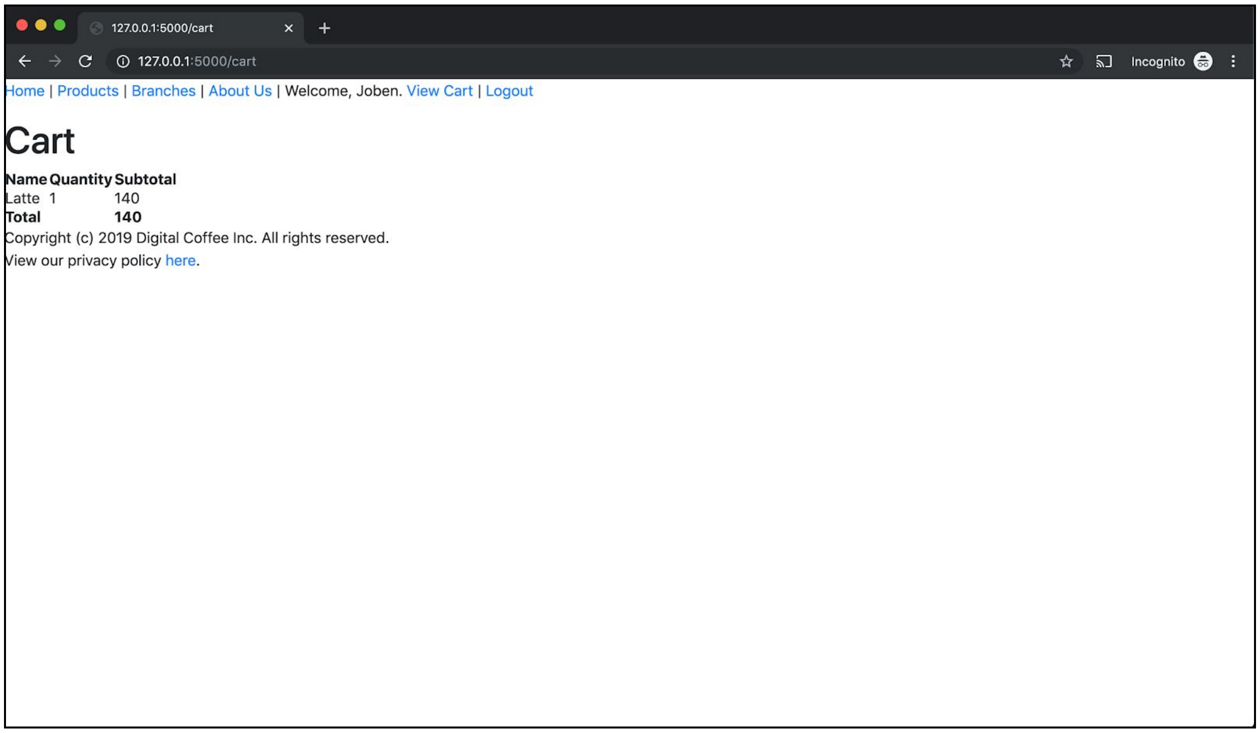
Product Details

Code	500
Description	Latte
Price	140

Add to Cart
Back
Copyright (c) 2019 Digital Coffee Inc. All rights reserved.
View our privacy policy here.

Click on the **Add to Cart** link.

You should be redirected to the Cart page. Your Shopping Cart now has one entry like so:



Home | Products | Branches | About Us | Welcome, Joben. View Cart | Logout

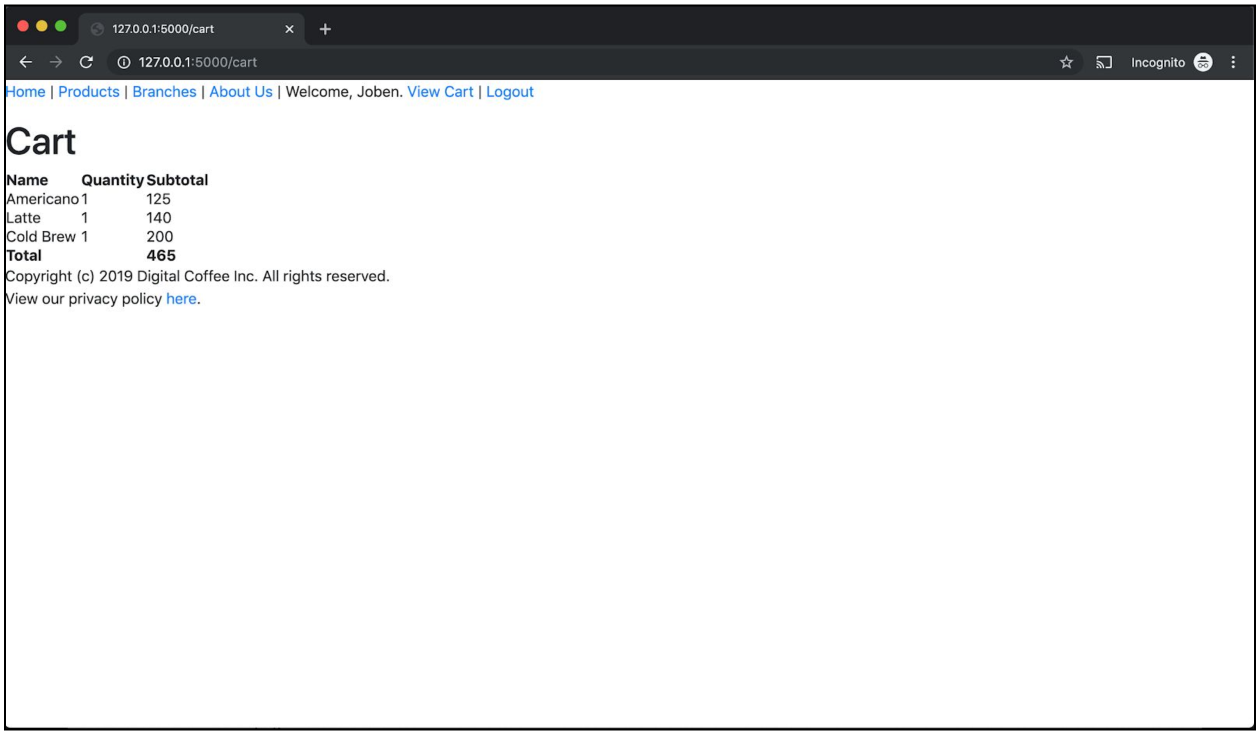
Cart

Name	Quantity	Subtotal
Latte	1	140
Total		**140**

Copyright (c) 2019 Digital Coffee Inc. All rights reserved.
View our privacy policy here.

Click on **Products** again and add new products to the shopping cart as you see fit.

Depending on what products you choose to add to the cart, you may see something like this:



Home | Products | Branches | About Us | Welcome, Joben. View Cart | Logout

Cart

Name	Quantity	Subtotal
Americano	1	125
Latte	1	140
Cold Brew	1	200
Total		**465**

Copyright (c) 2019 Digital Coffee Inc. All rights reserved.
View our privacy policy here.

Click on **Logout**. You should have been redirected to the Home page. You also won't be able to see the **View Cart** link.

Log in again using the following credentials:

Username or email address: chums@example.com
Password: Ch@ng3m3!

Once you are brought to the Home page, click on **View Cart**. You should see the Cart page with an empty shopping cart.

Take-Home Quiz #3

Total Points: 30
Due Date: Tuesday, November 12, 6:00PM

This will be a **Group** quiz.

Commit your Flask application folder into your Group GitHub repository. Once done, notify me via email at **jbilagan@ateneo.edu**.

This quiz will be a little more challenging than the previous one because there will be less guidance on what to do.

1. Enhance the Login page to include the following generic error **"Invalid username or password. Please try again."** if any of the following happen:

    a. Username entered is invalid
    b. Password is wrong
    c. Incomplete login data is passed (missing username or password or both)

2. In the Cart page, if there are items on display, add functionality to be able to add quantity. Allow the user to specify the actual quantity (hint: you may have to use HTML forms for this).

3. Also in the Cart page, add a function to be able to remove a specific item from the Cart.