CS421 Final Project

Paul Detloff
Wyatt Cannon
Saif Alam

Aug 2nd, 2023
CS421 Advance Web Application Development

Objective

The objective of this project was to create a functional website capable of various features such as a sign up/sign in system for users, including creating a username/password, as well as adding menu items to cart/placing and order, and including manager functionality to use admin services on the website. In this case, the website was an online coffee shop application.

Program

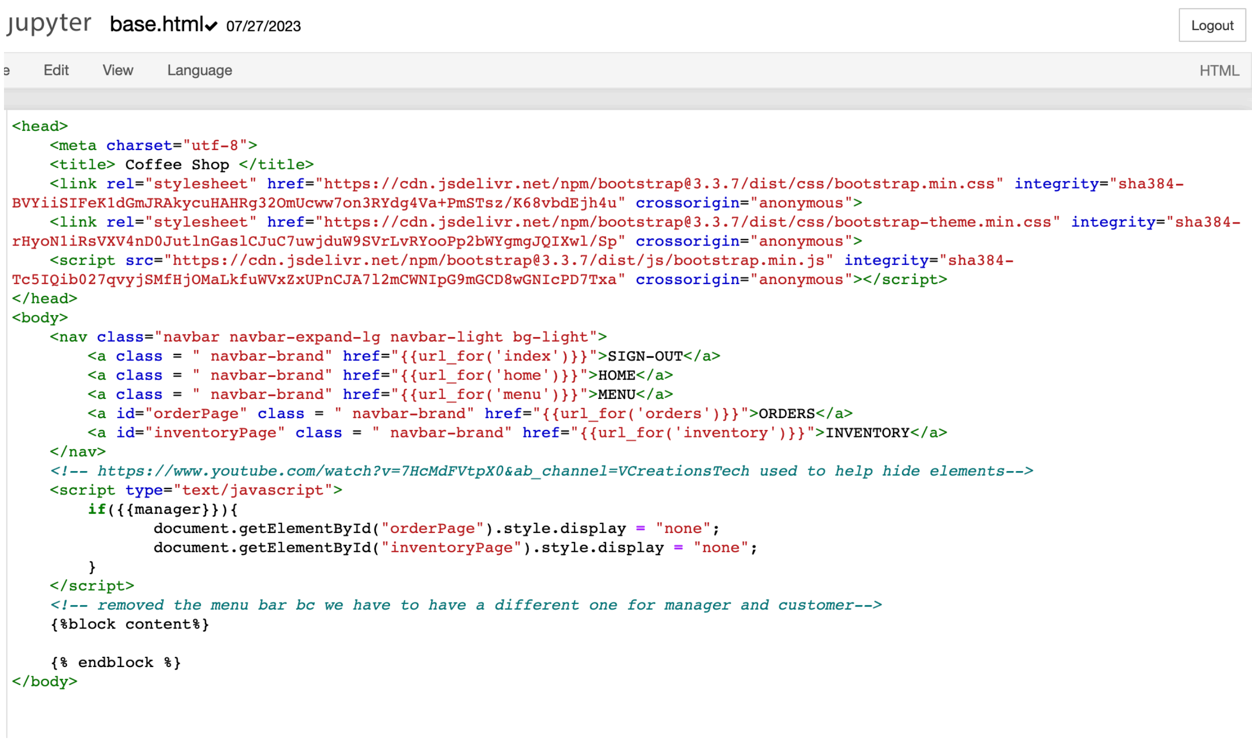
The project was written in multiple files under the folder cs421project-main. The following outlines were taken to complete the assignment:

- Create HTML Documents in accordance with each page required (home page, sign up, orders, etc.)
- Creating a sign in/sign up page for users with the ability to be stored into a database (SQLITE)
- Creating a unique key for manager privileges to the website that is otherwise hidden to the rest of the users, allowing for manager functionality tools
- Creating a python file to retrieve database information of inventory, as well as storing users information
- Creating a python file for back end services, tying in all the functionality

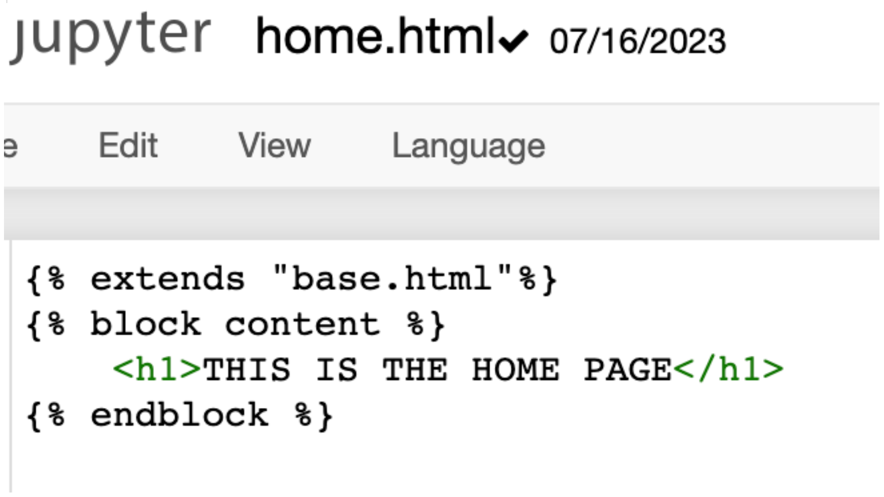
Code Design

The following is the design and implementation of the program:

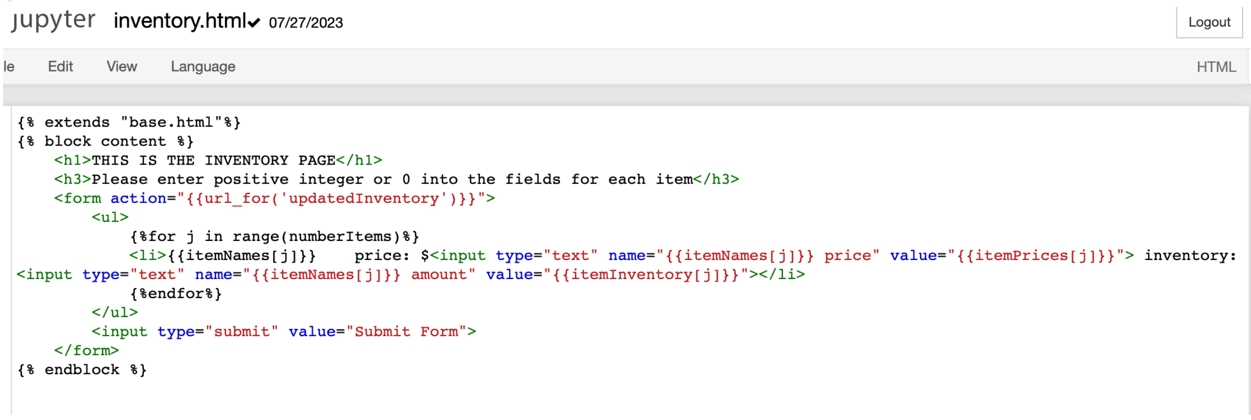
- Templates
    - Templates contained all the page files, written in HTML. The files were divided into 10 different pages. With an initial start page, users are redirected to sign up or log in.
    - Each HTML document inherits from the base HTML doc. They use forms to take in information and post information with python arrays and variables. Jinja was used for this. Here are screen shots of the HTML.



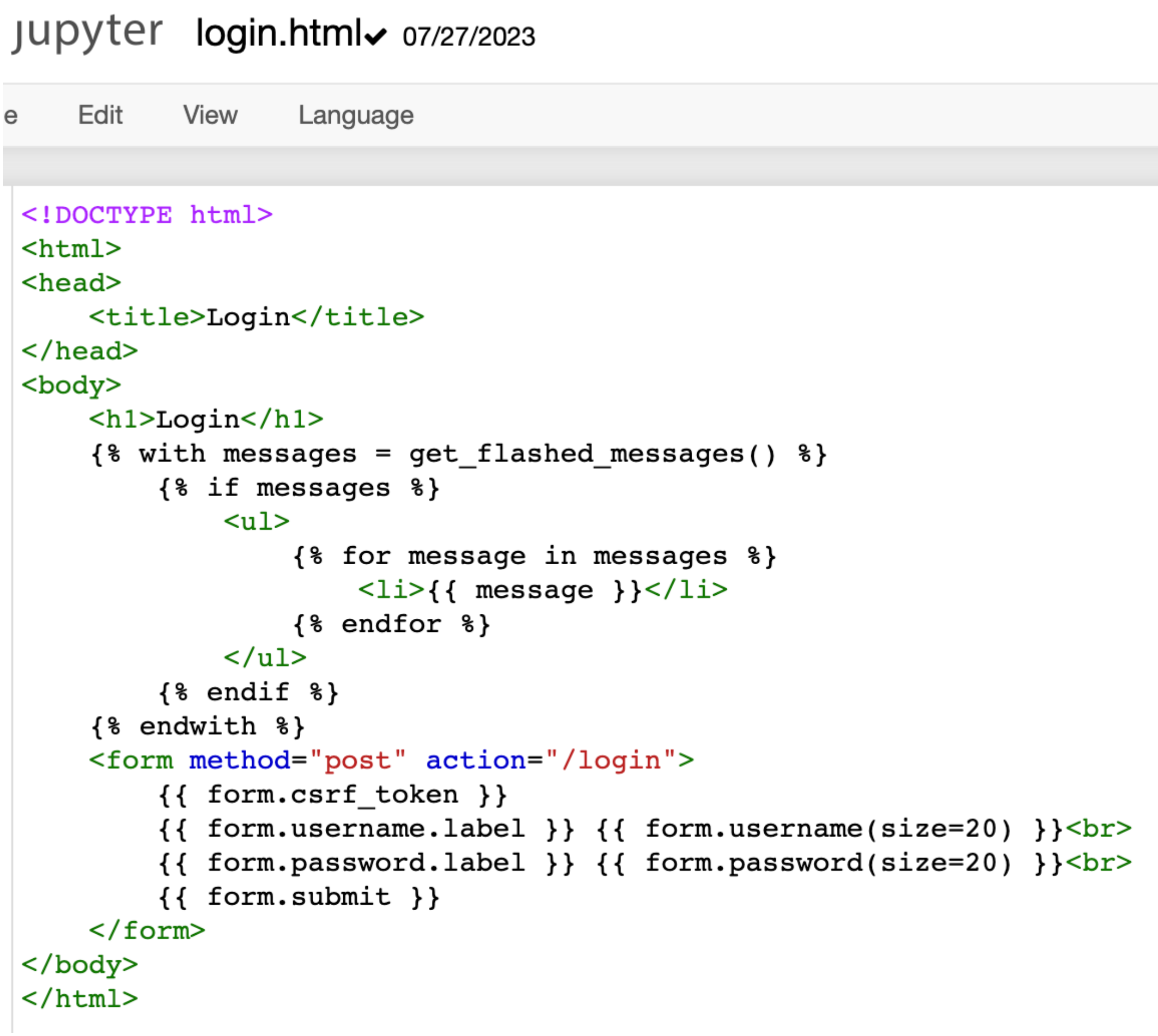
```html
<head>
    <meta charset="utf-8">
    <title> Coffee Shop </title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@3.3.7/dist/css/bootstrap.min.css" integrity="sha384-
BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@3.3.7/dist/css/bootstrap-theme.min.css" integrity="sha384-
rHyoN1iRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmgJQIXwl/Sp" crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@3.3.7/dist/js/bootstrap.min.js" integrity="sha384-
Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa" crossorigin="anonymous"></script>
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <a class = " navbar-brand" href="{{url_for('index')}}">SIGN-OUT</a>
        <a class = " navbar-brand" href="{{url_for('home')}}">HOME</a>
        <a class = " navbar-brand" href="{{url_for('menu')}}">MENU</a>
        <a id="orderPage" class = " navbar-brand" href="{{url_for('orders')}}">ORDERS</a>
        <a id="inventoryPage" class = " navbar-brand" href="{{url_for('inventory')}}">INVENTORY</a>
    </nav>
    <!-- https://www.youtube.com/watch?v=7HcMdFVtpX0&ab_channel=VCreationsTech used to help hide elements-->
    <script type="text/javascript">
        if({{manager}}){
                document.getElementById("orderPage").style.display = "none";
                document.getElementById("inventoryPage").style.display = "none";
        }
    </script>
    <!-- removed the menu bar bc we have to have a different one for manager and customer-->
    {%block content%}

    {% endblock %}
</body>
```



```html
{% extends "base.html"%}
{% block content %}
    <h1>THIS IS THE HOME PAGE</h1>
{% endblock %}
```

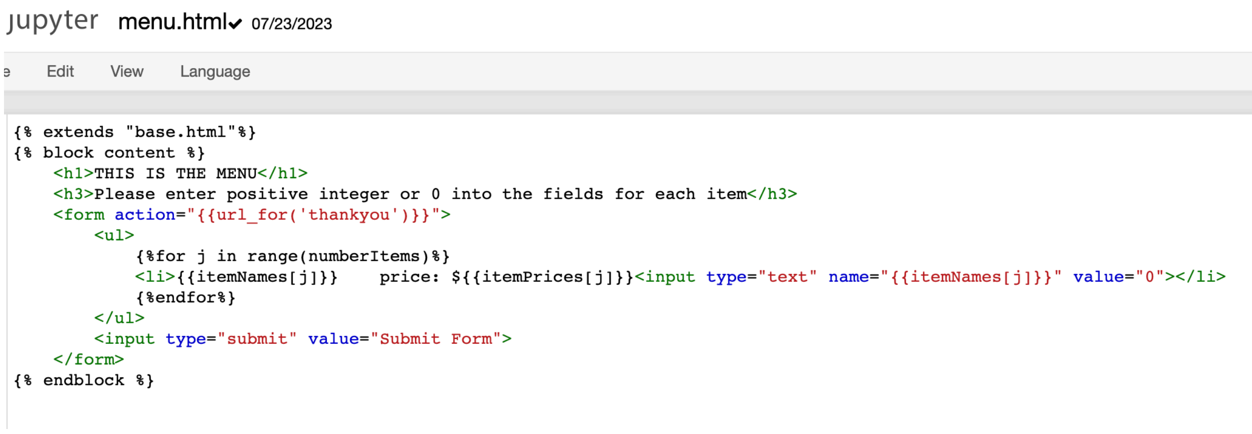


```html
{% extends "base.html"%}
{% block content %}
    <h1>THIS IS THE INVENTORY PAGE</h1>
    <h3>Please enter positive integer or 0 into the fields for each item</h3>
    <form action="{{url_for('updatedInventory')}}">
        <ul>
            {%for j in range(numberItems)%}
            <li>{{itemNames[j]}}    price: $<input type="text" name="{{itemNames[j]}} price" value="{{itemPrices[j]}}"> inventory:
<input type="text" name="{{itemNames[j]}} amount" value="{{itemInventory[j]}}"></li>
            {%endfor%}
        </ul>
        <input type="submit" value="Submit Form">
    </form>
{% endblock %}
```



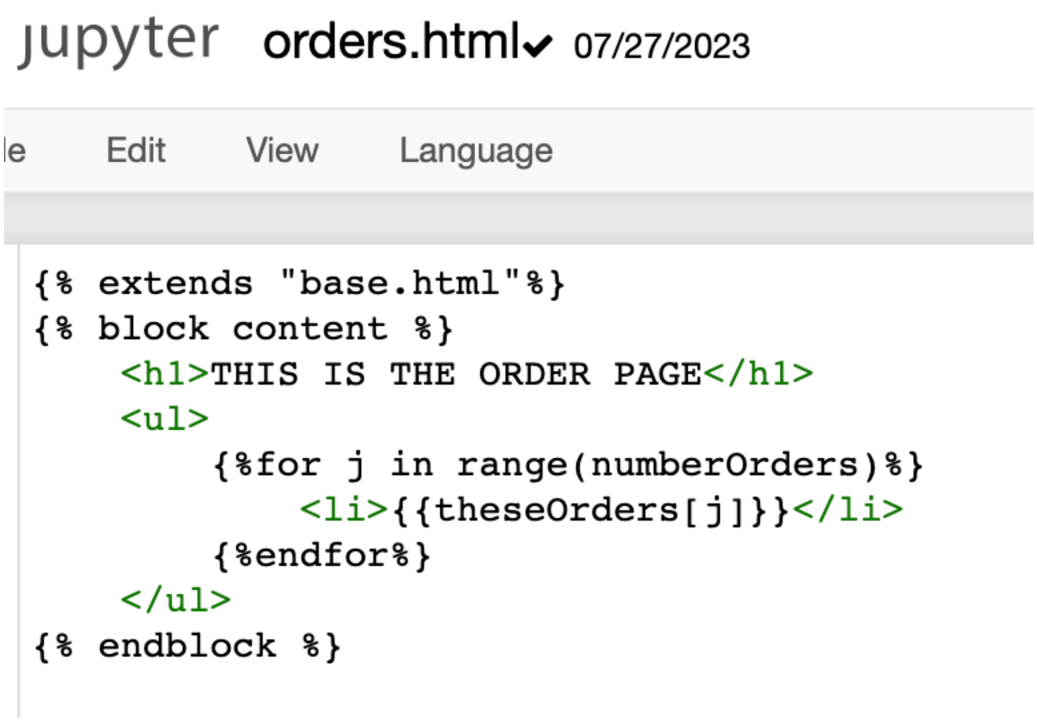
```
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
    <h1>Login</h1>
    {% with messages = get_flashed_messages() %}
        {% if messages %}
            <ul>
                {% for message in messages %}
                    <li>{{ message }}</li>
                {% endfor %}
            </ul>
        {% endif %}
    {% endwith %}
    <form method="post" action="/login">
        {{ form.csrf_token }}
        {{ form.username.label }} {{ form.username(size=20) }}<br>
        {{ form.password.label }} {{ form.password(size=20) }}<br>
        {{ form.submit }}
    </form>
</body>
</html>
```

○



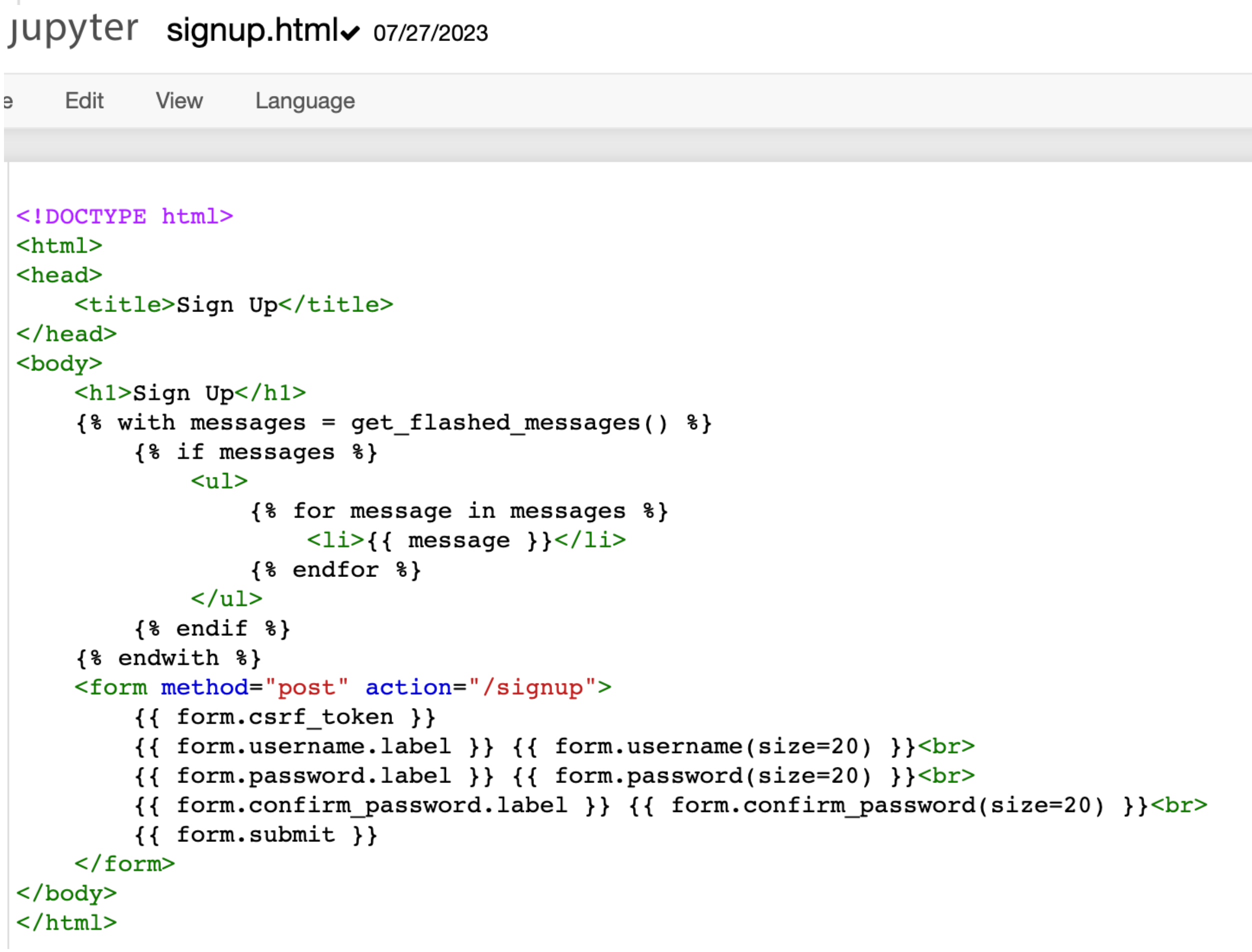
```
{% extends "base.html"%}
{% block content %}
    <h1>THIS IS THE MENU</h1>
    <h3>Please enter positive integer or 0 into the fields for each item</h3>
    <form action="{{url_for('thankyou')}}">
        <ul>
            {%for j in range(numberItems)%}
            <li>{{itemNames[j]}}    price: ${{itemPrices[j]}}<input type="text" name="{{itemNames[j]}}" value="0"></li>
            {%endfor%}
        </ul>
        <input type="submit" value="Submit Form">
    </form>
{% endblock %}
```

○



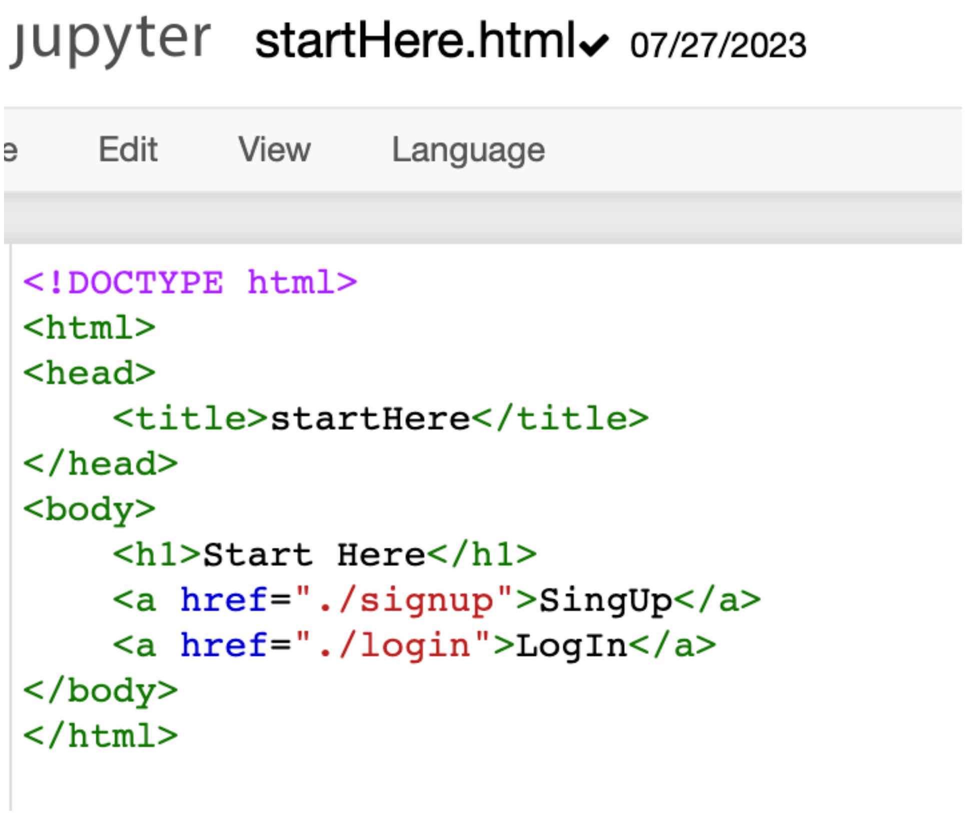
```
{% extends "base.html"%}
{% block content %}
    <h1>THIS IS THE ORDER PAGE</h1>
    <ul>
        {%for j in range(numberOrders)%}
            <li>{{theseOrders[j]}}</li>
        {%endfor%}
    </ul>
{% endblock %}
```

○

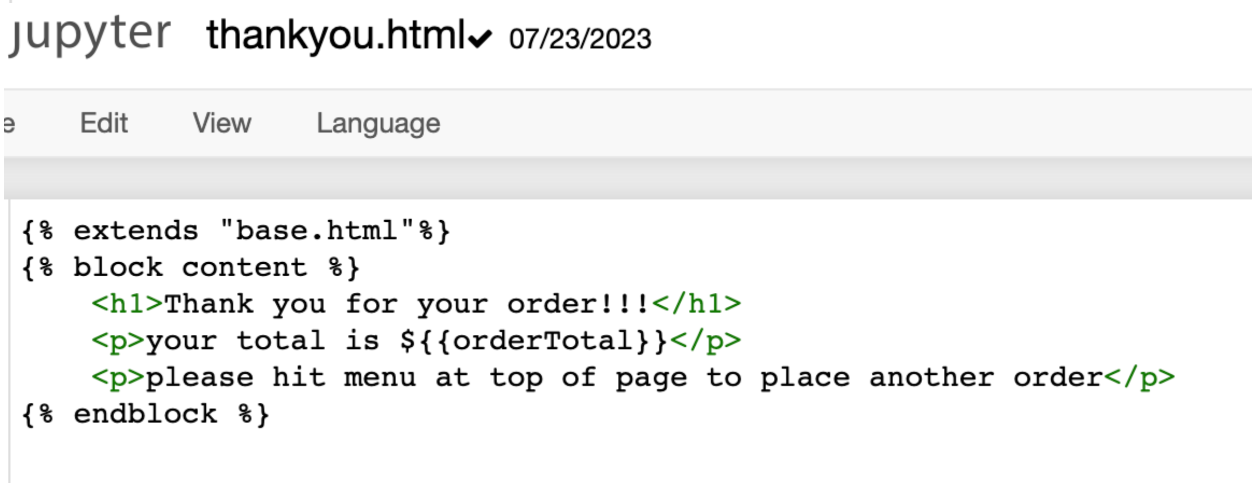


```
<!DOCTYPE html>
<html>
<head>
    <title>Sign Up</title>
</head>
<body>
    <h1>Sign Up</h1>
    {% with messages = get_flashed_messages() %}
        {% if messages %}
            <ul>
                {% for message in messages %}
                    <li>{{ message }}</li>
                {% endfor %}
            </ul>
        {% endif %}
    {% endwith %}
    <form method="post" action="/signup">
        {{ form.csrf_token }}
        {{ form.username.label }} {{ form.username(size=20) }}<br>
        {{ form.password.label }} {{ form.password(size=20) }}<br>
        {{ form.confirm_password.label }} {{ form.confirm_password(size=20) }}<br>
        {{ form.submit }}
    </form>
</body>
</html>
```

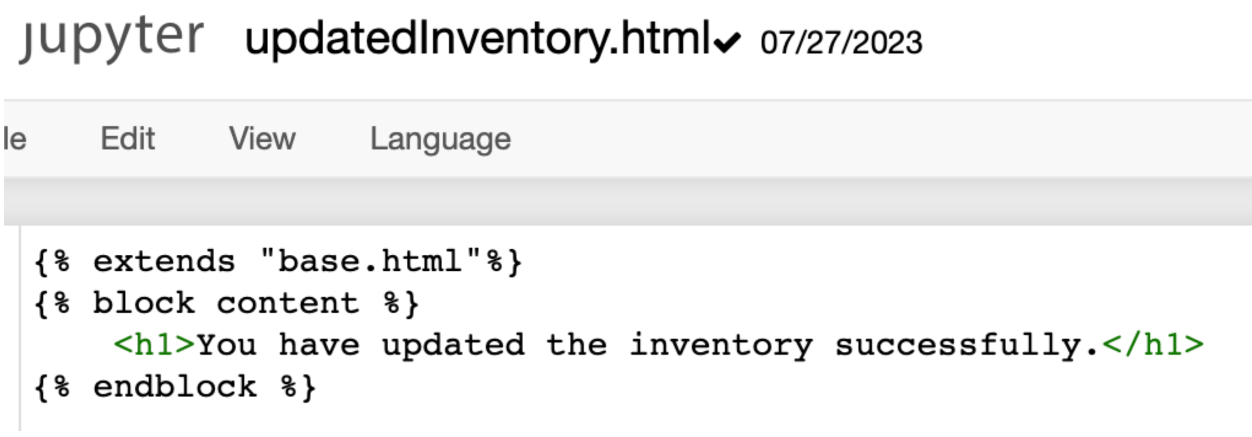
○



```
<!DOCTYPE html>
<html>
<head>
    <title>startHere</title>
</head>
<body>
    <h1>Start Here</h1>
    <a href="./signup">SingUp</a>
    <a href="./login">LogIn</a>
</body>
</html>
```

- 

```
{% extends "base.html"%}
{% block content %}
    <h1>Thank you for your order!!!</h1>
    <p>your total is ${{orderTotal}}</p>
    <p>please hit menu at top of page to place another order</p>
{% endblock %}
```

- 

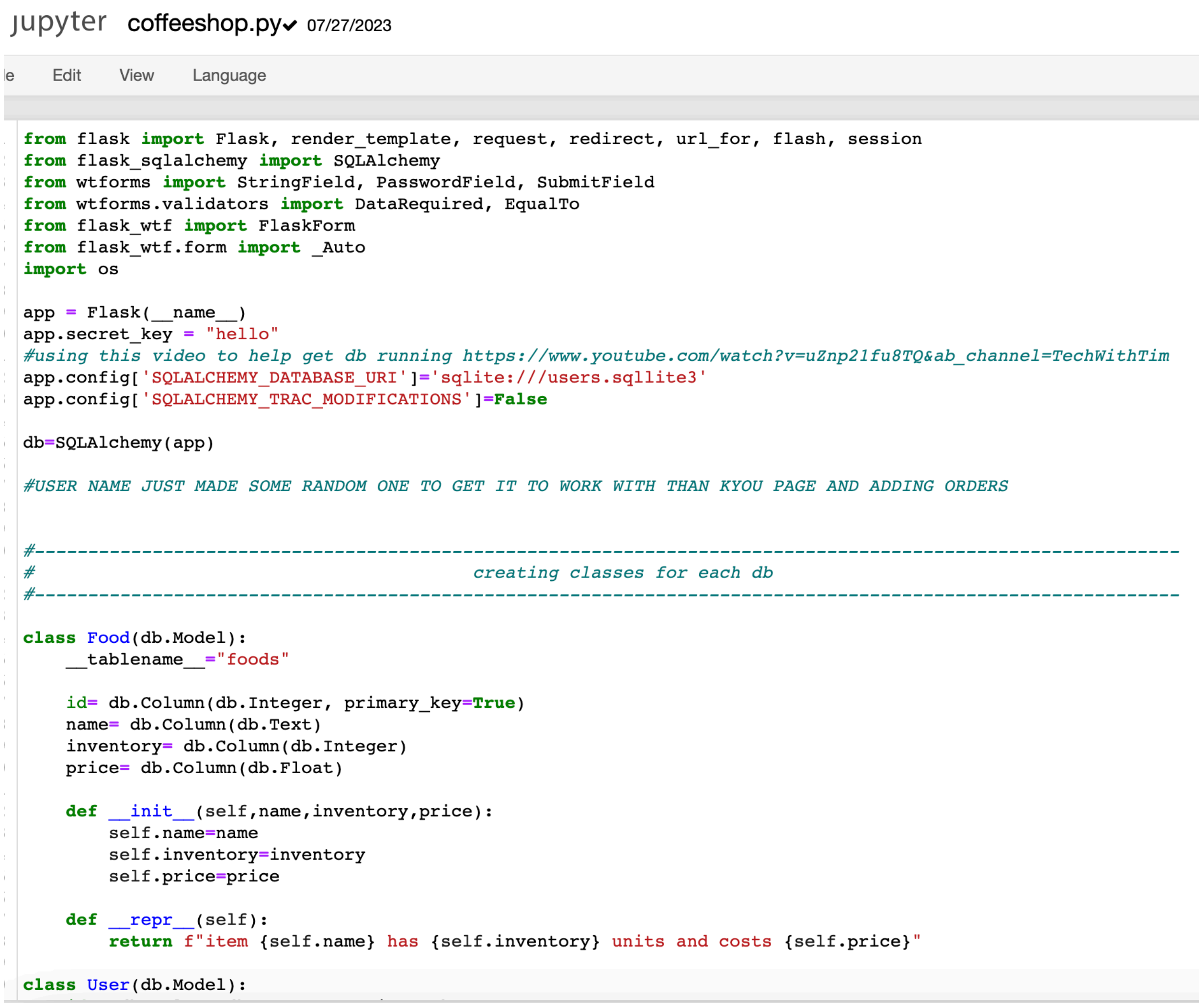
```
{% extends "base.html"%}
{% block content %}
    <h1>You have updated the inventory successfully.</h1>
{% endblock %}
```

-

- The Python code sets up a web application using the Flask framework to manage the shop. The functionalities are as follow:
    - **Imports and Configuration** – Importing the necessary modules like Flask, SQLAlchemy, and WTForms.
    - **Database Models**: Three classes (**Food**, **User**, and **Orders**) are defined as SQLAlchemy models representing the database tables for food items, users, and order information. These classes define the structure of each table and their relationships.
    - **Web Forms**: Two FlaskForm classes (**SignupForm** and **LoginForm**) are created using WTForms to handle user sign-up and login forms
    - **Database Creation and Initialization**: The code creates the necessary tables and initializes the database with some sample food items and a user
    - Routing and Views:
        - '/signup': Handles user sign-up. Validates form input and adds a new user to the database.
        - '/login': Handles user login. Validates form input and checks user credentials against the database.
        - '/home': Displays the home page, differentiating between managers and regular users.
        - '/menu': Displays the coffee shop menu, reading the food items and prices from the database.
        - '/thankyou': Processes user orders, calculates the total price, and updates the inventory in the database. It then shows a thank-you page.
        - '/orders': Shows all the orders placed so far.
        - '/inventory': Displays the inventory of food items, reading it from the database.
        - '/updatedInventory': Updates the inventory and prices of food items in the database
    - **Global Variables**: The code uses a global variable **manager** to differentiate between regular users and the manager
    -
- The Database code creates the SQLAlchemy database model.
    - **Database Model**: The code defines a class named **Food** as a SQLAlchemy model. This class represents the structure of the **foods** table in the database. Name, inventory, price are included.
        - **__init__ -** method to initialize instances of the Food class
        - **__repr__** - method to provide a string representation of the class instances.

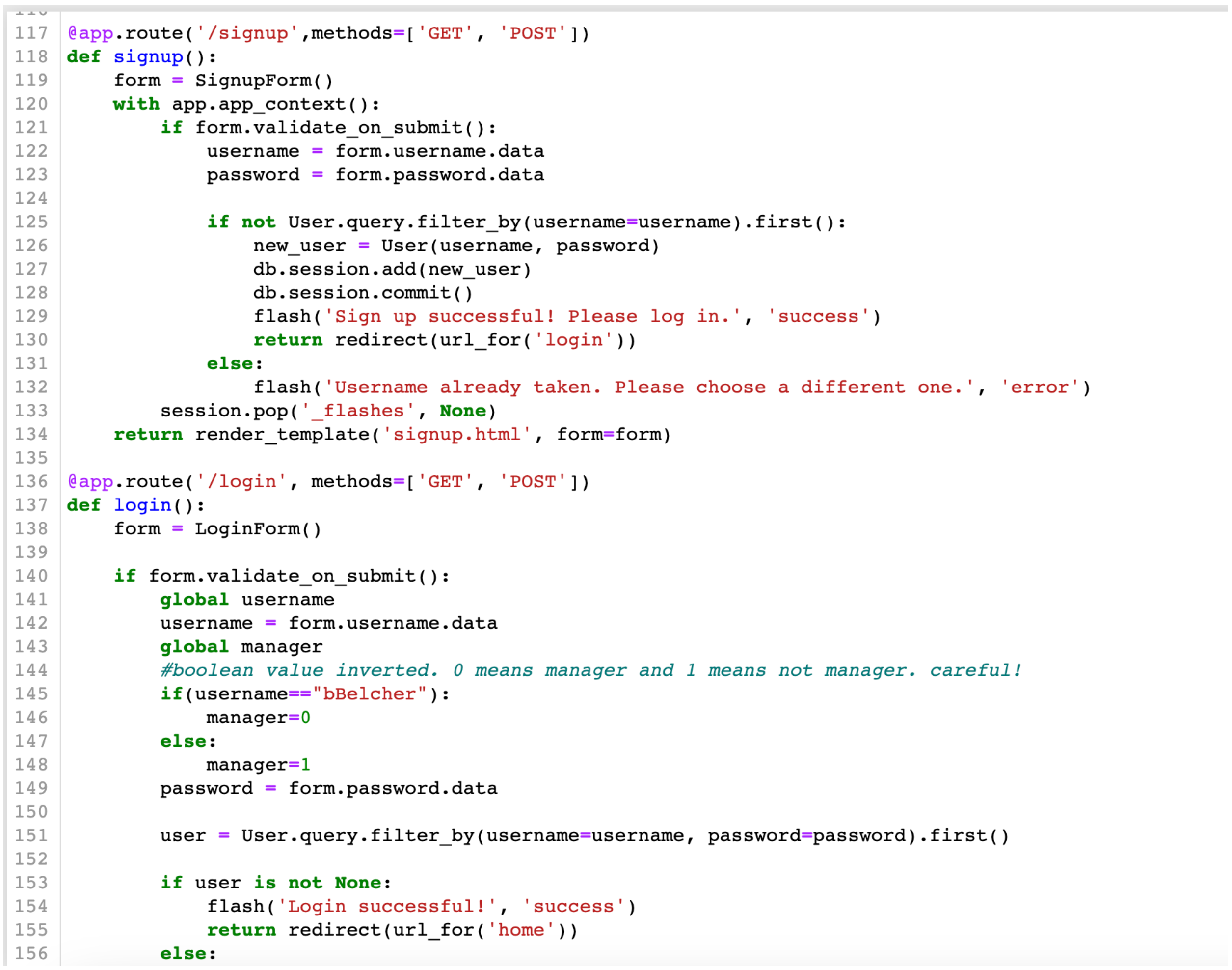
- **Database Configuration**: Creates a Flask application and configures the database URI to **data.sqlite**.
    - **SQLALCHEMY_TRAC_MODIFICATIONS: False -** to suppresses modification tracking. Additionally,
    - **SQLALCHEMY_ECHO**: **True** - prints all SQL statements generated by SQLAlchemy to the console.

Main Python Screeen Shots:

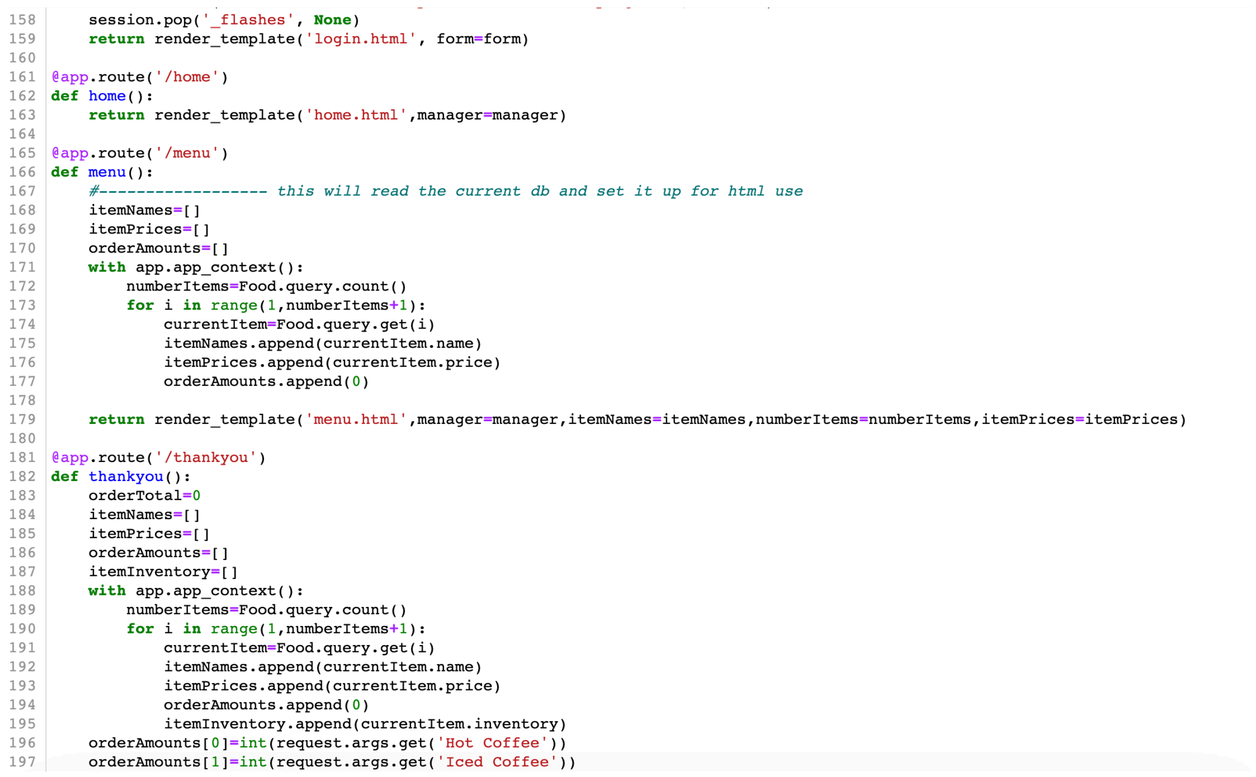


```
from flask import Flask, render_template, request, redirect, url_for, flash, session
from flask_sqlalchemy import SQLAlchemy
from wtforms import StringField, PasswordField, SubmitField
from wtforms.validators import DataRequired, EqualTo
from flask_wtf import FlaskForm
from flask_wtf.form import _Auto
import os

app = Flask(__name__)
app.secret_key = "hello"
#using this video to help get db running https://www.youtube.com/watch?v=uZnp21fu8TQ&ab_channel=TechWithTim
app.config['SQLALCHEMY_DATABASE_URI']='sqlite:///users.sqllite3'
app.config['SQLALCHEMY_TRAC_MODIFICATIONS']=False

db=SQLAlchemy(app)

#USER NAME JUST MADE SOME RANDOM ONE TO GET IT TO WORK WITH THAN KYOU PAGE AND ADDING ORDERS


#------------------------------------------------------------------------------------------------------------
#                                          creating classes for each db
#------------------------------------------------------------------------------------------------------------

class Food(db.Model):
    __tablename__="foods"

    id= db.Column(db.Integer, primary_key=True)
    name= db.Column(db.Text)
    inventory= db.Column(db.Integer)
    price= db.Column(db.Float)

    def __init__(self,name,inventory,price):
        self.name=name
        self.inventory=inventory
        self.price=price

    def __repr__(self):
        return f"item {self.name} has {self.inventory} units and costs {self.price}"

class User(db.Model):
```

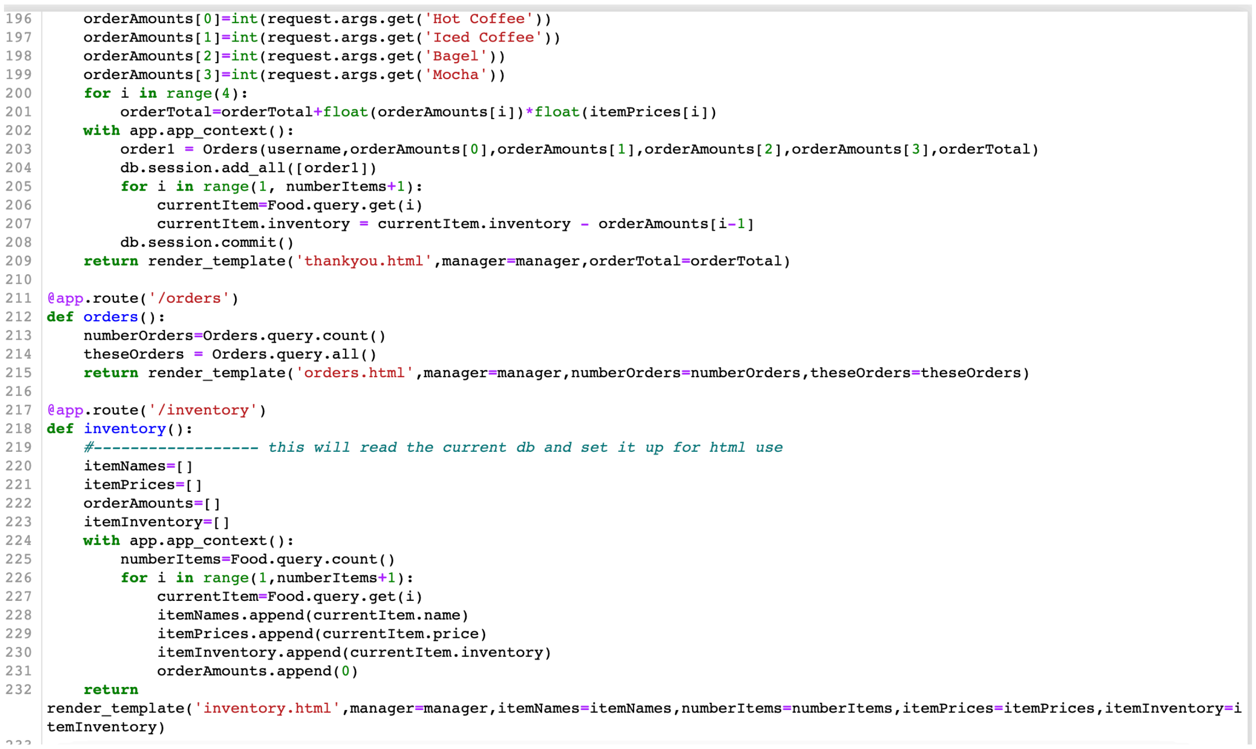
```python
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    password = db.Column(db.String(120), nullable=False)
    #@isManager = db.Column(db.Integer, nullable=False)

    def __init__(self,username,password):
        self.username=username
        self.password=password
        #self.isManager=isManager

    def __repr__(self):
        return f'<User {self.username}>'

class Orders(db.Model):
    __tablename__="orders"

    id= db.Column(db.Integer, primary_key=True)
    uName= db.Column(db.Text)
    amountHotCoffee= db.Column(db.Integer)
    amountIcedCoffee= db.Column(db.Integer)
    amountBagel= db.Column(db.Integer)
    amountMocha= db.Column(db.Integer)
    saleTotal= db.Column(db.Float)

    def __init__(self,uName,amountHotCoffee,amountIcedCoffee,amountBagel,amountMocha,saleTotal):
        self.uName=uName
        self.amountHotCoffee=amountHotCoffee
        self.amountIcedCoffee=amountIcedCoffee
        self.amountBagel=amountBagel
        self.amountMocha=amountMocha
        self.saleTotal=saleTotal

    def __repr__(self):
        return f"{self.uName} ordered {self.amountHotCoffee} Hot Coffee, {self.amountIcedCoffee} Iced Coffee, {self.amountBagel}Bagel, and {self.amountMocha} Mocha. Total is ${self.saleTotal}."


class SignupForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    confirm_password = PasswordField('Confirm Password', validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Sign Up')

class LoginForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Login')
#------------------------------------------------------------------------------------------------------------
#                                         creating db and adding foods
#------------------------------------------------------------------------------------------------------------

with app.app_context():
    db.create_all()


hotCoffee = Food('Hot Coffee',35,1.75)
icedCoffee = Food('Iced Coffee',33,2.50)
bagel = Food('Bagel',121,2.00)
mocha = Food('Mocha',12,4.25)
bBelcher = User("bBelcher","password123")

with app.app_context():
    Food.query.delete() #this clears the database bc it won't remove all the items from prior sessions
    Orders.query.delete()
    User.query.delete()
    db.session.add_all([hotCoffee,icedCoffee,bagel,mocha])
    db.session.add_all([bBelcher])
    db.session.commit()





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

@app.route('/signup',methods=['GET', 'POST'])
def signup():
    form = SignupForm()
```



```python
@app.route('/signup',methods=['GET', 'POST'])
def signup():
    form = SignupForm()
    with app.app_context():
        if form.validate_on_submit():
            username = form.username.data
            password = form.password.data

            if not User.query.filter_by(username=username).first():
                new_user = User(username, password)
                db.session.add(new_user)
                db.session.commit()
                flash('Sign up successful! Please log in.', 'success')
                return redirect(url_for('login'))
            else:
                flash('Username already taken. Please choose a different one.', 'error')
        session.pop('_flashes', None)
    return render_template('signup.html', form=form)

@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()

    if form.validate_on_submit():
        global username
        username = form.username.data
        global manager
        #boolean value inverted. 0 means manager and 1 means not manager. careful!
        if(username=="bBelcher"):
            manager=0
        else:
            manager=1
        password = form.password.data

        user = User.query.filter_by(username=username, password=password).first()

        if user is not None:
            flash('Login successful!', 'success')
            return redirect(url_for('home'))
        else:
```

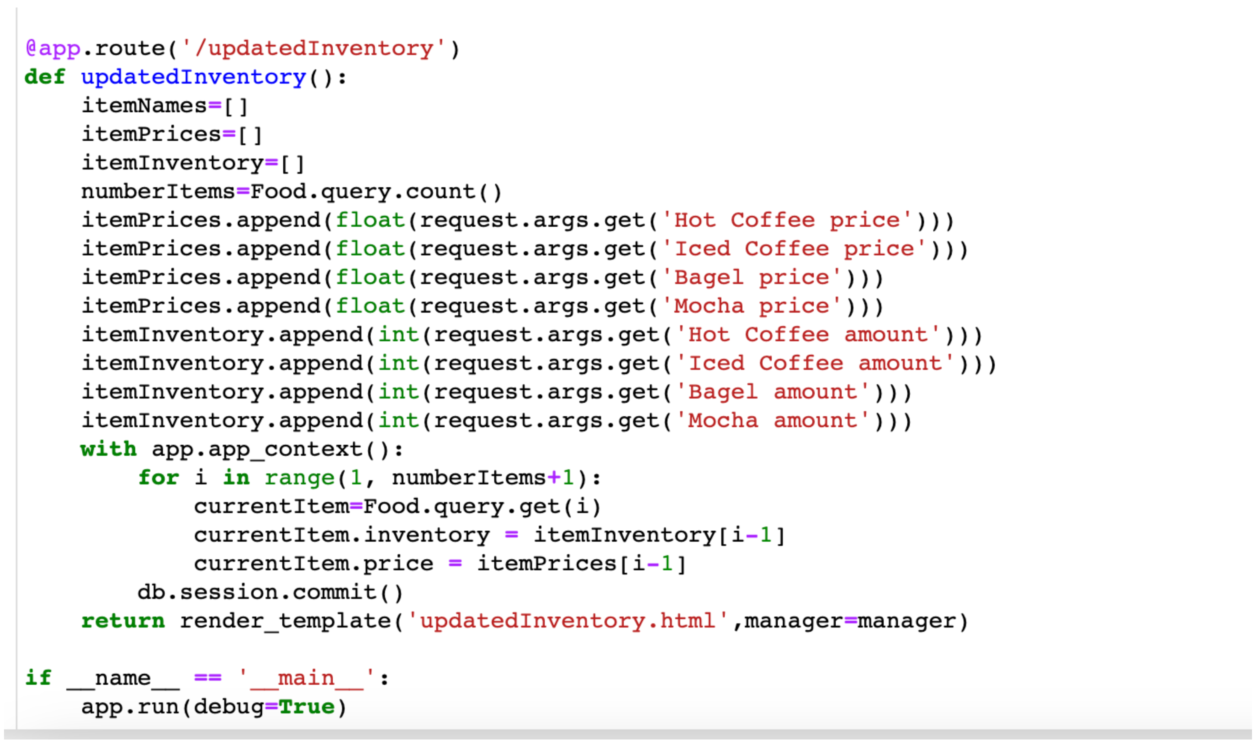


```python
    session.pop('_flashes', None)
    return render_template('login.html', form=form)

@app.route('/home')
def home():
    return render_template('home.html',manager=manager)

@app.route('/menu')
def menu():
    #------------------ this will read the current db and set it up for html use
    itemNames=[]
    itemPrices=[]
    orderAmounts=[]
    with app.app_context():
        numberItems=Food.query.count()
        for i in range(1,numberItems+1):
            currentItem=Food.query.get(i)
            itemNames.append(currentItem.name)
            itemPrices.append(currentItem.price)
            orderAmounts.append(0)

    return render_template('menu.html',manager=manager,itemNames=itemNames,numberItems=numberItems,itemPrices=itemPrices)

@app.route('/thankyou')
def thankyou():
    orderTotal=0
    itemNames=[]
    itemPrices=[]
    orderAmounts=[]
    itemInventory=[]
    with app.app_context():
        numberItems=Food.query.count()
        for i in range(1,numberItems+1):
            currentItem=Food.query.get(i)
            itemNames.append(currentItem.name)
            itemPrices.append(currentItem.price)
            orderAmounts.append(0)
            itemInventory.append(currentItem.inventory)
    orderAmounts[0]=int(request.args.get('Hot Coffee'))
    orderAmounts[1]=int(request.args.get('Iced Coffee'))
```



```python
    orderAmounts[0]=int(request.args.get('Hot Coffee'))
    orderAmounts[1]=int(request.args.get('Iced Coffee'))
```

```python
    orderAmounts[2]=int(request.args.get('Bagel'))
    orderAmounts[3]=int(request.args.get('Mocha'))
    for i in range(4):
        orderTotal=orderTotal+float(orderAmounts[i])*float(itemPrices[i])
    with app.app_context():
        order1 = Orders(username,orderAmounts[0],orderAmounts[1],orderAmounts[2],orderAmounts[3],orderTotal)
        db.session.add_all([order1])
        for i in range(1, numberItems+1):
            currentItem=Food.query.get(i)
            currentItem.inventory = currentItem.inventory - orderAmounts[i-1]
        db.session.commit()
    return render_template('thankyou.html',manager=manager,orderTotal=orderTotal)

@app.route('/orders')
def orders():
    numberOrders=Orders.query.count()
    theseOrders = Orders.query.all()
    return render_template('orders.html',manager=manager,numberOrders=numberOrders,theseOrders=theseOrders)

@app.route('/inventory')
def inventory():
    #------------------ this will read the current db and set it up for html use
    itemNames=[]
    itemPrices=[]
    orderAmounts=[]
    itemInventory=[]
    with app.app_context():
        numberItems=Food.query.count()
        for i in range(1,numberItems+1):
            currentItem=Food.query.get(i)
            itemNames.append(currentItem.name)
            itemPrices.append(currentItem.price)
            itemInventory.append(currentItem.inventory)
            orderAmounts.append(0)
    return
render_template('inventory.html',manager=manager,itemNames=itemNames,numberItems=numberItems,itemPrices=itemPrices,itemInventory=i
temInventory)
```



```
@app.route('/updatedInventory')
def updatedInventory():
    itemNames=[]
    itemPrices=[]
    itemInventory=[]
    numberItems=Food.query.count()
    itemPrices.append(float(request.args.get('Hot Coffee price')))
    itemPrices.append(float(request.args.get('Iced Coffee price')))
    itemPrices.append(float(request.args.get('Bagel price')))
    itemPrices.append(float(request.args.get('Mocha price')))
    itemInventory.append(int(request.args.get('Hot Coffee amount')))
    itemInventory.append(int(request.args.get('Iced Coffee amount')))
    itemInventory.append(int(request.args.get('Bagel amount')))
    itemInventory.append(int(request.args.get('Mocha amount')))
    with app.app_context():
        for i in range(1, numberItems+1):
            currentItem=Food.query.get(i)
            currentItem.inventory = itemInventory[i-1]
            currentItem.price = itemPrices[i-1]
        db.session.commit()
    return render_template('updatedInventory.html',manager=manager)

if __name__ == '__main__':
    app.run(debug=True)
```

Analysis/Conclusion

Overall, the application allows users to sign up, log in, view the coffee shop's menu, place orders, and view and update the inventory. The manager can access additional functionalities not available to regular users, such as viewing all orders and updating inventory and prices. Web Application combines all the moving parts to form functionality. Most troubleshooting tests compatibility with certain programs/files, or in simpler words, “getting everything to click.” One way to take this assignment a step further is implement framework for payment processing.