Austin Byrd

CS 3300

Sprint 1

Part 2: Professional Documentation

To create a django app you need to first the code environment for the app. This is done by installing python, and django. Django requires python.

1. Install python
    https://www.python.org/downloads/

2. Make a python virtual environment
```
py -m venv my_virtual_environment
```

3. Install Django
```
py -m pip install Django
```

4. Create your project
```
django-admin startproject my_project
```

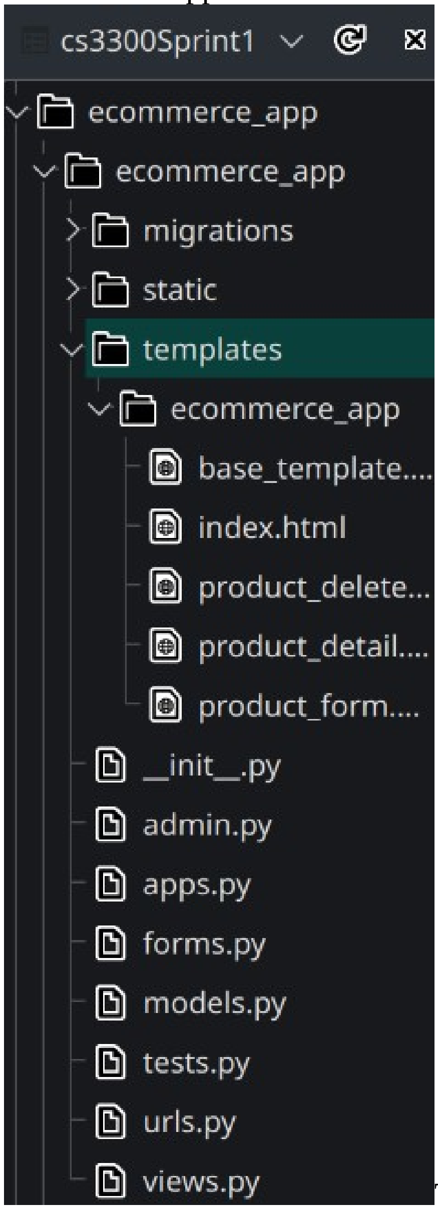
5. Start Django Project
```
python manage.py runserver
```

6. Now that your Project is up make your App
```
django-admin startproject my_app
```

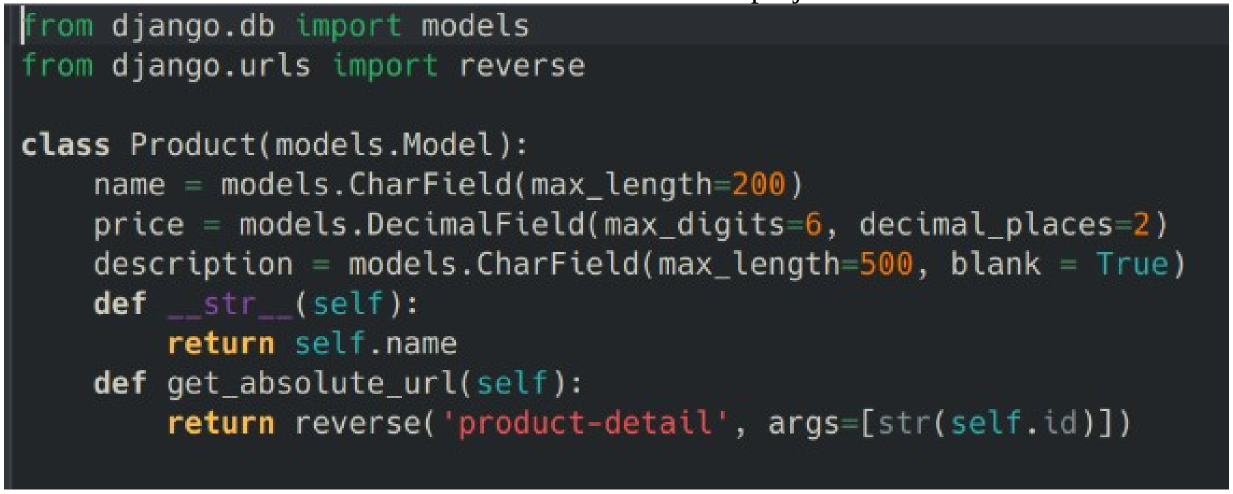
7. Your app will have Models, Views, Urls, Templates.

    Structure below

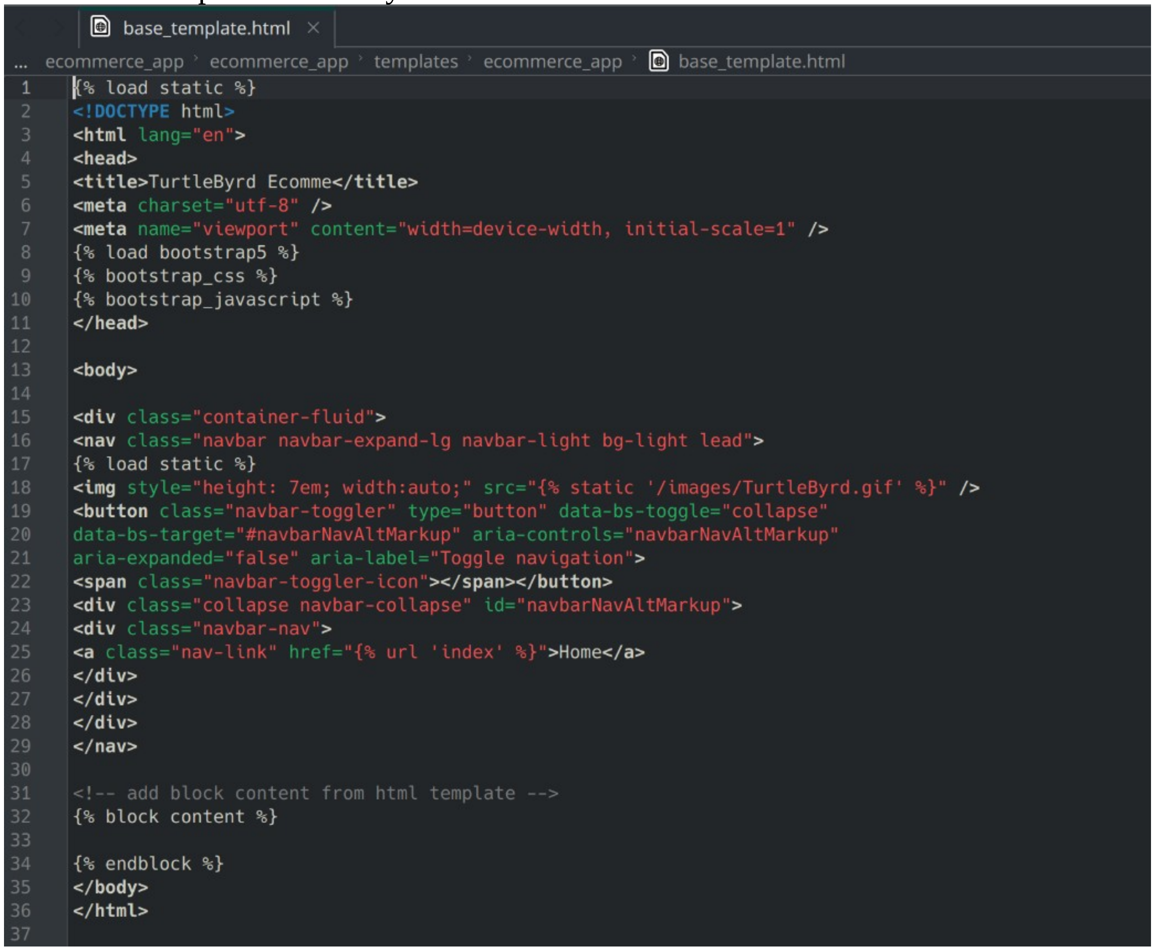
Structure of App Folder:


```
cs3300Sprint1
ecommerce_app
  ecommerce_app
    migrations
    static
    templates
      ecommerce_app
        base_template....
        index.html
        product_delete...
        product_detail....
        product_form....
    __init__.py
    admin.py
    apps.py
    forms.py
    models.py
    tests.py
    urls.py
    views.py
```

The models used for the project is shown below.


```python
from django.db import models
from django.urls import reverse

class Product(models.Model):
    name = models.CharField(max_length=200)
    price = models.DecimalField(max_digits=6, decimal_places=2)
    description = models.CharField(max_length=500, blank = True)
    def __str__(self):
        return self.name
    def get_absolute_url(self):
        return reverse('product-detail', args=[str(self.id)])
```

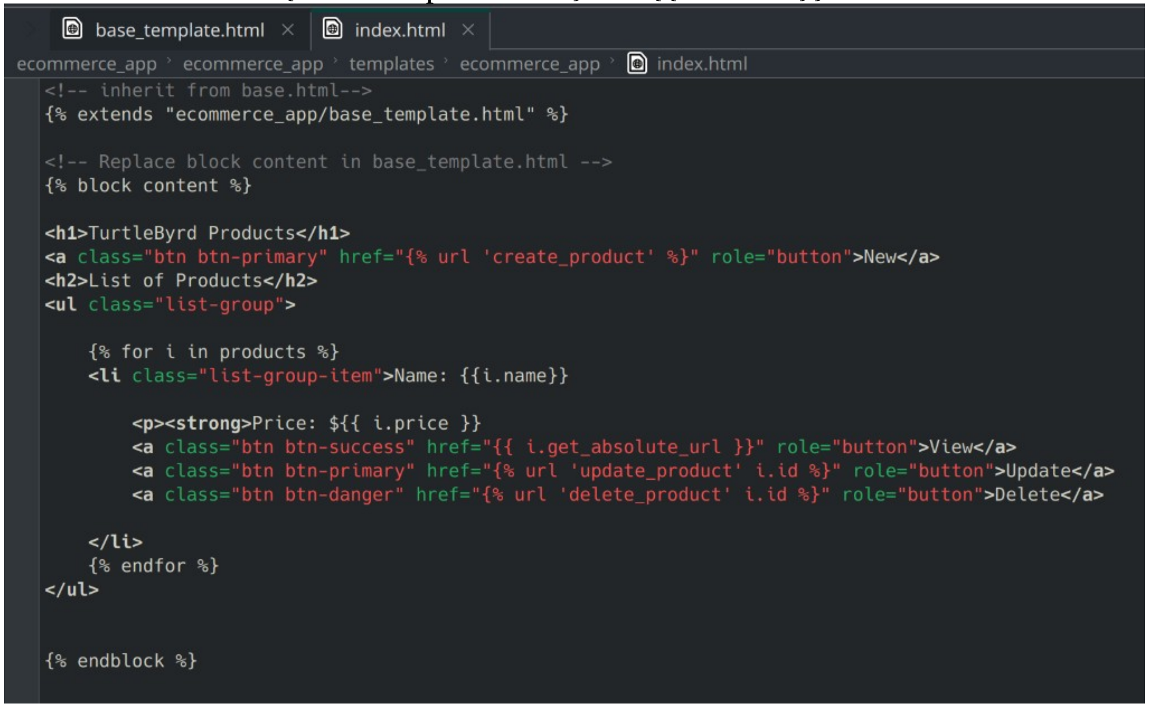
8. Make base template for website Home Page html
This has all the Bootstrap and HTML needed to structure the page and display it
It uses bootstrap classes for style and html divs for structure.



base_template.html
... ecommerce_app › ecommerce_app › templates › ecommerce_app › base_template.html

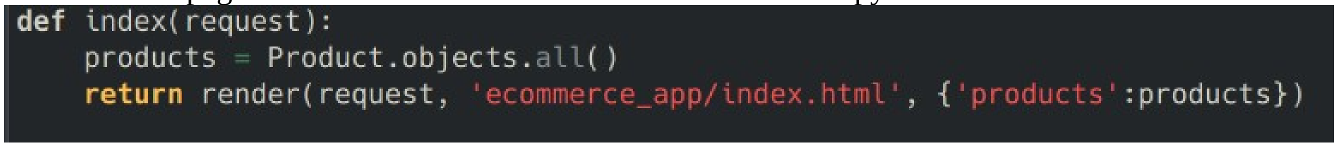
```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
<title>TurtleByrd Ecomme</title>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
{% load bootstrap5 %}
{% bootstrap_css %}
{% bootstrap_javascript %}
</head>

<body>

<div class="container-fluid">
<nav class="navbar navbar-expand-lg navbar-light bg-light lead">
{% load static %}
<img style="height: 7em; width:auto;" src="{% static '/images/TurtleByrd.gif' %}" />
<button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#navbarNavAltMarkup" aria-controls="navbarNavAltMarkup"
aria-expanded="false" aria-label="Toggle navigation">
<span class="navbar-toggler-icon"></span></button>
<div class="collapse navbar-collapse" id="navbarNavAltMarkup">
<div class="navbar-nav">
<a class="nav-link" href="{% url 'index' %}">Home</a>
</div>
</div>
</div>
</nav>

<!-- add block content from html template -->
{% block content %}

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

```

9. Index page for Home Page
# Notice this already includes functionality for the NEW, VIEW, UPDATE, and DELETE BUTTONS
# This will not be recovered but note that those buttons are there to go to different url's made later.
This uses liquid html to dynamically bring in data from the database for all the products in the products model database spreadsheet.
This is done with the {% for i in products %} and {{i.element}} html inserts.


base_template.html
index.html
ecommerce_app › ecommerce_app › templates › ecommerce_app › index.html

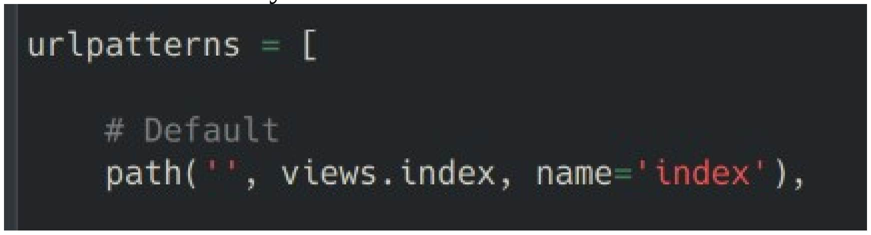
```
<!-- inherit from base.html-->
{% extends "ecommerce_app/base_template.html" %}

<!-- Replace block content in base_template.html -->
{% block content %}

<h1>TurtleByrd Products</h1>
<a class="btn btn-primary" href="{% url 'create_product' %}" role="button">New</a>
<h2>List of Products</h2>
<ul class="list-group">

    {% for i in products %}
    <li class="list-group-item">Name: {{i.name}}

        <p><strong>Price: ${{ i.price }}
        <a class="btn btn-success" href="{{ i.get_absolute_url }}" role="button">View</a>
        <a class="btn btn-primary" href="{% url 'update_product' i.id %}" role="button">Update</a>
        <a class="btn btn-danger" href="{% url 'delete_product' i.id %}" role="button">Delete</a>

    </li>
    {% endfor %}
</ul>


{% endblock %}
```

These HTML pages can be linked to via the Index View: In views.py



```
def index(request):
    products = Product.objects.all()
    return render(request, 'ecommerce_app/index.html', {'products':products})
```

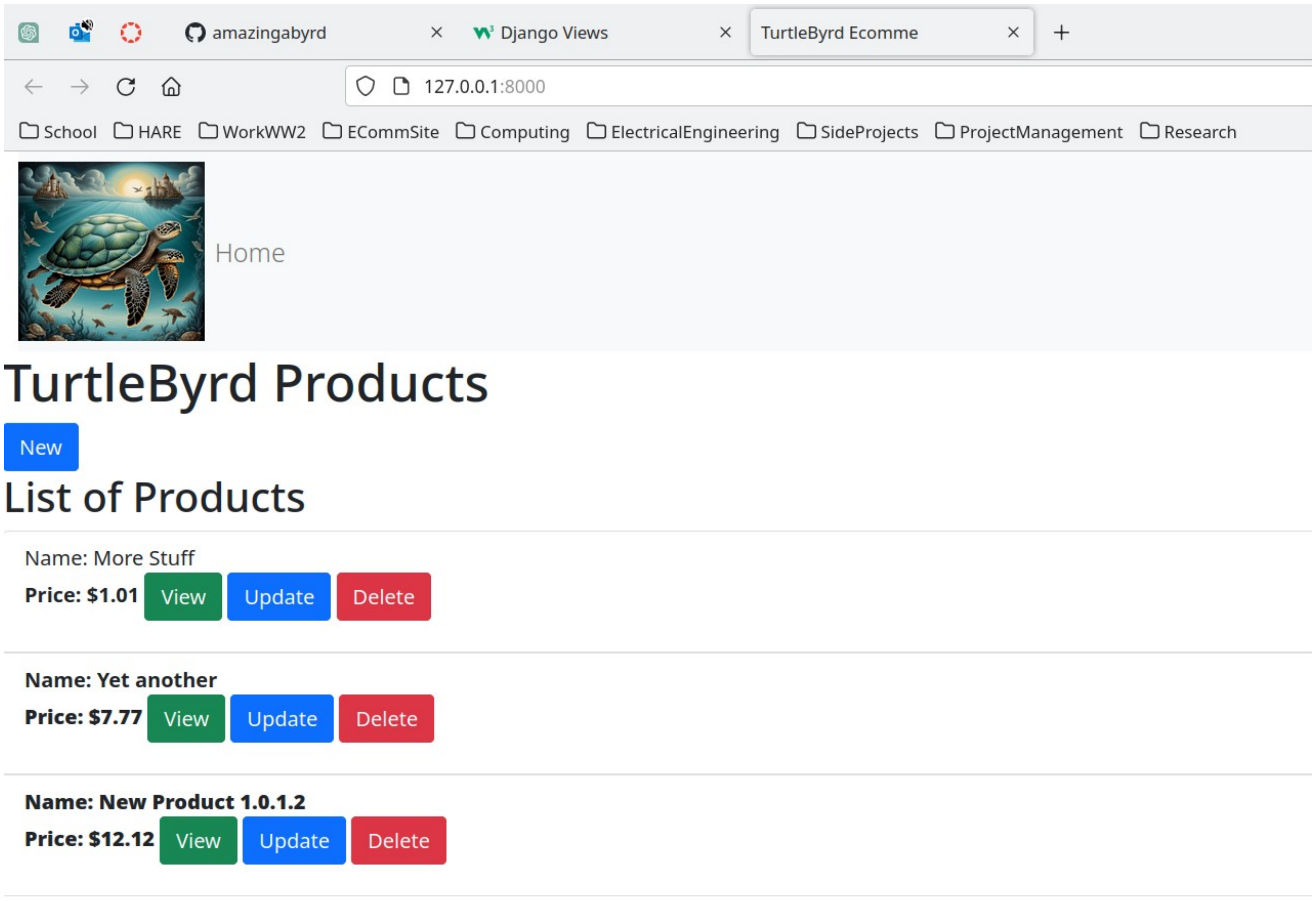
This view is called by the URL:



```
urlpatterns = [

    # Default
    path('', views.index, name='index'),
```

###
This URL when searched for from the browser can then go to the view and process the request to send the user the html required for the browser to render the site.
###

This allows the Home page to be displayed:


amazingabyrd | Django Views | TurtleByrd Ecomme

127.0.0.1:8000

School | HARE | WorkWW2 | ECommSite | Computing | ElectricalEngineering | SideProjects | ProjectManagement | Research

Home

TurtleByrd Products

New

List of Products

Name: More Stuff
Price: $1.01 View Update Delete

Name: Yet another
Price: $7.77 View Update Delete

Name: New Product 1.0.1.2
Price: $12.12 View Update Delete

Disregard the buttons for New, View, Update, and Delete

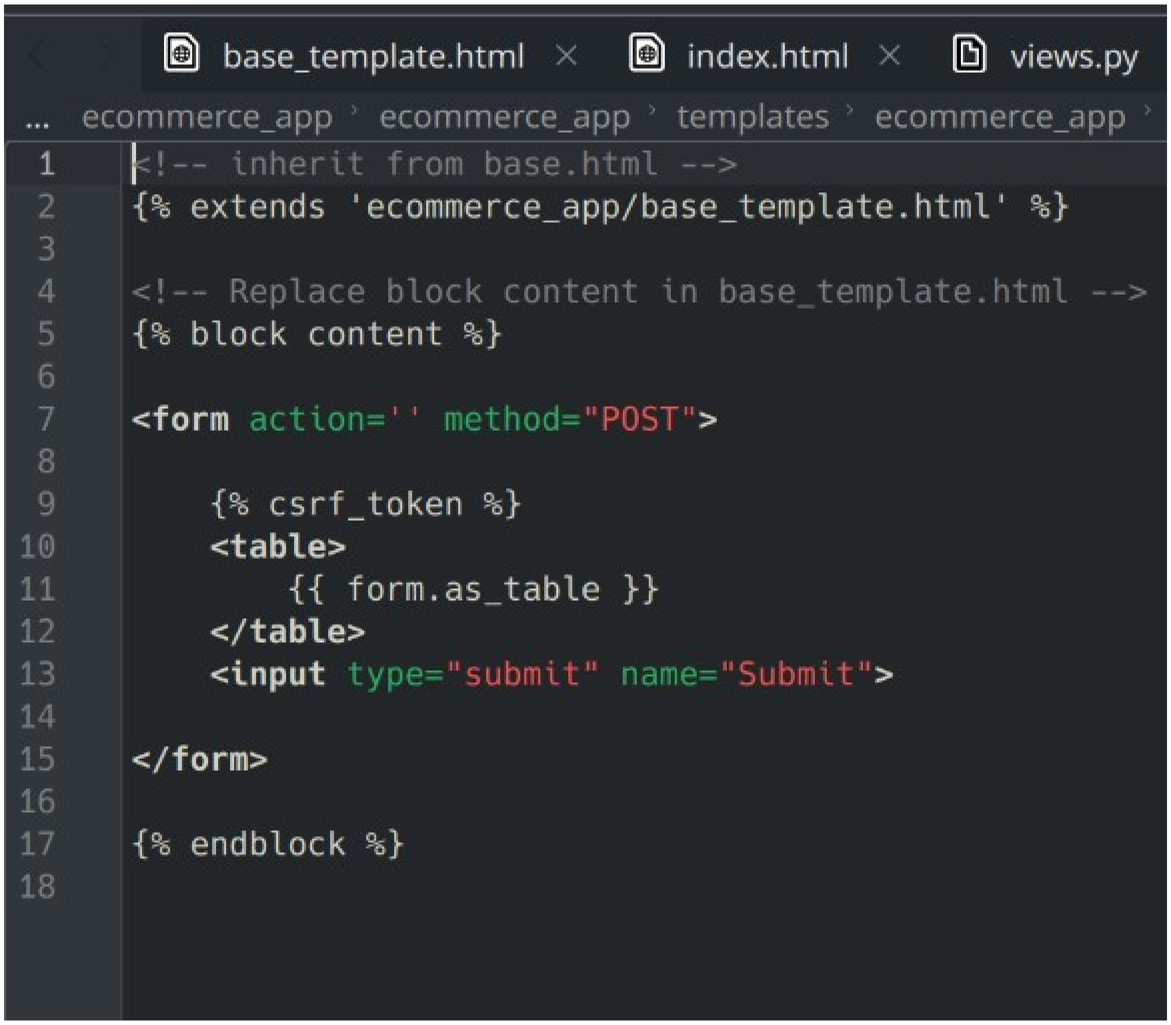
10. Get New Product Page/Form Setup
With a new URL

```
#Create Product
    path('product/create_product/', views.createProduct, name='create_product'),
```

View:

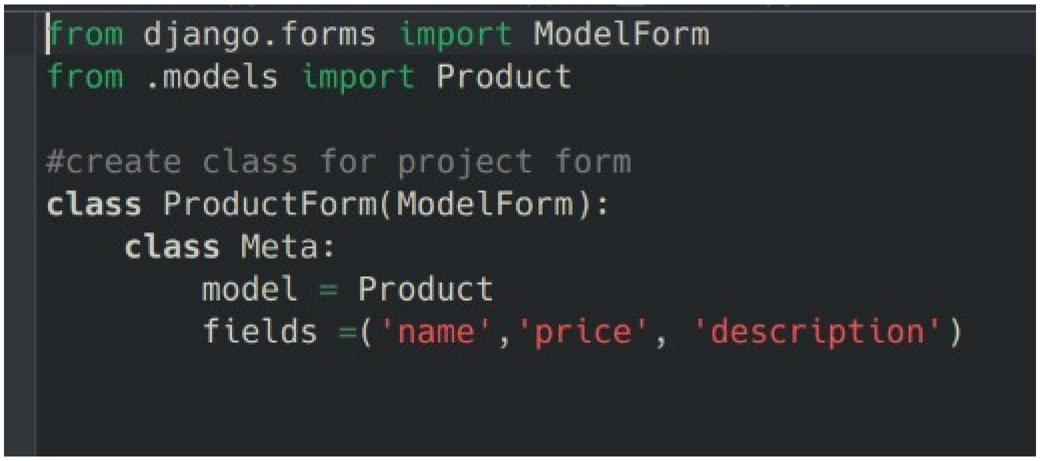
```
def createProduct(request):
    if request.method == 'POST':
        form = ProductForm(request.POST)
        if form.is_valid():
            # Save the form data to the database
            product = form.save()
            # Redirect back to the portfolio detail page or any other page you want
            return redirect('/')  # Change 'products' to the appropriate URL name
    else:
        form = ProductForm()

    context = {'form': form}
    return render(request, 'ecommerce_app/product_form.html', context)
```

HTML Template:


base_template.html
index.html
views.py
... ecommerce_app › ecommerce_app › templates › ecommerce_app ›

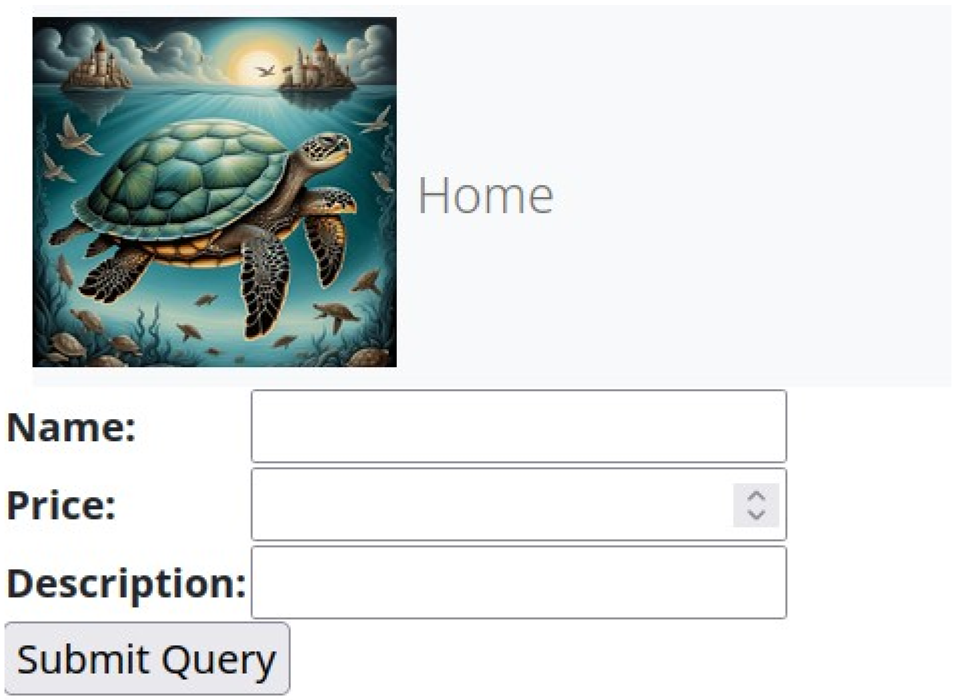
```
<!-- inherit from base.html -->
{% extends 'ecommerce_app/base_template.html' %}

<!-- Replace block content in base_template.html -->
{% block content %}

<form action='' method="POST">

    {% csrf_token %}
    <table>
        {{ form.as_table }}
    </table>
    <input type="submit" name="Submit">

</form>

{% endblock %}
```

and Form:



```
from django.forms import ModelForm
from .models import Product

#create class for project form
class ProductForm(ModelForm):
    class Meta:
        model = Product
        fields =('name','price', 'description')
```

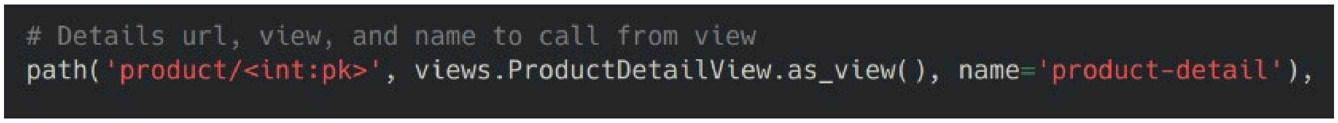
We have our create New Product Page



Thus making the Create Product functionality of the website.

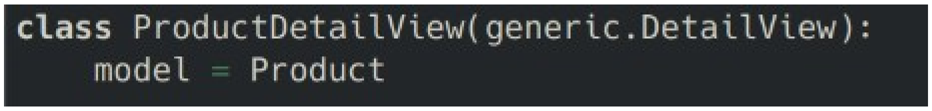
11. Add Detail view for each product.

With a URL



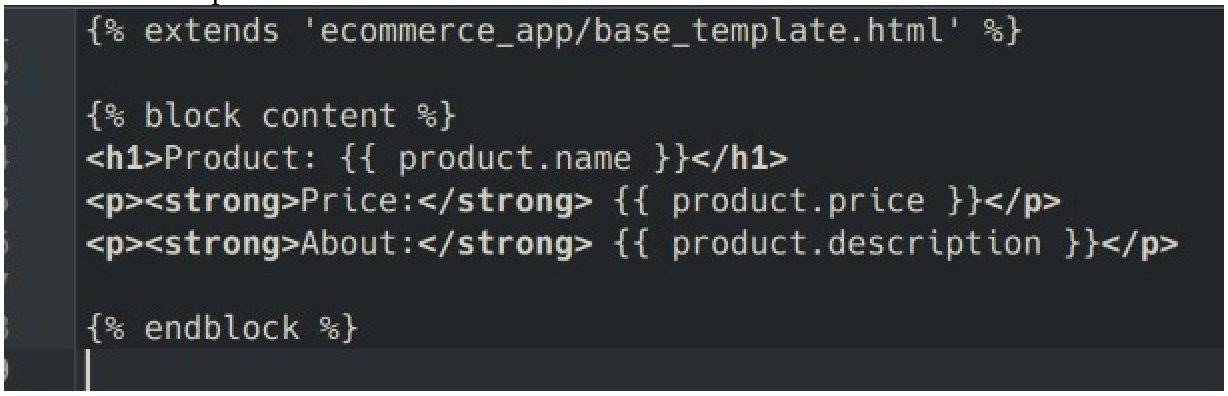
```
# Details url, view, and name to call from view
path('product/<int:pk>', views.ProductDetailView.as_view(), name='product-detail'),
```

View



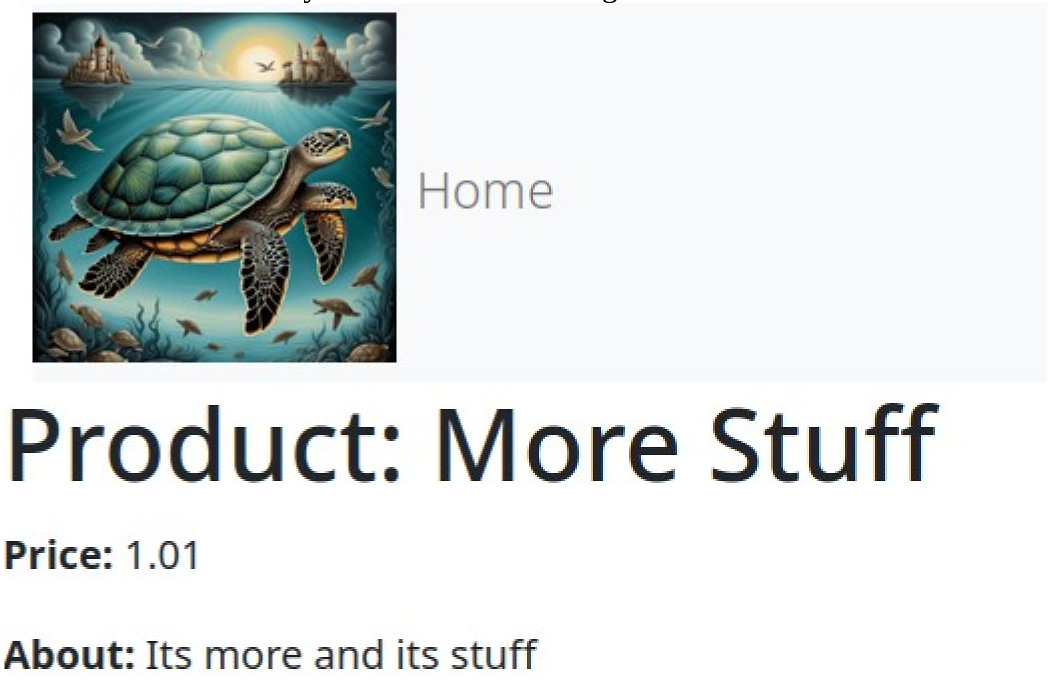
```
class ProductDetailView(generic.DetailView):
    model = Product
```

and HTML Template



```
{% extends 'ecommerce_app/base_template.html' %}

{% block content %}
<h1>Product: {{ product.name }}</h1>
<p><strong>Price:</strong> {{ product.price }}</p>
<p><strong>About:</strong> {{ product.description }}</p>

{% endblock %}
```

We add the functionality of the Detail View Page



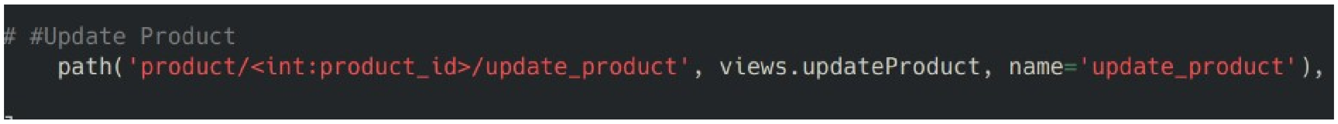
Home

Product: More Stuff

Price: 1.01

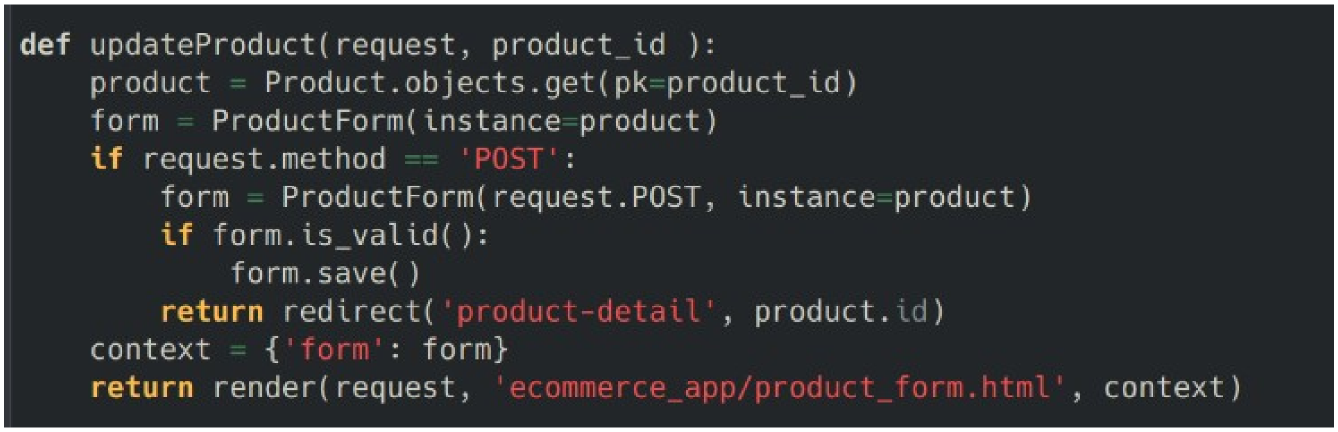
About: Its more and its stuff

12. Add Update to each
With a URL



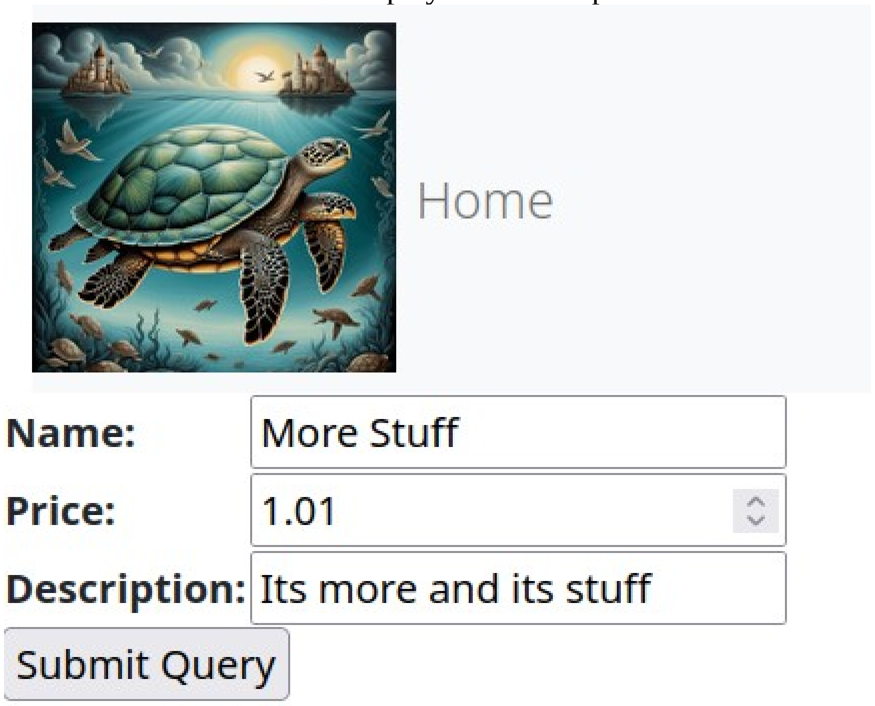
```
# #Update Product
    path('product/<int:product_id>/update_product', views.updateProduct, name='update_product'),
```

View



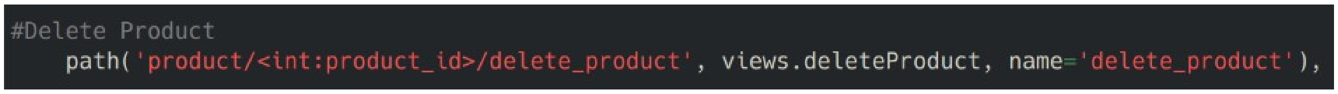
```
def updateProduct(request, product_id ):
    product = Product.objects.get(pk=product_id)
    form = ProductForm(instance=product)
    if request.method == 'POST':
        form = ProductForm(request.POST, instance=product)
        if form.is_valid():
            form.save()
        return redirect('product-detail', product.id)
    context = {'form': form}
    return render(request, 'ecommerce_app/product_form.html', context)
```

and Reusing the HTML Form Template
This allows the form to be displayed with the products information already filled in the form



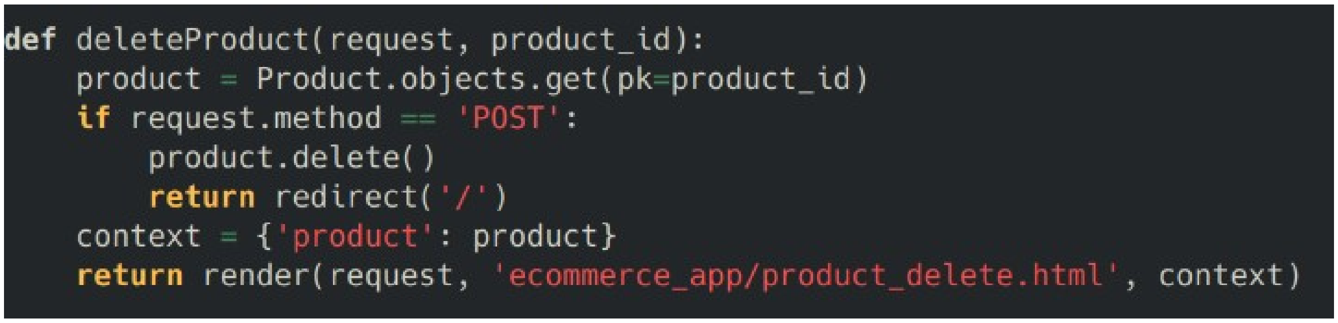
Name:	More Stuff
Price:	1.01
Description:	Its more and its stuff

13. Add Delete Option for each Product.
With a new URL for confirmation of deletion



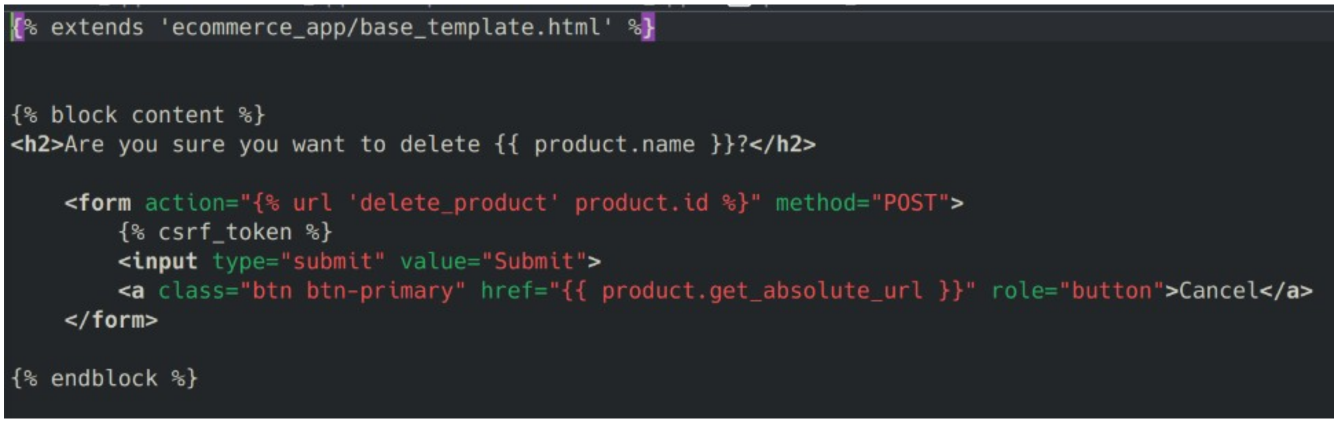
```
#Delete Product
    path('product/<int:product_id>/delete_product', views.deleteProduct, name='delete_product'),
```

View



```
def deleteProduct(request, product_id):
    product = Product.objects.get(pk=product_id)
    if request.method == 'POST':
        product.delete()
        return redirect('/')
    context = {'product': product}
    return render(request, 'ecommerce_app/product_delete.html', context)
```

HTML



```
{% extends 'ecommerce_app/base_template.html' %}


{% block content %}
<h2>Are you sure you want to delete {{ product.name }}?</h2>

    <form action="{% url 'delete_product' product.id %}" method="POST">
        {% csrf_token %}
        <input type="submit" value="Submit">
        <a class="btn btn-primary" href="{{ product.get_absolute_url }}" role="button">Cancel</a>
    </form>

{% endblock %}
```

this allows you to delete products from the listings.

The picture on the homepage is kept in the static/images folder and it holds the .gif of the Picture used at the top left of the site. This is put into the html with a static command. Allowing the asset to be brought into the site.