Comands for the terminal to start our django web server:

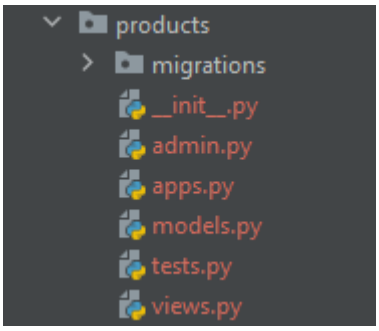
After install django (pip install django==(Version number))

The first command create a folder with all the .py files to get the app organized
The second command start the server

```
Terminal:  Local  +
(venv) PyShop $django-admin startproject pyshop
.
(venv) PyShop $python manage.py runserver
```

Create a new module in our project:

```
C:\Users\34634\Desktop\Python\Python tutorial\ecommerce>python manage.py startapp products
```



admin.py: How the administration panel of the app has to look like
apps.py: To store the configurations settings for the app
models.py: Where we define classes and new types for modeling the concepts in this app. (Ex: Products may have- product, category, review,..)

tests.py: Where the automatized tests should be written
views.py: What the user see when the user navigate to a certain page.

Starting our ecommerce logic:

Set the URL to a view:

1.) Create the functions (views) products/views.py file:

```
views.py U ×
products > views.py > ...
from django.http import HttpResponse
from django.shortcuts import render

# We have to map localhost.../product to index
# We are going to set it in the
# products/urls.py file


# Web page views by URL
def index(request):
    return HttpResponse('Hello word')

#That would be our localhost.../product/new
def product_new(request):
    return HttpResponse('Hello new word')
```

2.) We crate a .py file called urls.py and map each ‘URL’ to a view function:

```
urls.py M ×
products > urls.py > ...
from django.urls import path
# . = in the current folder
from . import views

# Array to set each 'URL' (/new,/sales,/other_view,..)
# to a function of our views file
urlpatterns = [
    path('', views.index),
    path('new', views.product_new)

]
```

3.) We map the, in this case, products/ path to the products.urls that we created before that way Django is going to know where find what it has to show. In autogenerated ecomerce/urls.py file:

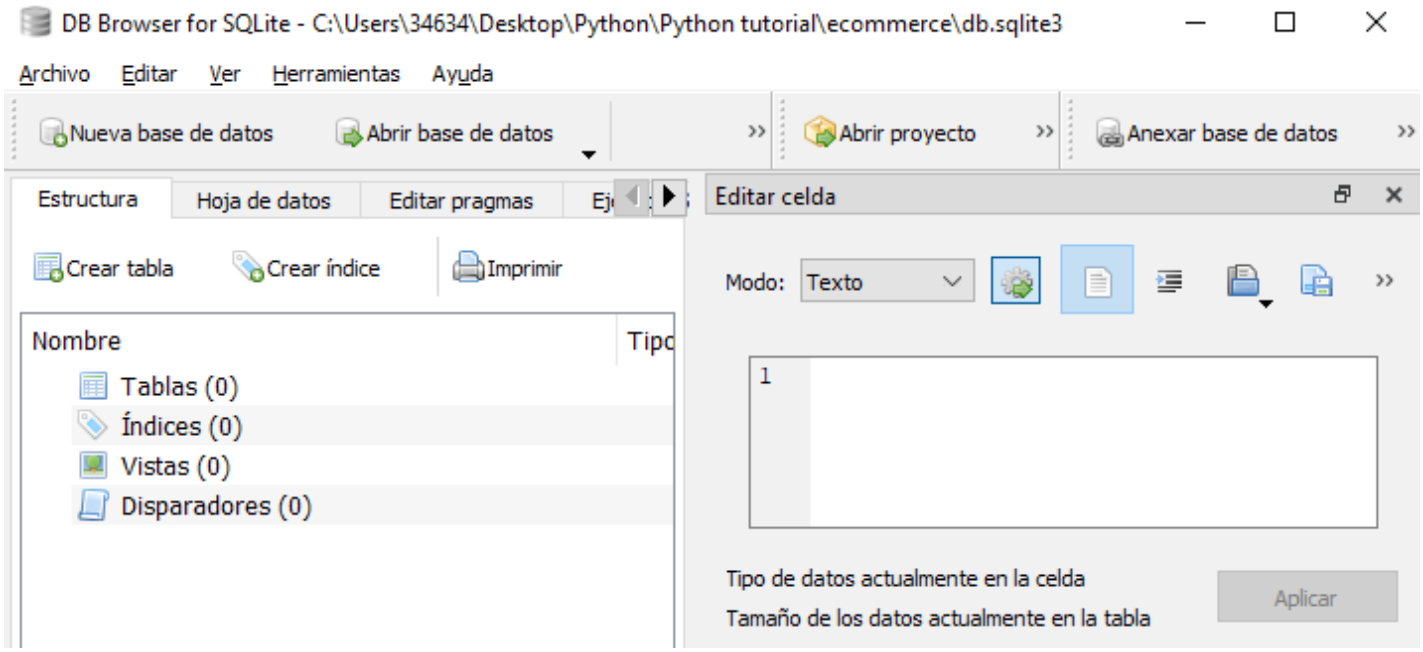
```
urls.py U ×
ecommerce > urls.py > ...
"""ecommerce URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/3.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    # That way we 'tell' django that any path that
    # start with products/ delegated to the product app
    path('products/',include('products.urls'))
]
```

Creating some representations of a real world concept (models):

```
models.py U ×
products > models.py > ...
from django.db.models.fields import CharField

class Product(models.Model):
    name= models.CharField(max_length=255) # Field that contain text data
    price = models.FloatField()
    stock = models.IntegerField()
    image_url = models.CharField(max_length=2038)
```

Create a data base for the storing products using our model:

This step is using sqlite which is fine for learning and smalls app.
We have to download https://sqlitebrowser.org/ to be able to visualize our data base.
We just have to drop our db.sqlite3 file in the DB Browser sqlite app.



Now how to make Django created our model:

First we have to tell django about our app and in order to get that:

We have a list called INSTALLED_APPS in the ecommerce.settings.py file where we could see all the apps installed by default and we have a class called ProductsConfif where we store the configuration of our app so we have to get in the list the path of out class like so:

```
ecommerce
  __pycache__
  __init__.py        U
  asgi.py            U
  settings.py        U
  urls.py            U
  wsgi.py            U
products
  __pycache__
  migrations
```

```python
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'products.apps.ProductsConfig'
]
```

And now we can, trough the terminal, order to django that we want to create a table using our model the commands are, python manage.py makemigrations to create a file with a class in it which is in charge of some a operation to create out table, as we can see:

```
PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

PS C:\Users\34634\Desktop\Python\Python tutorial\ecommerce> python manage.py makem
igrations
Migrations for 'products':
  products\migrations\0001_initial.py
    - Create model Product
```

The id field is created automatically by Django.

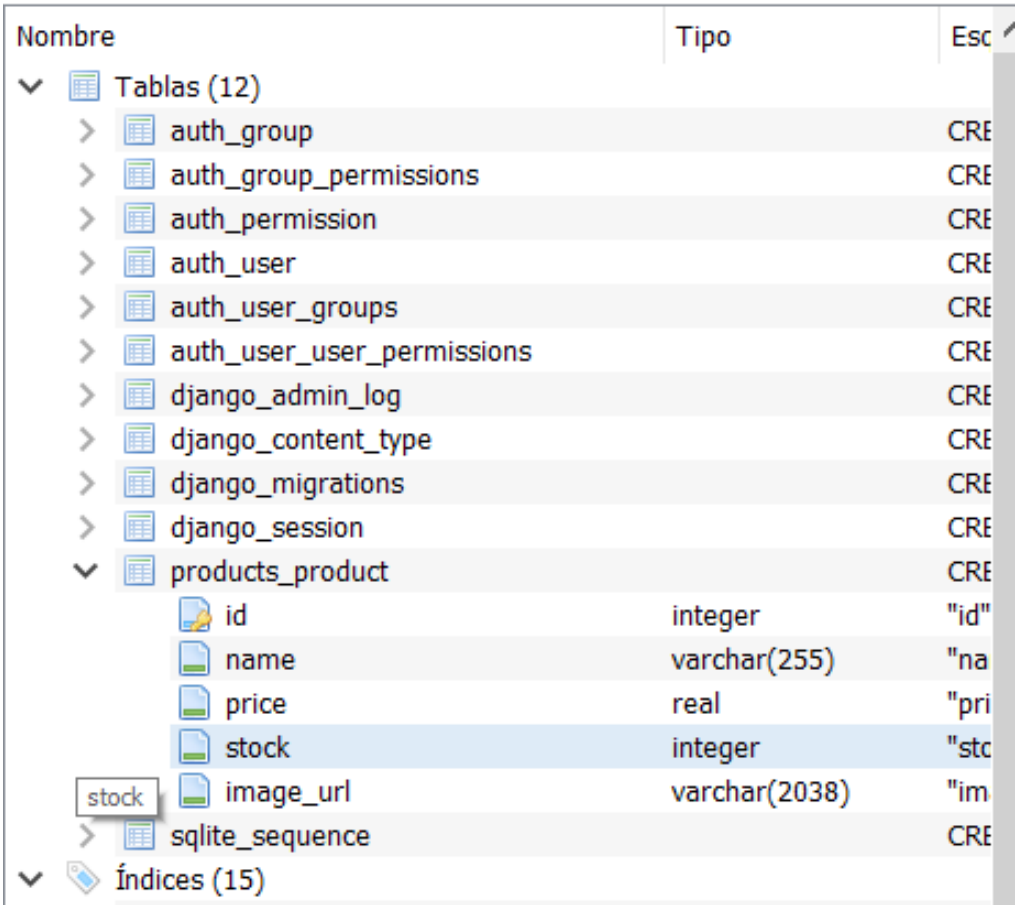
products > migrations > 0001_initial.py > Migration

```python
# Generated by Django 3.0 on 2021-07-24 18:41

from django.db import migrations, models


class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='Product',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID'))
                ('name', models.CharField(max_length=255)),
                ('price', models.FloatField()),
                ('stock', models.IntegerField()),
                ('image_url', models.CharField(max_length=2038)),
            ],
        ),
    ]
```

And now to create the table we use the command python manage.py migrate:

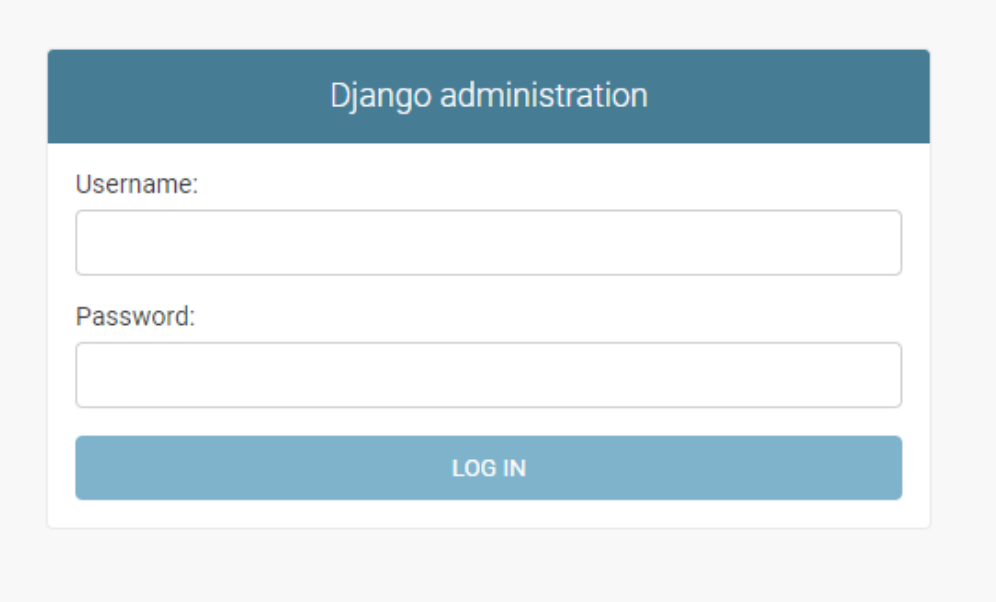
```
Applying products.0001_initial... OK
Applying sessions.0001_initial... OK
```

Now if we drop the data base again in the DB browser for sqlite we see:



Work with the administration panel generated by Django:

If we type http://localhost:(your Django port)/admin we are going to see a logging panel generated for Django:



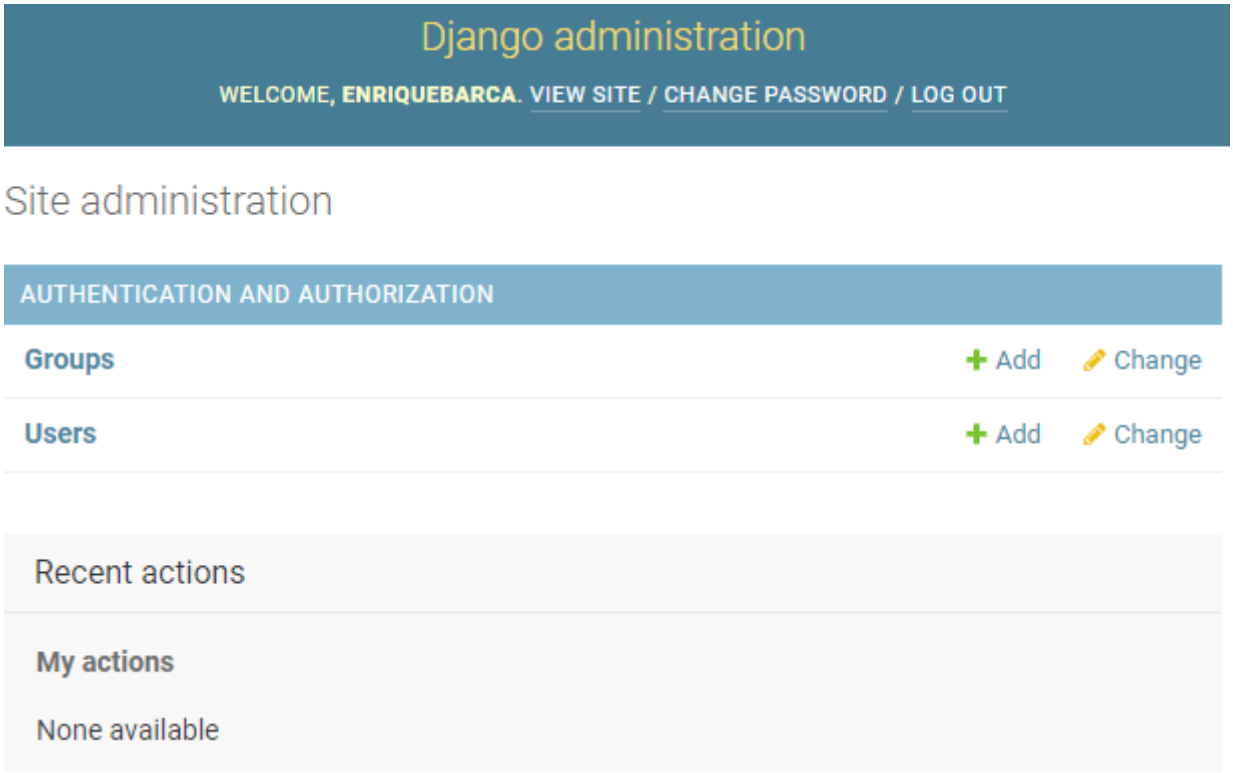
To sign in we have to created a super user using the terminal, like so:

command: python manage.py createsuperuser

and follow the instructions as we see in the picture below:

```
PS C:\Users\34634\Desktop\Python\Python tutorial\ecommerce> python manage.py createsuperuser
Username (leave blank to use '34634'): EnriqueBarca
Email address: fakeEmail7@gmail.com
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is too common.
This password is entirely numeric.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
```

Now we can log in and see the Django superuser default panel, as we see below:

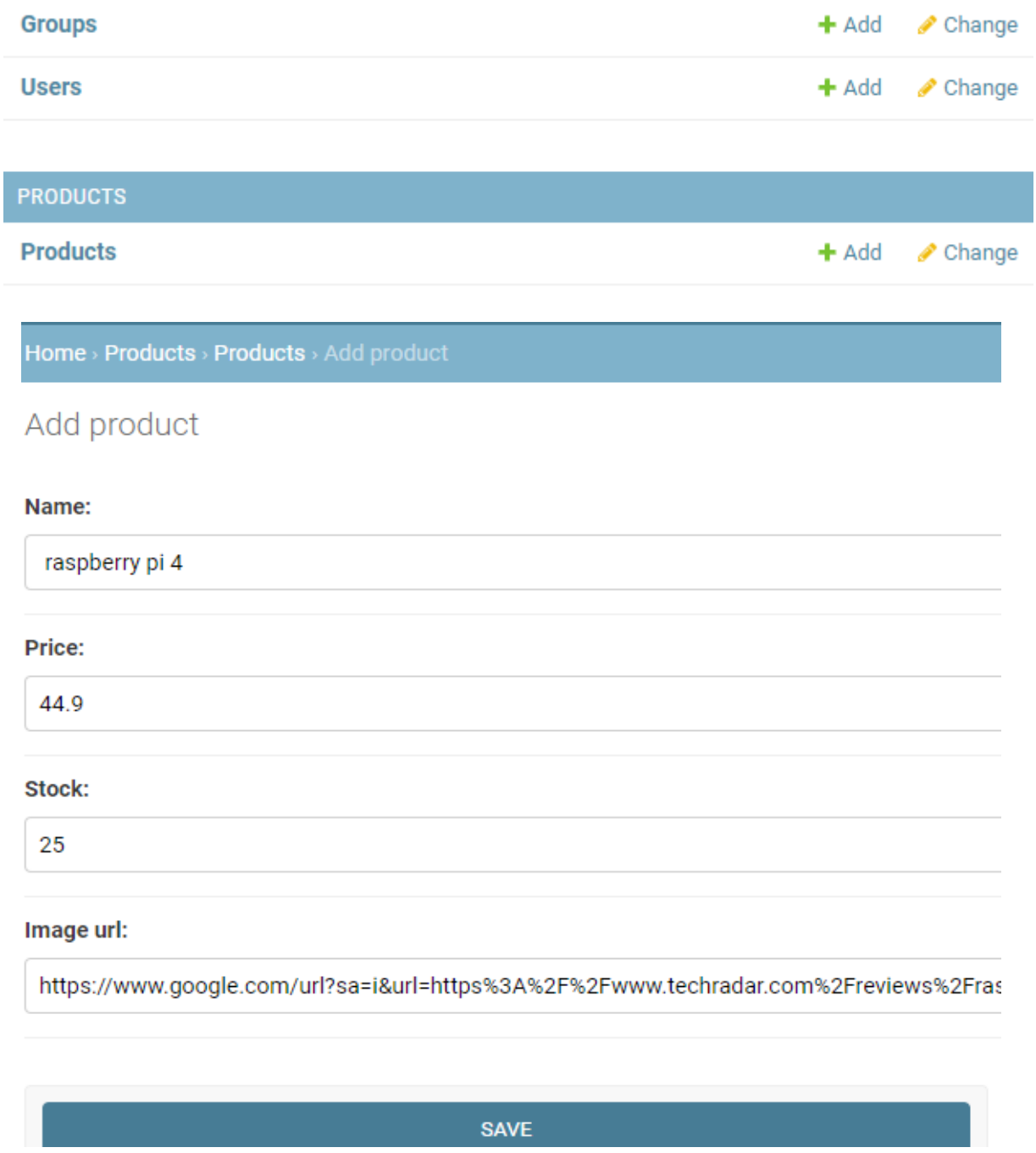


Now we are going to modify the panel to add our products to be able to administrate them from the panel.
We have to open the admin.py file and code the next lines:

products > admin.py

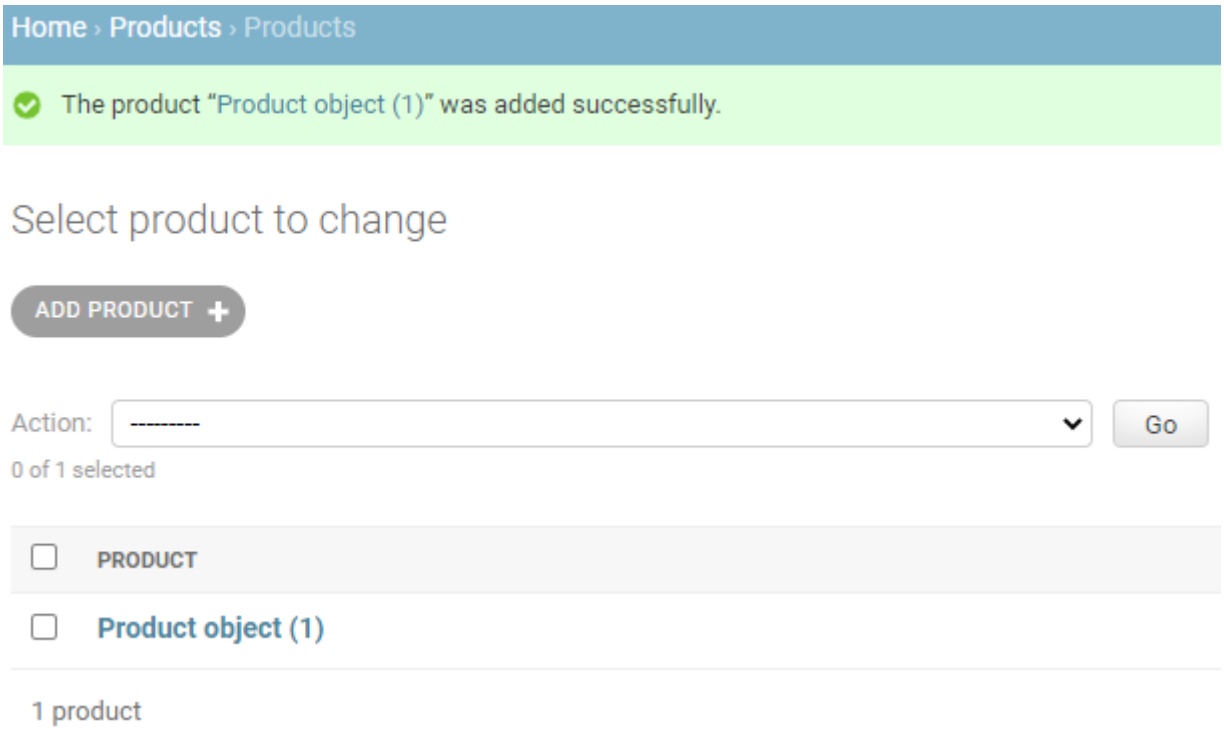
```python
from django.contrib import admin
from .models import Product

# We pass our Product class as an argument to
# tell Django that we want to administrate
# from the administration panel

admin.site.register(Product)
```

And if we go back to the web we are going to see how we have the new element to administrate from there:



Customizing the admin panel.

As we see in the image below Django shows as not in a very clear way what product we have but we can change:



In the same file that we modify before admin.py we can create a way to see ours products, like so:

products > admin.py > ProductAdmin

```python
from django.contrib import admin
from .models import Product

#That way we set a table view of our products in this case
class ProductAdmin(admin.ModelAdmin):

    # list_display it's override from ModelAdmin
    # Names ofeach column that we want to show
    list_display=('name','price','stock')

# We pass our Product class as an argument to
# tell Django that we want to administrate
# from the administration panel

admin.site.register(Product,ProductAdmin)
```

And now our management display looks like so:

Select product to change

Action: -------- Go 0 of 2 selected

	NAME	PRICE	STOCK
☐	Mouse	17.99	6
☐	raspberry pi 4	44.9	25

2 products

Show our data base registers in our ecommerce web.

1.) In our views.py we have to set in the functions the data of our data base.:

1.a) Import the products class, from .models import Products.
1.b) Using the methods we ask for the data that we want to show.
Ex: in the view function we can use:

Product.objects.all() → To show all the elements in the DB
Product.objects.filter() → to look for a certain product
Product.objects.get() → for a single product
Product.objects.save() → to add or update one

In this case we are going to use the all() method.

products > views.py > ...

```python
from django.http import HttpResponse
from django.shortcuts import render
from .models import Product

# We have to map localhost.../product to index
# We are going to set it in the
# products/urls.py file

# Web page views by URL
def index(request):
    products = Product.objects.all()
    return HttpResponse('Hello word')
```

2.) We have to create a html template:

2.a) Create a folder inside our products folder called templates, it's important use that name because is where Django is going to look at .
2.b) Create the template for the products.index, inside the template folder we have to create a file called index.html.

products > templates > index.html

```html
<h1>Products</h1>
<ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>

</ul>
```

3.) We set the created index.html file to show it to the client.

3.a) In our index function we have to use the render obj and pass it 3 parameters, request, which is the object that the function has as default as parameter, the template that we just created and a dictionary with the list of products that we took from the data base to show it in the template (Second argument). We have to something like:

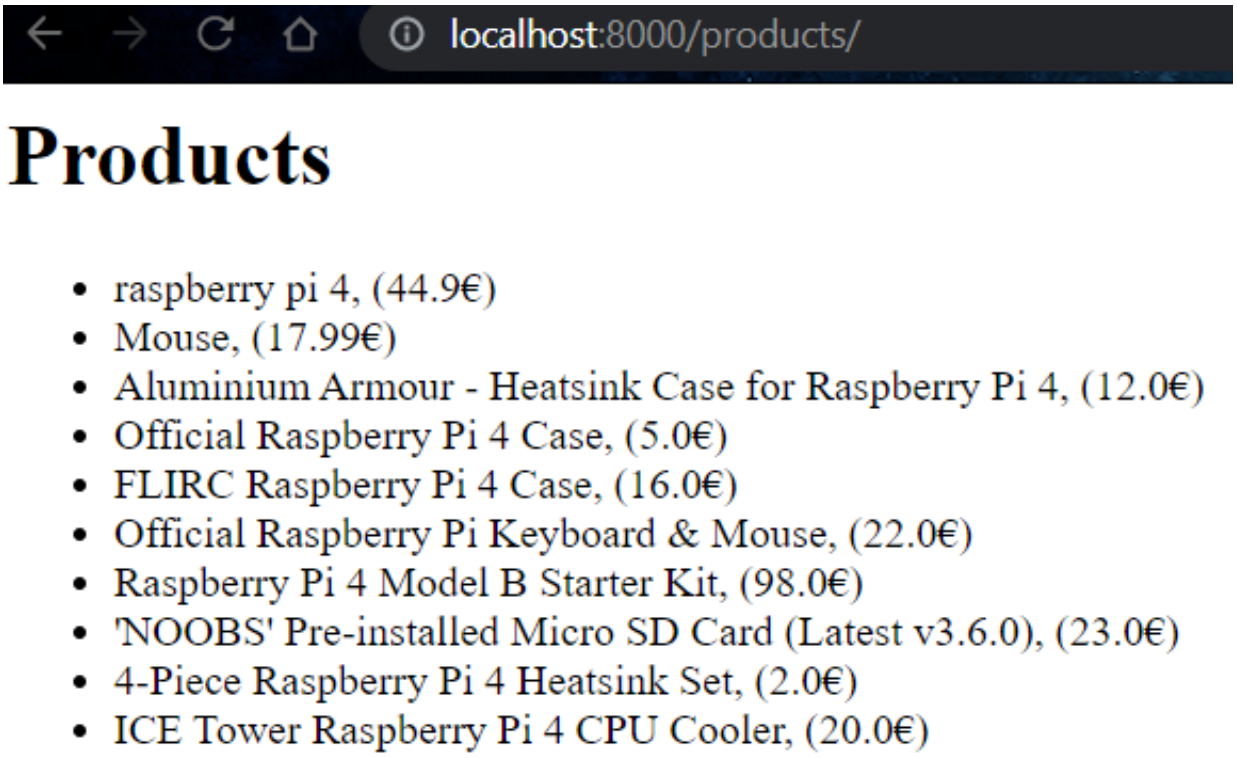
```python
# Web page views by URL
def index(request):
    products = Product.objects.all()
    return render(request, 'index.html',{'products': products})
```

3.b) We have to replace the Item tags with our products using a template tag → {% %} we use them to pass dynamic code, we would end with the template like so:

products > templates > index.html

```html
<h1>Products</h1>
<ul>
    <!-- 'products' is the context got it from the view.index -->
    {% for product in products %}
        <!-- Django will evaluate the expressions in double curly braces
        and render in the html-->
        <li>{{ product.name }}, ({{product.price}}€)</li>
    {% endfor %}

</ul>
```

Which have the next result:



Get better look at our web using bootstrap framework.

1.) We have to get the basic template in https://getbootstrap.com/docs/5.0/getting-started/introduction/ and going down to “Starter template” , copy the html code and pasted in a new file inside the templates folder of our project.

2.) We create a block content where is going to be our DB data, which we are going to get it from the index.html created before.

products > templates > base.html

```html
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet" integr

    <title>Hello, world!</title>
  </head>
  <body>
    <!-- Creating a block where is going to be the data -->
    {% block content %}

    {% endblock %}


    <!-- Option 1: Bootstrap Bundle with Popper -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js" integrity="sha384

    <!-- Option 2: Separate Popper and Bootstrap JS -->
    <!--
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js" integrity="sha384-IQs
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.min.js" integrity="sha384-cVKIPh
    -->
  </body>
</html>
```

3.) We have to create the communication between our block gap and the information that content the index.html file, like so:

products > templates > index.html

```html
{% extends 'base.html' %} <!-- extend the base.html template -->

{% block content %} <!-- Fill the hole (block) that we defined in the base.html file -->

    <h1>Products</h1>
    <ul>
        <!-- 'products' is the context got it from the view.index -->
        {% for product in products %}
            <!-- Django will evaluate the expressions in double curly braces
            and render in the html-->
            <li>{{ product.name }}, ({{product.price}}€)</li>
        {% endfor %}
    </ul>

{% endblock %}
```

4.) now we have the data let's add some style components to make the web look better.
We have to go to https://getbootstrap.com/docs/5.0/components/card/ and copy the code that is in the example section. We have to create a and inside a and inside them the code that we copy before. We should have something as it show below:

products > templates > index.html

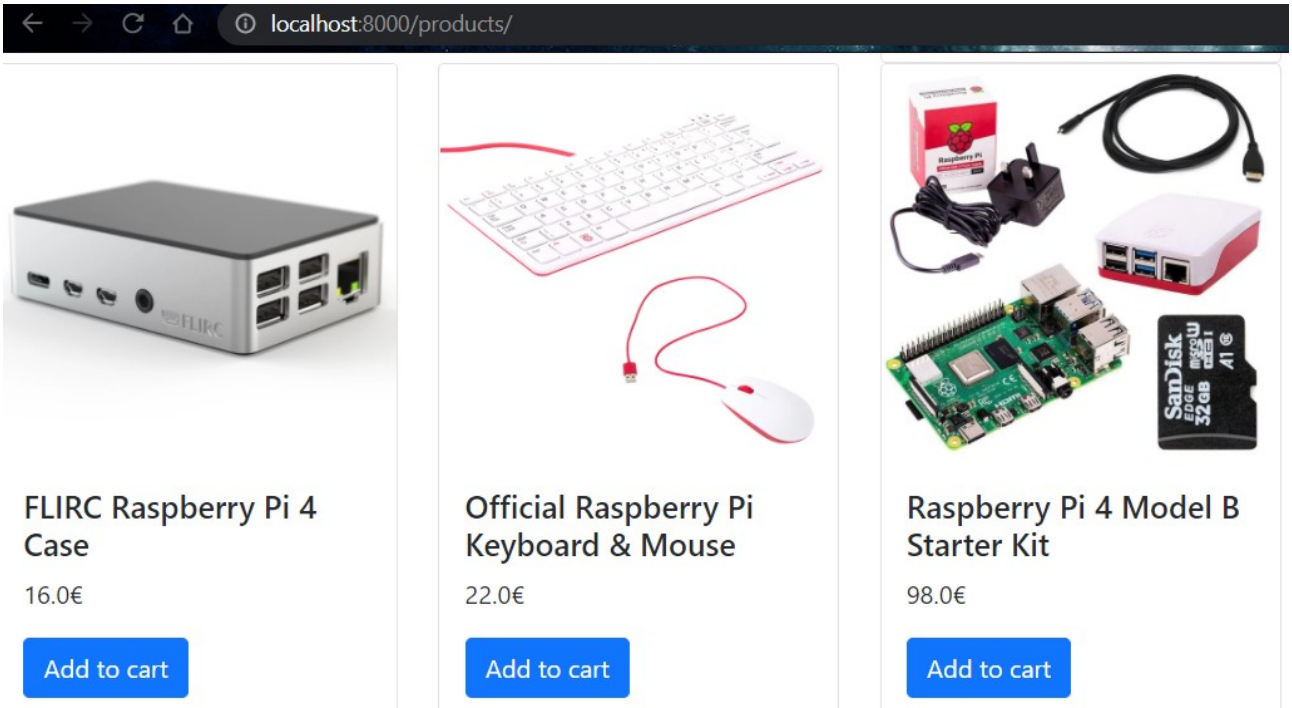
```html
{% extends 'base.html' %}
<!-- extend the base.html template -->

{% block content %}
<!-- Fill the hole (block) that we defined in the base.html file -->

    <h1>Products</h1>
    <div class="row">
        <div class="col">
            <div class="card" style="width: 18rem">
            <img src="..." class="card-img-top" alt="..." />
                <div class="card-body">
                    <h5 class="card-title">Card title</h5>
                    <p class="card-text">
                    Some quick example text to build on the card title and make up the
                    bulk of the card's content.
                    </p>
                    <a href="#" class="btn btn-primary">Go somewhere</a>
                </div>
            </div>
        </div>
    </div>
    <ul>
    <!-- 'products' is the context got it from the view.index -->
    {% for product in products %}
    <!-- Django will evaluate the expressions in double curly braces
                and render in the html-->
    <li>{{ product.name }}, ({{product.price}}€)</li>
    {% endfor %}
    </ul>

{% endblock %}
```

5.) Now we have to use the for loop already created and put it below our row div, that way we are going to render a column for each product. And in the different sections of the code paste we have to set the different values of our “Product”, we can delete the <ul> block created before.
The result should be something like so.

products > templates > index.html

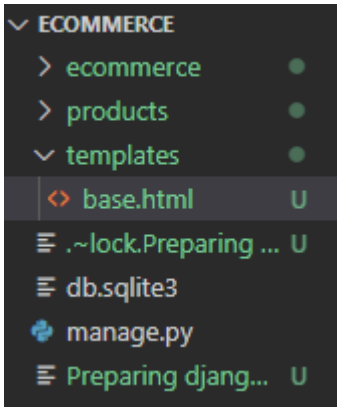
```
{% extends 'base.html' %}
<!-- extend the base.html template -->

{% block content %}
<!-- Fill the hole (block) that we defined in the base.html file -->

    <h1>Products</h1>
    <div class="row">
        {% for product in products %}
        <div class="col">
            <div class="card" style="width: 16rem">
            <img src="{{ product.image_url }}" class="card-img-top" alt="..." />
                <div class="card-body">
                    <h5 class="product-name">{{product.name}}</h5>
                    <p class="product_price">{{product.price}}€</p>
                    <a href="#" class="btn btn-primary">Add to cart</a>
                </div>
            </div>
        </div>
        {% endfor %}
    </div>

{% endblock %}
```

And we should end with a result like shown below:



Creating a problem to learn how to fix it.

We create before the folder template inside the products folder but our base.html file is going to be use in most of the web window so we have to take it out to be accessible from others folders so we have to create a template folder and put it somewhere central, like so:



Now in our web, we are going to see the next error:

TemplateDoesNotExist at /products/

base.html

Request Method:	GET
Request URL:	http://localhost:8000/products/
Django Version:	3.0
Exception Type:	TemplateDoesNotExist
Exception Value:	`base.html`
Exception Location:	C:\Users\34634\anaconda3\lib\site-packages\django\template\backends\django.py in reraise, line 84
Python Executable:	C:\Users\34634\anaconda3\python.exe
Python Version:	3.8.8
Python Path:	`['C:\\Users\\34634\\Desktop\\Python\\Python tutorial\\ecommerce', 'C:\\Users\\34634\\anaconda3\\python38.zip', 'C:\\Users\\34634\\anaconda3\\DLLs', 'C:\\Users\\34634\\anaconda3\\lib', 'C:\\Users\\34634\\anaconda3', 'C:\\Users\\34634\\anaconda3\\lib\\site-packages', 'C:\\Users\\34634\\anaconda3\\lib\\site-packages\\locket-0.2.1-py3.8.egg', 'C:\\Users\\34634\\anaconda3\\lib\\site-packages\\win32', 'C:\\Users\\34634\\anaconda3\\lib\\site-packages\\win32\\lib', 'C:\\Users\\34634\\anaconda3\\lib\\site-packages\\Pythonwin']`
Server time:	Sun, 25 Jul 2021 11:37:51 +0000

To fix it we have to tell Django that it have to see the new template folder in addition to install the app. In the setting.py file we have a BASE_DIR is set to the complete path where is storage this project so we have to add the template folder to it.

We have to look for TEMPLATES in setting.py and in the DIRS key we have to reference to the template folder like so:

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [

            os.path.join(BASE_DIR,'templates')
        ],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Now we should see our web as before.

Improving the look and feel of the web page.

Add a nav bar component.

1.) We have to go to https://getbootstrap.com/docs/5.0/components/navbar/ scroll down to brand and copy the code for this nav bar.

2.) We have to paste it in our base template because this template is gong to be in all our web windows. So we just have to paste it below the body tag, like so.

```
templates > <> base.html
    <body>
      <!-- As a link -->
      <nav class="navbar navbar-light bg-light">
        <div class="container-fluid">
          <a class="navbar-brand" href="#">Navbar</a>
        </div>
      </nav>

      <!-- As a heading -->
      <nav class="navbar navbar-light bg-light">
        <div class="container-fluid">
          <span class="navbar-brand mb-0 h1">Ecommerce</span>
        </div>
      </nav>
```

Add some padding with the help of bootstrap tags:

We are going to use the class container which has some padding by default, so the code would end looking like so:

```
templates > <> base.html
15
16      <title>Pyton ecommerce app!</title>
17    </head>
18    <body>
19      <!-- As a link -->
20      <nav class="navbar navbar-light bg-light">
21        <div class="container-fluid">
22          <a class="navbar-brand" href="#">Navbar</a>
23        </div>
24      </nav>
25
26      <!-- As a heading -->
27      <nav class="navbar navbar-light bg-light">
28        <div class="container-fluid">
29          <span class="navbar-brand mb-0 h1">Ecommerce</span>
30        </div>
31      </nav>
32      <div class="container">
33
34        {% block content %}
35        {% endblock %}
36
37      </div>
```

And our app like so: