

## Mini Market Paul Django Project

### 1.- Start the project

```
C:\Users\Paul Manriquez\Desktop\Django\MiniMarketPaul_Django>django-admin startproject MiniMarket

C:\Users\Paul Manriquez\Desktop\Django\MiniMarketPaul_Django>cd MiniMarket

C:\Users\Paul Manriquez\Desktop\Django\MiniMarketPaul_Django\MiniMarket>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 967C-7E8C

Directorio de C:\Users\Paul Manriquez\Desktop\Django\MiniMarketPaul_Django\MiniMarket

20/07/2024 15:58 <DIR>      .
20/07/2024 15:58 <DIR>      ..
20/07/2024 15:58          688 manage.py
20/07/2024 15:58 <DIR>      MiniMarket
                1 archivos    688 bytes
                3 dirs 164,012,806,144 bytes libres

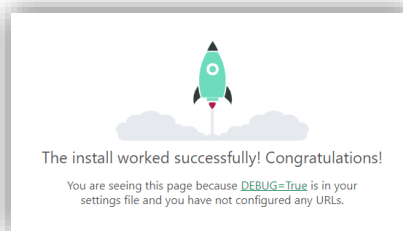
C:\Users\Paul Manriquez\Desktop\Django\MiniMarketPaul_Django\MiniMarket>python manage.py startapp ProjectWebMinimarketApp

C:\Users\Paul Manriquez\Desktop\Django\MiniMarketPaul_Django\MiniMarket>
```

Start the main project

Main App Module

```
C:\Users\Paul Manriquez\Desktop\Django\MiniMarketPaul_Django\MiniMarket>python manage.py runserver
```



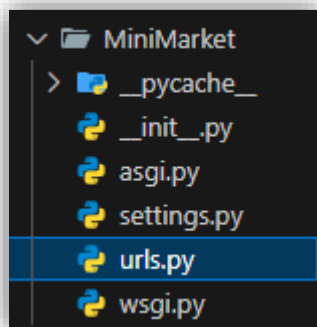
### 2.- Create your views and urls

ProjectWebMini...

- migrations
- \_\_init\_\_.py
- admin.py
- apps.py
- models.py
- tests.py
- views.py
- db.sqlite3
- manage.py

views.py

```
MiniMarket > ProjectWebMinimarketApp > views.py > Contact
1  from django.shortcuts import render,HttpResponse
2
3  # Create your views here.
4
5
6  def Home(request):
7      return HttpResponse('Home')
8
9  def Services(request):
10     return HttpResponse('Services')
11
12  def Store(request):
13     return HttpResponse('Store')
14
15  def Blog(request):
16     return HttpResponse('Blog')
17
18  def Contact(request):
19     return HttpResponse('Contact')
```

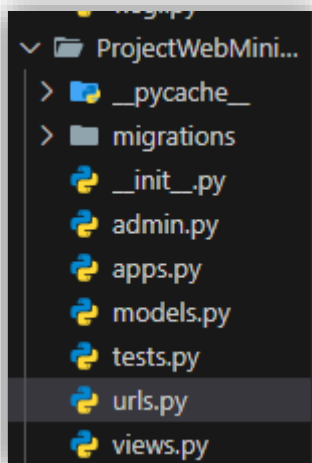


```
from django.contrib import admin
from django.urls import path
from ProjectWebMinimarketApp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.Home, name='Home'),
    path('Services/', views.Services, name='Services'),
    path('Store/', views.Store, name='Store'),
    path('Blog/', views.Blog, name='Blog'),
    path('Contact/', views.Contact, name='Contact'),
]
```

3.- Create a urls for the application to be more readable and modularization of the app.

Create a urls file in your app and add your urls



```
from django.urls import path

from ProjectWebMinimarketApp import views

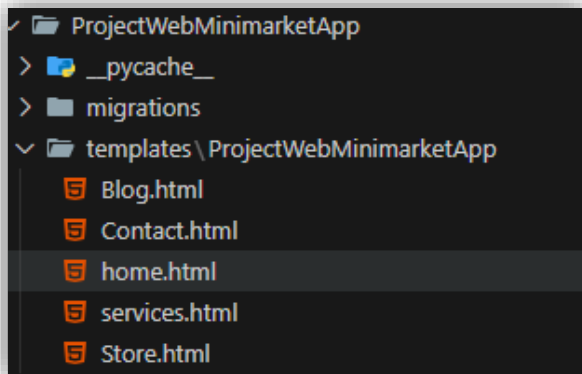
urlpatterns = [
    path('', views.Home, name='Home'),
    path('Services/', views.Services, name='Services'),
    path('Store/', views.Store, name='Store'),
    path('Blog/', views.Blog, name='Blog'),
    path('Contact/', views.Contact, name='Contact'),
]
```

Link the urls of the app in the main urls file

```
from django.contrib import admin
from django.urls import path, include

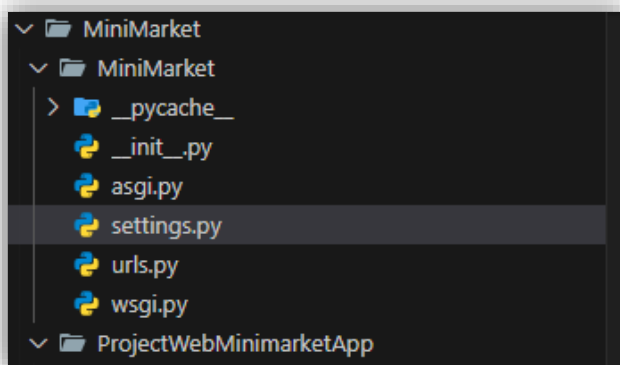
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('ProjectWebMinimarketApp.urls')),
]
```

4.- Create the html files for your app that will be used in the views and update the view file of the application to render your html files



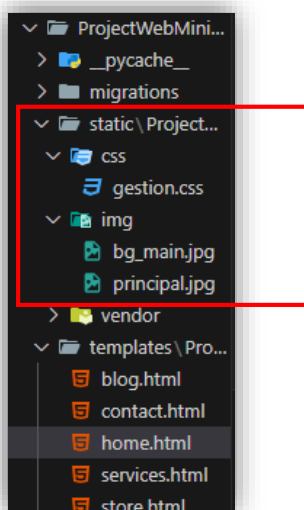
```
def Home(request):  
    return render(request, 'ProjectWebMinimarketApp/home.html')
```

5.- Register the app as installed app in the main project



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

6.- Create and configure the directories, that you will need in your html files, in this project we are using bootstrap and pre-build templates, but you can use your own html/css/bootstrap/js files.



7.- Since the template in this project was already created, the goal on this project is to learn how to navigate and change the current project based on your needs. In this part we modify the project, added some style and create the base.html, that are the codes that we will use along all our webpage.

This is a trick to load a current folder and avoid using all the url for the file were is

```
{% load static %}
<!-- Bootstrap -->
<link href="{% static 'ProjectWebMinimarketApp/vendor/bootstrap/css/bootstrap.min.css' %}" rel="stylesheet
```

Create the base of the pages of this app module

|                                 |    |                         |
|---------------------------------|----|-------------------------|
| templates\ProjectWebMinimark... | 59 |                         |
| base.html                       | 60 | <!--Changing content--> |
| blog.html                       | 61 | {% block content %}     |
|                                 | 62 | {% endblock %}          |

First we use the inheritance saying that in the current directory search for base.html file

Then we load the static file to load the current directory that are being used in our project

Now we use the block content on what we can use to insert diverse content inside the block

```
1
2
3 {% extends "../base.html"%}
4 {% load static %}
5
6 {% block content %}
7     <!-- Heading -->
8     <section class="page-sec
9         <div class="container">
10             <div class="intro">
11                 <img class="intro-i
12                 <div class="intro-t
13                     <h2 class="sectio
14                     <span class="se
15                     <span class="se
16                 </h2>
17                 <p class="mb-3">T
18             </p>
19             </div>
20         </div>
21     </div>
22 </div>
23 </section>
```

PROBLEMS OUTPUT DEBUG CONSOLE

```
raise new from exc
django.template.exceptions.TemplateDo
[20/Jul/2024 20:35:53] "GET / HTTP/1
[20/Jul/2024 20:39:09] "GET / HTTP/1
[20/Jul/2024 20:39:09] "GET /static/
[20/Jul/2024 20:39:09] "GET /static/
```

8.- Enable the links of the navbar in the base.html, since each url was named, you can referred to the url with the name that corresponds in the urls.py of the application



```
<li class="nav-item active px-lg-4">
  <a class="nav-link text-uppercase text-expanded" href="{% url 'Home' %}">Home</a>
</li>
```

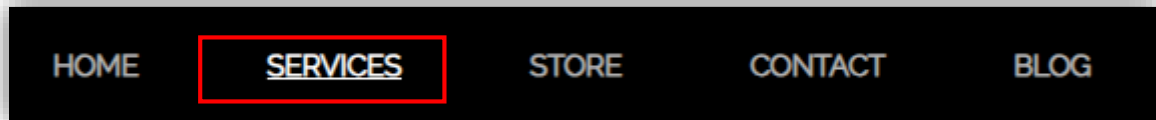
```
MiniMarket > ProjectWebMinimarketApp > urls.py > ...
1  from django.urls import path
2
3  from ProjectWebMinimarketApp import views
4
5  urlpatterns = [
6      path('', views.Home, name='Home'),
7      path('Services/', views.Services, name='Services'),
8      path('Store/', views.Store, name='Store'),
9      path('Blog/', views.Blog, name='Blog'),
10     path('Contact/', views.Contact, name='Contact'),
11 ]
```

9.- Inheritance, each page in our project will inherit the nav and footer, so this is the general initial code of each page where in the block content, we can add the html code that corresponds.

```
MiniMarket > ProjectWebMinimarketApp > templates > ProjectWebMinimarketApp > store.html
1  {% extends "../base.html"%}
2  {% load static %}
3
4  {% block Title %} Store {% endblock %}
5
6  {% block content%}
7
8  {% endblock %}
```

10.- Enable pointing to a url in the navbar if we are in that url

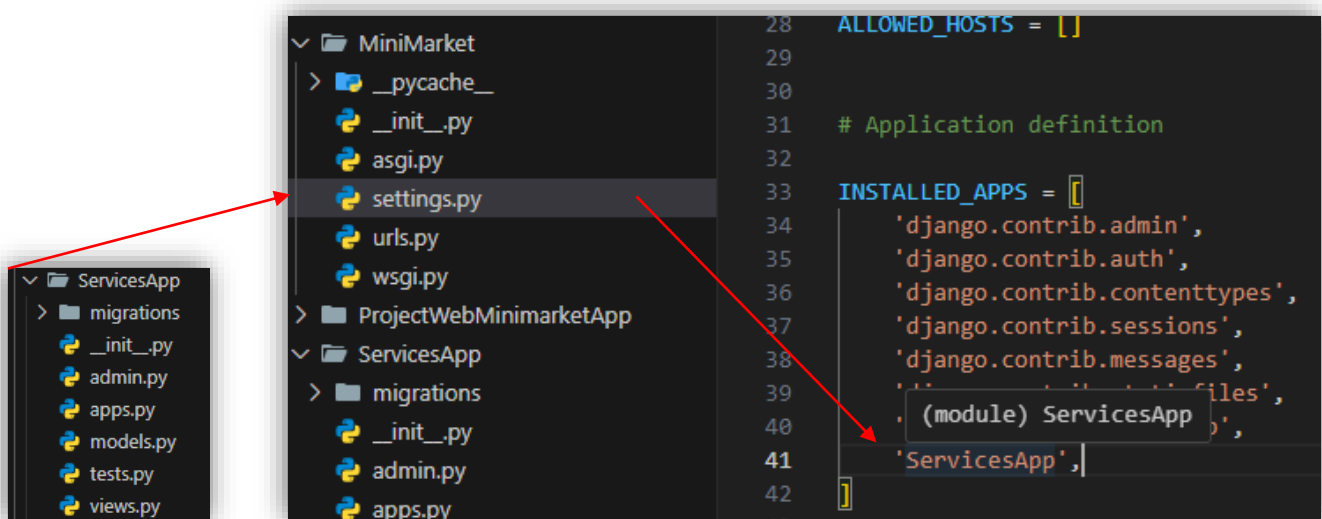
```
<li class="nav-item px-lg-4 {% if request.path == '/'%} active {% endif %}">
  <a class="nav-link text-uppercase text-expanded" href="{% url 'Home'%}">Home</a>
</li>
```



## Creation of the app module 'Services'

1.- To take use of the advantages of the modular creation of our app, we create the new module Services and registered in the main file, settings.py

```
C:\Users\Paul Manriquez\Desktop\Django\MiniMarketPaul_Django\MiniMarket>python manage.py startapp ServicesApp
```



## 2.- Mapping an ORM

Mapping an Object-Relational Mapping (ORM) in Django involves creating models that correspond to database tables. Django's ORM allows you to interact with your database using Python code instead of writing raw SQL queries.

In the models file of the ServicesApp , we create our model of the data base as follows:

```
1  from django.db import models
2
3  # Create your models here.
4  class Service(models.Model):
5      Title = models.CharField(max_length=50)
6      Content = models.CharField(max_length=50)
7      Image = models.ImageField()
8      Created = models.DateTimeField(auto_now_add=True)
9      Updated = models.DateTimeField(auto_now_add=True)
10
11     class Meta:
12         verbose_name = 'Service'
13         verbose_name_plural = 'Services'
14
15     def __str__(self) -> str:
16         return self.Title
```

This method defines the string representation of the model. When you print an instance of `Service`, it will return the value of the `Title` field. This is useful for the Django admin interface and other places where the object needs to be represented as a string.

### Meta Class in Django Models

The Meta class inside a Django model is used to define metadata options for the model. Metadata is "anything that's not a field," such as ordering options (how to order query results), database table name, or human-readable singular and plural names. Here are the specific attributes used in the provided example:

#### verbose\_name

- **Definition:** `verbose_name = 'Service'`
- **Purpose:** This defines a human-readable name for the model. This name is used in the Django admin interface and other parts of Django where the model name might be displayed. By default, Django would use the class name (in this case, `Service`) but you can customize it using `verbose_name`.

#### verbose\_name\_plural

- **Definition:** `verbose_name_plural = 'Services'`
- **Purpose:** This defines a human-readable plural name for the model. Similar to `verbose_name`, but it is used when referring to multiple instances of the model. For example, in the Django admin interface, the section for this model would be labeled "Services" instead of the default, which would be "Services" (the same as the model name, but with an 's' appended).

Now, execute the Migrations of the new Data base

| Command                                      | Description  | Short Explanation   |
|--|--|---|
| <code>python manage.py makemigrations</code> | Creates new migration files based on changes in models.  | Generates migration scripts for model changes.                            |
| <code>python manage.py migrate</code>        | Applies the migrations to the database, synchronizing the schema with the current state of models. | Applies migrations to update the database schema according to the models. |

```
C:\Users\Paul Manriquez\Desktop\Django\MiniMarketPaul_Django\MiniMarket>python manage.py makemigrations
Migrations for 'ServicesApp':
  ServicesApp\migrations\0001_initial.py
    - Create model Service

C:\Users\Paul Manriquez\Desktop\Django\MiniMarketPaul_Django\MiniMarket>python manage.py migrate
Operations to perform:
  Apply all migrations: ServicesApp, admin, auth, contenttypes, sessions
Running migrations:
  Applying ServicesApp.0001_initial... OK
  Applying contenttypes.0001_initial... OK
```

db.sqlite3  
manage.py

Now in the data base you can visualize the changes and the new data base added

| Tables (12)         |              |                                |
|---------------------|--------------|--------------------------------|
| ServicesApp_service |              |                                |
| id                  | integer      | "id" integer NOT NULL          |
| Title               | varchar(50)  | "Title" varchar(50) NOT NULL   |
| Content             | varchar(50)  | "Content" varchar(50) NOT NULL |
| Image               | varchar(100) | "Image" varchar(100) NOT NULL  |
| Created             | datetime     | "Created" datetime NOT NULL    |
| Updated             | datetime     | "Updated" datetime NOT NULL    |

### Register the new service in the Admin Panel

First, create a super user for this Project

```
C:\Users\Paul Manriquez\Desktop\Django\MiniMarketPaul_Django\MiniMarket>python manage.py createsuperuser
Username (leave blank to use 'paulmanriquez'): PaulM
Email address: paulmanriquezengineer@gmail.com
Password:
Password (again):
Superuser created successfully.
```

Run the server, go to admin url and access in the Administration panel

Django administration

Username:

Password:

Log in

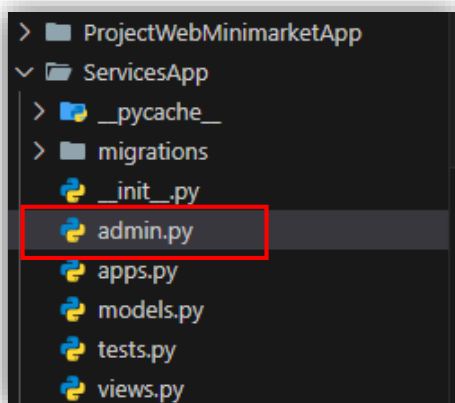
Django administration

Site administration

| AUTHENTICATION AND AUTHORIZATION |       |        |
|----------------------------------|-------|--------|
| Groups                           | + Add | Change |
| Users                            | + Add | Change |

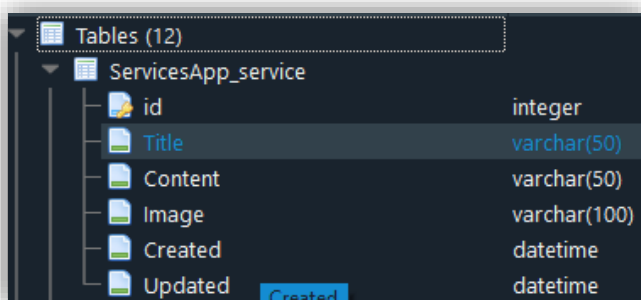
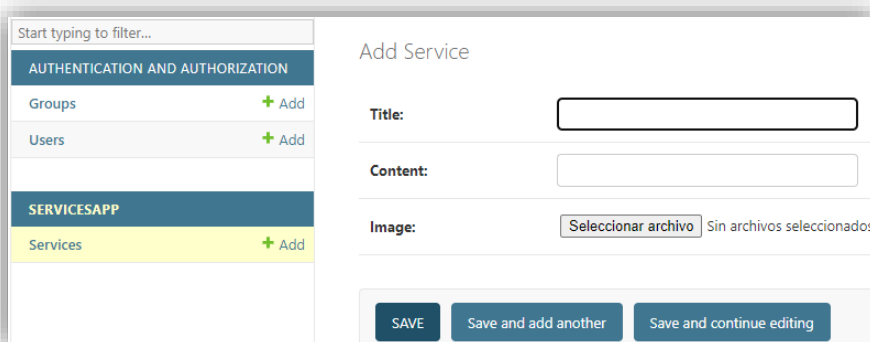
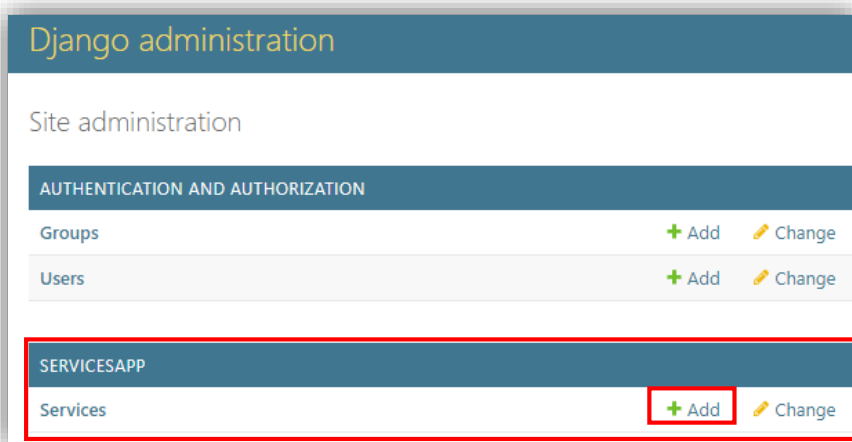


Now, in the admin file of the ServiceApp, we can add the new service as follows



```
from django.contrib import admin

from .models import Service
# Register your models here.
admin.site.register(Service)
```



If you want to visualize Created and Updated field, Modify the admin.py file as follows:

```
from django.contrib import admin

from .models import Service
# Register your models here.
class ServiceAdmin(admin.ModelAdmin):
    readonly_fields = ('Created', 'Updated')

admin.site.register(Service, ServiceAdmin)
```

Add Service

Title:

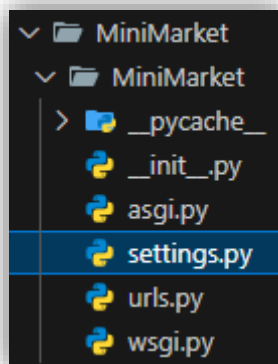
Content:

Image:  Sin archivos seleccionados

Created: -

Updated: -

*If you desire to change the language, you can do it as follows in this section:*

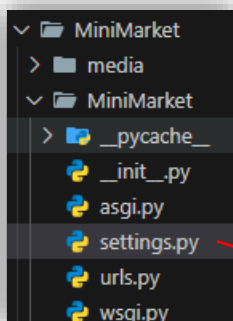


```
108 LANGUAGE_CODE = 'en-us'
109
110 TIME_ZONE = 'UTC'
111
112 USE_I18N = True
113
114 USE_TZ = True
115
```

Image:  Sin archivos seleccionados

*Since we don't configure where to store the files, they will be stored in the root of the project, thus, we need a special directory to store the media uploaded for each module, so, we need to configure as follows:*

Create the Media file and in the settings.py add the next configuration for the Constants:



```
121 STATIC_URL = 'static/'
122
123 MEDIA_URL = '/media/' #<-- Public url how to acces to the media
124 MEDIA_ROOT = os.path.join(BASE_DIR, 'media') #<-- Tell to Django where to search the Dir for media files
125
```

In the model of the data base we now tell where to store the images, we are telling that the media will be stored in the dir Services, if it doesn't exist, it will be created , simply change this and save.

```
MiniMarket > ServicesApp > models.py > Service
1  from django.db import models
2
3  # Create your models here.
4  class Service(models.Model):
5      Title = models.CharField(max_length=50)
6      Content = models.CharField(max_length=50)
7      Image = models.ImageField(upload_to='Services')
8      Created = models.DateTimeField(auto_now_add=True)
9      Updated = models.DateTimeField(auto_now_add=True)
10
11     class Meta:
12         verbose_name = 'Service'
13         verbose_name_plural = 'Services'
14
15     def __str__(self) -> str:
16         return self.Title
```

To be able to see the media that we have upload it is necessary to do the next configuration in the url.py where we were linking to the main urls.py:

MINIMARKETPAUL\_DJANGO

MiniMarket

media\Services

iphone.png

MiniMarket

\_\_pycache\_\_

\_\_init\_\_.py

asgi.py

settings.py

urls.py

wsgi.py

ProjectWebMinimarketApp

\_\_pycache\_\_

migrations

static

templates

\_\_init\_\_.py

admin.py

apps.py

models.py

tests.py

urls.py

views.py

MiniMarket > ProjectWebMinimarketApp > urls.py > ...

4 from django.conf.urls.static import static

5 #===== Adding where to search in media files =====

6 #To be able to see the media in the administration panel

7 from django.conf import settings

8 from django.conf.urls.static import static

9 #=====

15 urlpatterns = [

16 path('',views.Home,name='Home'),

17 path('Services/',views.Services,name='Services'),

18 path('Store/',views.Store,name='Store'),

19 path('Blog/',views.Blog,name='Blog'),

20 path('Contact/',views.Contact,name='Contact'),

21 ]

22 #===== Adding where to search in media files =====

23 #To be able to see the media in the administration panel

24 urlpatterns+=static(settings.MEDIA\_URL,document\_root=settings.MEDIA\_ROOT)

25 #=====

Configurations to see the media uploaded

Linker

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

July 21, 2024 - 01:50:43

Django version 5.0.7, using settings 'MiniMarket.settings'

In Django, configuring URLs to serve media files is crucial for correctly handling user-uploaded files during development. Here's an explanation of why it's necessary and what the code does:

## Purpose of the Configuration

1. **Serve Media Files in Development:** By default, Django does not serve media files (such as user-uploaded images) in development mode. This configuration allows you to access media files via URLs during development.
2. **Admin Panel Display:** When using Django's admin panel, uploaded media files need to be accessible through URLs. Without this configuration, you might see broken links or missing images in the admin interface.

## Code Explanation

Here's a breakdown of the key components in your `urls.py` configuration:

### 1. Import Statements

```
python
Copiar código
from django.conf import settings
from django.conf.urls.static import static
```

- `settings`: Provides access to Django's settings, including `MEDIA_URL` and `MEDIA_ROOT`.
- `static`: A utility function to serve static files during development.

### 2. `urlpatterns` Definition

```
python
Copiar código
urlpatterns = [
    path("", views.Home, name='Home'),
    path('Services/', views.Services, name='Services'),
    path('Store/', views.Store, name='Store'),
    path('Blog/', views.Blog, name='Blog'),
    path('Contact/', views.Contact, name='Contact'),
]
```

- Defines URL patterns for various views in your application.

### 3. Media Files Handling

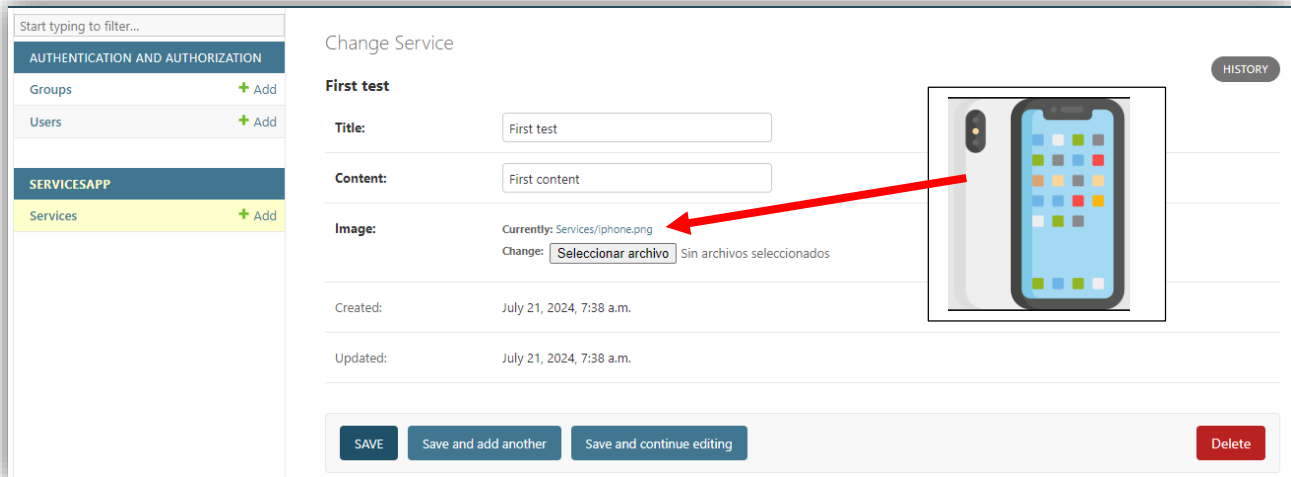
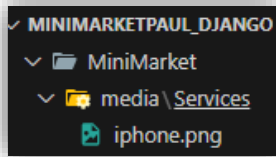
```
python
Copiar código
urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

- `settings.MEDIA_URL`: The URL prefix for serving media files (e.g., `/media/`).
- `settings.MEDIA_ROOT`: The filesystem path where media files are stored (e.g., `/path/to/media/`).
- `static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)`: This function appends a URL pattern to `urlpatterns` that tells Django to serve files from `MEDIA_ROOT` at the URL prefix specified by `MEDIA_URL`.

## Summary

This configuration is necessary for development purposes to ensure that media files uploaded by users can be served and viewed properly. In production environments, serving media files is typically handled by a dedicated web server like Nginx or through cloud storage services, rather than Django itself.

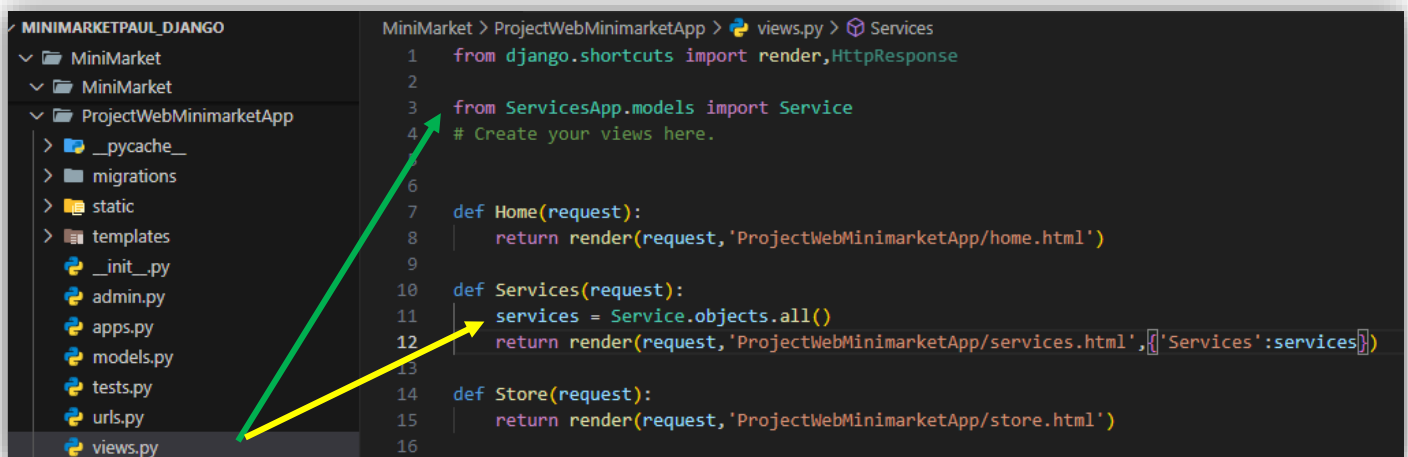
Now in the Panel administration, we add a new row in the Service data base and we can see that was created and uploaded correctly:



*Now The goal is to see displayed the services that we have created in the services page:*



1.- Pass the models objects services to the template of the services



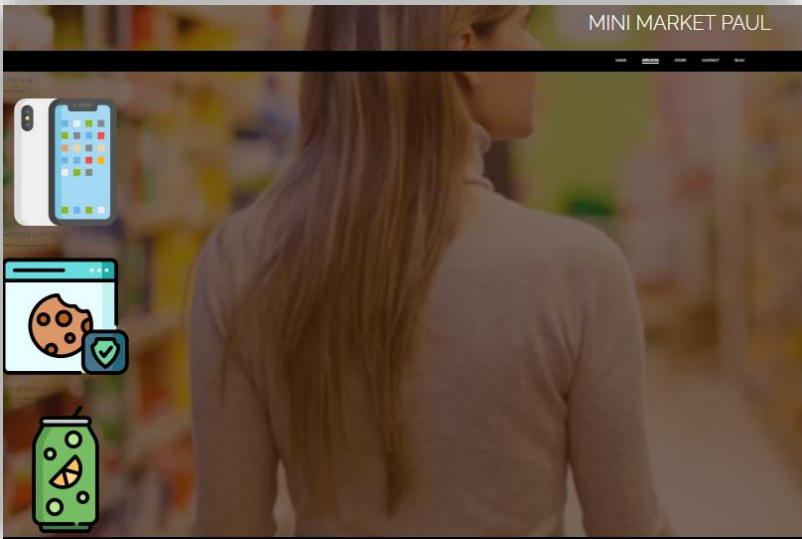
In the html file, since we will have several data in the services, we add a for each to pass through each data for the service as follows:

```
{% extends "../base.html"%}
{% load static %}

{% block Title %} Services {% endblock %}

<!-->
{% block content%}
    {% for service in Services %}
        <div>
            <p>
                <h2>{{service.Title}}</h2>
                <p>{{service.Content}}</p>
                <p><img src='{{service.Image.url}}'></p>
            </p>
        </div>
    {% endfor %}
{% endblock %}
```

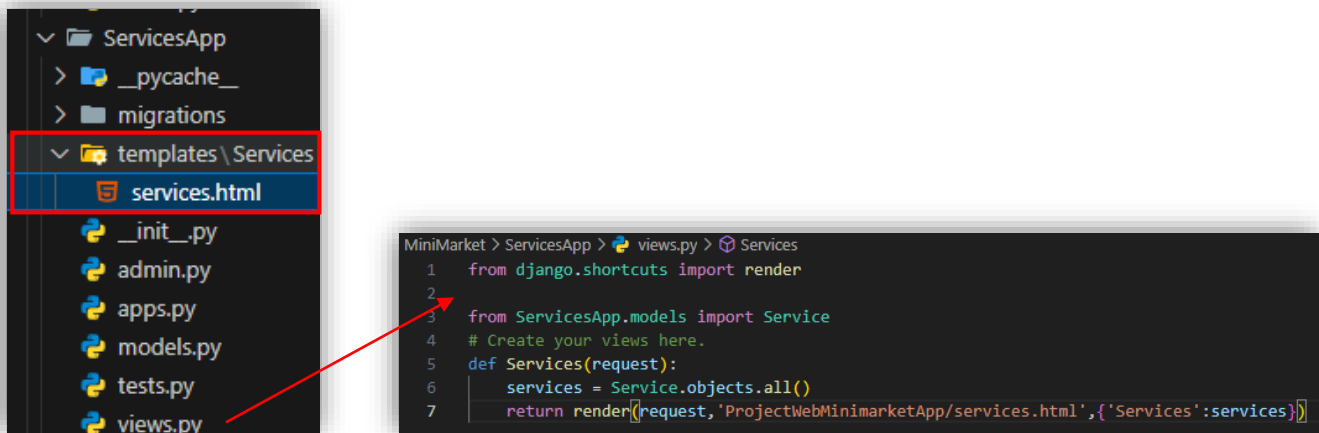
Now we can see that for each service added will be displayed, for the moment I don't added a Style css but this will be added.



| Table: ServicesApp_service |        |                |                            |                         |                            |               |
|----------------------------|--------|----------------|----------------------------|-------------------------|----------------------------|---------------|
|                            | id     | Title          | Content                    | Image                   | Created                    | Upd           |
|                            | Filter | Filter         | Filter                     | Filter                  | Filter                     | Filter        |
| 1                          | 1      | First test     | First content              | Services/iphone.png     | 2024-07-21 07:38:40.293148 | 2024-07-21 07 |
| 2                          | 2      | Second service | This is the second service | Services/cookies.png    | 2024-07-21 08:58:31.175301 | 2024-07-21 08 |
| 3                          | 3      | Third service  | This is the third service  | Services/soft-drink.png | 2024-07-21 08:59:22.880171 | 2024-07-21 08 |

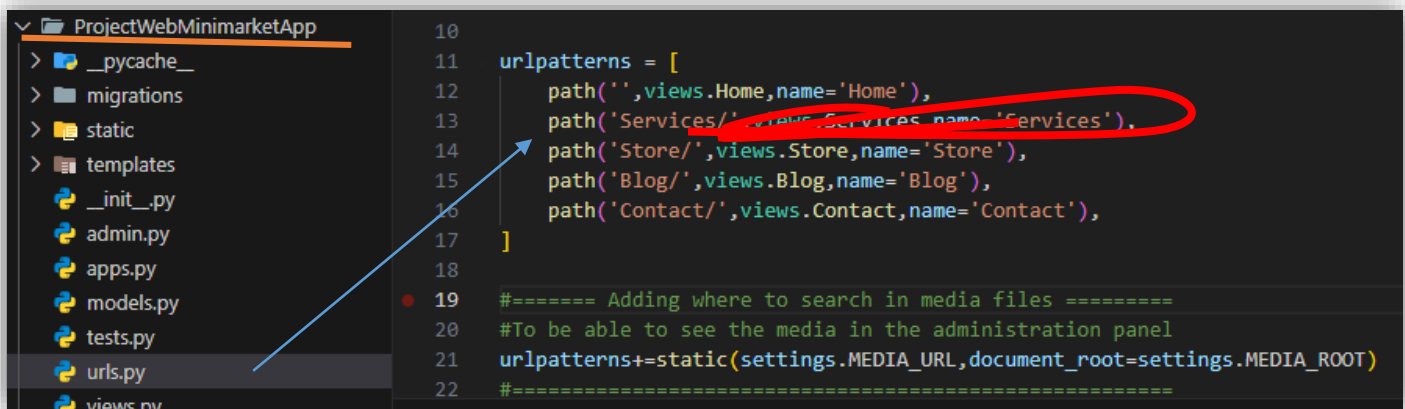
**Now that we have created the service, is much better to save the view of this app module in the module itself where belongs, to achieve this is as follows:**

Create the dirs. In the corresponding app in this case (service) and move the view of the service to the view py of the module where corresponds as follows:

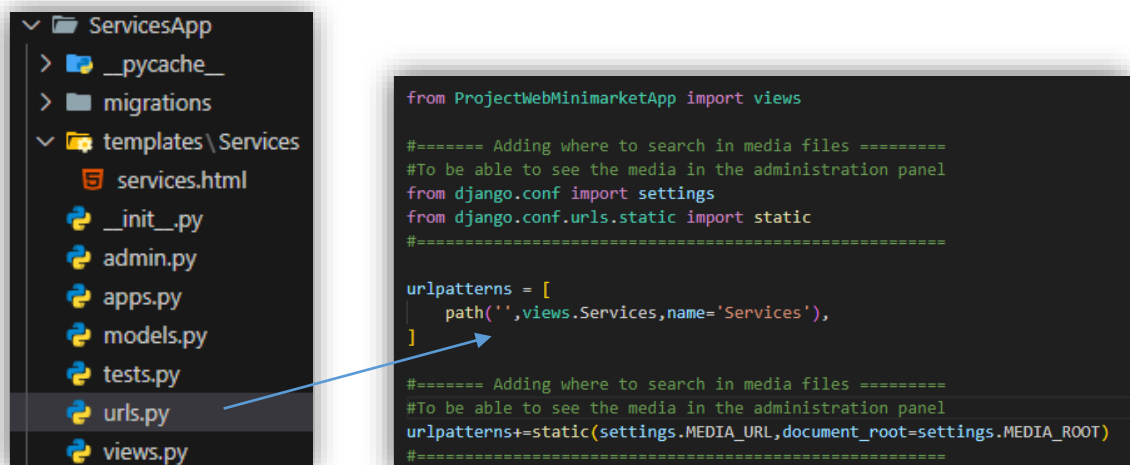


Since this view no longer exist in the original urls where we set all the urls, we need to delete the url and move itt to it own urls file of the application , so 1) delete the older url direction of the main application and 2) Move the url to the file where corresponds in the specific app

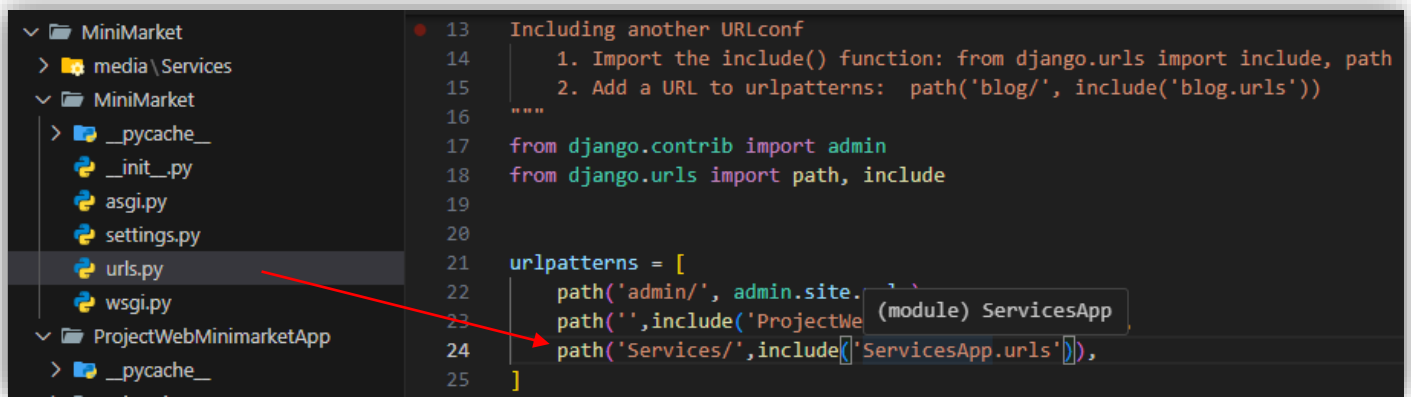
1) delete and move



2) Create the urls.py file of the module app and, since we are in the root of the app, we can set it as it is (a root)

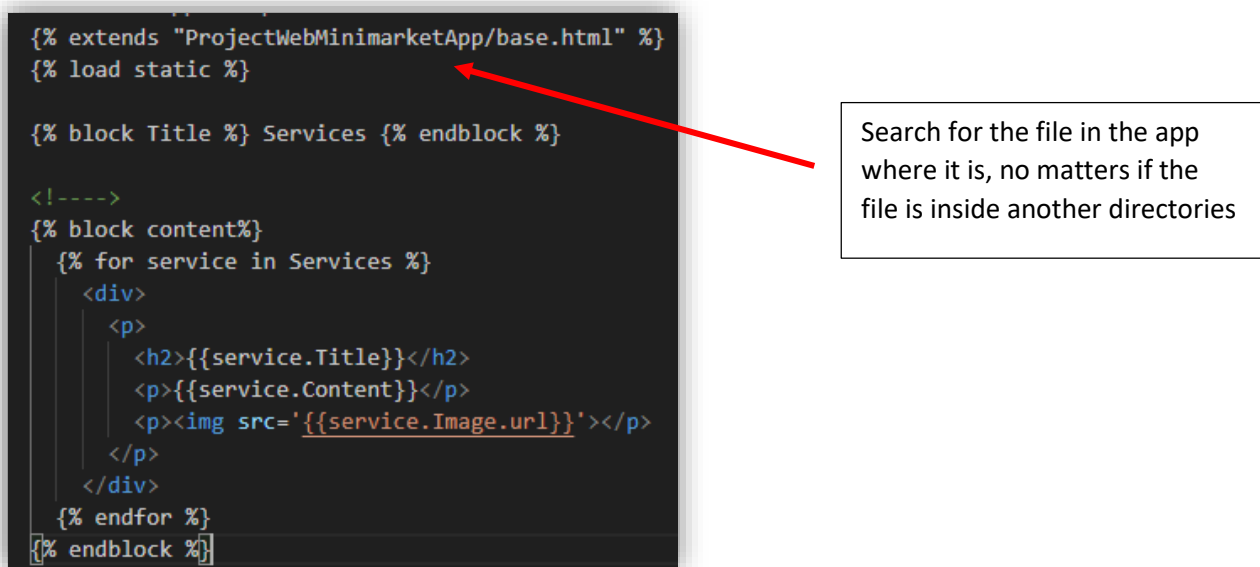


Include in the main app in the urls file the service where comes the urls of the module services



```
13 Including another URLconf
14 1. Import the include() function: from django.urls import include, path
15 2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
16 """
17 from django.contrib import admin
18 from django.urls import path, include
19
20
21 urlpatterns = [
22     path('admin/', admin.site.urls),
23     path('', include('ProjectWebMinimarketApp.urls')),
24     path('Services/', include('ServicesApp.urls')),
25 ]
```

**Quote:** Since Django search for the templates of each app, it can be referenced as follows



```
{% extends "ProjectWebMinimarketApp/base.html" %}
{% load static %}

{% block Title %} Services {% endblock %}

<!-->
{% block content%}
    {% for service in Services %}
        <div>
            <p>
                <h2>{{service.Title}}</h2>
                <p>{{service.Content}}</p>
                <p><img src='{{service.Image.url}}'></p>
            </p>
        </div>
    {% endfor %}
{% endblock %}
```

Search for the file in the app where it is, no matters if the file is inside another directories

In this part we re-use code of the home.html to display our services, from the admin now you can add a new service each time you want

```
{% extends "ProjectWebMinimarketApp/base.html" %}
{% load static %}

{% block Title %} Services {% endblock %}

<!-->
{% block content%}
    {% for service in Services %}
        <section class="page-section clearfix">
            <div class="container">
                <div class="intro">
                    
                    <div class="intro-text left-0 text-center bg-faded p-5 rounded">
                        <h2 class="section-heading mb-4">
                            <span class="section-heading-upper">{{service.Content}}</span>
                            <span class="section-heading-lower">{{service.Title}}</span>
                        </h2>
                    </div>
                </div>
            </div>
        </section>
    {% endfor %}
{% endblock %}
```

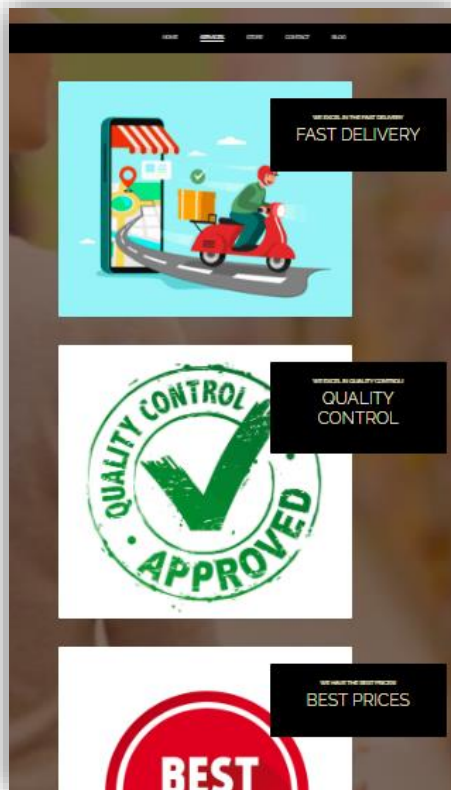


## Select Service to change

[ADD SERVICE +](#)

Action:   0 of 3 selected

- ☐ SERVICE
- ☐ Best prices
- ☐ Quality control
- ☐ Fast delivery



## Creation of a blog section

In this part we will focus on the creation of the blog section, we create the app and the model tables of the app

```
C:\Users\Paul Manriquez\Desktop\Django\MiniMarketPaul_Django\MiniMarket>python manage.py startapp BlogApp
```

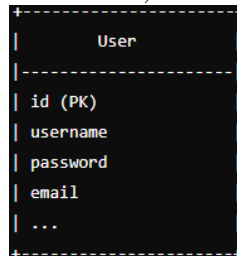
```
from django.contrib.auth.models import User
```

This line imports the User model from Django's built-in authentication system. The User model is used to handle user accounts and provides fields such as username, password, email, first name, last name, etc

```
Author = models.ForeignKey(User, on_delete=models.CASCADE)
```

- `Author = models.ForeignKey(User, on_delete=models.CASCADE)` defines a foreign key relationship between the Post model and the User model.
- `models.ForeignKey` is used to create a many-to-one relationship. This means that many posts can be associated with one user (the author).
- `User` is the model that this foreign key points to, which means each post will be associated with one specific user.
- `on_delete=models.CASCADE` specifies that if the referenced User is deleted, all related Post instances will also be deleted. This ensures referential integrity by not leaving orphaned posts with no associated author.

User is a model, so is a table



```
Categories = models.ManyToManyField(Category)
```

- `Categories = models.ManyToManyField(Category)` defines a many-to-many relationship between the Post model and the Category model.
- `models.ManyToManyField` is used to create a relationship where multiple categories can be associated with multiple posts. This allows a post to belong to multiple categories and a category to include multiple posts.
- `Category` is the model that this field is relating to, indicating that each post can have multiple categories and each category can have multiple posts.

```
from django.db import models

from django.contrib.auth.models import User
# Create your models here.
class Category(models.Model):
    Name = models.CharField(max_length=50)
    Created = models.DateTimeField(auto_now_add=True)
    Updated = models.DateTimeField(auto_now_add=True)

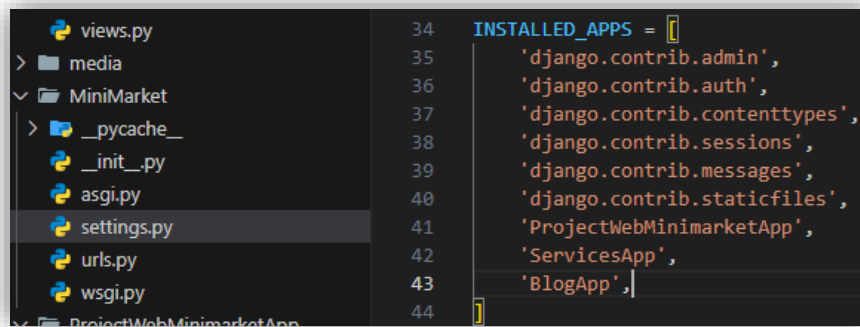
    class Meta:
        verbose_name = 'Category'
        verbose_name_plural = 'Categories'

    def __str__(self) -> str:
        return self.Name

class Post(models.Model):
    Title = models.CharField(max_length=50)
    Content = models.CharField(max_length=500)
    Image = models.ImageField(upload_to='Blog', null=True, blank=True)
    Author = models.ForeignKey(User, on_delete=models.CASCADE)
    Categories = models.ManyToManyField(Category)
    Created = models.DateTimeField(auto_now_add=True)
    Updated = models.DateTimeField(auto_now_add=True)

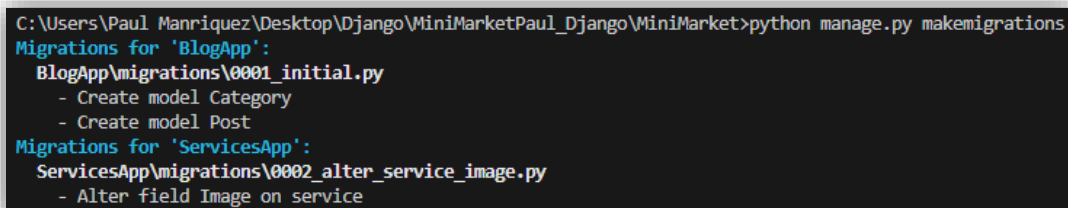
    class Meta:
        verbose_name = 'Post'
        verbose_name_plural = 'Posts'
```

Install the app in the settings.py of the main project

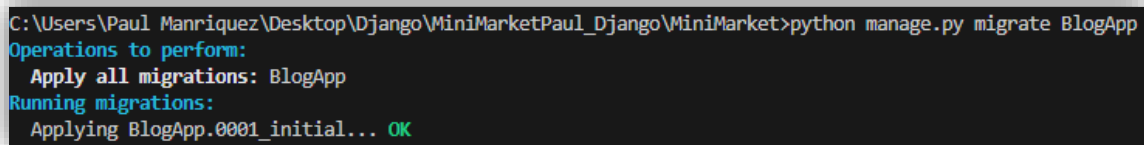


```
34 INSTALLED_APPS = [
35     'django.contrib.admin',
36     'django.contrib.auth',
37     'django.contrib.contenttypes',
38     'django.contrib.sessions',
39     'django.contrib.messages',
40     'django.contrib.staticfiles',
41     'ProjectWebMinimarketApp',
42     'ServicesApp',
43     'BlogApp',
44 ]
```

Execute migrations

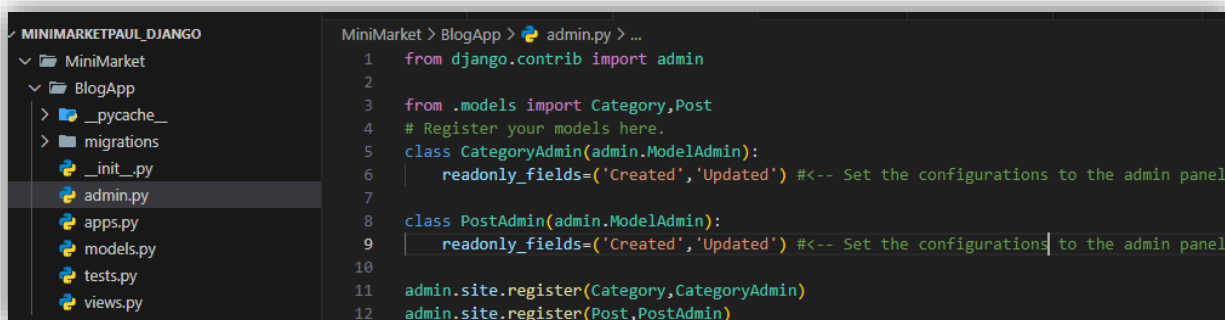


```
C:\Users\Paul Manriquez\Desktop\Django\MiniMarketPaul_Django\MiniMarket>python manage.py makemigrations
Migrations for 'BlogApp':
  BlogApp\migrations\0001_initial.py
    - Create model Category
    - Create model Post
Migrations for 'ServicesApp':
  ServicesApp\migrations\0002_alter_service_image.py
    - Alter field Image on service
```



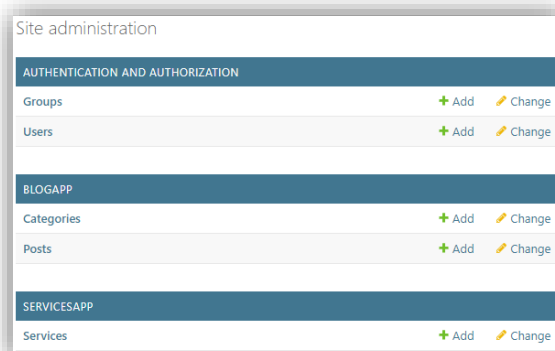
```
C:\Users\Paul Manriquez\Desktop\Django\MiniMarketPaul_Django\MiniMarket>python manage.py migrate BlogApp
Operations to perform:
  Apply all migrations: BlogApp
Running migrations:
  Applying BlogApp.0001_initial... OK
```

**Add to administration panel**



```
MiniMarket > BlogApp > admin.py > ...
1  from django.contrib import admin
2
3  from .models import Category, Post
4  # Register your models here.
5  class CategoryAdmin(admin.ModelAdmin):
6      |   readonly_fields=('Created','Updated') #<-- Set the configurations to the admin panel
7
8  class PostAdmin(admin.ModelAdmin):
9      |   readonly_fields=('Created','Updated') #<-- Set the configurations to the admin panel
10
11  admin.site.register(Category, CategoryAdmin)
12  admin.site.register(Post, PostAdmin)
```

Now we can see the new model in the admin panel



## Adding a category and a Post

Add Category

**Name:**

Created: -




Updated: -

Add Post

**Title:**

**Content:**


**Image:**  laptop.png

**Author:**    

**Categories:**

Internationals

Nationals

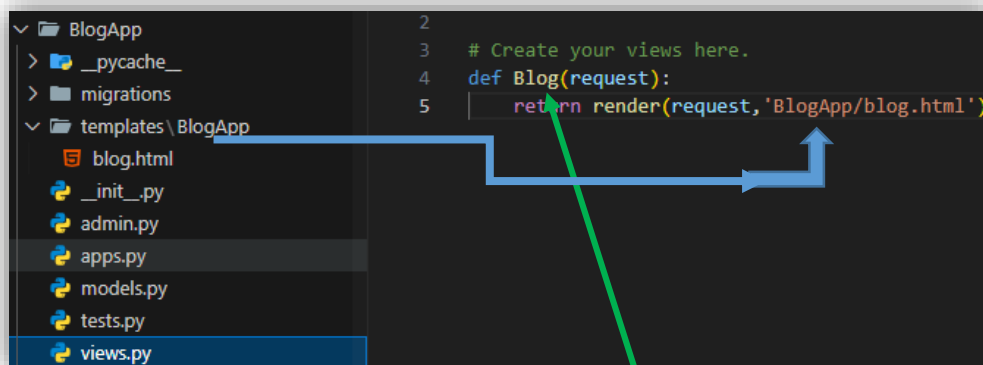


Hold down "Control", or "Command" on a Mac, to select more than one.

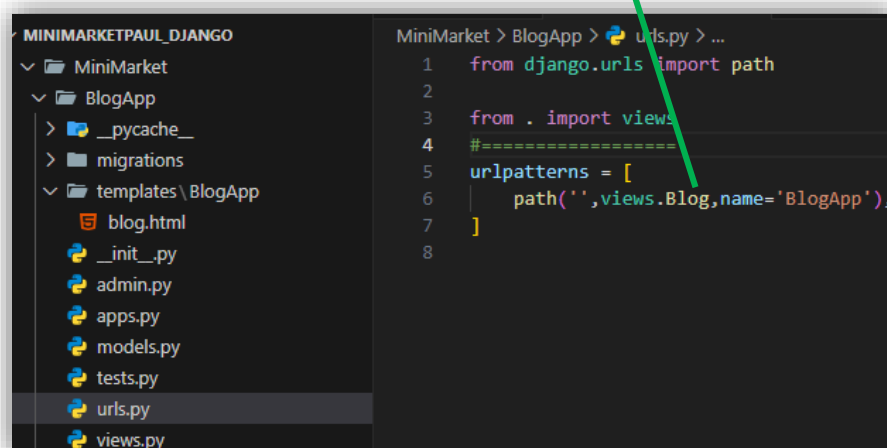
Created: -

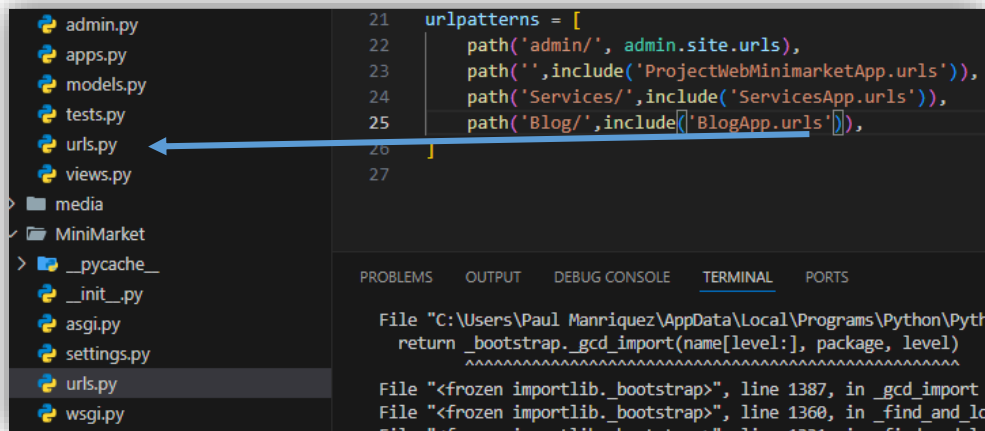
Updated: -

To use the page in our specific app, we need to set as follows



Register the new url to the main app



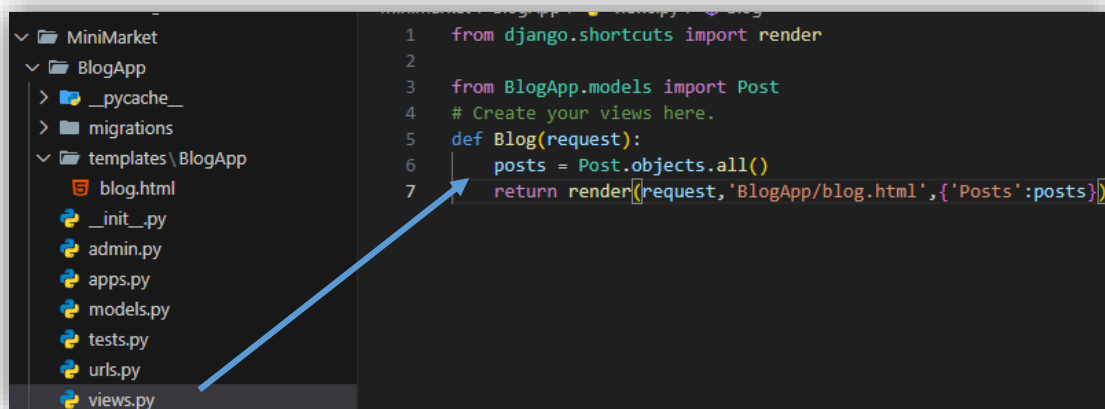


```
admin.py
apps.py
models.py
tests.py
urls.py
views.py
media
MiniMarket
  __pycache__
  __init__.py
  asgi.py
  settings.py
  urls.py
  wsgi.py
```

```
21 urlpatterns = [
22     path('admin/', admin.site.urls),
23     path('', include('ProjectWebMinimarketApp.urls')),
24     path('Services/', include('ServicesApp.urls')),
25     path('Blog/', include('BlogApp.urls')),
26 ]
27
```

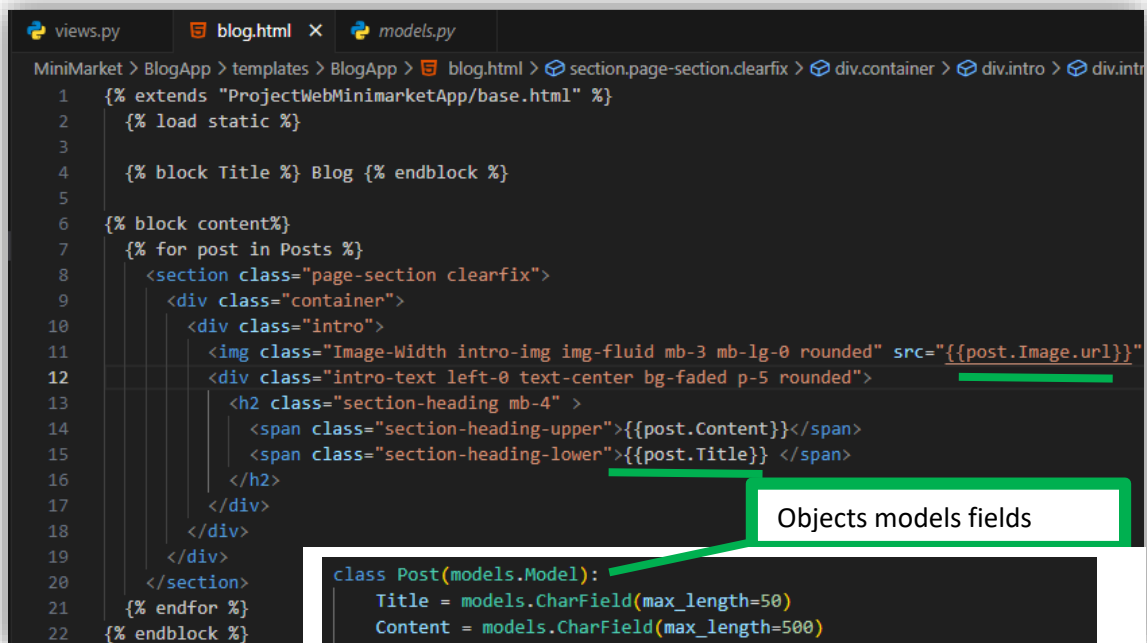
PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

File "C:\Users\Paul Manriquez\AppData\Local\Programs\Python\Python38\python.exe", line 1387, in \_gcd\_import  
File "<frozen importlib.\_bootstrap>", line 1360, in \_find\_and\_load  
File "<frozen importlib.\_bootstrap>", line 1331, in \_find\_and\_load\_unlocked



```
1 from django.shortcuts import render
2
3 from BlogApp.models import Post
4 # Create your views here.
5 def Blog(request):
6     posts = Post.objects.all()
7     return render(request, 'BlogApp/blog.html', {'Posts': posts})
```

MiniMarket > BlogApp > templates > BlogApp > blog.html



```
1 {% extends "ProjectWebMinimarketApp/base.html" %}
2 {% load static %}
3
4 {% block Title %} Blog {% endblock %}
5
6 {% block content %}
7     {% for post in Posts %}
8         <section class="page-section clearfix">
9             <div class="container">
10                 <div class="intro">
11                     
12                     <div class="intro-text left-0 text-center bg-faded p-5 rounded">
13                         <h2 class="section-heading mb-4">
14                             <span class="section-heading-upper">{{post.Content}}</span>
15                             <span class="section-heading-lower">{{post.Title}}</span>
16                         </h2>
17                     </div>
18                 </div>
19             </div>
20         </section>
21     {% endfor %}
22 {% endblock %}
```

Objects models fields



```
class Post(models.Model):
    Title = models.CharField(max_length=50)
    Content = models.CharField(max_length=500)
    Image = models.ImageField(upload_to='BlogApp', null=True, blank=True)
    Author = models.ForeignKey(User, on_delete=models.CASCADE)
    Categories = models.ManyToManyField(Category)
    Created = models.DateTimeField(auto_now_add=True)
    Updated = models.DateTimeField(auto_now_add=True)

    class Meta:
        verbose_name = 'Post'
        verbose_name_plural = 'Posts'

    def __str__(self) -> str:
        return self.Title
```

Now we can see displayed a post created from the admin panel



### Filtering by Category id parameter:

In this section we are adding a new view with goal is to filter by a parameter, get all the post but filtered by category, so since 'category\_id' comes like a parameter, this parameter is being used as the id of the category that corresponds and then we are getting the post corresponding to that category

```
def Category_view(request, category_id):
    category = Category.objects.get(id=category_id) #<--- Get all the categories according to the id related
    posts = Post.objects.filter(Categories=category) #<--- Show the post related to the category and get all
    return render(request, 'BlogApp/category.html', {'Category': category, 'Posts': posts})
```

Figure 1 views.py BlogApp

```
urlpatterns = [
    path('', views.Blog, name='BlogApp'),
    path('Category/<int:category_id>/', views.Category_view, name='Category')
]
```

Figure 2 urls.py BlogApp

**Quote: Since Category is the name of the model data base, the endpoint Category\_view cannot be called as Category**

This is how we can reference the link that we have created to redirect to our category

```
<!--Display Every Category-->
<section>
  <div class="categories-blog">
    Categories:
    {% for post in Posts%}
      {% for category in post.Categories.all %}
        <a href="{% url 'Category' category.id %}"> {{category.Name}} </a>
      {% endfor %}
    {% endfor %}
  </div>
</section>
```

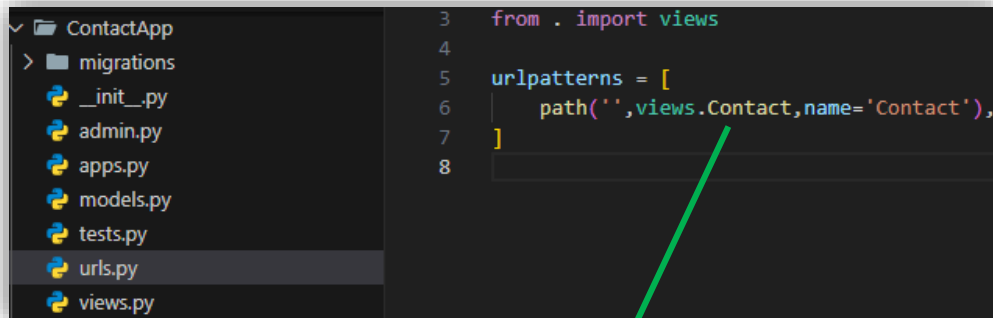
```
urlpatterns = [
    path('', views.Blog, name='BlogApp'),
    path('Category/<int:category_id>/', views.Category_view, name='Category')
]
```

## Contact module app

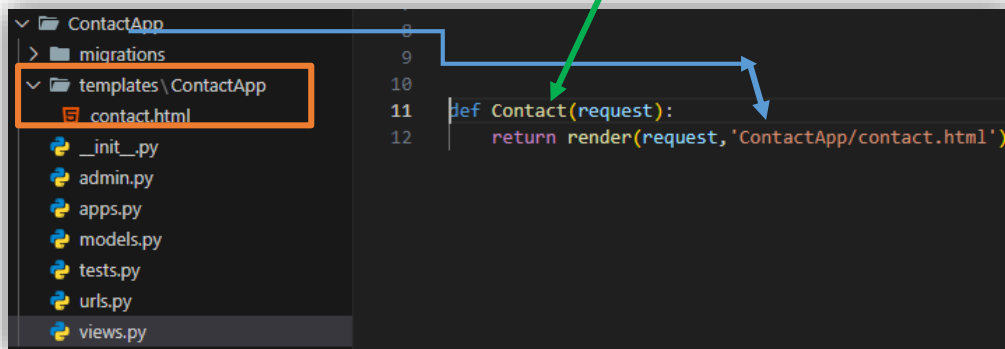
Create the module of the app

```
C:\Users\Paul Manriquez\Desktop\Django\MiniMarketPaul_Django\MiniMarket>python manage.py startapp ContactApp
```

Create the views.py and urls.py of the app as follows

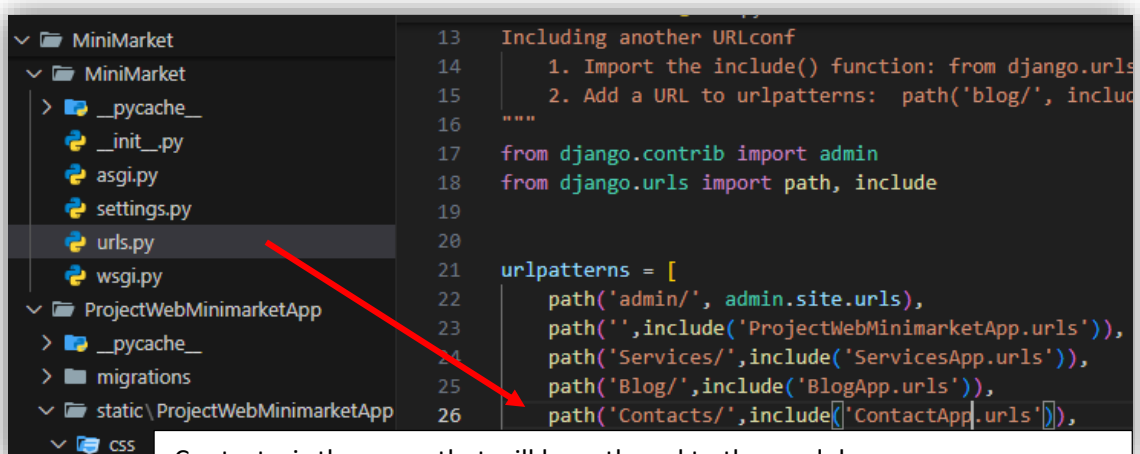


```
3 from . import views
4
5 urlpatterns = [
6     path('', views.Contact, name='Contact'),
7 ]
8
```




```
8
9
10
11 def Contact(request):
12     return render(request, 'ContactApp/contact.html')
```

Register the new url to contacts in the main app



```
13 Including another URLconf
14 1. Import the include() function: from django.urls
15 2. Add a URL to urlpatterns: path('blog/', include
16 """
17 from django.contrib import admin
18 from django.urls import path, include
19
20
21 urlpatterns = [
22     path('admin/', admin.site.urls),
23     path('', include('ProjectWebMinimarketApp.urls')),
24     path('Services/', include('ServicesApp.urls')),
25     path('Blog/', include('BlogApp.urls')),
26     path('Contacts/', include('ContactApp.urls')),
27 ]
```

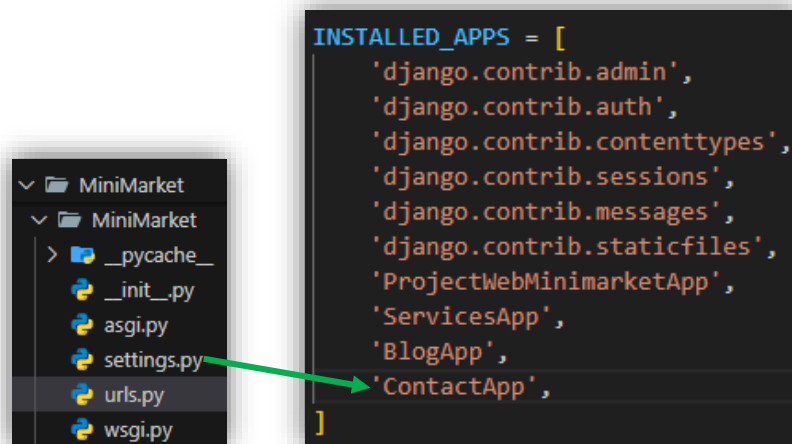
Contacts, is the name that will have the url to the module

 127.0.0.1:8000/Contacts/ Since in this part we only have ' ', as the root we access in the root simply with the path register in the main urls,

If you named of any other form is as follows: /Contacts/yournameroute/

```
urlpatterns = [
    path(' ', views.Contact, name='Contact'),
]
# {% url 'Contact' %} | Name enables to access or refers
# in a part of the project that you desire
```

Register the new app



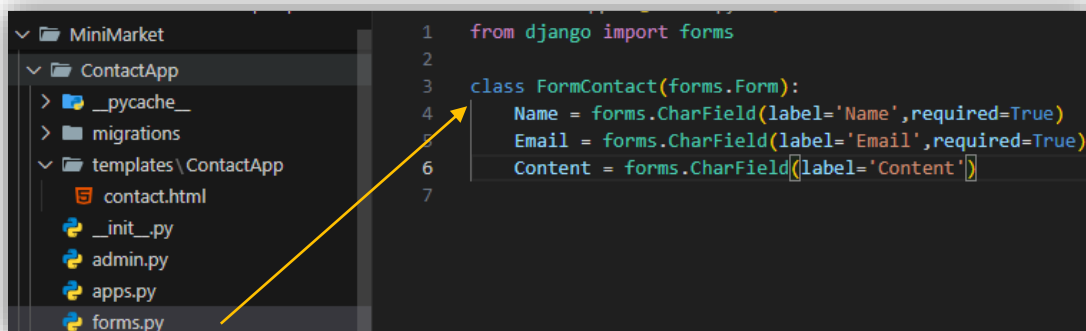
Now its all set, to conserve the underline where we are now, don't forget to name the root path exactly as how we access to it

```
<li class="nav-item px-lg-4" {% if request.path == '/Contacts/' %} active {% endif %}>
  <a class="nav-link text-uppercase text-expanded" href="{% url 'Contact' %}">Contact</a>
</li>
```



## Forms in Django

Create a forms.py in your app module as follows



In the views.py now you can create an instance of the form and make an instance

```
from .forms import FormContact
# Create your views here.
def Contact(request):
    Form_Contacts = FormContact() #Object of FormContact
    return render(request, 'ContactApp/contact.html', {'Form_Contact': Form_Contacts})
```

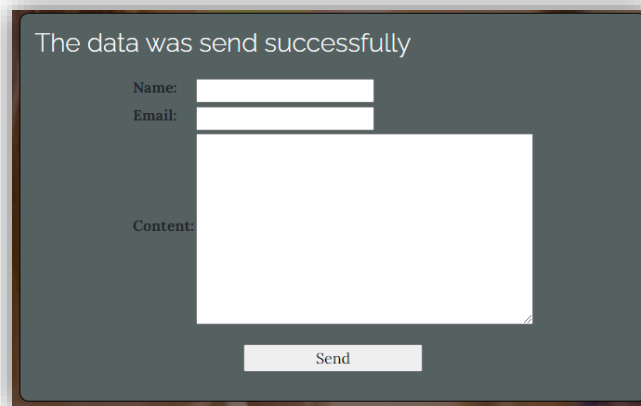


Now we can use the form that we have created in the contac.html file as follows:

You can also use the attribute of the object to choose how to display the table (.as\_table)

```
<div class="Form-contac">
  <form action="" method="POST" class="Form-contact-form">{%csrf_token%}
    <table class="Table-contact">
      {{Form_Contact.as_table}}
    </table>
    <input type="submit" value="Send" class="Btn-contact">
  </form>
</div>
{% endblock %}
```

Now we have the form like this, now it's up to you give it some more style



```
<div class="Form-contac">
  {% if 'valid' in request.GET %}
    <h3>The data was send successfully</h3>
  {% endif %}
  <form x action="" method="POST" class="Form-contact-form">{%csrf_token%}
    <table class="Table-contact">
      {{Form_Contact.as_table}}
    </table>
    <input type="submit" value="Send" class="Btn-contact">
  </form>
</div>
{% endblock %}
```

```
from .forms import FormContact
# Create your views here.
def Contact(request):
    Form_Contacts = FormContact() #Object of FormContact

    if request.method == 'POST':
        Form_Contacts = FormContact(data=request.POST) #Get the data from the post request
        Name = request.POST.get('Name')
        Email = request.POST.get('Email')
        Content = request.POST.get('Content')

        return redirect('/Contacts/?valid') #<--redirect with a parameter is like a flag
                                           #to tell to the html that is a post request

    return render(request, 'ContactApp/contact.html', {'Form_Contact':Form_Contacts})
```

If was a post method, it renders the same page but with a message saying: the data was send successfully

**Send an email in Django**, configure the settings.py of the main file as follows and test in the shell, if all was correct the returned value is a 1

```
#EMAIL SEND | Configuration
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_USE_TLS = True
EMAIL_PORT = 587
EMAIL_HOST_USER = 'paulmanriquezengineer@gmail.com'
EMAIL_HOST_PASSWORD = [REDACTED]
```

```
C:\Users\Paul Manriquez\Desktop\Django\MiniMarketPaul Django\MiniMarket>python manage.py shell
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from django.core.mail import send_mail
>>> send_mail('Asunto', 'Mensaje', 'paulmanriquezengineer@gmail.com', ['paulmanriquezengineer@gmail.com'], fail_silently=False)
1
```

```

from django.core.mail import EmailMessage

# Create an EmailMessage instance
email = EmailMessage(
    subject='Hello', # 1. subject: The subject of the email
    body='This is a test email.', # 2. body: The main content of the email
    from_email='your_email@example.com', # 3. from_email: The sender's email address
    to=['recipient1@example.com', 'recipient2@example.com'], # 4. to: List of recipient email addresses
    bcc=['bcc@example.com'], # 5. bcc: List of email addresses to send a blind carbon copy
    (optional)
    connection=None, # 6. connection: Email backend to use (optional)
    attachments=None, # 7. attachments: List of attachments (optional)
    headers={'Message-ID': 'foo'}, # 8. headers: Additional email headers (optional)
    cc=['cc@example.com'], # 9. cc: List of email addresses to send a carbon copy (optional)
    reply_to=['replyto@example.com'], # 10. reply_to: List of email addresses for the Reply-To header
    (optional)
)

```

```

from django.shortcuts import render, redirect

from .forms import FormContact
from django.core.mail import EmailMessage
# Create your views here.
def Contact(request):
    Form_Contacts = FormContact() #Object of FormContact

    if request.method == 'POST':
        Form_Contacts = FormContact(data=request.POST) #Get the data from the post request
        if Form_Contacts.is_valid(): # <-- Check all the data was filled
            Name = request.POST.get('Name')
            Email = request.POST.get('Email')
            Content = request.POST.get('Content')

            Email_Message = EmailMessage('This is a message from django',
                                         'The user: {} with Email:{} wrote this:\n{}'.format(Name,Email,Content),
                                         ['paulmanriquezengineer@gmail.com'],reply_to=[Email])

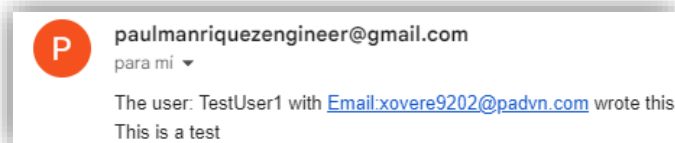
            try:
                Email_Message.send()


                return redirect('/Contacts/?valid') #<--redirect with a parameter is like a flag
                                                    #to tell to the 'html' that is a post request
            except:
                return redirect('/Contacts/?novalid')

    return render(request, 'ContactApp/contact.html',{'Form_Contact':Form_Contacts})

```

For this test I use an email generator and this are the results:



| REMITENTE   | ASUNTO                            | VER                  |
|---|-----------------------------------|----------------------|
|  Paul Manriquez<br>paulmanriquezengineer@gmail.com | Re: This is a message from django | <a href="#">&gt;</a> |

The data was send successfully

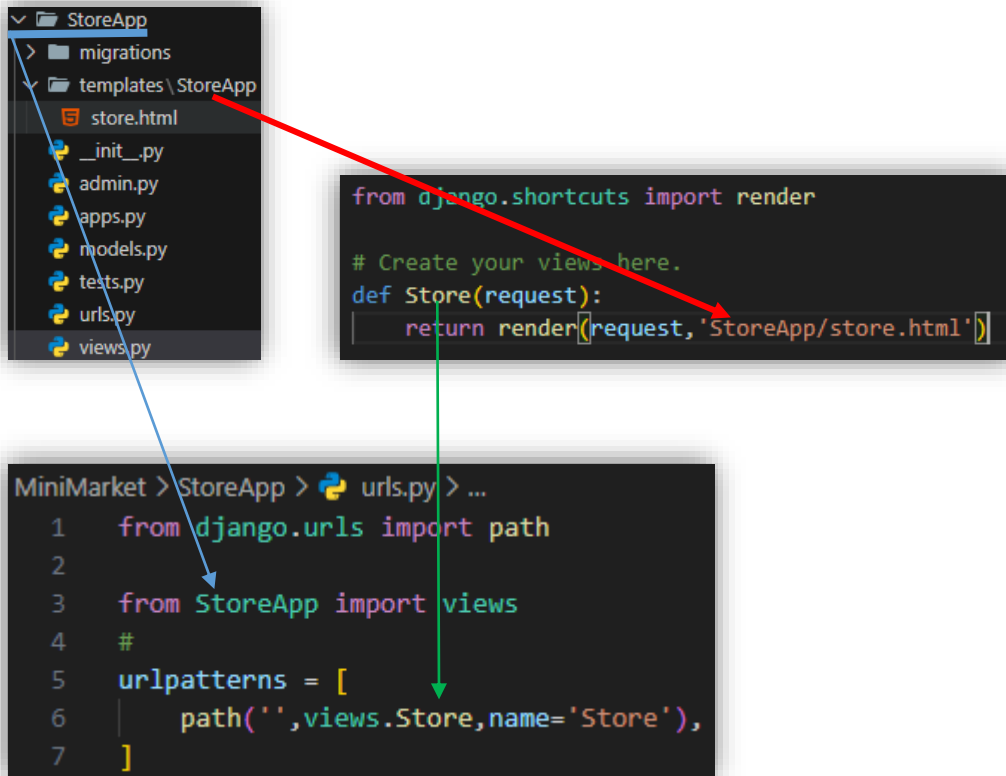
Name:   
 Email:   
 Content:

## Store app

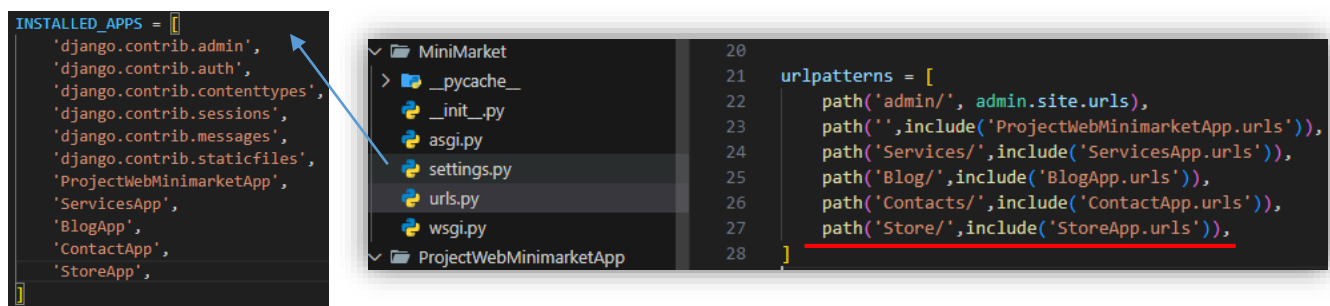
Creation of the application

```
C:\Users\Paul Manriquez\Desktop\Django\MiniMarketPaul_Django\MiniMarket>python manage.py startapp StoreApp
```

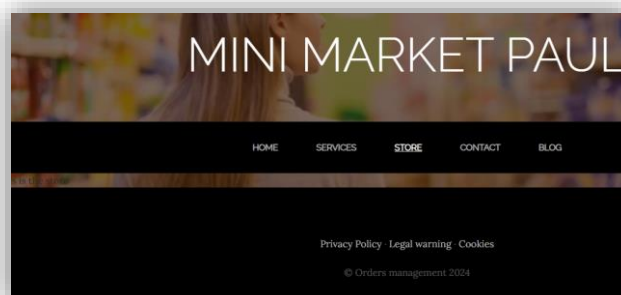
Add and configure the the views.py and urls.py and register the url direction of the new App module



Register the app and the url in the main application



Now you can see the view



## Models Data base

```
from django.db import models

# Create your models here.
class CategoryProduct(models.Model):
    Name = models.CharField(max_length=50)
    Created = models.DateTimeField(auto_now_add=True)
    Updated = models.DateTimeField(auto_now_add=True)

    class Meta: #<-- How will appear in the Panel Admin
        verbose_name = 'Category_Product'
        verbose_name_plural = 'Categories_Products'

    def __str__(self) -> str:
        return self.Name #<-- every call to the object
                           #gets the name of the category

class Product(models.Model):
    Name = models.CharField(max_length=22)
    Category = models.ForeignKey(CategoryProduct, on_delete=models.CASCADE)
    Image = models.ImageField(upload_to='StoreApp', null=True, blank=True)
    Price = models.FloatField()
    Availability = models.BooleanField(default=True)

    class Meta: #<-- How will appear in the Panel Admin
        verbose_name = 'Product'
        verbose_name_plural = 'Products'
```

In this part, create a folder called StoreApp and save it in media folder of the application

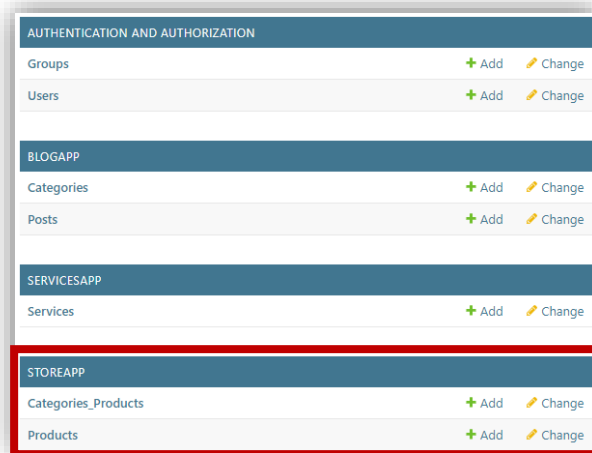
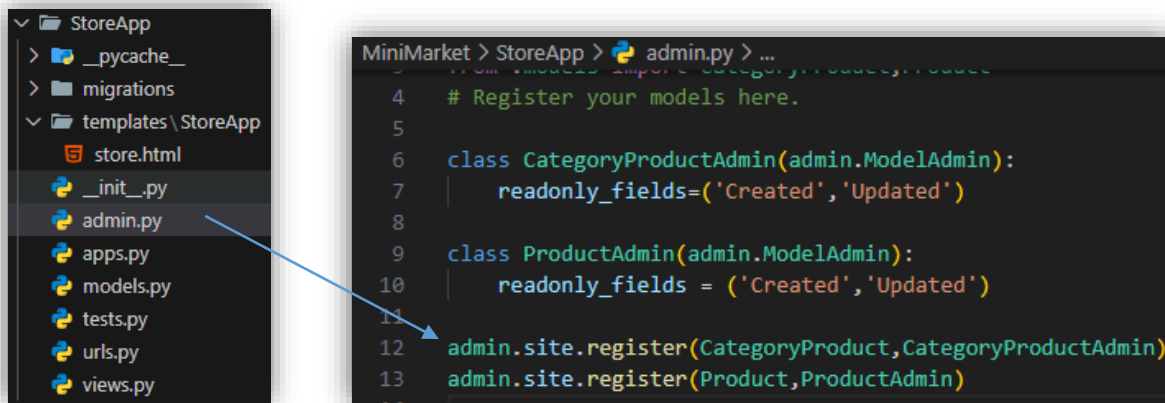
## Perform Migrations

```
C:\Users\Paul Manriquez\Desktop\Django\MiniMarketPaul_Django\MiniMarket>python manage.py makemigrations
Migrations for 'StoreApp':
  StoreApp/migrations/0001_initial.py
    - Create model CategoryProduct
    - Create model Product

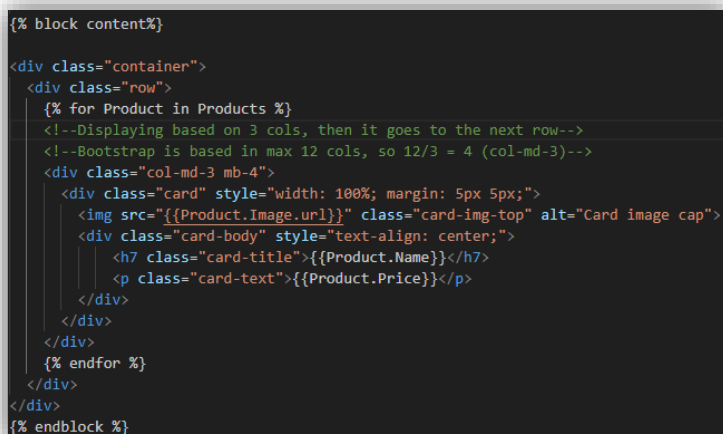
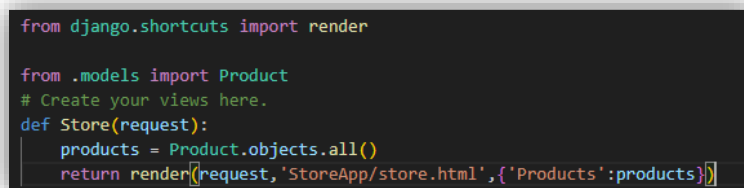
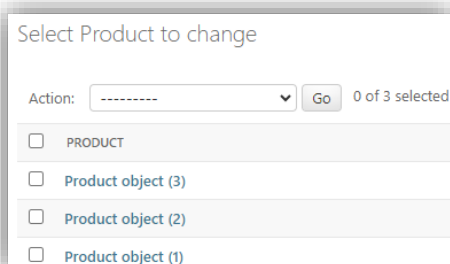
C:\Users\Paul Manriquez\Desktop\Django\MiniMarketPaul_Django\MiniMarket>python manage.py migrate
Operations to perform:
  Apply all migrations: BlogApp, ServicesApp, StoreApp, admin, auth, contenttypes, sessions
Running migrations:
  Applying ServicesApp.0002_alter_service_image... OK
  Applying StoreApp.0001_initial... OK
```

|                          |              |                                |
|--------------------------|--------------|--------------------------------|
| StoreApp_categoryproduct |              | CREATE TABLE "StoreApp_categor |
| id                       | integer      | "id" integer NOT NULL          |
| Name                     | varchar(50)  | "Name" varchar(50) NOT NULL    |
| Created                  | datetime     | "Created" datetime NOT NULL    |
| Updated                  | datetime     | "Updated" datetime NOT NULL    |
| StoreApp_product         |              | CREATE TABLE "StoreApp_product |
| id                       | integer      | "id" integer NOT NULL          |
| Name                     | varchar(22)  | "Name" varchar(22) NOT NULL    |
| Image                    | varchar(100) | "Image" varchar(100)           |
| Price                    | real         | "Price" real NOT NULL          |
| Availability             | bool         | "Availability" bool NOT NULL   |
| Category_id              | bigint       | "Category_id" bigint NOT NULL  |

## Register in the admin panel



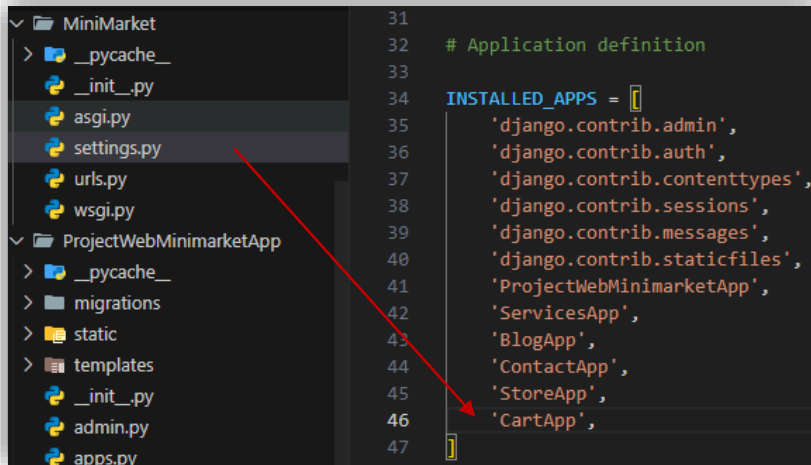
## Create some products and display in the template



## Cart App

Create the app, add to the apps of the main application and start to write the class functionality of the cart

```
C:\Users\Paul Manriquez\Desktop\Django\MiniMarketPaul_Django\MiniMarket>python manage.py startapp CartApp
```



| Concept       | Description   | Usage in `Cart` Class  |
|---------------|---|--|
| `request`     | Represents the HTTP request that the user made to the server. It contains metadata about the request. | The `request` object is passed to the `Cart` class constructor to access session data associated with the request. |
| HTTP Method   | Indicates the type of request (GET, POST, etc.).  | Not directly used in the `Cart` class, but can be used to determine the type of request in views.                  |
| GET/POST Data | The data sent by the user in the request, accessible through `request.GET` and `request.POST`.        | Not directly used in the `Cart` class, but used in views to handle form submissions and other data.                |
| Headers       | Metadata sent with the request, like user-agent, cookies, etc.  | Not directly used in the `Cart` class.   |
| User          | The user making the request, available through `request.user` if the user is authenticated.           | Not directly used in the `Cart` class.   |

|                   |   |  |
|-------------------|---|--|
| Session           | The session data associated with the request.   | Accessed through `request.session` to store and retrieve cart data.                                      |
| `request.session` | A dictionary-like object that allows you to store and retrieve arbitrary data on a per-site-visitor basis. This data is stored on the server. | Used to store and manage cart data across multiple requests.   |
| Authentication    | Storing user login status.  | Not directly used in the `Cart` class.   |
| User Preferences  | Remembering user settings.  | Not directly used in the `Cart` class.   |
| Shopping Carts    | Storing cart contents in an e-commerce application.   | The primary use of `request.session` in the `Cart` class.  |
| Methods in `Cart` |   |  |
| `__init__`        | Initializes the cart by accessing `request.session` and retrieving or creating the cart data.   | <code>`self.session = request.session`</code><br><code>`self.cart = self.session.get('cart', {})`</code> |
| `Add`             | Adds a product to the cart or increments the quantity if the product is already in the cart.  | <code>`self.cart[Product.id] = {...}`</code><br><code>`self.Save_Cart()`</code>                          |

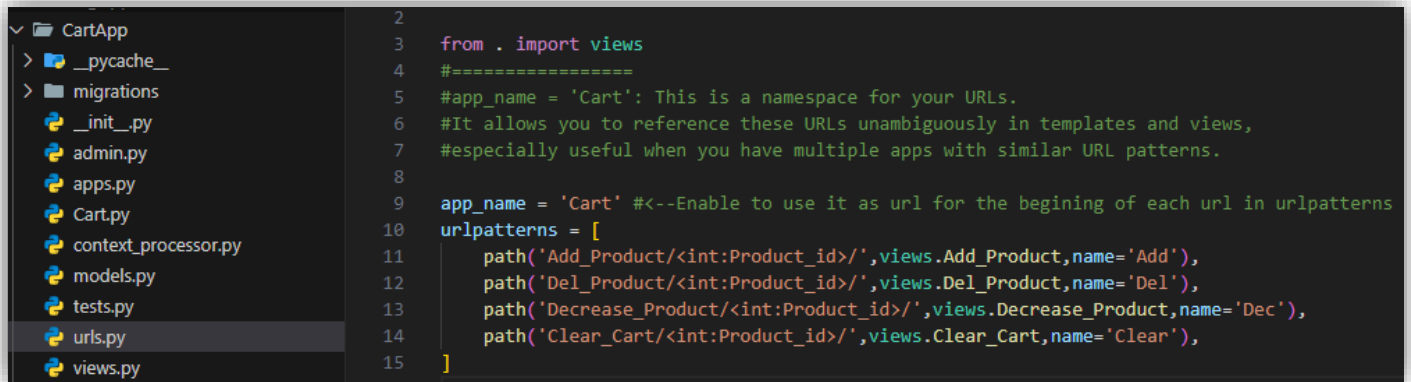
|                               |   |  |
|-------------------------------|---|--|
| <code>Save_Cart</code>        | Saves the current state of the cart to the session and marks the session as modified.   | <code>`self.session['cart'] = self.cart` &lt;br&gt;`self.session.modified = True`</code>   |
| <code>Delete_Product</code>   | Removes a product from the cart by its ID and updates the session.  | <code>`del self.cart[str(Product.id)]` &lt;br&gt;`self.Save_Cart()`</code>   |
| <code>Decrease_Product</code> | Decreases the quantity of a product in the cart, and if the quantity reaches zero, deletes the product from the cart and updates the session. | <code>`dic['Quantity'] -= 1` &lt;br&gt;`if dic['Quantity'] == 0:`<br/>`self.Delete_Product(Product)` &lt;br&gt;`self.Save_Cart()`</code> |
| <code>Clean_Cart</code>       | Clears the cart by setting it to an empty dictionary and marks the session as modified.   | <code>`self.session['cart'] = {}` &lt;br&gt;`self.session.modified = True`</code>  |

```
MiniMarket > CartApp > Cart.py > Add
1  #My cart Class
2  class Cart:
3      #Constructor
4      def __init__(self,request):
5          self.request=request          #<-- Get the request
6          self.session=request.session#<-- Get the current session
7          #===== Check if the cart exist in the session, otherwise create it
8          #This ensures that self.cart always references the current cart,
9          # whether it was just created or already existed in the session.
10         # The other methods in the class then use self.cart to access and modify the cart's contents.
11         cart = self.session.get('cart')
12         if not cart: #<-- if the cart dont exist in the session.get it creates a cart in the session
13             self.cart = self.session['cart'] = {} #<--- Creates a dictionary for each product
14             #Cart dictionary exist in both, in .cart and in session
15         else:
16             self.cart = cart #Save the content of the old cart
```

Create the views for the cart

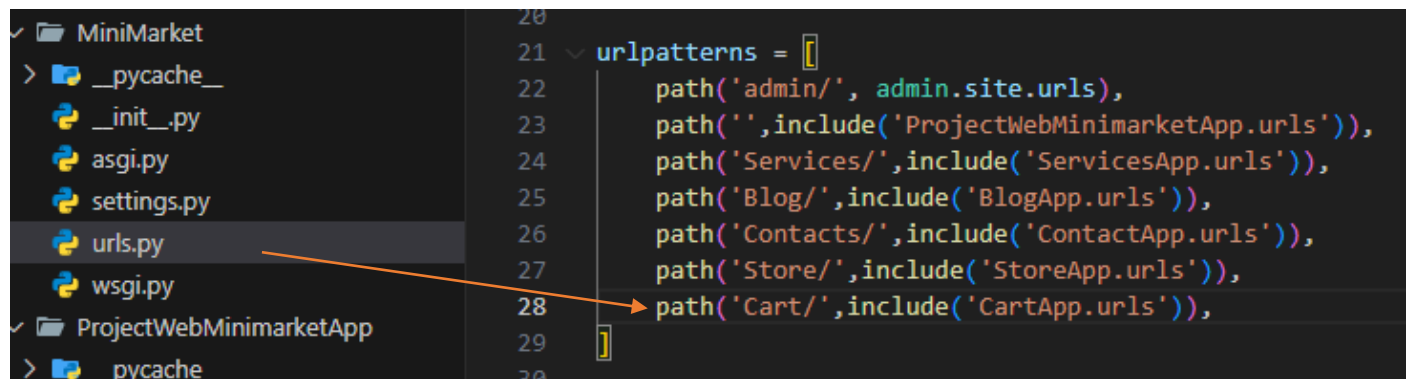
```
MiniMarket > CartApp > views.py > Clear_Cart
1  from django.shortcuts import render,redirect
2
3  from .Cart import Cart #<-- Class
4  from StoreApp.models import Product #<--Product object model
5  # Create your views here.
6
7
8  #===== Add a new product to cart
9  def Add_Product(request,Product_id):
10     MyCart = Cart(request) #Create an object cart
11
12     ProductAdd = Product.objects.get(id=Product_id) #Get the product with the id
13
14     if ProductAdd:
15         MyCart.Add(ProductAdd)
16
17     redirect('Store')
```

Add the urls for each endpoint



```
2
3 from . import views
4 #=====
5 #app_name = 'Cart': This is a namespace for your URLs.
6 #It allows you to reference these URLs unambiguously in templates and views,
7 #especially useful when you have multiple apps with similar URL patterns.
8
9 app_name = 'Cart' #<--Enable to use it as url for the begining of each url in urlpatterns
10 urlpatterns = [
11     path('Add_Product/<int:Product_id>/',views.Add_Product,name='Add'),
12     path('Del_Product/<int:Product_id>/',views.Del_Product,name='Del'),
13     path('Decrease_Product/<int:Product_id>/',views.Decrease_Product,name='Dec'),
14     path('Clear_Cart/<int:Product_id>/',views.Clear_Cart,name='Clear'),
15 ]
```

Register the urls for the cart app in the main application



```
20
21 urlpatterns = [
22     path('admin/', admin.site.urls),
23     path('',include('ProjectWebMinimarketApp.urls')),
24     path('Services/',include('ServicesApp.urls')),
25     path('Blog/',include('BlogApp.urls')),
26     path('Contacts/',include('ContactApp.urls')),
27     path('Store/',include('StoreApp.urls')),
28     path('Cart/',include('CartApp.urls')),
29 ]
```



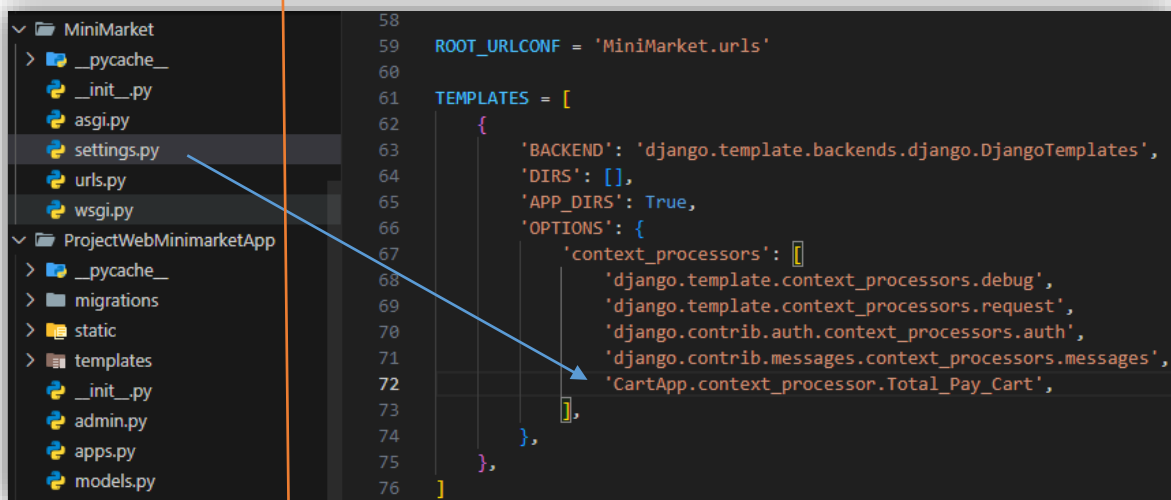
## Global Variable in Django (Total) - context processor

Create the file py (context\_processor.py) in thee CartApp

```
def Total_Pay_Cart(request):
    total = 12
    if request.user.is_authenticated:
        cart = request.session.get('cart', {})
        for key, product in cart.items():
            total += float(product['Price']) * product['Quantity']
    return {'Total_Pay': total}

#===== request.session['cart'] | is a dict of dicts
#key | Estands for each id of the product
# product | Product specification
```

Register the global variable and how to use it



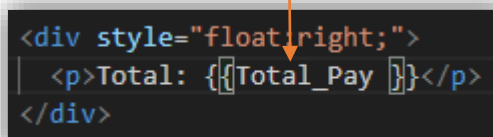
The screenshot shows a file explorer on the left with the following structure:

- MiniMarket
  - \_\_pycache\_\_
  - \_\_init\_\_.py
  - asgi.py
  - settings.py
  - urls.py
  - wsgi.py
- ProjectWebMinimarketApp
  - \_\_pycache\_\_
  - migrations
  - static
  - templates
  - \_\_init\_\_.py
  - admin.py
  - apps.py
  - models.py

The main editor shows the contents of `settings.py` (lines 58-76):

```
58
59 ROOT_URLCONF = 'MiniMarket.urls'
60
61 TEMPLATES = [
62     {
63         'BACKEND': 'django.template.backends.django.DjangoTemplates',
64         'DIRS': [],
65         'APP_DIRS': True,
66         'OPTIONS': {
67             'context_processors': [
68                 'django.template.context_processors.debug',
69                 'django.template.context_processors.request',
70                 'django.contrib.auth.context_processors.auth',
71                 'django.contrib.messages.context_processors.messages',
72                 'CartApp.context_processor.Total_Pay_Cart',
73             ],
74         },
75     },
76 ]
```

An orange arrow points from the `settings.py` file in the file explorer to the `context_processors` list in the code. A blue arrow points from the `'CartApp.context_processor.Total_Pay_Cart'` entry in the list to the `context_processors` key in the `OPTIONS` dictionary.



The screenshot shows a snippet of a Django template:

```
<div style="float:right;">
| <p>Total: {{Total_Pay }}</p>
</div>
```

An orange arrow points from the `settings.py` file in the file explorer to the `context_processors` list in the code. A blue arrow points from the `'CartApp.context_processor.Total_Pay_Cart'` entry in the list to the `context_processors` key in the `OPTIONS` dictionary.

Using the namespace in the store.html to refer to the endpoint of CartApp to add

```
StoreApp
├── __pycache__
├── migrations
├── templates
│   └── StoreApp
│       ├── Cart_app
│       │   ├── widget.html
│       │   └── store.html
│       └── ...
└── ...
```

```
<div class="card-text text-center" style="margin-bottom: 5px;">
  <a href="{% url 'Cart:Add' Product.id %}" class="btn btn-success">Add To Cart</a>
</div>
```

```
app_name = 'Cart' #<--Enable to use it as url for the begining of each url in urlpatterns
urlpatterns = [
    path('Add_Product/<int:Product_id>/',views.Add_Product,name='Add'),
```

## Widget and flow process

The widget to display the current cart is placed in store.html like this

```
<div style="float:right;">
  {% include 'StoreApp/Cart_app/widget.html'%}
</div>
```

Widget.html works with the objects **IN THE SESSION** to make the changes

CartApp/views.py

```
def Add_Product(request,Product_id):
    MyCart = Cart(request) #Create an object cart
    ProductAdd = Product.objects.get(id=Product_id) #Get the product with the id
    if ProductAdd:
        MyCart.Add(ProductAdd)
    return redirect('Store')
```



| Product         | Quantity | sub total |
|-----------------|----------|-----------|
| Boxing-Gloves   | 2        | - + 600.0 |
| Scissors        | 2        | - + 24.0  |
| Doll            | 1        | - + 13.0  |
| Doll Bear       | 3        | - + 150.0 |
| Total: \$ 774.0 |          |           |

Cart.py

```
def Cart(request):
    if not request.session.get('cart'):
        request.session['cart'] = {}
    return request.session['cart']

def Add_Product(request,Product_id):
    Product = Product.objects.get(id=Product_id)
    cart = Cart(request)
    cart[Product.id] = {
        'Product_id':Product.id,
        'Name':Product.Name,
        'Price':str(Product.Price),
        'Quantity':1,
        'Image':Product.Image.url,
        'SubTotal':str(Product.Price)
    }
    return cart
```

```
<!--Body Table-->
<tbody>
  <!--ALL THIS DATA COMES FROM Cart.py THAT CREATE A SESSION CART-->
  <!--Retrieve the data from the cart in SESSION-->
  {% if request.session.cart.items %}<!--If in the session exist items-->
  {% for key, product in request.session.cart.items %}<!--Pass the product to the template-->
    <!--//////////Row//////////-->
    <tr class="text-center">
      <td>{{product.Name}}</td>
      <td>{{product.Quantity}}</td>
      <td>
        <a href="{% url 'Cart:Dec' product.Product_id%}" class="btn btn-sm btn-success">-</a>
        <a href="{% url 'Cart:Add' product.Product_id%}" class="btn btn-sm btn-success">+</a>
        {{product.SubTotal}}
      </td>
    </tr>
  {% endfor %}
  {% else %}
    <tr>
      <td colspan="3">Your cart is empty</td>
    </tr>
  {% endif %}
</tbody>
```

We use the models of the product **to create the object** of the product

```
from .Cart import Cart #<-- Class
from StoreApp.models import Product #<--Product object model

# * Each endpoint is created with the Product_id that comes from store.html and widget.html
# * Create the object Cart and perform the respective operation
# if the product with the id exist in cart SESSION <----
# * Cart.py comes from CartApp that is the class that create the SESSION'cart'

# Create your views here.
#===== Add a new product to cart
def Add_Product(request,Product_id):
    MyCart = Cart(request) #Create an object cart

    ProductAdd = Product.objects.get(id=Product_id) #Get the product with the id

    if ProductAdd:
        MyCart.Add(ProductAdd)

    return redirect('Store')
```

The difference is that in the store.html we are displaying the products by the model and adding widget.html but the objects are not being created till the endpoint is being called in views.py in CartApp

Playing with the Class Cart.py

Created with the models of the product

```
<div style="float:right;">
    {% include 'StoreApp/Cart_app/widget.html'%}
</div>

<div class="row">
    {% for Product in Products %}
    <!--Displaying based on 3 cols, then it goes to the next row-->
    <!--Bootstrap is based in max 12 cols, so 12/3 = 4 (col-md-3)-->
    <div class="col-md-3 mb-4">
        <div class="card" style="width: 100%; margin: 5px 5px;">
            

            <div class="card-body" style="text-align: center;">
                <h7 class="card-title">{{Product.Name}}</h7>
                <p class="card-text">{{Product.Price}}</p>

            </div>

            <div class="card-text text-center" style="margin-bottom: 5px;">
                <a href="{% url 'Cart:Add' Product.id %}" class="btn btn-success">Add To Cart</a>
            </div>

        </div>
    </div>
```

This is playing with the endpoint and the parameter that is being passed is a field of the model Product

## Authentication App

```
C:\Users\Paul Manriquez\Desktop\Django\MiniMarketPaul_Django\MiniMarket>python manage.py startapp AuthenticationApp
```

Create the endpoint, the url, install the app and register in the urls of the main app

```
MiniMarket > AuthenticationApp > views.py > Authenticate
1  from django.shortcuts import render
2
3  # Create your views here.
4  def Authenticate(request):
5      return render(request, 'authenticationapp/register.html')

from django.urls import path

from . import views

urlpatterns = [
    path('', views.Authenticate, name='authenticate'),
]
```

```
> StoreApp
v MiniMarket
  > __pycache__
  > __init__.py
  > asgi.py
  > settings.py
  > urls.py
  > wsgi.py
v ProjectWebMinimarketApp
  > __pycache__
  > migrations
  > static
  > templates
34 INSTALLED_APPS = [
35     'django.contrib.admin',
36     'django.contrib.auth',
37     'django.contrib.contenttypes',
38     'django.contrib.sessions',
39     'django.contrib.messages',
40     'django.contrib.staticfiles',
41     'ProjectWebMinimarketApp',
42     'ServicesApp',
43     'BlogApp',
44     'ContactApp',
45     'StoreApp',
46     'CartApp',
47     'AuthenticationApp',
48 ]

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('ProjectWebMinimarketApp.urls')),
    path('Services/', include('ServicesApp.urls')),
    path('Blog/', include('BlogApp.urls')),
    path('Contacts/', include('ContactApp.urls')),
    path('Store/', include('StoreApp.urls')),
    path('Cart/', include('CartApp.urls')),
    path('Authentication/', include('AuthenticationApp.urls')),
]
```

Create the template folder of the app

```
v AuthenticationApp
  > migrations
  v templates\authenticationapp
    > register.html
```



