Mini Market Paul Django Project

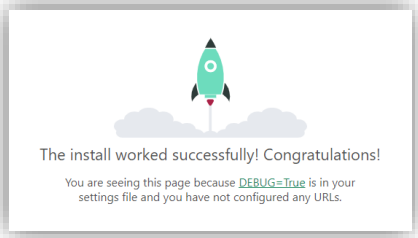
1.- Start the project

```
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>
```

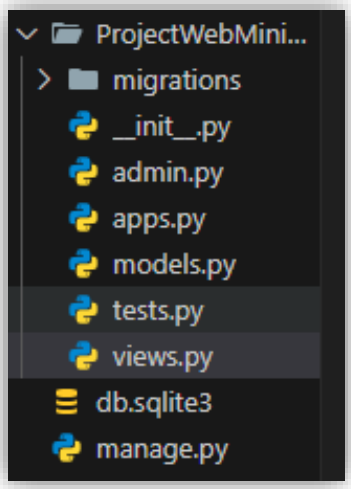
Start the main project

Main App Module

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



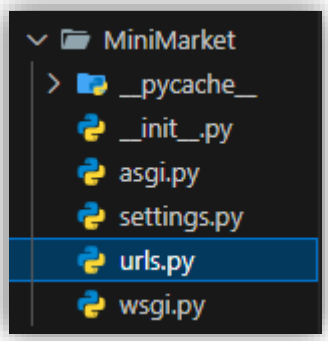
2.- Create your views and urls



views.py

MiniMarket > ProjectWebMinimarketApp > views.py > Contact

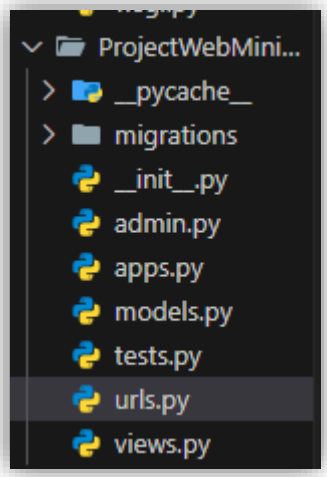
```python
from django.shortcuts import render,HttpResponse

# Create your views here.


def Home(request):
    return HttpResponse('Home')

def Services(request):
    return HttpResponse('Services')

def Store(request):
    return HttpResponse('Store')

def Blog(request):
    return HttpResponse('Blog')

def Contact(request):
    return HttpResponse('Contact')
```




```python
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

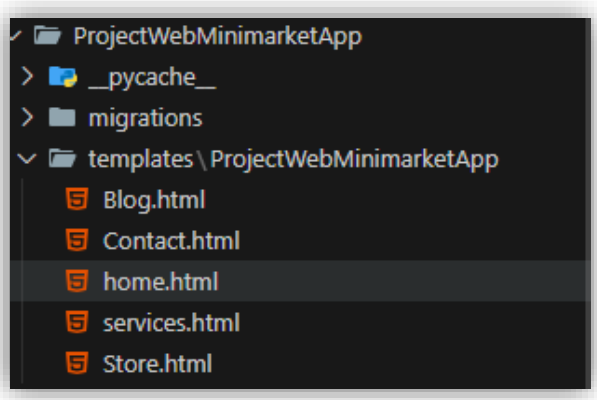



```python
from django.urls import path

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

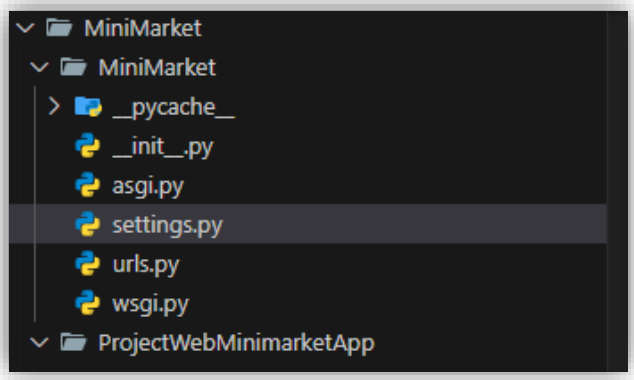
```python
from django.contrib import admin
from django.urls import path, include


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



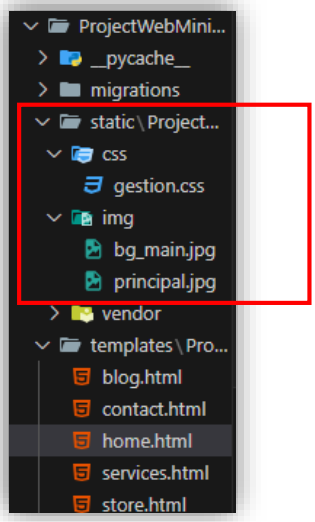

```python
def Home(request):
    return render(request,'ProjectWebMinimarketApp/home.html')
```

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




```python
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.

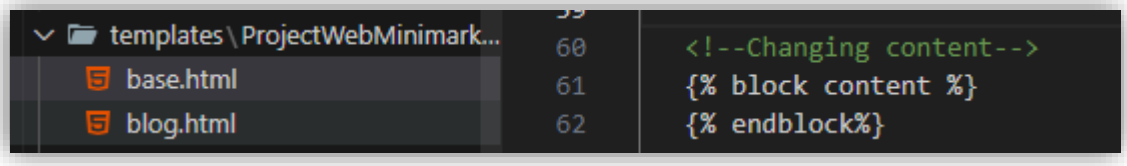


7.- Since the template in this project was already created, the goal on this project is to learn how to navigate and change the current project based on your needs. In this part we modify the project, added some style and create the base.html, that are the codes that we will use along all our webpage.

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
```

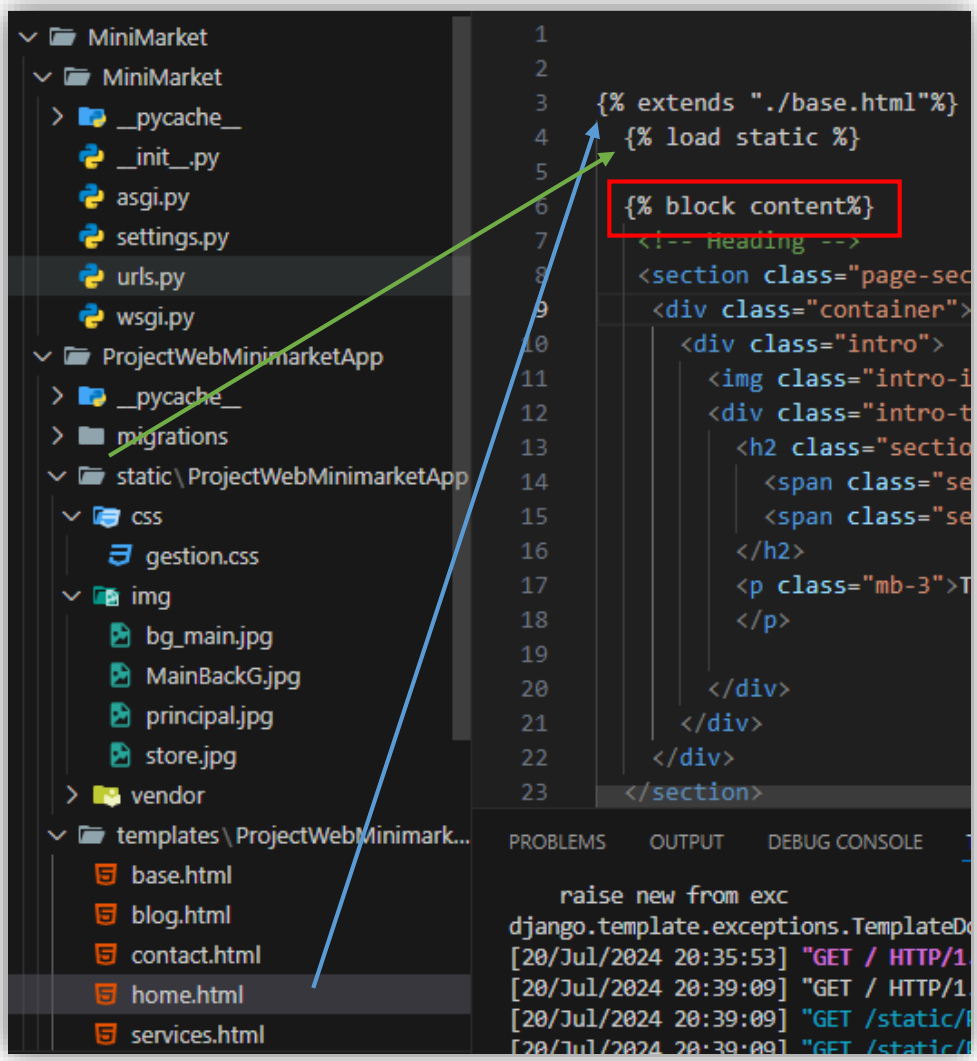
Create the base of the pages of this app module



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



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 > ...

```
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'),
]
```

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

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

  {% block Title %} Store {% endblock %}

  {% block content%}

  {% endblock %}
```

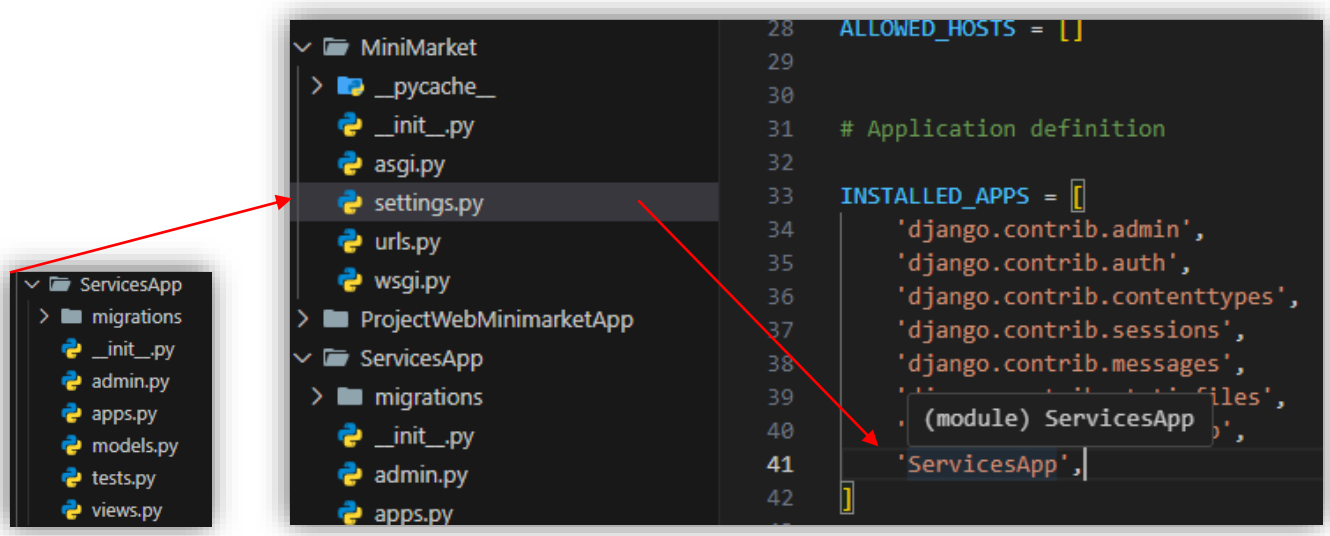
10.- Enable pointing to a url in the navbar if we are in that url

```
<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:

```python
from django.db import models

# Create your models here.
class Service(models.Model):
    Title = models.CharField(max_length=50)
    Content = models.CharField(max_length=50)
    Image = models.ImageField()
    Created = models.DateTimeField(auto_now_add=True)
    Updated = models.DateTimeField(auto_now_add=True)

    class Meta:
        verbose_name = 'Service'
        verbose_name_plural = 'Services'

    def __str__(self) -> str:
        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.

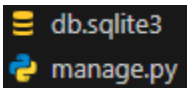
verbose_name_plural

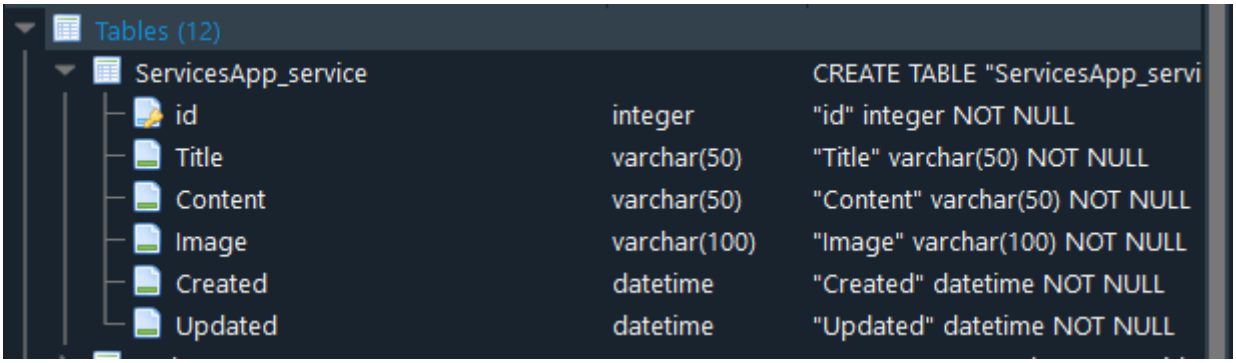
- **Definition**: verbose_name_plural = 'Services'
- **Purpose**: This defines a human-readable plural name for the model. Similar to verbose_name, but it is used when referring to multiple instances of the model. For example, in the Django admin interface, the section for this model would be labeled "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
`python manage.py makemigrations`	Creates new migration files based on changes in models.	Generates migration scripts for model changes.
`python manage.py migrate`	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
```

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

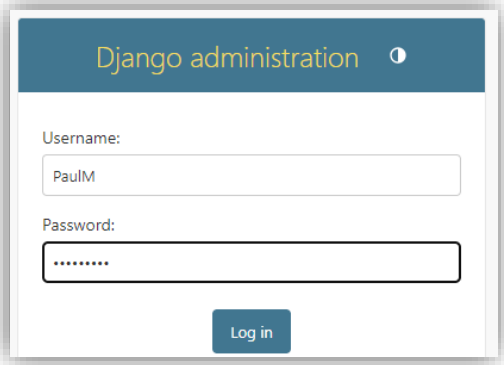


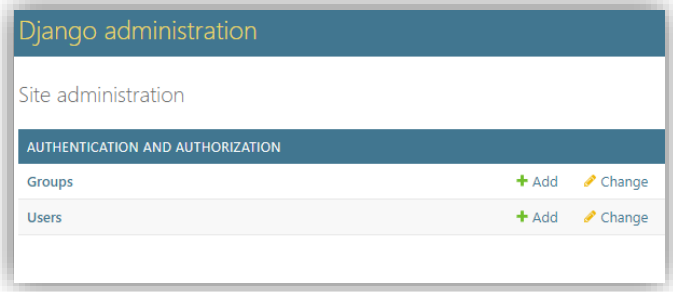
Register the new service in the Admin Panel

First, create a super user for this Project

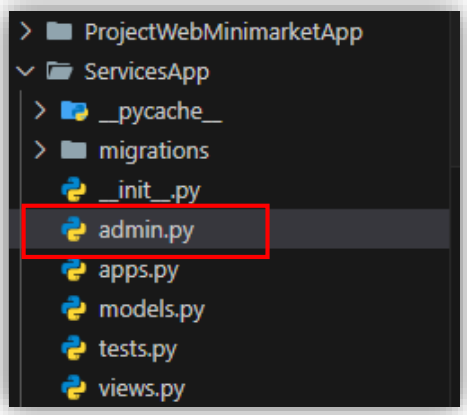
```
C:\Users\Paul Manriquez\Desktop\Django\MiniMarketPaul_Django\MiniMarket>python manage.py createsuperuser
Username (leave blank to use 'paulmanriquez'): PaulM
Email address: paulmanriquezengineer@gmail.com
Password:
Password (again):
Superuser created successfully.
```

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

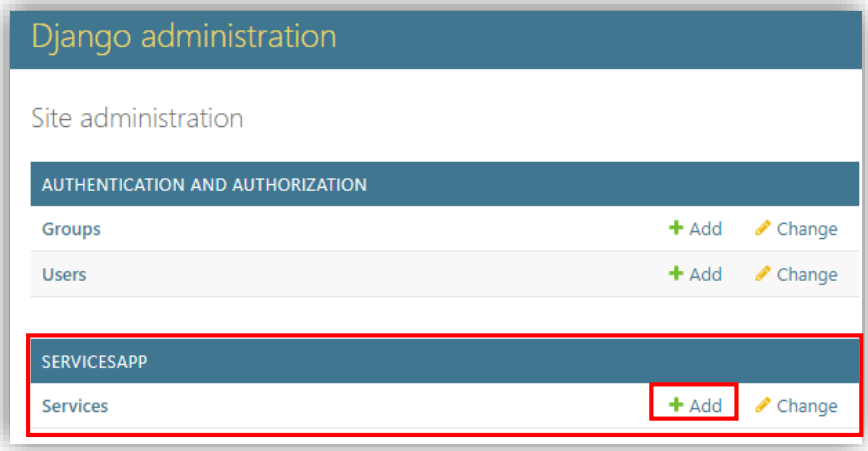


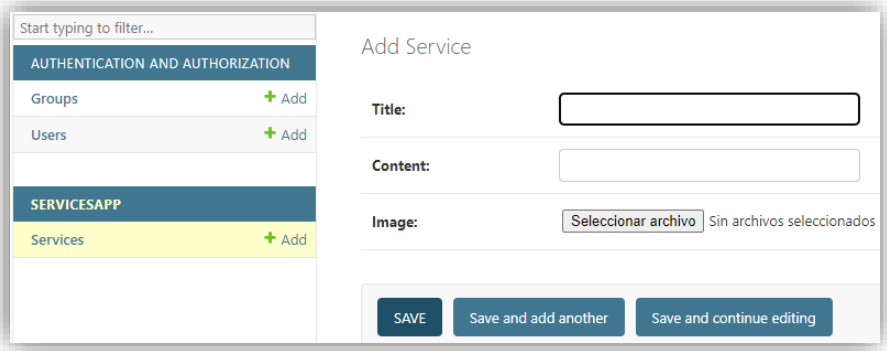


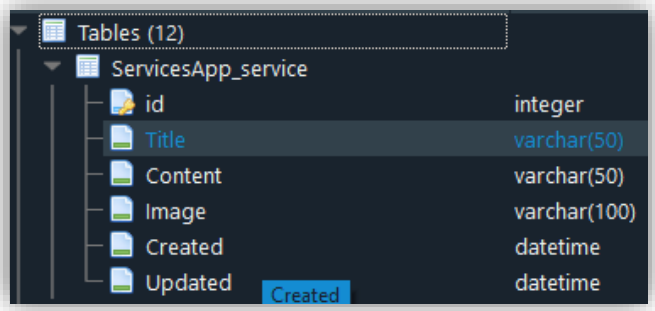
Now, in the admin file of the ServiceApp, we can add the new service as follows




```python
from django.contrib import admin

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

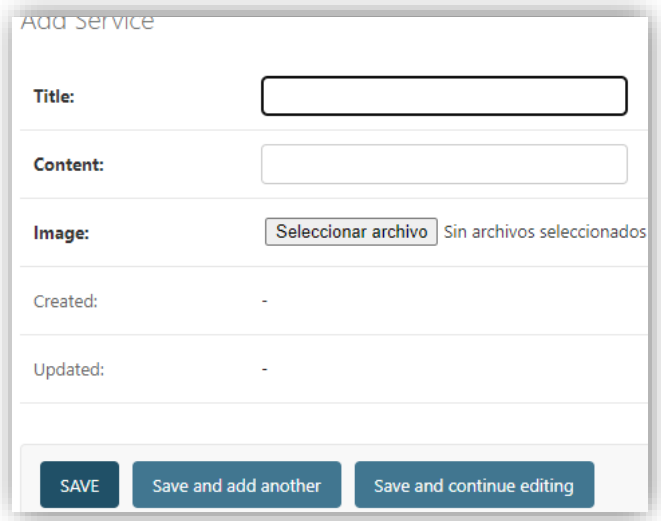




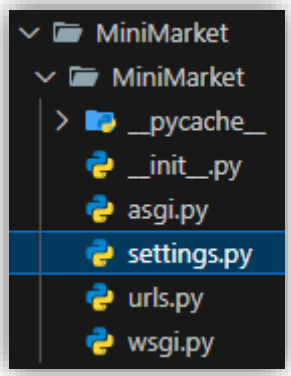


If you want to visualize Created and Updated field, Modify the admin.py file as follows:

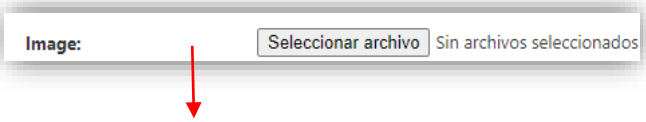
```python
from django.contrib import admin

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

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



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

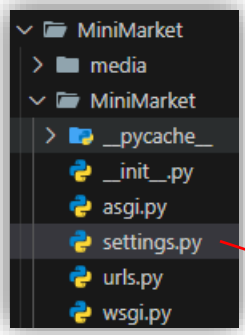



```python
LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True
```



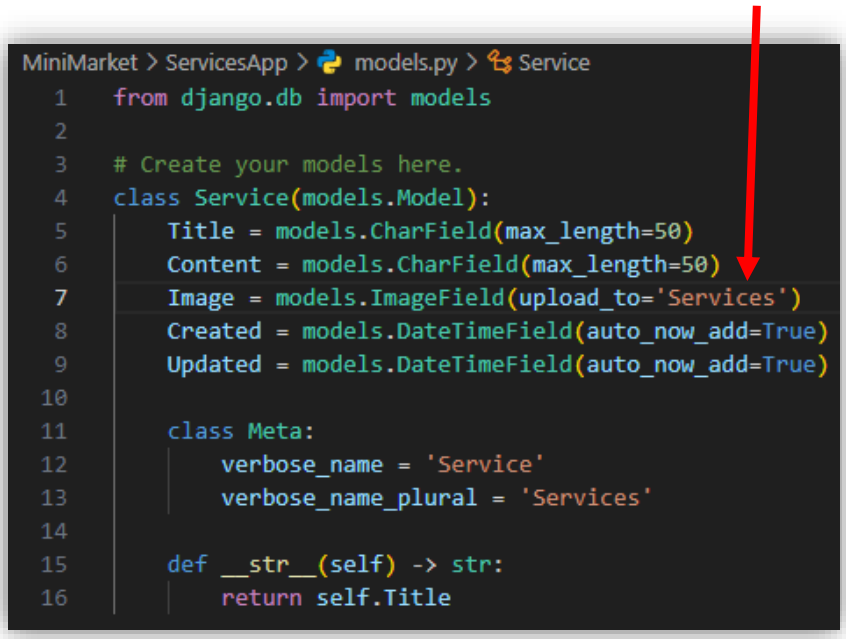
Since we don't configure where to store the files, they will be stored in the root of the project, thus, we need a special directory to store the media uploaded for each module, so, we need to configure as follows:

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

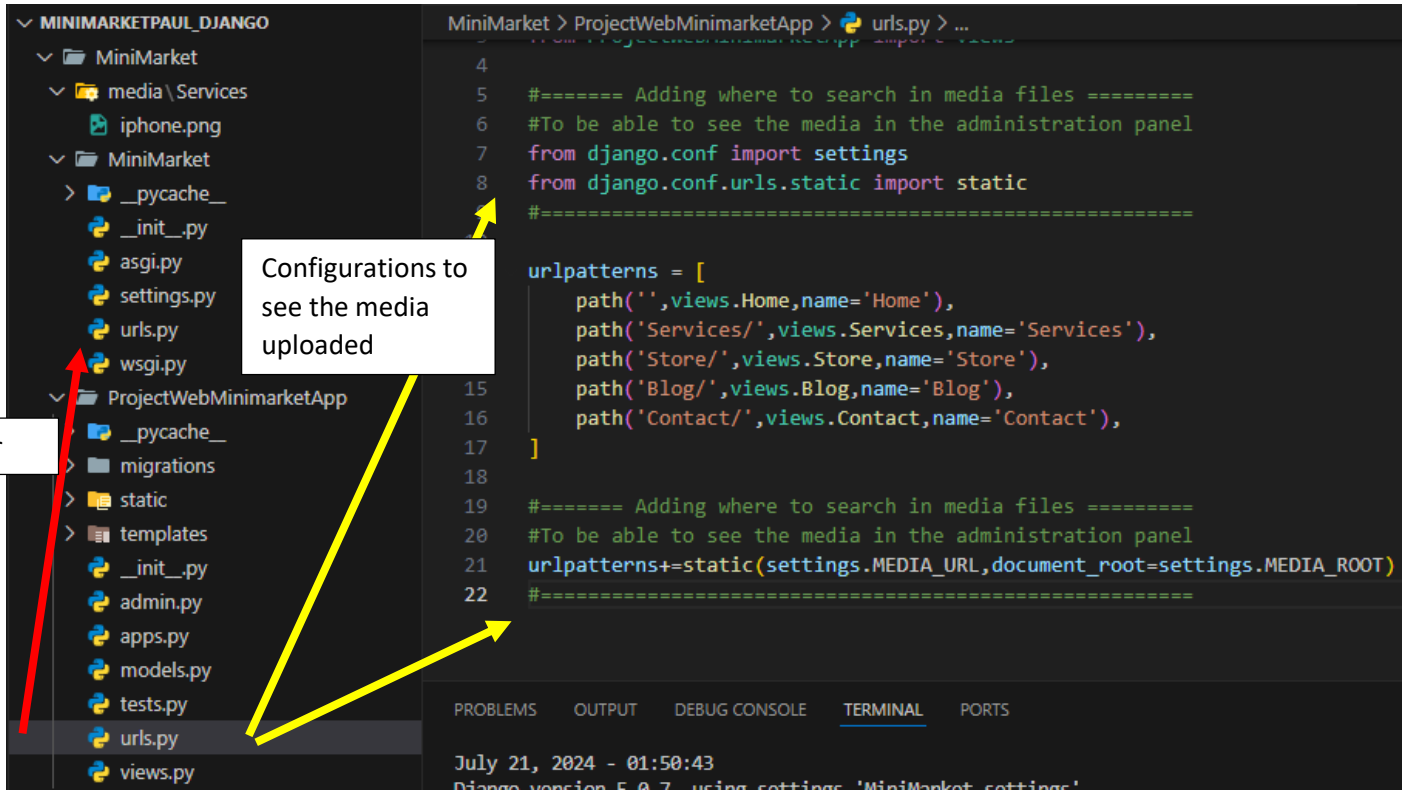



```python
STATIC_URL = 'static/'

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

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.



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:



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**

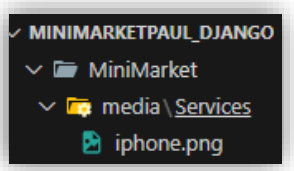
   python
   Copiar código
   urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)

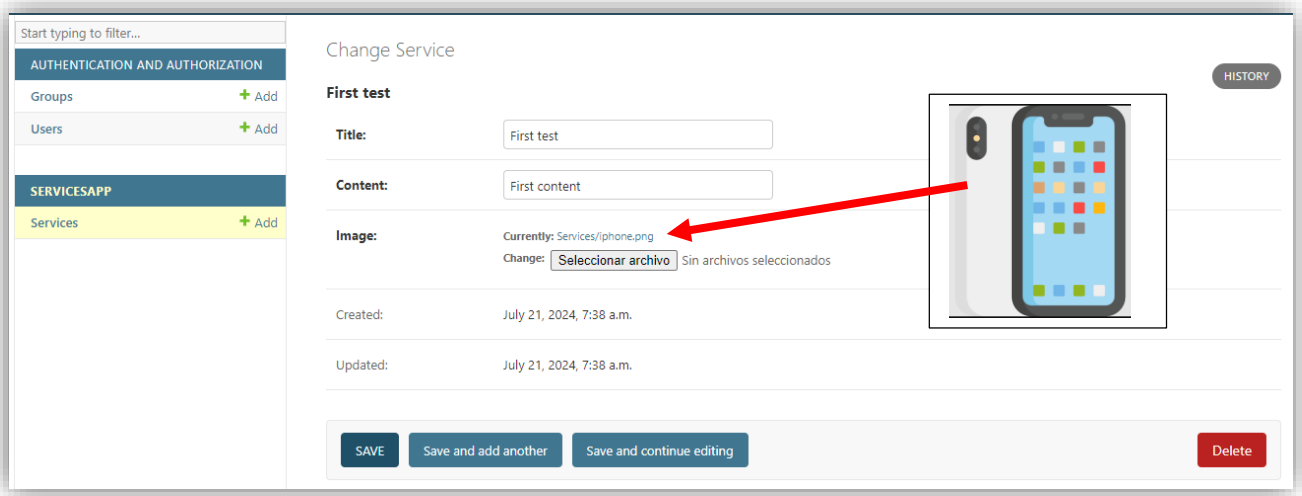
   - settings.MEDIA_URL: The URL prefix for serving media files (e.g., /media/).
   - settings.MEDIA_ROOT: The filesystem path where media files are stored (e.g., /path/to/media/).
   - static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT): This function appends a URL pattern to urlpatterns that 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.

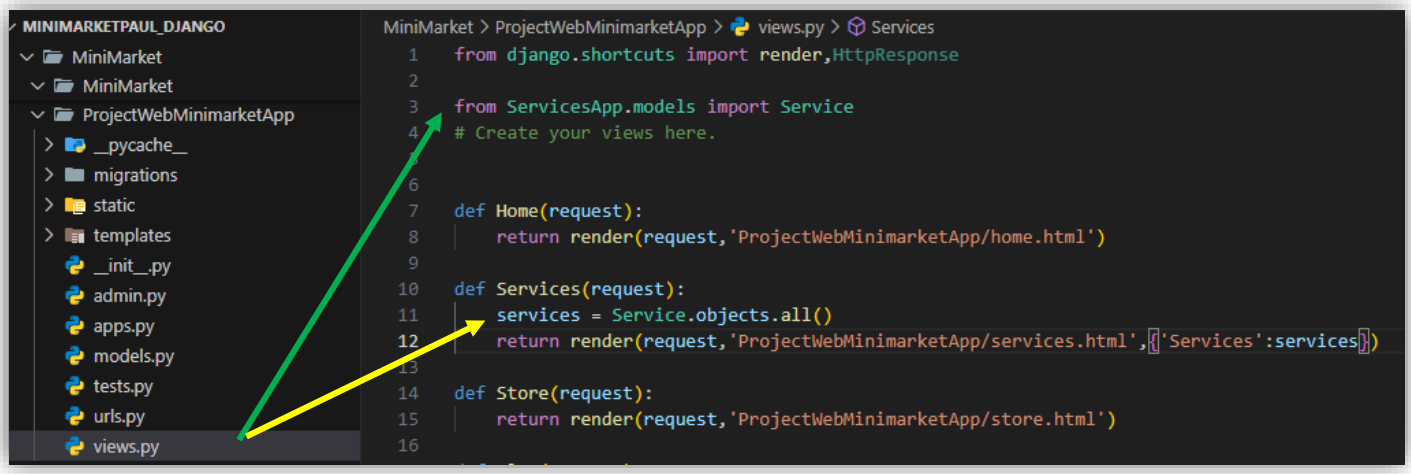
Now in the Panel administration, we add a new row in the Service data base and we can see that was created and uploaded correctly:





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.

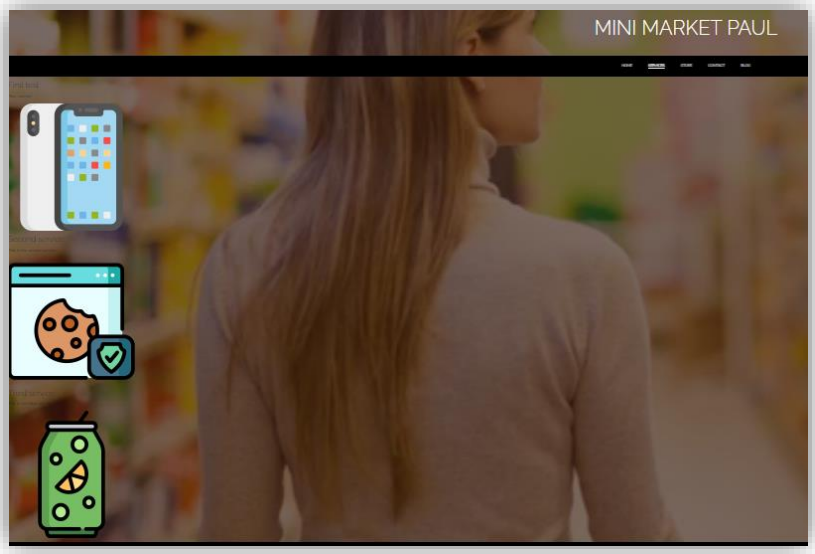


Table: ServicesApp_service

Filter in any co...

	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