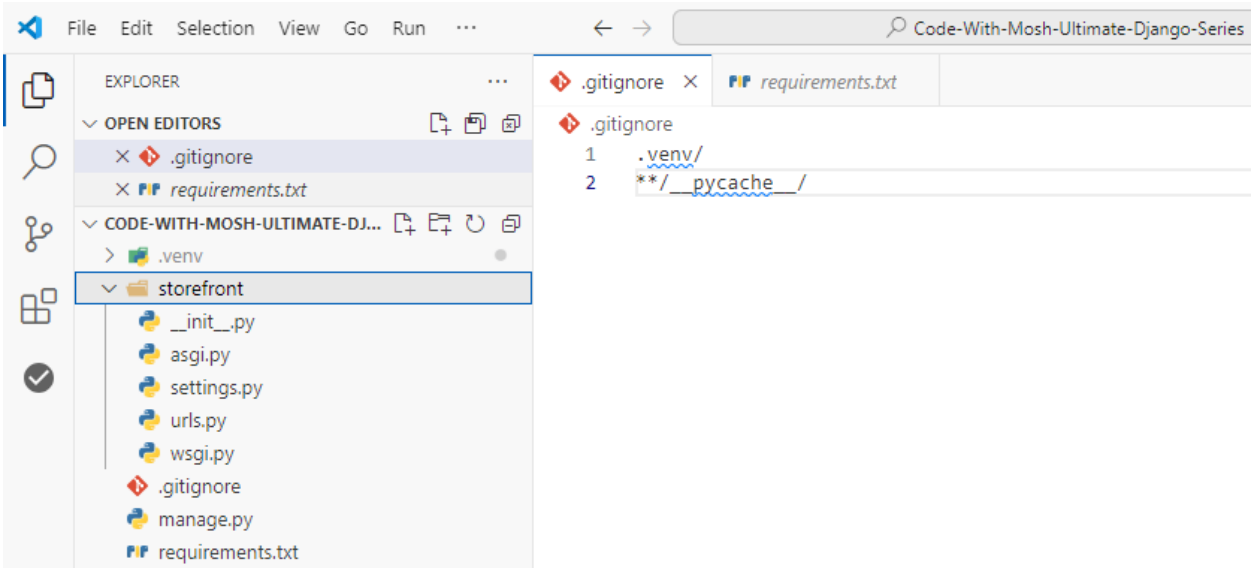
****

To Create New Project Inside The Current Directory:

- **Run Command**: *django-admin startproject storefront .*



****

The **apps.py** is The Config File For Our App.

****

The **views.py** Is The ***Request Handler*** For The App.

****

Note: To Install The App Inside settings.py We JUST NEED To Append The Name Of App To INSTALLED_APPS-Array.

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

    # ! This Line Added By Jafar Loka
    'playground'
]
```

****

A View Function Is: Function That Take Request And Return Response.

****

To Map View Function To Specific Point:

- Create **urls.py** inside the app-directory.
- Import The Necessary Modules.
- Create urlpatterns-array that contain list of URLs.

```
from django.urls import path;
from . import views;

#URLConf
urlpatterns = [
    path('hello/', views.say_hello),
];
```

****

To Reference The urls.py Of App From urls.py Of Project, We Can Use include Function:

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

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

****

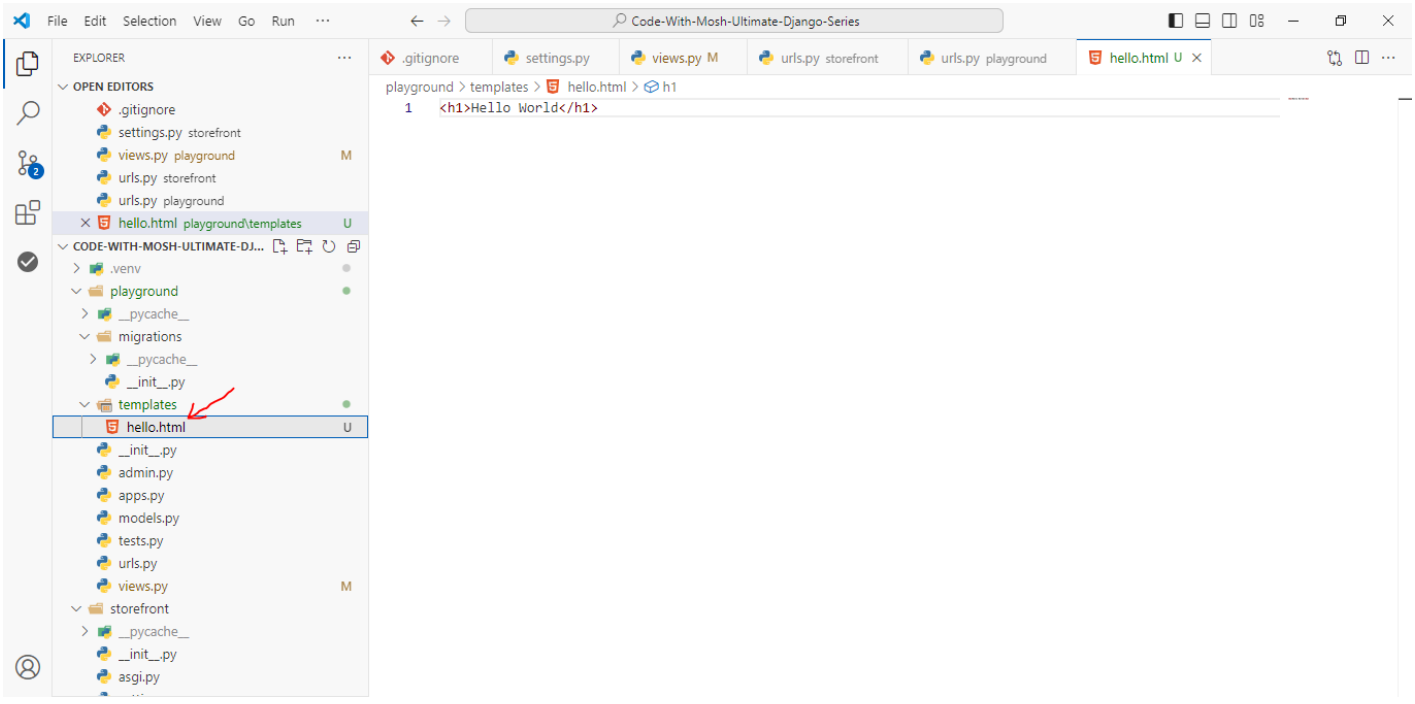
Note 1: We Must Be Attention When Using Slashes For urls.py Files Inside The App OR Project.

Note 2: We Must Always Set / To The End Of URL Pattern.

****

To Return HTML Code Instead Of Simple Response:

- Create New Folder Inside The App Directory Called: templates
- Create The File With Any Name, Ex: hello.html
- Then, Inside The View Function (Request handler Function), We Use render()-function.
    - Code: `return render(request, 'hello.html');`



****

To Pass Parameters To render-function:

```
return render(request, 'hello.html', { 'name': 'Jafar Loka'});
```

****

To Add Logic To HTML Code:

```
<!-- <h1>Hello World</h1> -->
{% if name %}
<h1>Hello {{ name }}</h1>
{% else %}
<h1>Hello World</h1>
{% endif %}
```

****

The auto_now Make That: Each Time We Update The Field The Value Of This Field Will Change.

```
class Product(models.Model):
    title = models.CharField(max_length=255);
    description = models.TextField();
    price = models.DecimalField(max_digits=6, decimal_places=2);
    inventory = models.IntegerField();
    last_update = models.DateTimeField(auto_now=True);
```

****

```
class Customer(models.Model):

    first_name = models.CharField(max_length=255);
    last_name  = models.CharField(max_length=255);
    email      = models.EmailField(unique=True);
    phone      = models.CharField(max_length=10);
    birth_date = models.DateField(null=True);
```

****

Django Will Create The Reverse Of Relationship And Set The Value.

Here We Set primary_key To Avoid The Problem Of One-To-Many Relationship.

```
class Address(models.Model):
    street = models.CharField(max_length=255);
    city   = models.CharField(max_length=255);

    customer = models.OneToOneField(to=Customer, on_delete=models.CASCADE,
primary_key=True);
```

****

```
class Order(models.Model):

    PAYMENT_STATUS_PENDING  = 'P';
    PAYMENT_STATUS_COMPLETE = 'C';
    PAYMENT_STATUS_FAILED   = 'F';

    PAYMENT_STATUS_CHOICES = [
        (PAYMENT_STATUS_PENDING, 'Pending'),
        (PAYMENT_STATUS_COMPLETE, 'Complete'),
        (PAYMENT_STATUS_FAILED, 'Failed'),
    ];

    placed_at = models.DateTimeField(auto_now_add=True);

    payment_status = models.CharField(max_length=1,
choices=PAYMENT_STATUS_CHOICES, default=PAYMENT_STATUS_PENDING);
**********************************************************************************************
```

To Define One-To-Many Relationship:

```
class Product(models.Model):
    title = models.CharField(max_length=255, unique=True);
    description = models.TextField();
    price = models.DecimalField(max_digits=6, decimal_places=2);
    inventory = models.IntegerField();
    last_update = models.DateTimeField(auto_now=True);

    collection = models.ForeignKey(to=Collection, on_delete=models.PROTECT)
**********************************************************************************************

class OrderItem(models.Model):
    order = models.ForeignKey(to=Order, on_delete=models.PROTECT);
    product = models.ForeignKey(to=Product, on_delete=models.PROTECT);
    quantity = models.PositiveSmallIntegerField();
    unit_price = models.DecimalField(max_digits=6, decimal_places=2)
**********************************************************************************************
```

Note (To Remember): When We Create *One To One Relationship* In Any Class, The Related Class Will Have Also Automatically The Reverse Relationship.

****

Note (To Remember): If We Can't Organize Classes That Represents Models; Then We Can Pass The Model Name To Foreign Key AS String.

```
collection = models.ForeignKey('Collection', on_delete=models.PROTECT)
```

****

```
class Promotion(models.Model):
    description = models.CharField(max_length=255);
    discount    = models.FloatField();

    # Also Here We Can Set Start And End Date For Each Promotion
```

****

Here We Define Many To Many Relationship For Product-Model.

The Reverse Relationship Inside The Promotion-Model, Will Be: product_set, To Change The Name We Use related_name:

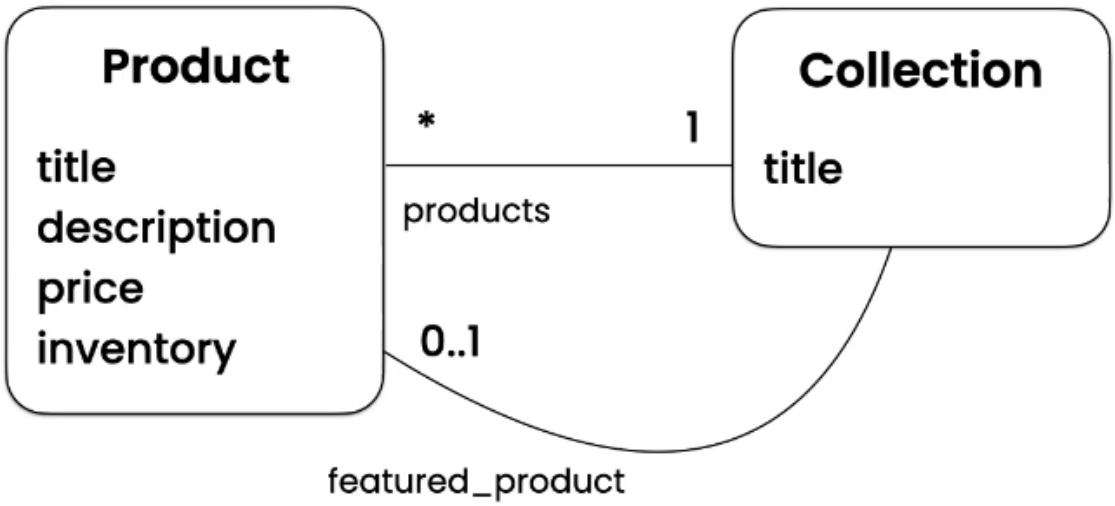
```
class Product(models.Model):
    title = models.CharField(max_length=255, unique=True);
    description = models.TextField();
    price = models.DecimalField(max_digits=6, decimal_places=2);
    inventory = models.IntegerField();
    last_update = models.DateTimeField(auto_now=True);

    collection = models.ForeignKey(to=Collection, on_delete=models.PROTECT);

    promotions = models.ManyToManyField(to=Promotion, related_name='products');
```

****

Each Class Depends On Other:



CIRCULAR DEPENDENCY

```
(.venv) G:\Web\Django\Code-With-Mosh-Ultimate-Django-Series>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

Exception in thread django-main-thread:
Traceback (most recent call last):
  File "C:\Program Files\Python311\Lib\threading.py", line 1038, in _bootstrap_inner
    self.run()
  File "C:\Program Files\Python311\Lib\threading.py", line 975, in run
    self._target(*self._args, **self._kwargs)
  File "G:\Web\Django\Code-With-Mosh-Ultimate-Django-Series\.venv\Lib\site-packages\django\utils\autoreload.py", line 64, in wrapper
    fn(*args, **kwargs)
  File "G:\Web\Django\Code-With-Mosh-Ultimate-Django-Series\.venv\Lib\site-packages\django\core\management\commands\runserver.py", line 134, in inner_run
    self.check(display_num_errors=True)
  File "G:\Web\Django\Code-With-Mosh-Ultimate-Django-Series\.venv\Lib\site-packages\django\core\management\base.py", line 563, in check
    raise SystemCheckError(msg)
django.core.management.base.SystemCheckError: SystemCheckError: System check identified some issues:

ERRORS:
store.Collection.featured_product: (fields.E303) Reverse query name for 'store.Collection.featured_product' clashes with field name 'store.Product.collection'.
        HINT: Rename field 'store.Product.collection', or add/change a related_name argument to the definition for field 'store.Collection.featured_product'.
```

****

To Solve The Above Problem, We Must Discard The Reverse Relationship That Django Automatically Created:

Note: When We Set + For related_name We Discard The Reverse Relationship For Product-Model.

```
featured_product = models.ForeignKey(
      to='Product', on_delete=models.SET_NULL, null=True, related_name='+'
   )
```

****

When We Use ContentTypes We Can Build Generic Relationships Between Classes.

From INSTALLED_APPS In setings.py:

```
INSTALLED_APPS = [

    'django.contrib.contenttypes',

]
```

****

In This Way We Can Define Generic Relationship Between Models:

ContentType is The Class That Represent What We Connect.

Content_object is The Object That We Want To Check.

The Limited Of This Implementation If The Id Of Related Model Is Not Integer, Like In NoSQL.

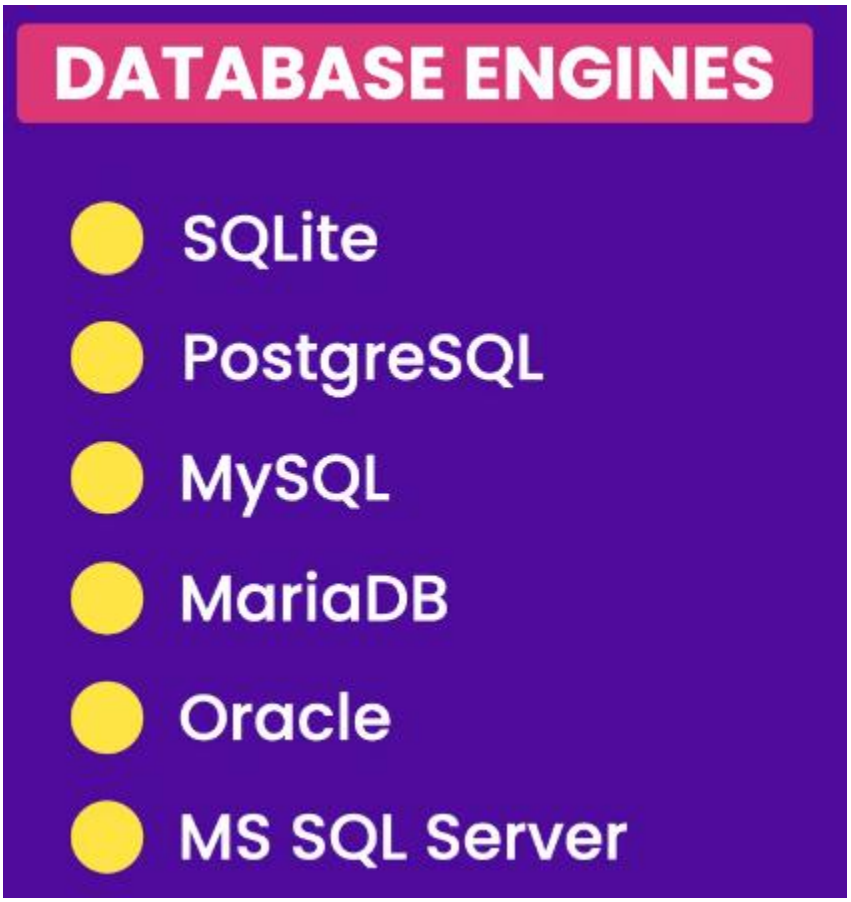
```
from django.contrib.contenttypes.models import ContentType;
from django.contrib.contenttypes.fields import GenericForeignKey;

class Tag(models.Model):
    label = models.CharField(max_length=255);

class TaggedItem(models.Model):
    tag = models.ForeignKey(to=Tag, on_delete=models.CASCADE);
    content_type = models.ForeignKey(to=ContentType, on_delete=models.CASCADE);
    object_id = models.PositiveIntegerField();
    content_object = GenericForeignKey();
```

****

For MS-SQL We Need Another Extension To Support:



****

To Create The Migrations For Our Django Project:

- Set The Apps Inside The *INSTALLED_APPS*
- Run Command: *python manage.py makemigrations*

```
(.venv) G:\Web\Django\Code-With-Mosh-Ultimate-Django-Series>python manage.py makemigrations
Migrations for 'store':
  store\migrations\0001_initial.py
    + Create model Customer
    + Create model Cart
    + Create model Collection
    + Create model Promotion
    + Create model Address
    + Create model Order
    + Create model Product
    + Create model OrderItem
    + Add field featured_product to collection
    + Create model CartItem
Migrations for 'likes':
  likes\migrations\0001_initial.py
    + Create model LikedItem
Migrations for 'tags':
  tags\migrations\0001_initial.py
    + Create model Tag
    + Create model TaggedItem

(.venv) G:\Web\Django\Code-With-Mosh-Ultimate-Django-Series>
```

****