pip install pipenv

pipenv install django

(it will install Django inside virtual environment)

pipenv shell

(it will activate the virtual environment)

Use django-admin to execute commands (it will show the list of commands)

django-admin startproject storefront

django-admin startproject storefront . (use current dir as project dir hence not duplicate)

manage.py is a wrapper around django-admin and takes the settings of application into account

using python manage.py will show the same commands as django-admin

python manage.py runserver

python manage.py runserver 7000 (change port number)

In settings.py : every time we create an app we should register it here :

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'playground',

]

python manage.py startapp playground

view is a request handler in Django :

views.py :

from django.shortcuts import render

from django.http import HttpResponse

**def** say\_hello(request):

    return HttpResponse('Hello World ')

urls.py : in here we create routs that must be end with / . note that Django search for urlpatterns variable :

we use path function to create a urlpattern object

from django.urls import path

from . import views

*# URLConf*

urlpatterns = [

    path('hello/', views.say\_hello)

]

In main project inside urls.py we should add route with include function that gets the application urls.py file

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

    path('admin/', admin.site.urls),

    path('playground/', include('playground.urls'))

]

We make templates folder and add html file inside then in views.py we call render function

from django.shortcuts import render

from django.http import HttpResponse

**def** say\_hello(request):

*# return HttpResponse('Hello World ')*

    return render(request, 'hello.html')

we can send parameter to template to show with context parameter of render function inside views.py

**def** say\_hello(request):

*# return HttpResponse('Hello World ')*

    return render(request, 'hello.html', {'name': 'Mosh'})

inside template

<h1>Hello World {{ name }}</h1>

We can use logic inside of template like this :

{% if name %}

<h1>Hello  {{ name }}</h1>

{% else %}

<h1>Hello World</h1>

{%% endif %}

pipenv install django-debug-toolbar

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'playground',

    'debug\_toolbar'

]

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

    path('admin/', admin.site.urls),

    path('playground/', include('playground.urls')),

    path('\_\_debug\_\_/', include('debug\_toolbar.urls')),

]

MIDDLEWARE = [

    "debug\_toolbar.middleware.DebugToolbarMiddleware",

    'django.middleware.security.SecurityMiddleware',

    'django.contrib.sessions.middleware.SessionMiddleware',

    'django.middleware.common.CommonMiddleware',

    'django.middleware.csrf.CsrfViewMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddleware',

    'django.contrib.messages.middleware.MessageMiddleware',

    'django.middleware.clickjacking.XFrameOptionsMiddleware',

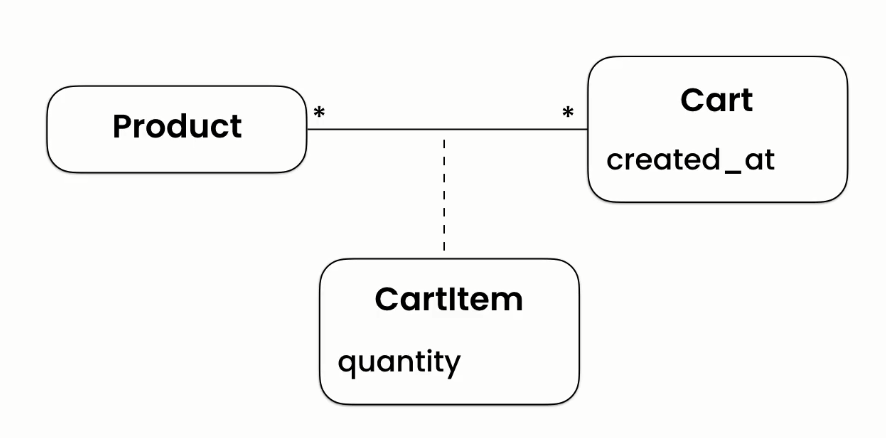
]

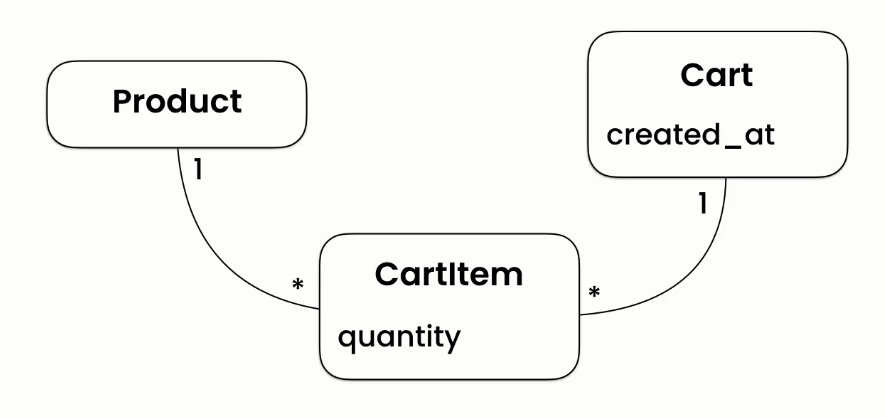
INTERNAL\_IPS = [

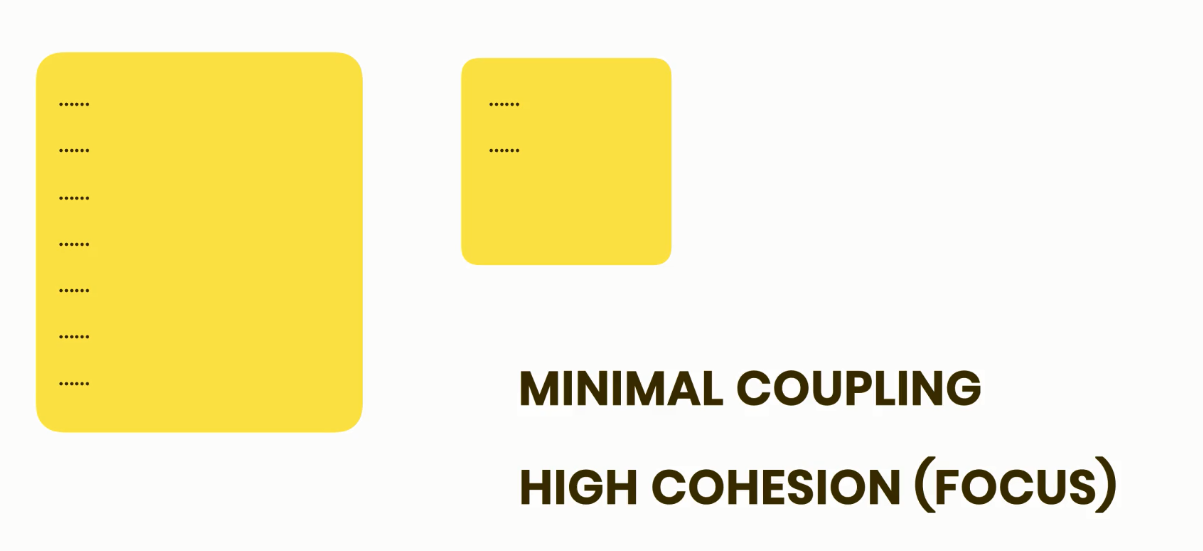
    "127.0.0.1"

]

The association class is between 2 many to many relationship and we can break it into 2 one to many relationship







In the models.py (or models module) , we create a model class that inherits from Model class in django.db. (we have imported the models module from django.db package) :

from django.db import models

*# Create your models here.*

**class** Product(models.Model):

django field types : in the model field reference website we can see all the built-in field types :

<https://docs.djangoproject.com/en/4.2/ref/models/fields/>

Django will create the id field automatically but if we want to do it manually we will set the parameter primary\_key=True

from django.db import models

*# Create your models here.*

**class** Product(models.Model):

     title = models.CharField(max\_length=255)

     description = models.TextField()    *# long text*

*# 9999.99 consider is the max price value*

     price = models.DecimalField(max\_digits=6, decimal\_places=2)

*# float will round the values we should use decimal*

     inventory = models.IntegerField()

     last\_update = models.DateTimeField(auto\_now=True)

*# every time we update a product object django will update this field*

**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=255)

     birth\_date = models.DateField(null=True)

with the field option choices we can specify which values can be possible for the field and do it with a list of tuples consisting values and human reading name (membership field)

**class** Customer(models.Model):

     MEMBERSHIP\_BRONZE = 'B'

     MEMBERSHIP\_Silver = 'S'

     MEMBERSHIP\_Gold = 'G'

     MEMBERSHIP\_CHOICES = [

          (MEMBERSHIP\_BRONZE, 'Bronze'),

          (MEMBERSHIP\_Silver, 'Silver'),

          (MEMBERSHIP\_Gold, 'Gold'),

     ]

     first\_name = models.CharField(max\_length=255)

     last\_name = models.CharField(max\_length=255)

     email = models.EmailField(unique=True)

     phone = models.CharField(max\_length=255)

     birth\_date = models.DateField(null=True)

     membership = models.CharField(max\_length=1, choices=MEMBERSHIP\_CHOICES, default=MEMBERSHIP\_BRONZE, )

**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\_statuses = models.CharField(max\_length=1, choices=PAYMENT\_STATUS\_CHOICES, default=PAYMENT\_STATUS\_PENDING)

Implementing one to one relationship :

**class** Address(models.Model):

     street = models.CharField(max\_length=255)

     city = models.CharField(max\_length=255)

     customer = models.OneToOneField(Customer, on\_delete=models.CASCADE )      *# the address is gonna get deleted*

     customer = models.OneToOneField(Customer, on\_delete=models.SET\_NULL )

*# the address is not gonna get deleted and customer column will be null but here it can not be*

     customer = models.OneToOneField(Customer, on\_delete=models.SET\_DEFAULT )    *# the address is not gonna get deleted and customer column will beset to default value*

     customer = models.OneToOneField(Customer, on\_delete=models.PROTECT )

*# the address is not gonna get deleted and customer column will beset to default value*

We should set the parameter primary\_key = True because otherwise Django will create the id primary key and it will change to one to many relationship

We don’t need to define the reverse relationship in customer class

**class** Address(models.Model):

     street = models.CharField(max\_length=255)

     city = models.CharField(max\_length=255)

     customer = models.OneToOneField(Customer, on\_delete=models.CASCADE, primary\_key= True )      *# the address is gonna get deleted*

The **one to many relation** ship will be implemented like this :

**class** Address(models.Model):

     street = models.CharField(max\_length=255)

     city = models.CharField(max\_length=255)

     customer = models.ForeignKey(Customer, on\_delete=models.CASCADE )

we create the collection class before product class in order to be able to reference it in product class otherwise we can reference it by string

**class** Cart(models.Model):

     created\_at = models.DateTimeField(auto\_now\_add=True)

**class** CartItem(models.Model):

     cart = models.ForeignKey(Cart, on\_delete=models.CASCADE)

     Product = models.ForeignKey(Product, on\_delete=models.CASCADE)

     quantity = models.PositiveSmallIntegerField()

**Many to Many relationship**

For implementng many to many we do like this : (default property will be product\_set but we can rename it in related\_name

**class** Product(models.Model):

     title = models.CharField(max\_length=255)

     description = models.TextField()    *# long text*

*# 9999.99 consider is the max price value*

     price = models.DecimalField(max\_digits=6, decimal\_places=2)    *# float will round the values we should use decimal*

     inventory = models.IntegerField()

     last\_update = models.DateTimeField(auto\_now=True)   *# every time we update a product object django will update this field*

     collection = models.ForeignKey(Collection, on\_delete=models.PROTECT)

     promotions = models.ManyToManyField(Promotion, related\_name= "products")  *# jango will create products property in promotion class*

**class** Promotion(models.Model):

     Description = models.CharField(max\_length=255)

     discount = models.FloatField()

*#product\_set*

**Circular dependency**

In this case both Collection and Product are dependent to each other

‘Product’ is entered with string because it is not recognized because Collection is defined before Product hen ce it has been entered with string

**class** Collection(models.Model):

     title = models.CharField(max\_length=255)

     featured\_product = models.ForeignKey('Product', on\_delete=models.SET\_NULL, null=True, related\_name='+')

*# jango cannot create the reverse realtionship because we have collection in product*

*# + means don't create reverse relation*

**class** Product(models.Model):

     title = models.CharField(max\_length=255)

     description = models.TextField()    *# long text*

*# 9999.99 consider is the max price value*

     price = models.DecimalField(max\_digits=6, decimal\_places=2)    *# float will round the values we should use decimal*

     inventory = models.IntegerField()

     last\_update = models.DateTimeField(auto\_now=True)   *# every time we update a product object django will update this field*

     collection = models.ForeignKey(Collection, on\_delete=models.PROTECT)

     promotions = models.ManyToManyField(Promotion, related\_name= "products")  *# jango will create products property in promotion class*

There is an abstract model called **ContentType** that comes with django and it is installed in setting.py. by using it we can create generic relationships between our models.

ContentType represent type of object in our application

And also there is an object called **GenericForeignKey** that holds the actual object that we refer

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'playground',

    'debug\_toolbar',

    'store',

    'tags'

]

from django.db import models

from django.contrib.contenttypes.models import ContentType

from django.contrib.contenttypes.fields import GenericForeignKey

*# Create your models here.*

**class** Tag(models.Model):

    label = models.CharField(max\_length=255)

**class** TaggedItem(models.Model):

    tag = models.ForeignKey(Tag, on\_delete=models.CASCADE)

*# Type (product, video, article)*

*# ID*

    content\_type = models.ForeignKey(ContentType, on\_delete=models.CASCADE)

*# if object type is removed we wantto remove all associated tags*

    object\_id = models.PositiveIntegerField()

    content\_object = GenericForeignKey()

*# using this field we can read an acutual object that particular tag is applied to*

Django has User object that can be used in models

from django.db import models

from django.contrib.auth.models import User

from django.contrib.contenttypes.models import ContentType

from django.contrib.contenttypes.fields import GenericForeignKey

*# Create your models here.*

**class** LikedItem(models.Model):

    user = models.ForeignKey(User, on\_delete=models.CASCADE)

    content\_type = models.ForeignKey(ContentType, on\_delete=models.CASCADE)

    object\_id = models.PositiveIntegerField()

    content\_object = GenericForeignKey()

**Creating migrations**

python manage.py makemigrations

django will create new migration file for all the installed apps

it will create migration files inside each app within migrations folder

if we change the name of a migration file we should change the name in all the places that it is referred.

**Slug**

A slug can contains only numbers hyphens and letters and it is used in urls for the purpose of search engine optimization and make it easier for web search engine to find out content

**class** Product(models.Model):

     title = models.CharField(max\_length=255)

**slug = models.SlugField()**

     description = models.TextField()    *# long text*

*# 9999.99 consider is the max price value*

     unit\_price = models.DecimalField(max\_digits=6, decimal\_places=2)    *# float will round the values we should use decimal*

     inventory = models.IntegerField()

     last\_update = models.DateTimeField(auto\_now=True)   *# every time we update a product object django will update this field*

     collection = models.ForeignKey(Collection, on\_delete=models.PROTECT)

     promotions = models.ManyToManyField(Promotion, related\_name= "products")  *# jango will create products property in promotion class*

after migrations django ask us to prepare a default value for Slug field because it is not nullable so we should make a default value or make it nullable

slug = models.SlugField(default='-')

slug = models.SlugField(null=True)

if we don’t supply the value in model django will ask us to set it and it is just placed in migration file and in the model the default value is not set and hence not shown

python manage.py migrate

django go through all the apps and execute pending migrations

python manage.py sqlmigrate store 0003 (the third migration in the store app)

using this command we can see actual sql commands forwatd to the database

Django model metadata :

We can specify meta data for entity by creating Meta class inside an entity (search Django model metadata)

**class** Meta:

          db\_table = 'store\_customers'

          indexes = [

               models.Index(fields=['last\_name', 'first\_name'])

          ]

Then we make migrations and migrate to see the changes

If we want to revert the migration and for example change the name of the table

You can delete the code then make migration and migrate and you should delete the migration but it is not a good way the best way is to undo the commit and revert it to the last version

git reset –hard HEAD~1