PyShop project by Mosh

Implementing a new Django project

1. Installing Django
2. Pip install Django==4.0.1
3. django-admin startproject pyshop .
   1. python –version  
      python -m django –version
4. python manage.py runserver

Creating products app

1. python manage.py startapp products

url mapping

1. define function in view.py :

from django.http import HttpResponse

def index(request):

return HttpResponse(‘Hello World!’)

1. map url to function

in products.urls.py

from django.urls import path

from . import views

urlpatterns = [path(‘ ’,views.index),

path(‘new/’, view.new)

]

1. introduce products urls(which is in app) to pyshop urls(which is in project)

from Django.urls import path, include

urlpatterns= [path(‘products/’, include

(‘products.urls’))]

PyShop model

From Django.db import models

Class Product(models.Model):

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

price = models.FloatField()

stokc = models.IntegerField()

image\_url = models.CharField(max\_length=2083)

migrations

PyShop.settings

INSTALLED\_APPS = [

'django.contrib.admin',

'products.apps.ProductsConfig'

]

T : python manage.py makemigrations

T: python manage.py migrate

Creating Offer table (model)

class Offer(models.Model):

code = models.CharField(max\_length=20)

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

discount = models.FloatField()

T:python maage.py makemigrations

T: python manage.py migrate

Managing Admin Panel

T:Python manage.py createsuperuser

Add our model to admin panel

In admin.py add  
from .model import Product  
admin.site.register(Product)

Customize Admin panel

In admin.py add:  
 class ProductAdmin(admin.ModelAdmin):  
 list\_display = ('name', 'price', 'stock')  
 admin.site.register(Product, ProductAdmin)

* After creating a App we should add it in Intalled\_app in settings.py
* Apps : means configuration file
* Views.py : means url handler

vsCode’s Shortcuts

Ctl+B : close and open : Collapse left panel

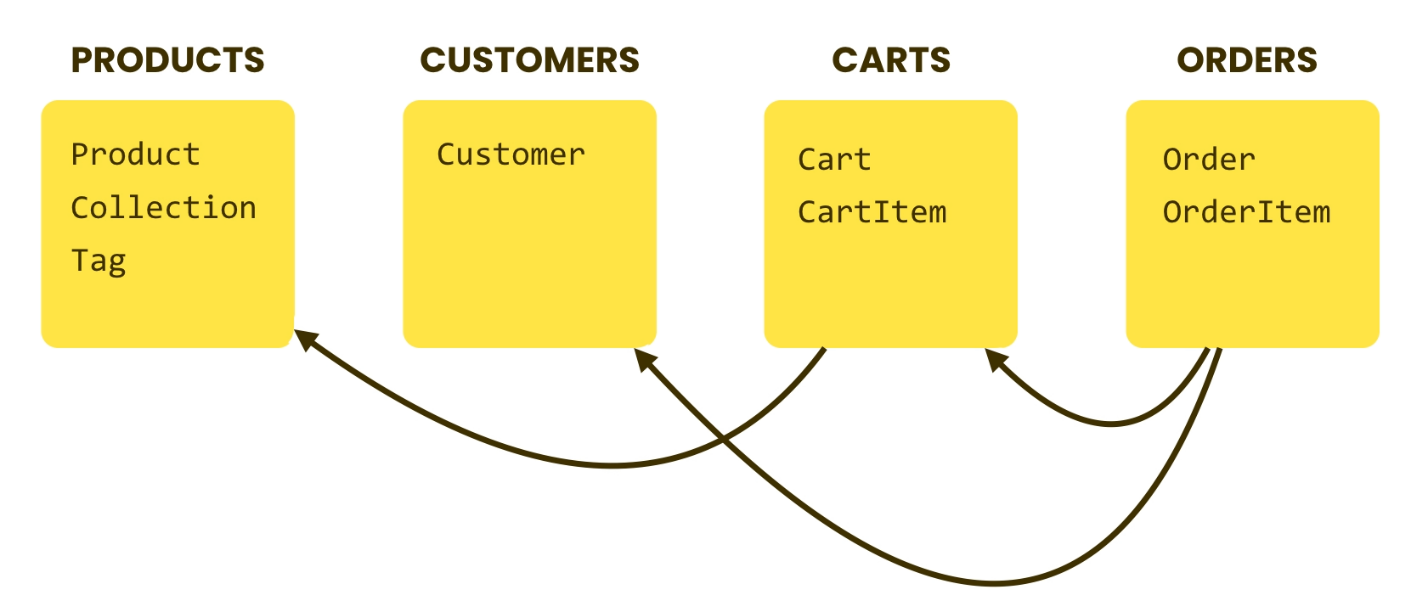
Ctl+L : clear Terminal windows

StoreFront Project

Monolith Design : hard to understand, hard to maintain and hard to reuse

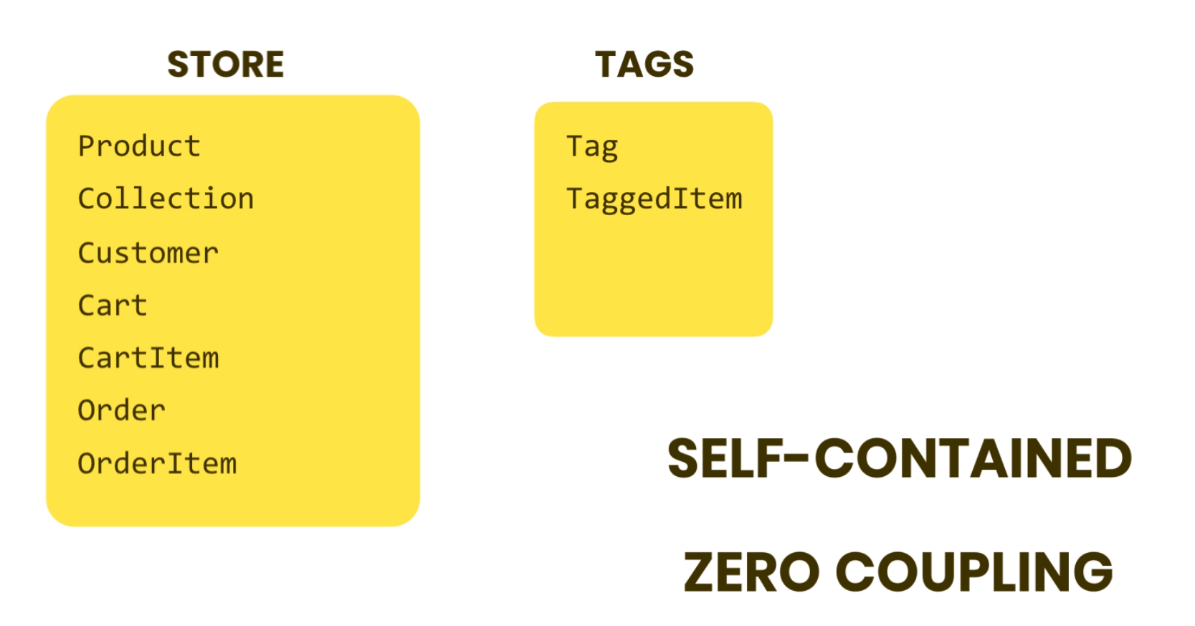


Too small app design



Best philosophy : Each App do one thing and do it well

Minimal coupling  
High cohesion ( Focused)



**4.Setting Up the Database**

Django officially supported Dbs: SqlLite, MySQL,Posgresql,Oracle,MariaDB

T:> py manage.py makemigrations : Django will create a migration file for all Installed\_App(in settings.py)

Every migration file has a sequence number and a descriptive name 0001\_initial.py

SK : ctrl+T : find symbol in project

\*changing migration file name means that you should change all migration dependency files

\*missing to add app name in INSTALLED\_APPS means Django wouldn’t take consider that app’s migration changes

Slug = models.Slugfield(null=true) creating slug field which is a filed that is used in SEO (filed just consist of letter, number, hyphen and underline)

**Running migration**

**Instlling sqllite extension :**install sqlite extention and then in command plette choose sqlite:open my database

Changing Metadata :

In customer models define inner class

Class Meta:

Db\_table = ‘store\_customers’  
 indexes = [models.Index(fields=[‘last\_name’,’first\_name’])]

Recommendation: stick to the Django default convention (do not change tbl name or other conventions)

Recommendation: write a specific code to solve specific problem then migrate

**Revertin migration :**

Consider reverting last change (removing Meta changes)

Py manage.py migrate Store 0003 #migrate from current version(0004) to version 0003

for complete reverting we must delete migration 0004 and all changes in code to avoid applying which is defficult so we use a version control such as **GIT**

**MYSQL**

**Running custom SQL for Migration**

Py manage.py makemigrations store –empty

Edit migration file :

operations = [migrations.RunSQL(“””

INSERT INTO store\_collection(title) VALUES (‘collection1’)“””,

“””

DELETE FROM store\_collection where title=’collection1’ “””) ]

T> py manage.py migrate  
T>py manage.py migrate Store 0003 #revert last migration

**Connect Vscode with MySQL**

1. Install mysql-installer-community
2. Install Vscode extension : mysql
3. Add mysql path in windows path
4. Change project settings file and define connection string

**Episode 5 : Django ORM (Object-Relational Mapper)**

A good software Engineer delivers Working software in time

As the name implies map objects to relational records­

oRM translates our python oop code to SQL code at runtime.

for high performance solutions we need to write our own optimized sql code and don’t use ORM

\*Migrations are part of Django ORM

**Connectng storefront db project**

DATABASES = {

    'default': {

     'ENGINE': 'django.db.backends.mysql',

        'NAME': 'storefront',

        'USER': 'root',

        'PASSWORD': 'passsXXDD',

        'HOST': '127.0.0.1',

        'PORT': '3306',

    }

Create database storefront  
py manage.py migrate  
populate <https://www.mockaroo.com/> data in db

Every **model** in Django have an attribute called **object** which return a **manager Object.** A manager object is an **interface** to database, like a remote control with a bunch of buttons

def say\_hello(request):

    query\_set = Product.objects.all()

    #list(query\_set()) : query sets are lazy means evaluated at later time

    #query\_set[0:5]

    return render(request,'hello.html',{'name':'Javad','products':list(query\_set)})

query set is lazy to build complex queries :

    query\_set=query\_set.filter().filter().order\_by()

when we iterated the query or converte it ro a list(query\_set) that query get valued.

   product= Product.objects.get(pk=1) returns a single product with pk=1

from django.core.exceptions import ObjectDoesNotExist

def say\_hello(request):

    try:

        query\_set = Product.objects.get(pk=0)

    except ObjectDoesNotExist:

        pass

we should try exception because get method return exception if don’t find pk=0

but if we uset first() method it reurns non and we don’t get exception

    query\_set = Product.objects.filter(pk=0).first()

    query\_set = Product.objects.filter(unit\_price\_\_gte=2)  
\* filter method expression should be a logical expression

    query\_set = Product.objects.filter(inventory\_\_lt=20,unit\_price\_\_gt=100)

Product.objects.filter(unit\_price\_\_range=(10,20))

Product.objects.filter(inventory\_\_lt=20).filter(unit\_price\_\_gt=100)

#----------------------------------

    query\_set = Product.objects.filter(collection\_id\_\_title\_\_Icontains='car')

    #filter all products which belongs to a collection which it's title contains car

    query\_set = Product.objects.filter(last\_update\_\_year=2021)

query\_set = Product.objects.filter(description\_\_isnull=True)

    query\_set = Product.objects.filter(inventory\_\_gt=10,unit\_price\_\_lt=200)

OR operations using Q class

from django.db.models import Q,F

# using Q with can use | , & operaators

Product.objects.filter(Q(inventory\_\_gt=10) | Q(unit\_price\_\_lt=200))

Product.objects.filter(Q(inventory\_\_gt=10) | Q~(unit\_price\_\_lt=200))

# ~ means not

#using F for comparing two fields

F(reference to a particular field)

query\_set = Product.objects.filter(inventory=F('unit\_price'))

Google : Queryset Api #all loockup fields

**Creating dummy data :** https://www.mockaroo.com

VSCode Suortcuts :

Shift+pgUp/pgDown : scroll termial up and down

Ctrl+j : show / hide terminal

Ctrl+Shift+O : symbol search

Alt+Shift+F: arrange code