Django

Backend of e-commerce

Commands:

To create django project

- pip3 install pipenv
- pipenv install django
- pipenv shell
- pipenv –venv
- django startproject store .

To create django app(django can have many apps)

- python manage.py startapp play

Writing views

- In play/view.py

```python
from django.shortcuts import render
from django.http import HttpResponse
# Create your views here.

def say_hello(request):
   return HttpResponse('Hello')
```

Mapping view to play/urls.py

- Create urls.py in play

Pass to project urls

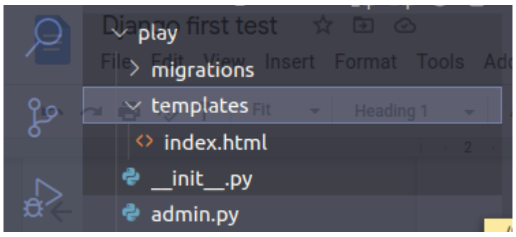
- store/urls.py

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

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

Using template:

- First create templates folder then create .html files



- In views.py render the 'template' file

```python
from django.shortcuts import render
from django.http import HttpResponse
# Create your views here.


def say_hello(request):
   # return HttpResponse('<h1> Hello </h1>')
   # we render
   return render(request, 'index.html')
```

- play/url

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

urlpatterns = [
   path('index/', views.say_hello)
]
```

- store/url

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

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

- To pass by reference use dictionary in views.py

```
def say_hello(request):
   # return HttpResponse('<h1> Hello </h1>')
   # we render
   return render(request, 'index.html', {'name':
'Back'})
```

- Then in '.html' file

```
<h1>Welcome {{ name }}</h1>
```

- Also can write logic in html file

```
<!-- to write logic -->
{% if name %}
<h1>Welcome {{ name }}</h1>

{% else %}

<h1> Welcome sir</h1>

{% endif %}
```

- Debugging in vscode ????
- Django debug bar
    - steps
        1. $ pipenv install django-debug-toolbar
        2. Add to settings.py installed app

```
INSTALLED_APPS = [
   'django.contrib.admin',
   'django.contrib.auth',

'django.contrib.contenttypes',
   'django.contrib.messages',

'django.contrib.staticfiles',
   'play',
   'debug_toolbar'
]
```

3. Add url pattern in main url.py

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

4. Add the Middleware in settings.py

```
MIDDLEWARE = [

'debug_toolbar.middleware.DebugToolba
rMiddleware',
#...
]
```

5. For localhost

```
INTERNAL_IPS = [
    # ...
    "127.0.0.1",
    # ...
]
```

In our template '.html ' file to see debug tool bar we must pass proper html

Creating e-commerce model

- Apps should be as minimal as possible.
- To minimise our complexity of a project

So our models are

- Store_list
    - Product
    - Collection
    - Customer
    - Cart
    - CartItem
    - Order
    - OrderItem

- Tag
    - Tag
    - TaggedItem

Then create a model class for these apps

- In store_list app(folder) / models.py

  Model field types

  `CharField` has two extra arguments:

  `CharField.max_length`

  `CharField.db_collation`

  - We create model classes

```
class Product(models.Model):
   # model field types
   # id created automatically created by django
   title  = models.CharField(max_length=255)
   description = models.TextField()
   # let say max price is 9999.99
   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=255)
   birth_date = models.DateField(null=True)
```

- Choice fields:

  A sequence consisting of iterables of exactly two items (e.g. `[(A, B), (A, B) ...]`) to use as choices for this field. If choices are given, they're enforced by model validation and the default form widget will be a select box with these choices instead of the standard text field.

  We use choice in 2 classes in customer and order.

```

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.Models):
   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)
```

- Defining 1 to 1 relationships

  With customer and address

```
class Address(models.Model):
   street = models.CharField(max_length=255)
   city = models.CharField(max_length=255)
   customer = models.OneToOneRel(Customer,
on_delete=models.CASCADE, primary_key=True)
   # because we don't want to create id for address
that cause many to many relation
```

- Defining 1 to many relationships

```
class Collection(models.Model):
   title = models.CharField(max_length=255)

   # product = models.ForeignKey(Product, on_delete=CASCADE)
   # this should be defined in product class
class Product(models.Model):
   # model field types
   # id created automatically created by django
   title  = models.CharField(max_length=255)
   description = models.TextField()
   # let say max price is 9999.99
   price = models.DecimalField(max_digits=6,
decimal_places=2)
   inventory = models.IntegerField()
   last_update = models.DateTimeField(auto_now=True)
   collection = models.ForeignKey(Collection,
on_delete=models.PROTECT)
   # if collection deleted but not product
```

1 Collection to * Product

```
class Customer(models.Model):
      #...

      #...

class Order(models.Models):
      #...

      #...
   customer = models.ForeignKey(Customer,
on_delete=models.PROTECT)
```

1 Customer to * orders

- Many to Many

  …