Create new Django project

1. Create new folder on machine
2. Open command prompt and go to newly created directory

cd C:\Users\<user-name>\Desktop\Learning\New

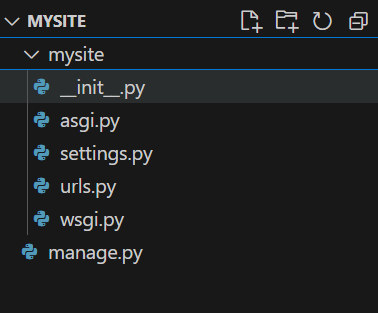
1. Create a django project using below command

django-admin startproject mysite

where,

‘mysite’ is a project name.

1. Open ‘mysite’ project in VS code.



manage.py file perform all administrative task

\_\_init\_\_.py represents it is python package

settings.py file contain all project settings like apps, database etc

urls.py file contains all input urlpatterns

Running django site on development server.

1. Go to project directory

cd C:\Users\<user-name>\Desktop\Learning\New\mysite

1. Run the below command

python manage.py runserver

click <http://localhost-ip:8000/>

Creating app in django:-

1. Go to project directory

cd C:\Users\<user-name>\Desktop\Learning\New\mysite

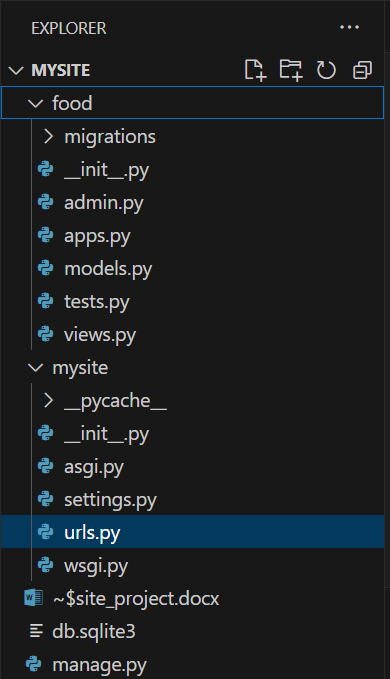
1. Create new app with below command

python manage.py startapp food

or

django-admin startup app projectApp

1. Check in VS code



Creating view:-

from django.shortcuts import render

from django.http import HttpResponse

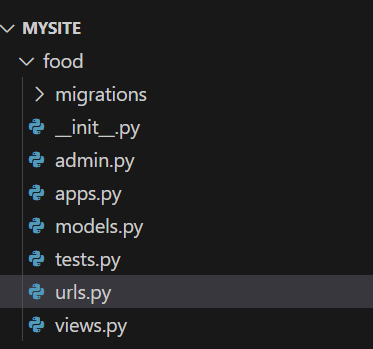
# Create your views here.

def index(request):

    return HttpResponse("Hello World")

Link view with <url:->

1. Create urls.py file in food app



1. Link view in url
2. from django.urls import path
3. from . import views
4. urlpatterns = [
5. path('', views.index, name='index'),
6. ]

Include app urls.py to project urls.py

from django.urls import include, path

urlpatterns = [

    path('admin/', admin.site.urls),

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

]

Run the url as <http://localhost:8000/food>

Hello World view will display

Add another view

def item(request):

    return HttpResponse("This is new Item")

link view to app url

 path('item/', views.item, name='item'),

run :- http:localhost:8000/food/item

**Models:-**

* Blue print to create database table.
* Models are class
* Written in models.py file

Run the below command to migration or create predefined tables in database

python manage.py migrate

this command checks the INSTALLED\_APPS In settigs.py file and create necessary table in DB.

write model:-

(in models.py file)

class Item(models.Model):

    item\_name = models.CharField(max\_length=200)

    item\_desc = models.CharField(max\_length=200)

    item\_price = models.IntegerField()

Add your app in your django project:-

* Go to settings.py file and see INSTALLED\_APPS field
* Add your app name in INSTALLED\_APPS

Format:-

appname.apps.classname

where,

appname = your actual app name

apps represent apps.py file in your app

classname is class name present in apps.py file your app

updated INSTALLED\_APPS:-

INSTALLED\_APPS = [

    'food.apps.FoodConfig',

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

]

Create model in django

python manage.py makemigrations appname

e.g. python manage.py makemigrations food

create models for food app

Create DB table:-

python manage.py sqlmigrate food 0001 (optional)

python manage.py migrate

Store Data in DB table:-

Steps:-

Database abstraction API 🡪 Create Object 🡪 Update Object 🡪 Delete Object

Python Shell:-

Using this functionality you can interact with database.

CMD :- python manage.py shell

* Import the model class

>>> from food.models import Item

Where,

food is a app

models refers to models.py file

Item is model class name.

* Read all data from Item table

>>> Item.objects.all()

Return Queryset[]

* Insert Data

>>> obj1 = Item(item\_name='Pizza',item\_desc='Cheesy Pizza',item\_price=20)

* Save insert data to table

>>> obj1.save()

Insert another row

>>> obj2 = Item(item\_name='Burger',item\_desc='Cheesy Burger',item\_price=10)

>>> obj2.save()

* Read Item table

>>> Item.objects.all()

<QuerySet [<Item: Item object (1)>, <Item: Item object (2)>]>

This return QuerySet.

String representation to display item name instead of object1, object2 etc

class Item(models.Model):

    item\_name = models.CharField(max\_length=200)

    item\_desc = models.CharField(max\_length=200)

    item\_price = models.IntegerField()

    def \_\_str\_\_(self):

        return self.item\_name

Now, restart the existing active Python shell or open new python shell and try to read all data from Item table

>>> from food.models import Item

>>> Item.objects.all()

<QuerySet [<Item: Pizza>, <Item: Burger>]>

**Creating SuperUser and Django admin panel: -**

Execute below command to create Django admin user

python manage.py createsuperuser

Provide username and password

Open Django admin panel

Python manage.py runserver

<https://localhost:8000/admin/>

We can see default Django admin page which has Groups and Users tables.

Adding app model/Tables to admin panel

-Open app admin.py file

- import model

- add model to Admin panel

from django.contrib import admin

from .models import Item

# Register your models here.

admin.site.register(Item)

Refresh the server and http url and see the admin panel which will add app panel under dmin panel with app table name.

Now Using admin panel, You can perform CRUD operation on table.

* RETRIEVE DATA FROM DATABASE

In Django data is retrieves using Queryset.

Queryset is a collect of objects stored in your database.

Manger – TO construct Queryset

A manager is nothing but is something every model which create has

Every Model has default manager called objects.

e.g. Item.objects.all()

Item 🡪 Model

Objects 🡪 manager

All() 🡪 method

* **Retrieve data from database and display in food app webpage(view).**

1. Open views.py file of food app
2. Import Item model
3. Retrieve data in view and display
4. Refresh the server and webpage

from .models import Item

def index(request):

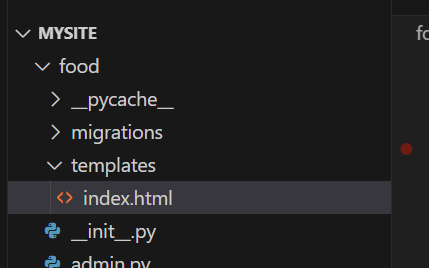
    item\_list = Item.objects.all()

    return HttpResponse(item\_list)

**Template: -**

Templates in Django allows us to create HTML dynamically.

* Create ‘templates’ directory/folder in app folder.
* Create a new directory under template (Optional).
* Create HTML template file.



<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <h1>This is juat a template</h1>

</body>

</html>

* Import Django template loader
* Use template using get\_template() function
* Create context{}
* Render a template in httpResponse
* from django.template import loader

def index(request):

    item\_list = Item.objects.all()

    template = loader.get\_template('food/index.html')

    context = {

          }

    return HttpResponse(template.render(context,request))

Retrieving database object to templates/ Passing context to templates/ render template

def index(request):

    item\_list = Item.objects.all()

    context = {

        'item\_list': item\_list

          }

    return render(request, 'food/index.html', context)

{% for item in item\_list %}

<ul>

    <li>

        {{ item.id }}  --- {{ item.item\_name }}

    </li>

</ul>

{% endfor %}

Creating detailed view:-

This will display item details.

Write detail() view :-

def detail(request,item\_id):

    return HttpResponse("This is item no/id: %s" % item\_id)

Update view to urls.py

   path('<int:item\_id>/', views.detail, name='detail'),

webpage:-

http://127.0.0.1:8000/food/2/

Detail view continue:-

def detail(request,item\_id):

    item = Item.objects.get(pk=item\_id)

    context = {

        'item': item

    }

#    return HttpResponse("This is item no/id: %s" % item\_id)

    return render(request,'food/detail.html', context)

modify index.html (dynamic url)

{% for item in item\_list %}

<ul>

    <li>

 <!--      {{ item.id }}  --- {{ item.item\_name }} -->

        <a href="/food/{{ item.id }}">{{ item.id }}  --- {{ item.item\_name }} </a>

    </li>

</ul>

{% endfor %}

**-DTL (Django Template langauge)**

**-** Django Default Engine

- Jinja2 – Templating engine

- Define variable in double curly braces {{}}.

- Define Tag, Tags are defined in {% %}.

* **Removing hard coded URLs:- (Use of url Tag)**

We can replace hard code url with path function name attribute.

Previous: -

<a href="/food/{{ item.id }}">{{ item.id }} --- {{ item.item\_name }}</a>

New: -

<a href="{% url 'detail' item.id %}">{{ item.id }}  --- {{ item.item\_name }} </a>

Where,

‘detail’ represents name value of path function of urls.py

(  path('<int:item\_id>/', views.detail, name='detail'),)

* **Namespacing: -**

URL namespaces allow you to uniquely reverse named URL patterns even if different applications uses the same URL names.

Use:-

- Go to urls.py file of your app and add below line before url patterns.

app\_name = 'food'

* Use ‘food’ namespace wherever you use path names.
* <a href="{% url 'food:detail' item.id %}">{{ item.id }}  --- {{ item.item\_name }} </a>

Where, food is a namespace & detail is a path name

* **STATIC files:-**

These are css file to design web pages.

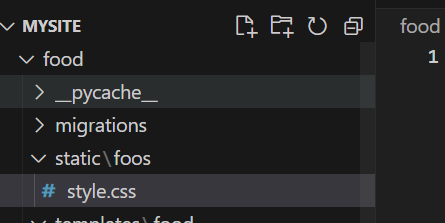
Path:-

Go to app directory

Create ‘static’ directory/folder.

Create app name folder. (Optional)

Create css file.



Load static file in html file

* Images, javascript or css are static files.
* Static file configured in setting.py file

In Installed\_apps

* 'django.contrib.staticfiles',
* STATIC\_URL = 'static/'
* Add DTL Tag to load static files in html file.
* {% load static %}
* Link css file in html <head> tag
* <link rel="stylesheet" href="{% static 'food/style.css' %}">

If you did not see background colour change, then restart the server.

* **Creating NavBar**

Add BootstrapCDN in <head> tag of HTML file.

Go to bootstrap website ad copy and paste bootstrapCDN in your html page head tag

<script src="https://cdn.jsdelivr.net/npm/bootstrap@4.3.1/dist/js/bootstrap.min.js" integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM" crossorigin="anonymous">

</script>

* Create NavBar in html body with bootstrap classes.
* <nav class="navbar navbar-dark bg-dark">
* <a href="#" class="navbar-brand">FoodApp</a>
* <div class="navbar">
* <a class="nav-Item nav-link" href="#">Add Item</a>
* <a class="nav-Item nav-link" href="#">View Item</a>
* <a class="nav-Item nav-link" href="#">Delete Item</a>
* </div>
* </nav>

**-Creating Base Template**

Create base.html file template.

Copy links and NavBar from Index.html file to base.html

Load static file

{% load static %}

Create block body DTL tags

    {% block body %}

    {% end block body %}

Load base file to index.html and detail.html files.

{% extends 'food/base.html'%}

Add block body tag to index.html and detail.html files.

Final files.

Base.html file

{% load static %}

<!DOCTYPE html>

<html lang="en">

<head>

    <title>Document</title>

    <link rel="stylesheet" href="{% static 'food/style.css' %}">

    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.3.1/dist/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.3.1/dist/js/bootstrap.min.js" integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM" crossorigin="anonymous">

    </script>

</head>

<body>

    <nav class="navbar navbar-dark bg-dark">

        <a href="#" class="navbar-brand">FoodApp</a>

        <div class="navbar">

            <a class="nav-Item nav-link" href="#">Add Item</a>

            <a class="nav-Item nav-link" href="#">View Item</a>

            <a class="nav-Item nav-link" href="#">Delete Item</a>

        </div>

    </nav>

    {% block body %}

    {% endblock %}

</body>

</html>

Index.html file: -

{% extends 'food/base.html'%}

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    {% block body %}

    {% for item in item\_list %}

    <ul>

        <li>

 <!--      {{ item.id }}  --- {{ item.item\_name }} -->

            <a href="{% url 'food:detail' item.id %}">{{ item.id }}  --- {{ item.item\_name }} </a>

        </li>

    </ul>

    {% endfor %}

    {% endblock %}

</body>

</html>

detail.html file: -

{% extends 'food/base.html'%}

{% block body %}

<h2> {{ item.item\_name }} </h2>

<h3> {{ item.item\_desc }} </h3>

<h4> {{ item.item\_price }} </h4>

{% endblock %}

* **Adding Image field to model: -**

Modify model with image files as below.

class Item(models.Model):

    item\_name = models.CharField(max\_length=200)

    item\_desc = models.CharField(max\_length=200)

    item\_price = models.IntegerField()

    item\_image = models.CharField(max\_length=500, default="https://plakarestaurant.ca/wp-content/themes/twentytwentythree-child/img/food-placeholder.png")

    def \_\_str\_\_(self):

        return self.item\_name

Generate Django migration

python manage.py makemigrations food

python manage.py sqlmigrate food 0002

Commit changes to database

python manage.py migrate

* Access image in web page

Download image place holder, adding button for details page

Remove all html code from index.html body and update with new code.

<body>

    {% block body %}

    {% for item in item\_list %}

    <div class="row">

        <div class="col-md-3 offset-md-2">

            <img class="card" height="150px" src="{{ item.item\_image }}"/>

        </div>

        <div class="col-md-4">

            <h3>{{ item.item\_name }}</h3>

            <h4>{{ item.item\_desc }}</h4>

            <h5>${{ item.item\_price}}</h5>

        </div>

        <div class="col-md-2">

            <a href="{% url 'food:detail' item.id %}" class="btn btn-success">Details</a>

        </div>

    </div>

    {% endfor %}

    {% endblock %}

</body>

Add actual images

Copy images url and insert into Item table using admin panel

Refresh food app and see the result.

* Designing detail view: -

Modify detail.html file

<div class="row">

    <div class="col-md-6">

        <img class="card" height="300px" src="{{ item.item\_image }}"/>

    </div>

    <div class="col-md-6">

        <h2> {{ item.item\_name }} </h2>

        <h3> {{ item.item\_desc }} </h3>

        <h4> ${{ item.item\_price }} </h4>

    </div>

</div>

* **Django form: -**

1. Create a forms.py file in food app and write model form.

from django import forms

from .models import Item

# Creating model form

class ItemForm(forms.ModelForm):

    class Meta:

        model=Item

        fields = ['item\_name','item\_desc','item\_price','item\_image']

1. Create a template for the new form.
2. Write a view for the form

from .forms import ItemForm

def create\_item(request):

    # user form class

    form=ItemForm(request.POST or None)

    if form.is\_valid():

        form.save()

        return redirect('food:index')

    return render(request,'food/Item-form.html', {'form':form})

1. Pass form object to html template file

 <form method="POST">

        {% csrf\_token %}

        {{ form }}

    <button type="submit">Save</button>

    </form>

1. Add view to urlpattern path in urls.py file

path('add/', views.create\_item, name='create\_item'),

Check the webpage and add new item

1. Link Add\_item

Go to base directory and add url

<a class="nav-Item nav-link" href="{% url 'food:create\_item' %}">Add Item</a>

1. Render base.html to item-form.html

-**Edit functionality:-**

**View: -**

# Update items

def update\_item(request, id):

    item = Item.objects.get(id=id)

    # create for object

    form = ItemForm(request.POST or None, instance=item)

    if form.is\_valid():

        form.save()

        return redirect('food:index')

    return render(request, 'food/Item-form.html', {'form':form, 'item':item})

urls.py: -

 path('update/<int:id>/', views.update\_item, name='update\_item'),

**-Delete item**

def delete\_item(request, id):

    item = Item.objects.get(id=id)

    # create for object

    form = ItemForm(request.POST)

    if request.method == 'POST':

        item.delete()

        return redirect('food:index')

    return render(request, 'food/Item-delete.html', {'item':item})

{% extends 'food/base.html'%}

{% block body %}

<form method="POST">

    {% csrf\_token %}

    <h2>Are you sure you want to delete {{ item.item\_name }}</h2>

    <button type="submit">Confirm</button>

</form>

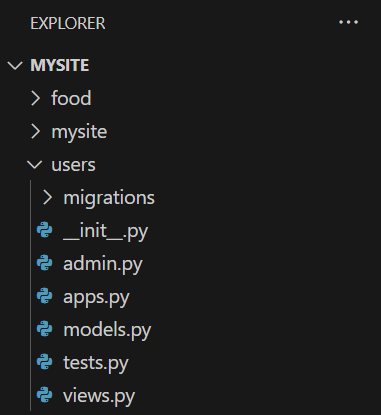
{% endblock %}

path('delete/<int:id>/', views.delete\_item, name='delete\_item'),

* **User registration form: -**

1. Create users app.

python manage.py startapp users



1. Add new app to project setting.py file INSTALLED\_APPS

   'users.apps.UsersConfig',

1. Write a new view with in-built UserCreationForm in users app

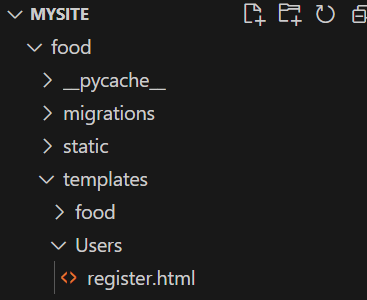
from django.contrib.auth.forms import UserCreationFrom

def register(request):

    form = UserCreationFrom()

    return render (request,'/user/register.html',{'form':form})

1. Write a template for user creation.



<form method="POST">

    {% csrf\_token %}

    {{ form }}

</form>

1. Update project urls.py file.

path('register/', user\_views.register, name='register'),

* **Registration success message: -**

1. Update view file

from django.contrib import messages

from django.shortcut import render,redirect

# Create your views here.

def register(request):

    if request.method == 'POST':

        form = UserCreationForm(request.POST)

        if form.is\_valid():

            username = form.cleaned\_data.get('username')

            messages.success(request, f"Welcome {username}, Your account is created")

            return redirect('food:index')

    else:

        form = UserCreationForm()

    return render (request,'users/register.html',{'form':form})

1. Update base.html file

{% if messages %}

        {% for message in messages %}

            <div class="alert alert-{{message.tags}}">{{ message }}</div>

        {% endfor %}

    {% endif %}

**-Saving users: -**

Django UserCreationForm does not allow duplicate username. Default validation.

1. Add below line of code in user form

form.save()

1. This will save new user in user table, check via admin panel.