**Module – 5 (DB and Python Framework)**

**1) Why Django should be used for web-development? Explain how you can create a project in Django?**

Django promotes fast development by including numerous built-in modules such as user authentication, admin interface, URL routing, and form handling. This allows developers to focus on writing application logic rather than reinventing the wheel. Django comes with built-in authentication and user management features, further strengthening application security. Django has a large, active community that contributes to its continuously growing ecosystem. Developers have access to thousands of third-party packages that can be integrated to extend the framework's functionality.

* pip install Django
* django-admin startproject myproject
* cd myproject
* python manage.py runserver

**2) How to check installed version of Django?**

django-admin —version

**3) Explain what does django-admin.py make messages command is used for?**

When you want your website to support multiple languages, you need to:

Identify and mark text in your app that should be translated.

Extract these texts into language-specific files.

Translate them.

Compile them so Django can use them.

The makemessages command handles step 2 — it scans your code for strings marked for translation and creates .po files for translators to work with.

**4) What is Django URLs? Make program to create Django urls?**

In Django, URLs (Uniform Resource Locators) are used to map web addresses (such as example.com/home) to views, which are Python functions or classes that handle the request and return a response.

Django uses a URL dispatcher that allows you to define URL patterns and link them to the appropriate views.

from django.http import HttpResponse

def home(request):

return HttpResponse("Welcome to the Home Page!")

def about(request):

return HttpResponse("This is the About Page.")

**5) What is a QuerySet? Write program to create a new Post object in database:**

A QuerySet in Django represents a collection of database queries that fetch data from a model (i.e., a table in the database). It is Django’s way of interacting with the database using Object-Relational Mapping (ORM) instead of writing raw SQL queries.

from django.db import models

class Post(models.Model):

title = models.CharField(max\_length=100)

content = models.TextField()

author = models.CharField(max\_length=50)

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

def \_\_str\_\_(self):

return self.title

**6) Mention what command line can be used to load data into Django?**

In Django, data can be loaded into the database using a variety of command-line tools. One of the most commonly used methods is the loaddata command, which is designed to import data from fixture files into the database.

python manage.py loaddata <fixture\_file\_name>.json

**7) Explain what does django-admin.py make messages command is used for?**

When you want your website to support multiple languages, you need to:

Identify and mark text in your app that should be translated.

Extract these texts into language-specific files.

Translate them.

Compile them so Django can use them.

The makemessages command handles step 2 — it scans your code for strings marked for translation and creates .po files for translators to work with.

**8) Make Django application to demonstrate following things o There will be 2 modules(Admin,Product manager) o Admin can add product name (ex.Product id and product name) ex. (1, Samsung), (2, Apple)…etc. Data should store in?**

Step 1: models.py

from django.db import models

class Product(models.Model):

product\_id = models.IntegerField(unique=True)

product\_name = models.CharField(max\_length=100)

def \_\_str\_\_(self):

return f"{self.product\_id} - {self.product\_name}"

Step 2: admin.py

from django.contrib import admin

from .models import Product

class ProductAdmin(admin.ModelAdmin):

list\_display = ('product\_id', 'product\_name')

admin.site.register(Product, ProductAdmin)

Step 3: Create Superuser for Admin Module

python manage.py createsuperuser

(Admin can now log in at http://127.0.0.1:8000/admin)

Step 4: settings.py (Add App to Installed Apps)

INSTALLED\_APPS = [

...

'myapp',

]

Step 5: Apply Migrations

python manage.py makemigrations

python manage.py migrate

Step 6: URL Configuration

from django.shortcuts import render

from .models import Product

def product\_list(request):

products = Product.objects.all()

return render(request, 'product\_list.html', {'products': products})

python

CopyEdit

# myapp/urls.py

from django.urls import path

from . import views

urlpatterns = [

path('products/', views.product\_list, name='product\_list'),

]

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('', include('myapp.urls')),

]

Step 7: Template

<!DOCTYPE html>

<html>

<head><title>Product Manager</title></head>

<body>

<h1>All Products</h1>

<ul>

{% for product in products %}

<li>{{ product.product\_id }} - {{ product.product\_name }}</li>

{% endfor %}

</ul>

</body>

</html>

**8) Product\_mst table with product id as primary key o Admin can add product subcategory details Like (Product price, product image, Product model, product Ram) data should store in Product\_sub\_cat table o Admin can get product name as foreign key from product\_mst table in product\_sub\_category\_details page Admin can view, update and delete all registered details of product manager can search product on search bar and get all details about product.**

1. models.py

from django.db import models

class ProductMst(models.Model):

product\_id = models.AutoField(primary\_key=True)

product\_name = models.CharField(max\_length=100)

def \_\_str\_\_(self):

return self.product\_name

class ProductSubCat(models.Model):

product = models.ForeignKey(ProductMst, on\_delete=models.CASCADE)

product\_price = models.DecimalField(max\_digits=10, decimal\_places=2)

product\_image = models.ImageField(upload\_to='product\_images/')

product\_model = models.CharField(max\_length=100)

product\_ram = models.CharField(max\_length=50)

def \_\_str\_\_(self):

return f"{self.product.product\_name} - {self.product\_model}"

2. admin.py

from django.contrib import admin

from .models import ProductMst, ProductSubCat

class ProductSubCatAdmin(admin.ModelAdmin):

list\_display = ('product', 'product\_price', 'product\_model', 'product\_ram')

admin.site.register(ProductMst)

admin.site.register(ProductSubCat, ProductSubCatAdmin)

3. forms.py

from django import forms

from .models import ProductSubCat

class ProductSearchForm(forms.Form):

search\_query = forms.CharField(label='Search Product', max\_length=100)

4. views.py

from django.shortcuts import render

from .models import ProductSubCat

from .forms import ProductSearchForm

def product\_search(request):

form = ProductSearchForm()

products = []

if request.method == 'GET':

query = request.GET.get('search\_query', '')

if query:

products = ProductSubCat.objects.filter(product\_\_product\_name\_\_icontains=query)

return render(request, 'product\_search.html', {'form': form, 'products': products})

5. urls.py

from django.urls import path

from . import views

urlpatterns = [

path('search/', views.product\_search, name='product\_search'),

]

In myproject/urls.py:

from django.contrib import admin

from django.urls import path, include

from django.conf import settings

from django.conf.urls.static import static

urlpatterns = [

path('admin/', admin.site.urls),

path('', include('myapp.urls')),

]

if settings.DEBUG:

urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

6. templates/product\_search.html

<!DOCTYPE html>

<html>

<head>

<title>Search Product</title>

</head>

<body>

<h2>Search Product</h2>

<form method="get">

{{ form.as\_p }}

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

</form>

{% if products %}

<h3>Search Results:</h3>

<ul>

{% for item in products %}

<li>

<strong>{{ item.product.product\_name }}</strong><br>

Model: {{ item.product\_model }}<br>

RAM: {{ item.product\_ram }}<br>

Price: ${{ item.product\_price }}<br>

<img src="{{ item.product\_image.url }}" width="150" />

</li>

{% endfor %}

</ul>

{% elif form.cleaned\_data.search\_query %}

<p>No products found.</p>

{% endif %}

</body>

</html>

7. settings.py (Important!)

Add this to handle image uploads:

python

CopyEdit

# settings.py

MEDIA\_URL = '/media/'

MEDIA\_ROOT = BASE\_DIR / 'media'