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 is a high-level Python web framework that simplifies the development of robust, scalable, and maintainable web applications.

🡪 How to Create a Project in Django?

1. **Install Django**

pip install Django

**2. Start a Django Project**

django-admin startproject project\_name

3. **Navigate to the Project Directory**

cd project\_name

4. **Run the Development Server**

python manage.py runserver

**5. Create an App**

python manage.py startapp app\_name

6. **Register the App**

INSTALLED\_APPS =

[

'app\_name',

]

**8. Apply Migrations**

python manage.py makemigrations

python manage.py migrate

9. **Create Views and URLs**

# views.py

from django.http import HttpResponse

def home(request):

return HttpResponse("Hello, Django!")

# urls.py

from django.urls import path

from.import views urlpatterns = [

path('', views.home, name='home'),

]

🡪 URLs in the project's urls.py

from django.contrib import admin from django.urls import path, include urlpatterns = [ path('admin/', admin.site.urls), path('', include('app\_name.urls')), ]

🡪 Run the Server

python manage.py runserver

(2) How to check installed version of Django?

🡪 Run the following command in your terminal or command prompt:

**django-admin –version**

🡪 Using pip show

**pip show django**

(3) Explain what does django-admin.py makemessages command is used for?

🡪 The django-admin.py makemessages command is used in Django for internationalization (i18n).

1.Scans the Codebase: It looks for strings marked with translation functions like gettext.

2.Generates Message Files: It creates .po files (Portable Object files) in the locale directory of your app. These files store the original strings and their placeholders for translations.

3.Supports Multiple Languages: You can specify the language for which you want to generate message files.

**cmd : django-admin makemessages -l <language\_code>**

<language\_code> is the language code (e.g.fr for French, es for Spanish).

(4) What are Django URLs? make program to create django urls.

🡪 In Django, urls like addresses for different pages of your website. They connect specific web requests (like visiting a page) to the code (called views) that handles the request.

Example:

* Visiting http://yourwebsite.com/home/ might show the homepage.

🡪 Django uses a URL dispatcher to match these addresses to the correct views.

How to create Django urls?

**1. Write a View (What Happens on a Page)**: **In views.py**

from django.http import HttpResponse

def home(request):

return render(request,"Welcome to the Home Page!")

**2. Add URLs in urls.py (Connect Address to View)**: **In urls.py:**

from django.urls import path

from . import views

urlpatterns = [

path('home/', views.home, name='home'), # Home Page

]

**3.Run the Server and Test**: Start the server:

python manage.py runserver

Visit : http://127.0.0.1:8000/home

Shows "Welcome to the Home Page!"

Explanation:

Urls: Address(e.g. /home/) for pages

Views: Code that runs when someone visits an address.

Dispather: Matches the address to the correct code.

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

🡪 A QuerySet in Django is a collection of database queries that represent a set of objects retrieved from your database. It allows you to filter, order, and manipulate data using Django’s ORM (Object-Relational Mapping).

Creating a New Post Object in the Database:

1. **Define the Post Model**: In models.py:

from django.db import models

class Post(models.Model):

title = models.CharField(max\_length=200)

content = models.TextField()

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

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

return self.title

2. **Apply Migrations**: Run the following commands to create the database table:

python manage.py makemigrations

python manage.py migrate

3. **Create a New Post Object**: Open the Django shell:

python manage.py shell

Inside the shell, create a new object:

from myapp.models import Post

# Create and save a new Post object

post = Post.objects.create(

title="My First Post",

content="This is the content of my first post."

)

print(post.id) # Check the ID of the created post

4. **Check the Database**: You can confirm the object was created by opening the Django admin panel or querying it again:

all\_posts = Post.objects.all()

print(all\_posts)

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

🡪 In Django, you can use the *loaddata* command to import data into your database from a file (like JSON, XML, or YAML).

🡪 Command Syntax:

python manage.py loaddata <fixture\_file>

🡪If you have a fixture file named data.json in your project, you can load it using:

python manage.py loaddata data.json

🡪Steps to Load Data Using loaddata

1. **Create a Fixture File**: Export data or manually create a fixture file.

For example, a data.json file might look like this:

[

{

"model": "appname.modelname",

"pk": 1,

"fields": {

"field1": "value1",

"field2": "value2"

}

},

{

"model": "appname.modelname",

"pk": 2,

"fields": {

"field1": "value3",

"field2": "value4"

}

}

]

2. **Place the Fixture File**: Save the fixture file in a directory Django can access, such as appname/fixtures/.

**3.Run the Command**: Use the loaddata command to load the data into your database:

🡪python manage.py loaddata appname/fixtures/data.json

4. **Verify Data**: Check the database or use the Django admin panel to confirm the data was loaded successfully.

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

🡪 The makemessages command in Django is used to set up translations for your app. It finds text marked for translation and creates a file where you can add translations in different languages.

*How to Use the Command*

**django-admin makemessages -l <language\_code>**

<language\_code>: Language you want to translate to (e.g., fr for French, es for Spanish).

Workflow:

1. **Mark Translatable Strings**: Mark strings in your templates or Python code using \_():

from django.utils.translation import gettext as \_

greeting = \_("Hello, World!")

2. **Run makemessages**: Generate the .po file:

django-admin makemessages -l fr

3. **Translate the Strings**: Open the .po file, and add translations for each string.

4. **Compile Messages**: Compile the .po file into a binary .mo file using:

django-admin compilemessages

5.**Enable Translations**: Django will now use the translated strings when the corresponding language is active.

Options for makemessages

-a: Updates all languages.

-e: Specifies file extensions to process (e.g., .html).

--ignore: Excludes specific files or directories.

(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.

🡪 Steps to Create the Application

1. **Set Up the Project and App**

django-admin startproject product\_management

cd product\_management

python manage.py startapp products

**2. Define Models**

products/models.py,

create models for ProductMst and ProductSubCat.

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, related\_name="subcategories")

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

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

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

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

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

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

**3. Apply Migrations**

🡪 python manage.py makemigrations

🡪 python manage.py migrate

**4. Create Forms**

products/forms.py

define forms for adding and searching products.

from django import forms

from .models import ProductMst, ProductSubCat

class ProductMstForm(forms.ModelForm):

class Meta:

model = ProductMst

fields = ['product\_name']

class ProductSubCatForm(forms.ModelForm):

class Meta:

model = ProductSubCat

fields = ['product', 'price', 'image', 'model', 'ram']

class ProductSearchForm(forms.Form):

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

**5. Create Views**

products/views.py,

views for Admin and Product Manager functionality.

from django.shortcuts import render, get\_object\_or\_404, redirect

from .models import ProductMst, ProductSubCat

from .forms import ProductMstForm, ProductSubCatForm, ProductSearchForm

def admin\_add\_product(request):

form = ProductMstForm(request.POST or None)

if form.is\_valid():

form.save()

return redirect('admin\_product\_list')

return render(request, 'admin\_add\_product.html', {'form': form})

def admin\_add\_subcategory(request):

form = ProductSubCatForm(request.POST or None, request.FILES or None)

if form.is\_valid():

form.save()

return redirect('admin\_product\_list')

return render(request, 'admin\_add\_subcategory.html', {'form': form})

def admin\_product\_list(request):

products = ProductMst.objects.all()

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

def admin\_edit\_product(request, pk):

product = get\_object\_or\_404(ProductMst, pk=pk)

form = ProductMstForm(request.POST or None, instance=product)

if form.is\_valid():

form.save()

return redirect('admin\_product\_list')

return render(request, 'admin\_edit\_product.html', {'form': form})

def admin\_delete\_product(request, pk):

product = get\_object\_or\_404(ProductMst, pk=pk)

product.delete()

return redirect('admin\_product\_list')

def product\_manager\_search(request):

form = ProductSearchForm(request.GET or None)

products = None

if form.is\_valid():

query = form.cleaned\_data['search\_query']

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

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

**6. Create Templates**

Create HTML templates for each view in products/templates.

-admin\_add\_product.html

- admin\_add\_subcategory.html

-admin\_product\_list.html

-admin\_edit\_product.html

-product\_manager\_search.html

**7. Configure URLs**

products/urls.py, define the app’s URLs.

from django.urls import path

from . import views

urlpatterns = [

path('admin/add-product/', views.admin\_add\_product, name='admin\_add\_product'),

path('admin/add-subcategory/', views.admin\_add\_subcategory, name='admin\_add\_subcategory'),

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

path('admin/edit-product/<int:pk>/', views.admin\_edit\_product, name='admin\_edit\_product'),

path('admin/delete-product/<int:pk>/', views.admin\_delete\_product, name='admin\_delete\_product'),

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

]

Include the app’s URLs in the project’s urls.py:

from django.urls import path, include

urlpatterns = [

path('admin/', include('products.urls')),

]

**8.Run the server**

🡪 python manage.py runserver