c. Use the admin panel to add at least three sample books with different details.

a. Create a view to display a list of all books in the database. Use a template to render this list.
b. Create a view to display detailed information about a single book, including all its fields.
c. Create templates for both views, ensuring they have appropriate HTML structure.

a. Define URL patterns to route requests to the views you created in Task 4.
b. Implement a homepage that displays a list of all books.
c. Implement URLs for displaying detailed book information.

a. Implement a search functionality that allows users to search for books by title or author.

ANS)
Step 1: Create a new Django project named "Bookstore."
```
django-admin startproject Bookstore
```

Step 2: Set up a Django app named "books."
```
cd Bookstore
python manage.py startapp books
```

Step 3: Define a Django model named "Book" with the specified fields in the "books/models.py" file.
```
from django.db import models

class Book(models.Model):
    title = models.CharField(max_length=100)
    author = models.CharField(max_length=100)
    published_date = models.DateField()
    price = models.DecimalField(max_digits=10, decimal_places=2)
    isbn = models.CharField(max_length=13)

    def __str__(self):
        return self.title
```

Step 4: Create and apply the necessary database migrations to create the "Book" model.
```
python manage.py makemigrations books
python manage.py migrate
```

Step 5: Register the "Book" model in the Django admin panel.
In the "books/admin.py" file, register the "Book" model as follows:
```
from django.contrib import admin
from .models import Book

admin.site.register(Book)
```

Step 6: Create a superuser account with the following command:
```
python manage.py createsuperuser
```

Follow the prompts to create a superuser account with a username and password.

Step 7: Use the admin panel to add at least three sample books with different details.

Access the Django admin panel at http://localhost:8000/admin/ and log in with the superuser account. Then, use the admin panel to add sample books with their details.

Step 8: Create views and templates for displaying the list of books and detailed book information.

a. Create a view to display a list of all books in the "books/views.py" file:

```
from django.shortcuts import render
from .models import Book

def book_list(request):
    books = Book.objects.all()
    return render(request, 'books/book_list.html', {'books': books})
```

b. Create a view to display detailed information about a single book:

```
from django.shortcuts import render, get_object_or_404
from .models import Book

def book_detail(request, book_id):
    book = get_object_or_404(Book, pk=book_id)
    return render(request, 'books/book_detail.html', {'book': book})
```

c. Create templates for both views in a "templates/books" directory.

Create "templates/books/book_list.html" for the list of books.
Create "templates/books/book_detail.html" for detailed book information.
Step 9: Define URL patterns to route requests to the views created in Task 8.

In the "books/urls.py" file, define URL patterns as follows:

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

urlpatterns = [
    path('', views.book_list, name='book_list'),
    path('<int:book_id>/', views.book_detail, name='book_detail'),
]
```

Step 10: Implement a homepage that displays a list of all books.

In the project's "Bookstore/urls.py" file, include the "books" app's URLs as follows:

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

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

Step 11: Implement URLs for displaying detailed book information.

In the homepage template (e.g., "books/book_list.html"), include links to individual book pages using the {% url 'book_detail' book.id %} template tag.

Step 12: Implement a search functionality that allows users to search for books by title or author.

a. Create a new view for searching books:

```
from django.db.models import Q
from django.shortcuts import render
from .models import Book

def book_search(request):
    query = request.GET.get('q')
    if query:
        books = Book.objects.filter(Q(title__icontains=query) |
Q(author__icontains=query))
    else:
        books = Book.objects.all()
    return render(request, 'books/book_search.html', {'books': books,
'query': query})
```

b. Define a URL pattern for the search view in the "books/urls.py" file:

```
path('search/', views.book_search, name='book_search'),
```

c. Create a template for the search results (e.g., "books/book_search.html") that displays the search form and search results.

With these steps, you'll have a Django project named "Bookstore" with a "books" app that includes a "Book" model, admin panel integration, views for listing and displaying book details, and a search functionality.

17)

- **Create python Django project with name ‘moviereview’**
- **Create an app called movie**
- **Create home.html file in movieapp.**
- **Code for home.html**

```
<body>
<h1>My movie app </h1>
<h3>Enter data </h3>
<form action="" >
<label for="data">Data:</label>
<input type="text" name=" " ><br><br>
<button type="submit" >Search</button>
</form>
</body>
```

- **Create model named Movie with attributes Title,Actor,Date of Relaease.**
- **Create super user with your enrollment number and password will be your name.(it is compulsory)**
- **Log in to the django admin portal with this user and Enter the following data in Movie table.**

Title	Actor	Date of Release
JAWAN	SRK	8-Sept-2023
GADAR-2	SunnyD	25-Aug-2023

OH MY GOD-2 Akshay K 18-Aug-2023
• Make necessary adjustment to your code to let user search for data from this database by Title on home page

ANS)
Step 1: Create a Django project named "moviereview."
```
django-admin startproject moviereview
```

Step 2: Create an app called "movie."
```
cd moviereview
python manage.py startapp movie
```

Step 3: Create a "home.html" file in the "movie" app's "templates/movie" directory.

Here's the code for "home.html":
```
<!DOCTYPE html>
<html>
<head>
    <title>My Movie App</title>
</head>
<body>
    <h1>My Movie App</h1>
    <h3>Enter data</h3>
    <form action="" method="get">
        <label for="title">Title:</label>
        <input type="text" name="title"><br><br>
        <button type="submit">Search</button>
    </form>
</body>
</html>
```

Step 4: Create a model named "Movie" with attributes Title, Actor, and Date of Release in the "movie/models.py" file.
```
from django.db import models

class Movie(models.Model):
    title = models.CharField(max_length=100)
    actor = models.CharField(max_length=100)
    date_of_release = models.DateField()

    def __str__(self):
        return self.title
```

Step 5: Create a superuser with your enrollment number as the username and your name as the password.
```
python manage.py createsuperuser
```

Follow the prompts to create the superuser account.

Step 6: Log in to the Django admin portal with the superuser account and enter the following data in the "Movie" table:

Title: JAWAN
Actor: SRK
Date of Release: 2023-09-08

Title: GADAR-2

```
Actor: SunnyD
Date of Release: 2023-08-25

Title: OH MY GOD-2
Actor: Akshay K
Date of Release: 2023-08-18
```

Step 7: Make necessary adjustments to your code to let users search for data by Title on the home page.

Modify the "movie/views.py" file to handle the search functionality:

```
from django.shortcuts import render
from .models import Movie

def home(request):
    title = request.GET.get('title')
    movies = Movie.objects.filter(title__icontains=title) if title else
[]
    return render(request, 'movie/home.html', {'movies': movies})
```

Update the "movie/urls.py" file to include the URL pattern for the home view:

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

urlpatterns = [
    path('', views.home, name='home'),
]
```

18)
Create python Django project with name 'myproject'
• Create an app called myapp
• Create home.html file in myapp.
• Code for home.html

```
<body>
<h1>My app</h1>
<h3>Enter data </h3>
<form action="" >
<label for="data">Data:</label>
<input type="text" name=" " ><br><br>
<button type="submit" >Search</button>
</form>
</body>
```

• Create model named Mydata with attributes name,branch,roll no.
• Create super user with your enrollment number and password will be your name.(it is compulsory)
• Log in to the django admin portal with this user and Enter the following data in Mydata table.

name	branch	roll no
Yaksh	**CE**	**111**
Rohan	**IT**	**222**
Radha	**CST**	**333**

• Make necessary adjustment to your code to let user search for data from this database by name on home page.

ANS)
Step 1: Create a Django project named "myproject."

```
django-admin startproject myproject
```

Step 2: Create an app called "myapp."

```
cd myproject
python manage.py startapp myapp
```

Step 3: Create a "home.html" file in the "myapp" app's "templates/myapp" directory.

Here's the code for "home.html":

```
<!DOCTYPE html>
<html>
<head>
    <title>My App</title>
</head>
<body>
    <h1>My App</h1>
    <h3>Enter data</h3>
    <form action="" method="get">
        <label for="name">Name:</label>
        <input type="text" name="name"><br><br>
        <button type="submit">Search</button>
    </form>
</body>
</html>
```

Step 4: Create a model named "Mydata" with attributes name, branch, and roll no. in the "myapp/models.py" file.

```
from django.db import models

class Mydata(models.Model):
    name = models.CharField(max_length=100)
    branch = models.CharField(max_length=100)
    roll_no = models.IntegerField()

    def __str__(self):
        return self.name
```

Step 5: Create a superuser with your enrollment number as the username and your name as the password.

```
python manage.py createsuperuser
```

Follow the prompts to create the superuser account.

Step 6: Log in to the Django admin portal with the superuser account and enter the following data in the "Mydata" table:

Name: Yaksh
Branch: CE
Roll No: 111

Name: Rohan
Branch: IT
Roll No: 222

Name: Radha
Branch: CST
Roll No: 333

Step 7: Make necessary adjustments to your code to let users search for data by name on the home page.

Modify the "myapp/views.py" file to handle the search functionality:

```
from django.shortcuts import render
from .models import Mydata

def home(request):
    name = request.GET.get('name')
    mydata = Mydata.objects.filter(name__icontains=name) if name else []
    return render(request, 'myapp/home.html', {'mydata': mydata})
```

Update the "myapp/urls.py" file to include the URL pattern for the home view

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

urlpatterns = [
    path('', views.home, name='home'),
]
```

19)
1. Create a Django Project named "music"
2. Create an App named 'song'
3. Create Home Page by making template home.html in 'song' App.
4. Code for 'home.html' is as below.

```
<body>
<h2> Songs </h2>
<h4> Enter Song Name </h4>
<form action="" >
<label for="search">Search for Song </label>
<input type="text" name="SearchSong" />
<button type="submit" >Search</button>
</form>
</body>
```

5. Create Model with name 'Song' with attributes songname, singers, musicdirector, year.
6. Create a superuser and using the username and password, enter the details of Song from admin panel. Details are given as below.
7. Make Necessary changes to your code to show all the songs on home page ordered by year.
8. Search the particular song using search box should show the particular searched song details after clicking search button.

ANS)
Step 1: Create a Django project named "music."

```
django-admin startproject music
```

Step 2: Create an app named "song."

```
cd music
python manage.py startapp song
```

Step 3: Create a "home.html" file in the "song" app's "templates/song" directory.

Here's the code for "home.html":

```
<!DOCTYPE html>
<html>
<head>
    <title>Songs</title>
</head>
<body>
    <h2>Songs</h2>
    <h4>Enter Song Name</h4>
    <form action="" method="get">
        <label for="SearchSong">Search for Song</label>
        <input type="text" name="SearchSong">
        <button type="submit">Search</button>
    </form>
    <h3>All Songs</h3>
    <ul>
        {% for song in songs %}
            <li>{{ song.songname }} - {{ song.singers }} ({{ song.year
}})</li>
        {% endfor %}
    </ul>
</body>
</html>
```

Step 4: Create a model named "Song" with attributes songname, singers, musicdirector, and year in the "song/models.py" file.

```
from django.db import models

class Song(models.Model):
    songname = models.CharField(max_length=100)
    singers = models.CharField(max_length=100)
    musicdirector = models.CharField(max_length=100)
    year = models.PositiveIntegerField()

    def __str__(self):
        return self.songname
```

Step 5: Create a superuser with the following command:

```
python manage.py createsuperuser
```

Follow the prompts to create the superuser account.

Step 6: Log in to the Django admin panel with the superuser account and enter the details of songs from the admin panel.

Step 7: Make necessary changes to your code to show all the songs on the home page ordered by year.

Modify the "song/views.py" file to retrieve the songs ordered by year

```
from django.shortcuts import render
from .models import Song

def home(request):
    songs = Song.objects.order_by('year')
    return render(request, 'song/home.html', {'songs': songs})
```

Step 8: Implement search functionality to display the particular searched song details.

Modify the "song/views.py" file to handle the search functionality:

```
from django.shortcuts import render, get_object_or_404
from .models import Song

def home(request):
    songs = Song.objects.order_by('year')

    search_song = request.GET.get('SearchSong')
    if search_song:
        searched_song = get_object_or_404(Song, 
songname__icontains=search_song)
        return render(request, 'song/home.html', {'songs': songs, 
'searched_song': searched_song})

    return render(request, 'song/home.html', {'songs': songs})
```

20)
DJANGO TEMPLATE ENGINE PROJECT
Task 1: Project Setup and Template Configuration
1. Task: Verify project setup and template configuration.
- Description: Confirm that the Django project and app have been created, and that template settings in `settings.py` are correctly configured.
Task 2: Create a Basic Template
2. Task: Create a basic HTML template.
- Description: Develop a simple HTML template named `hello.html` inside the app's `templates` directory, as shown in the project setup.
Task 3: Create a View to Render the Template
3. Task: Develop a view to render the template.
- Description: Create a view function named `hello_view` in the app's `views.py` that renders the `hello.html` template.
Task 4: Define a URL Pattern for the View
4. Task: Define a URL pattern for the `hello_view` in the app's `urls.py`.
- Description: Create a URL pattern that maps to the `hello_view` function, making sure it includes the `/demo/hello/` URL path.
Task 5: Configure Main URLs
5. Task: Verify main URL configuration.
- Description: Confirm that the app's URLs are included in the main project's `urls.py` correctly.
Task 6: Start the Development Server
6. Task: Run the development server.
- Description: Start the Django development server using the command `python manage.py runserver`. Verify that the server runs without errors.
Task 7: Access the Template via URL
7. Task: Access the template via its URL.
- Description: Access the template at `http://localhost:8000/demo/hello/` using a web browser or a tool like `curl`. Ensure that the template is displayed as expected, showing "Hello, Django User!"
Task 8: Modify the Template Context
8. Task: Modify the template context.
- Description: In the `hello_view`, change the value of the `name` variable in the context to a different name (e.g., "John"). Verify that the template updates accordingly.
Task 9: Template Inheritance

9. Task: Implement template inheritance .
- Description: Create a base template that includes common elements like headers and footers. Then, create a child template that extends the base template and adds content unique to the
child template.
Task 10: Template Tags
10. Task: Explore and use additional template tags
- Description: Experiment with Django's template tags (e.g., `for`, `if`, `include`) to enhance the template's functionality or appearance

ANS)
Task 1: Project Setup and Template Configuration
Ensure you have already set up your Django project and configured the template settings in the settings.py file to include the app's templates directory.

Task 2: Create a Basic Template
Create a new HTML template file named hello.html inside the templates directory of your app. This file should contain the basic HTML structure you want to display.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Hello Template</title>
</head>
<body>
    <h1>Hello, Django User!</h1>
</body>
</html>
```

Task 3: Create a View to Render the Template
In your app's views.py, create a view function named hello_view that will render the hello.html template.

```
from django.shortcuts import render

def hello_view(request):
    return render(request, 'hello.html')
```

Task 4: Define a URL Pattern for the View
In your app's urls.py file, define a URL pattern that maps to the hello_view function, ensuring it includes the /demo/hello/ URL path.

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

urlpatterns = [
    path('demo/hello/', views.hello_view, name='hello'),
]
```

Task 5: Configure Main URLs
Make sure that your app's URLs are included in the main project's urls.py file.

Task 6: Start the Development Server
Run the Django development server using the command:

```
python manage.py runserver
```

Verify that the server starts without errors.

Task 7: Access the Template via URL
Access the template in your web browser by navigating to http://localhost:8000/demo/hello/. You should see the "Hello, Django User!" message.

Task 8: Modify the Template Context
In the hello_view function, you can modify the template context to change the displayed name. For example:

```
from django.shortcuts import render

def hello_view(request):
    context = {'name': 'John'}
    return render(request, 'hello.html', context)
```

This will update the template to say "Hello, John!"

Task 9: Template Inheritance
To implement template inheritance, create a base template that includes common elements, such as headers and footers. Then, create a child template that extends the base template and adds unique content.

Here's an example:

base.html (Base Template)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}My Site{% endblock %}</title>
</head>
<body>
    <header>
        <h1>My Website</h1>
    </header>

    <nav>
        <ul>
            <li><a href="/">Home</a></li>
            <li><a href="/about/">About</a></li>
            <li><a href="/contact/">Contact</a></li>
        </ul>
    </nav>

    <main>
        {% block content %}
        {% endblock %}
    </main>

    <footer>
        &copy; 2023 My Website
    </footer>
</body>
</html>
```

child.html (Child Template)

```
{% extends "base.html" %}

{% block title %}About Us{% endblock %}
{% block content %}
    <h2>About Us</h2>
    <p>We are a company dedicated to creating amazing things.</p>
{% endblock %}

<ul>
    {% for item in items %}
        <li>{{ item }}</li>
    {% endfor %}
</ul>

{% if user.is_authenticated %}
    <p>Welcome, {{ user.username }}!</p>
{% else %}
    <p>Please log in.</p>
{% endif %}
```