Django Framework

Error/ Warning

Information

Flashback

Class Exercise

AGENDA

1. Django - New Project in Django, Basics of creating app
2. Project Files, Admin Page, Django Shell
3. Django Components - Models in Django
4. URLS and View Functions, Models with Views
5. Templates Basics, Passing data to templates

1. Django - New Project in Django, Basics of Creating App

Django - new project in Django, basics of creating app

Create a Django app

In windows Terminal navigate to your desired project directory and execute the following commands -

Step No.	Action	Command
1	Create a virtual environment	python –m venv myEnv
2	Activate virtual environment	myEnv\Sctripts\activate
3	Install Django	python –m pip install Django
4	Install Django REST framwork	python –m pip install djangorestframework
5	Create a Django project	django-admin startproject myProject
6	Create a Django app	cd myProject python manage.py startapp myApp

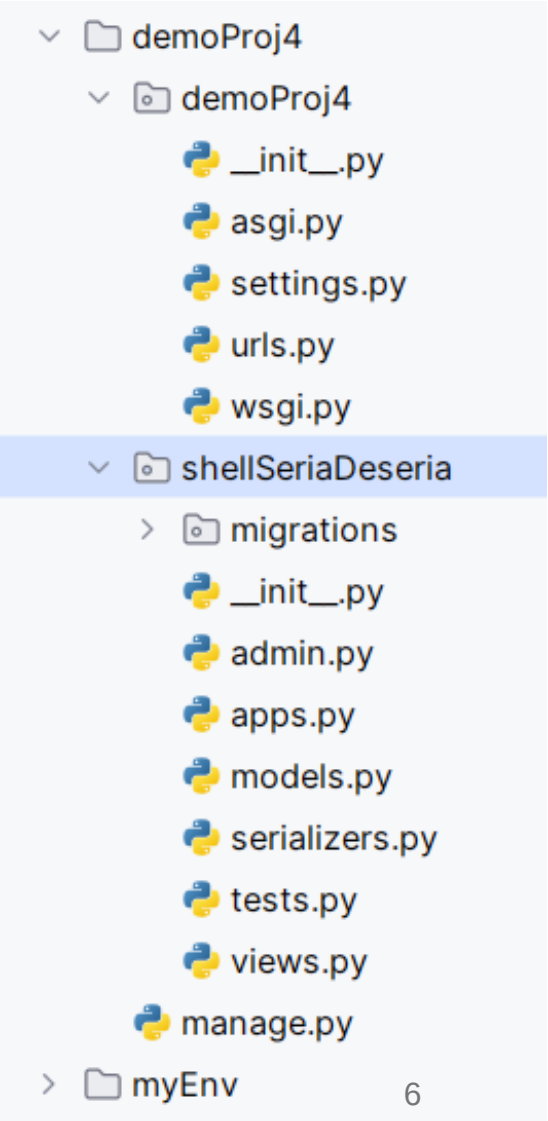
Django - new project in Django, basics of creating app

Registering the Django app

This will be directory structure after executing the all the previous commands.

Add the newly created app to the INSTALLED_APPS list in “demoProj4/settings.py”.

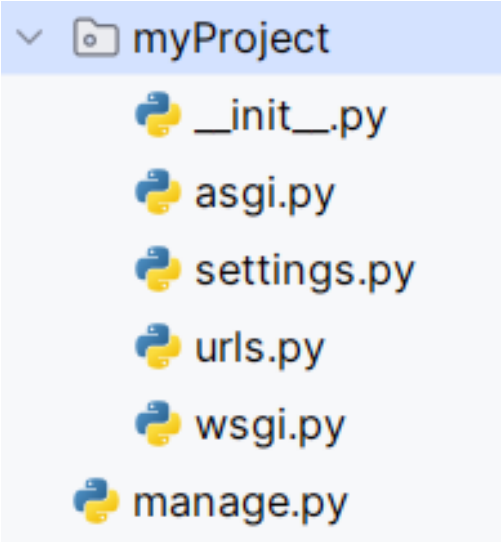
```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'shellSeriaDeseria',
]
```



2. Project Files, Admin Page, Django Shell

Project files

When you create a new Django project using the command `django-admin startproject myproject`, it generates the following file structure:

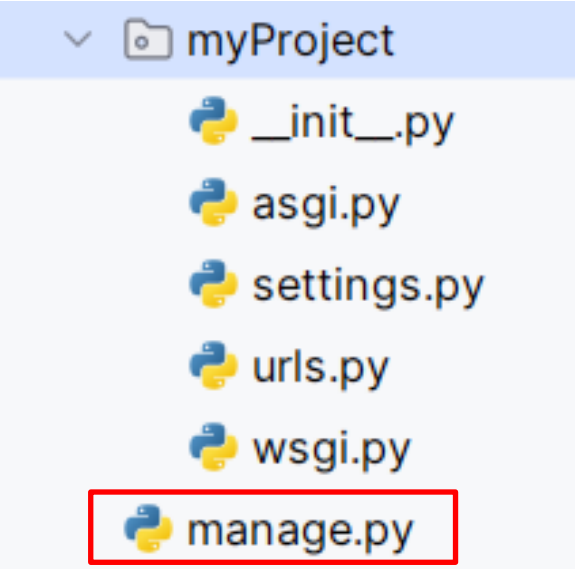


Project files, admin page, Django shell

Project files

Explanation – “manage.py”

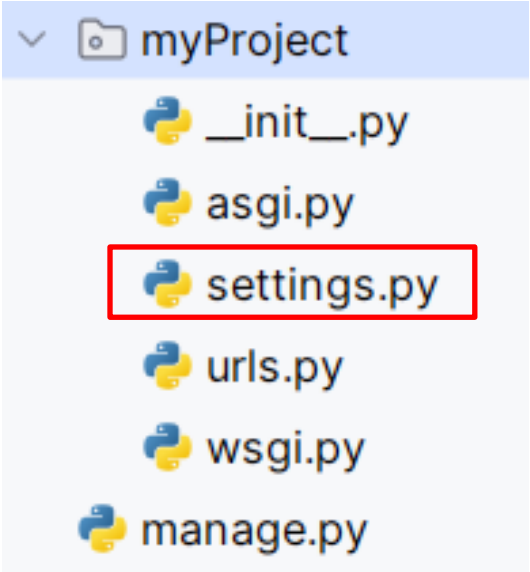
- Purpose: A command-line utility that allows to interact with Django project in various ways.
- Common Commands:
  - runserver: Starts the development server.
  - migrate: Applies database migrations.
  - makemigrations: Creates new migrations based for changes detected in models
  - createsuperuser: Creates a superuser for the admin interface.
  - shell: Opens the Django shell.



Project files

Explanation – “myProject/settings.py”

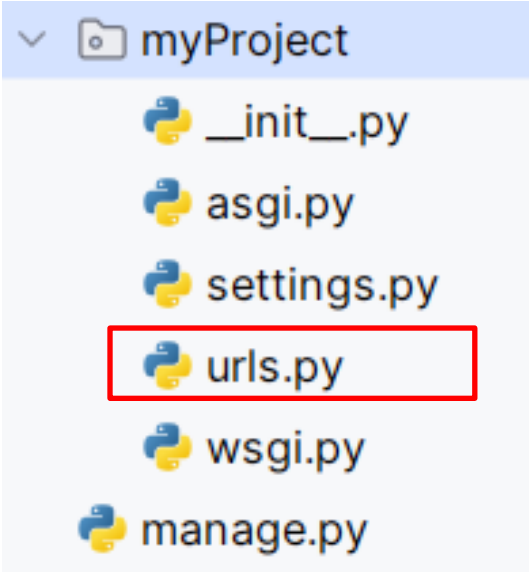
- Purpose: Contains all the configuration settings for a Django project.
- Key sections:
  - INSTALLED_APPS: A list of apps that are enabled in the project.
  - MIDDLEWARE: Middleware classes used by the application.
  - DATABASES: Configuration for the database (e.g., SQLite, PostgreSQL).
  - TEMPLATES: Configuration for template engines.
  - STATIC_URL: URL to use when referring to static files.



Project files

Explanation – “myProject/urls.py”

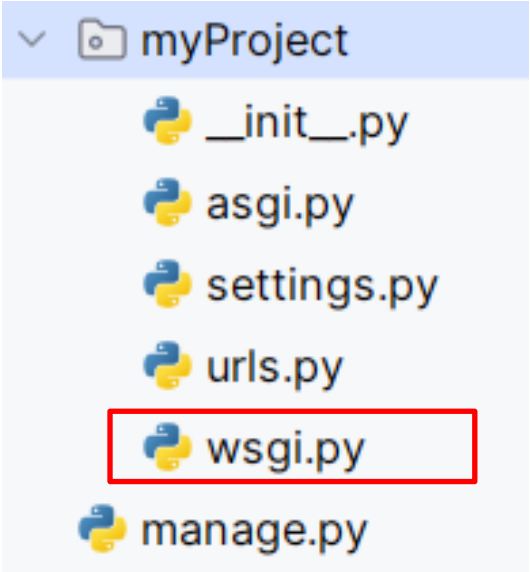
- Purpose: Defines the URL patterns for the project.
- Key points:
  - urlpatterns: A list of URL patterns mapped to view functions or classes.
  - Includes the URLs of installed apps using include.



Project files

Explanation – “myProject/wsgi.py”

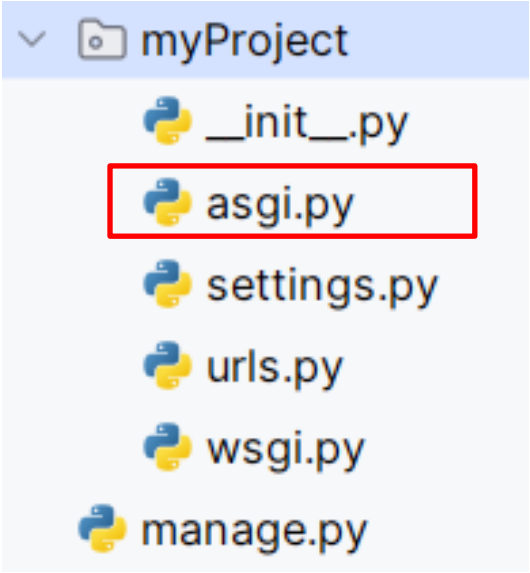
- Purpose: Acts as an entry-point for WSGI-compatible web servers to serve your project. It's used in deployment.



Project files

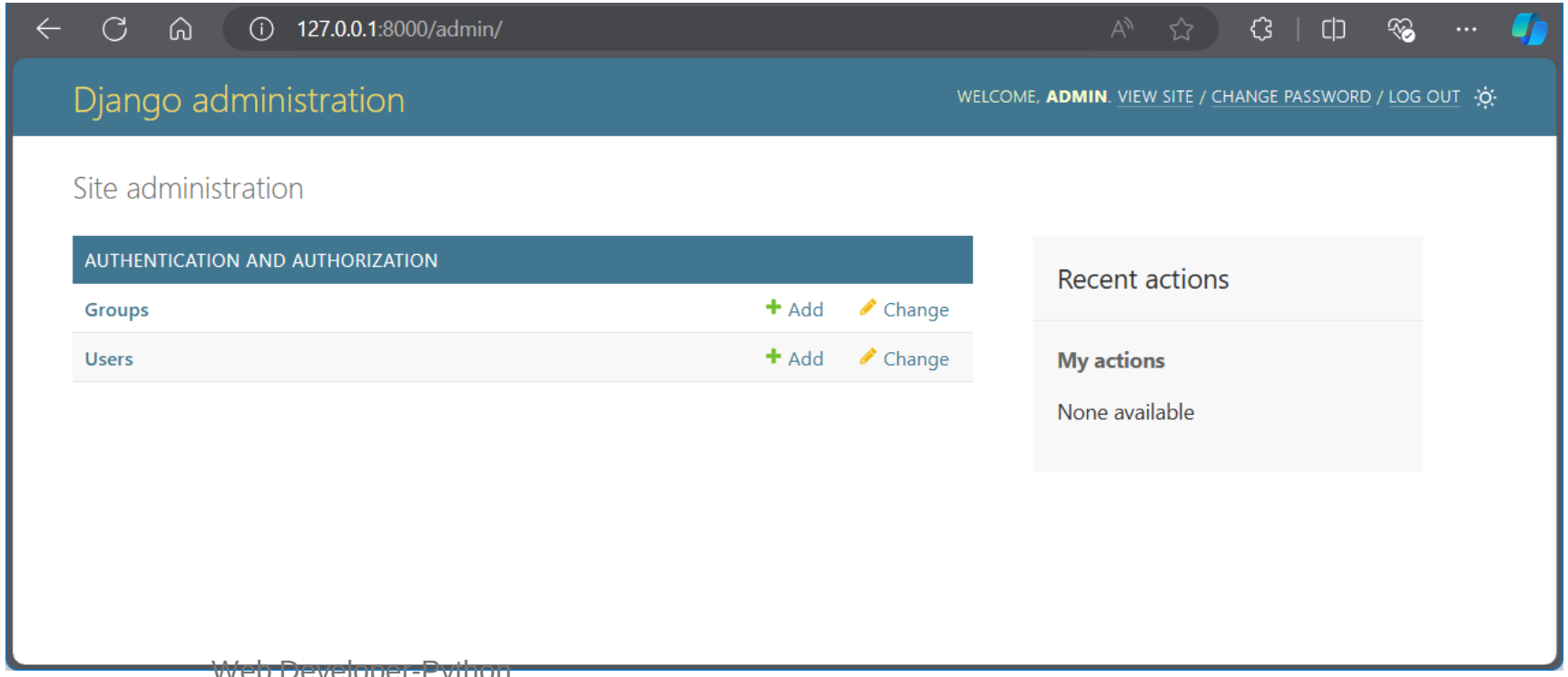
Explanation – “myProject/asgi.py”

- Purpose: Similar to wsgi.py, but for ASGI-compatible web servers, allowing for asynchronous capabilities.



Admin page

- The Django admin page is a powerful feature that comes with Django by default.
- It provides a web-based interface for managing the content of your project.



Admin page

Setting up the admin page

1. Run the database migrations to create the necessary tables for Django's built-in authentication system.

```
python manage.py makemigrations
python manage.py migrate
```

2. Create a Superuser: To access the admin page, you need to create a superuser.

```
python manage.py createsuperuser
```

3. Run the Development Server:

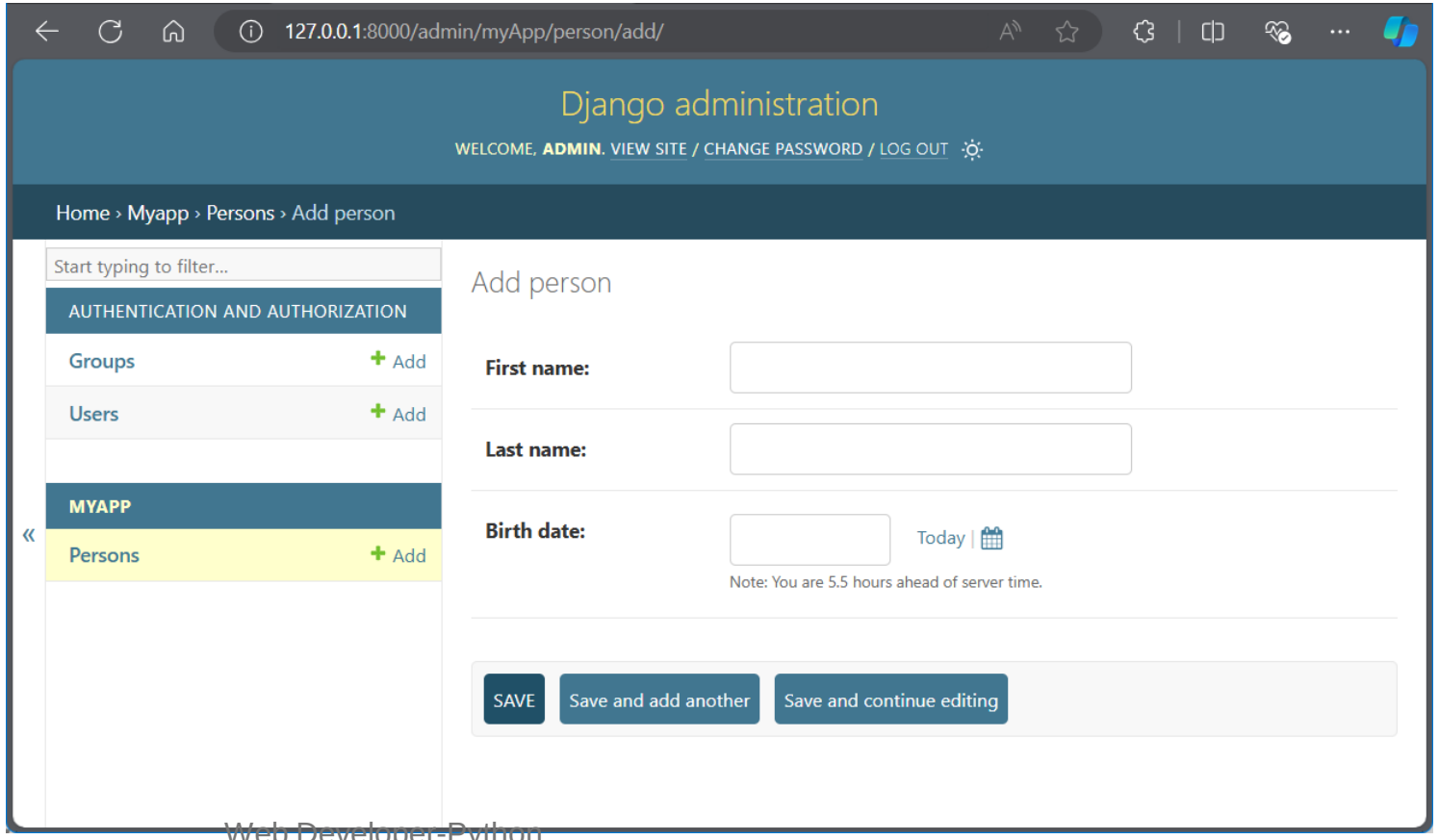
```
python manage.py runserver
```

4. Access the Admin Interface:

   Navigate to http://127.0.0.1:8000/admin and log in with your superuser credentials.

Admin page

Register and customize the admin interface for the “Person” model



Admin page

Register and customize the admin interface for the “Person” model

Step 1: Define the Person model in “myapp/models.py”.

Admin page

Register and customize the admin interface for the “Person” model

Step 2: Register the Person model in “myapp/admin.py” to make it manageable through the Django admin interface.

admin.py

```python
from django.contrib import admin
from .models import Person

# Register your models here.
admin.site.register(Person)

```

ICTACADEMY

Admin page

Register and customize the admin interface for the “Person” model

Step 3: To customize the admin interface for the “Person” model create a “PersonAdmin” class in “myapp/admin.py” and configure it to display specific fields, add search functionality, and more.

admin.py

```python
from django.contrib import admin
from .models import Person


class PersonAdmin(admin.ModelAdmin):
    list_display = ('first_name', 'last_name', 'birth_date')
    search_fields = ('first_name', 'last_name')


# Register your models here.
admin.site.register(Person)
```

Admin page

Register and customize the admin interface for the “Person” model

Step 4: To verify the Admin Interface run this command in Django shell and navigate to http://127.0.0.1:8000/admin.

```
python manage.py runserver
```

Django shell

- The Django shell is an interactive Python shell that loads your project settings and allows you to interact with your models and data directly.
- Useful for testing, debugging and performing database operations manually.
- To exit the Django shell, simply type exit() or press Ctrl+D.

Run the following command to open the Django shell -

```
python manage.py shell
```

```
Terminal    Local ×
(myEnv) PS D:\Algoritmo\3.0 Web development using Django\myProject> python manage.py shell
Python 3.10.5 (tags/v3.10.5:f377153, Jun  6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```

ICTACADEMY

Django shell

Steps to work with models in shell

Once inside the Django shell, you can import and interact with your models and perform database operations.

Step1: Assume you have a model Person defined in myapp/models.py:

models.py

```python
from django.db import models


class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
    birth_date = models.DateField()

```

Django shell

Steps to work with models in shell

Once inside the Django shell, you can import and interact with your models and perform database operations.

Step2: Run the migration commands.

```
python manage.py makemigrations
python manage.py migrate
```

Step3: Import the Person model into the Django shell:

```
from myapp.models import Person
```

Project files, admin page, Django shell

- Running `makemigrations` generates migration files based on the model definitions.
- `migrate` applies these changes to the database.

Django shell

Example 01 - Create a new record.

```
person = Person(first_name="John", last_name="Doe", birth_date="1980-01-01")
person.save()
```

Note: Repeat step4 with different values for first_name, last_name and birth_date to create multiple records.

Django shell

Example 02 - Querying to retrieve all records and print those in shell.

```
people = Person.objects.all()
for person in people:
        print(person.first_name, person.last_name)
```

Django shell

Example 03 - Querying to filter specific records.

```
people = Person.objects.filter(last_name="Doe")
for person in people:
        print(person.first_name, person.last_name)
```

Django shell

Example 04 - Querying to update a record.

```
person = Person.objects.get(id=1)
person.first_name = "Jonathan"
person.save()
```

Django shell

Example 05 - Querying to delete a records.

```
person = Person.objects.get(id=1)
person.delete()
```

3. Django Components - Models in Django

Django components - Models in Django

Model

- Django Models are a way to define the structure of your database in a declarative manner.
- They provide an abstraction layer that allows you to work with your data in a Pythonic way.

Model

Example – Create model

Models are defined in the “myProject/models.py” file of your app. Let's create a simple model for a Book.

models.py

```python
from django.db import models


class Book(models.Model):
    title = models.CharField(max_length=100)
    author = models.CharField(max_length=100)
    publication_date = models.DateField()
    isbn = models.CharField(max_length=13, unique=True)
    price = models.DecimalField(max_digits=6, decimal_places=2)

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

Django components - Models in Django

Model

Example – Use model

- You can now use the Book model to create, retrieve, update, and delete records.
- Open the Django shell to interact with the model.

Django components - Models in Django

Model

Example – Create a new Book record

```
>>> from myApp.models import Book
>>> book = Book(
...     title="Django for Beginners",
...     author="William S. Vincent",
...     publication_date="2018-12-02",
...     isbn="9781735467207",
...     price=29.99
... )
>>> book.save()
>>> 
```

Django components - Models in Django

Model

Example – Retrieve records

```
>>> # Retrieve all books
>>> books = Book.objects.all()
>>> for book in books:
...     print(book.title)
...
Django for Beginners
>>> # Retrieve a single book by its primary key (id)
>>> book = Book.objects.get(id=1)
>>> print(book.title)
Django for Beginners
>>>
```

Model

Example – Update a record

```
>>> # Update the price of the book
>>> book.price = 24.99
>>> book.save()
>>> 
```

Model

Example – Delete a record

```
>>> book.delete()
(1, {'myApp.Book': 1})
>>>
```

Django components - Models in Django

Adding more features

Django models support many features, such as relationships, custom managers, and query sets.

Example

Adding foreign key to “author” of “Book” model pointing to “Author” model.

models.py

```python
from django.db import models


class Author(models.Model):
    name = models.CharField(max_length=100)
    birthdate = models.DateField()

    def __str__(self):
        return self.name


class Book(models.Model):
    title = models.CharField(max_length=100)
    author = models.ForeignKey(Author, on_delete=models.CASCADE)
    publication_date = models.DateField()
    isbn = models.CharField(max_length=13, unique=True)
    price = models.DecimalField(max_digits=6, decimal_places=2)

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

Adding more features

Explanation

`author=models.ForeignKey(Author, on_delete=models.CASCADE)`:

This line defines an author field for the Book model.

`ForeignKey` is used to create a many-to-one relationship with the Author model.

The `on_delete=models.CASCADE`

parameter specifies that if the referenced `Author` object is deleted, all related Book objects should also be deleted.

Django components - Models in Django

TRY IT

Case Study: Managing a Library System with Django

Objective

- Create a simple Django project to manage a library system.
- The system should include a model for books and allow you to add, update, and view books using the Django admin interface and the Django shell.

Django components - Models in Django

TRY IT

Case Study: Managing a Library System with Django

Instruction for model.py

- In “library_app/models.py” define a model Book with the following fields:
  - title (CharField, max_length=100)
  - author (CharField, max_length=50)
  - publication_date (DateField)
  - isbn (CharField, max_length=13)

Django components - Models in Django

TRY IT

Case Study: Managing a Library System with Django

Instruction to customize admin interface

- Customize the admin interface to display the title, author, and publication_date in the list view.
- Add search functionality for the title and author fields.
- Make interface to add new books.

Django components - Models in Django

TRY IT

Case Study: Managing a Library System with Django

Instruction to interact with the database

- Open the Django shell.
- Create a few Book instances and save them to the database.
- Query the database to retrieve all books.
- Update the author of one of the books.
- Delete one of the books.

4. URLS and View Functions, Models with Views

Configuring URLs

Update the main URL configuration to include the URLs for new app in “myProject/urls.py”.

urls.py

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('myapp/', include('myApp.urls')),  # Include myapp URLs
]

```

The main project is now set up to route requests to the “myApp” app.

Note: Before configuring URLs create a Django App named – “myApp”.

URLS and View Functions, Models with Views

Add URLs

Create a URL configuration for maApp in “myApp/urls.py”. This tells Django how to handle requests for app.

urls.py

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


urlpatterns = [
    # URL pattern for the index view
    path('', views.index, name='index'),
]

```

It is a basic URL routing setup for myApp.

URLS and View Functions, Models with Views

Creating view functions

A view function processes requests and returns responses. Code for this should be added in “myApp/views.py” file.

views.py

```python
from django.shortcuts import render

# Create your views here.
from django.http import HttpResponse


def index(request):
    return HttpResponse("Hello, world. You're at the myApp index.")

```

This view returns a simple HTTP response.

Creating model

Models are Python classes that map to database tables. Let's create add a "Item" model in "myApp/models.py".

```
class Item(models.Model):
    name = models.CharField(max_length=200)
    description = models.TextField()

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

Apply migrations

```
python manage.py makemigrations
python manage.py migrate
```

Model named "Item" with two fields - name and description and the necessary database table has been created.

URLS and View Functions, Models with Views

Connecting models to views

- Display data from your models in your views.
- This involves querying the database and passing data to templates.
- Create a file – “myapp/templates/index.html”

URLS and View Functions, Models with Views

Connecting models to views

Create the template - index.html

index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Item List</title>
</head>
<body>
    <h1>Items</h1>
    <ul>
        {% for item in items %}
            <li>{{ item.name }}: {{ item.description }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

URLS and View Functions, Models with Views

Update the view

File - myApp/index.html

views.py

```python
from django.shortcuts import render
from .models import Item


def index(request):
    items = Item.objects.all()
    return render(request, template_name: 'index.html', context: {'items': items})

```

This view now queries the “Item” model and passes the data to a template for rendering.

Configuring templates directory

Ensure Django knows where to find templates by configuring the templates directory in “myProject/settings.py”.

settings.py

```python
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'myApp/templates')],   # Add this line
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

URLS and View Functions, Models with Views

Running the development server

```
python manage.py runserver
```

Navigate to - http://127.0.0.1:8000/myapp/

URLS and View Functions, Models with Views

Summary of file changes

- myProject/urls.py: Included app URLs.
- myApp/urls.py: Defined URL patterns for the app.
- myApp/views.py: Created view functions.
- myApp/models.py: Defined a model.
- myApp/templates/index.html: Created a template.
- myProject/settings.py: Configured the templates directory.

5. Templates Basics, Passing Data to Templates

Templates basics, passing data to templates

Create template

- Start a Django project – “DjangoTemplate”
- Start a Django app – “myApp”.
- Create a directory named “templates” inside the “myApp” directory.
- Create a basic HTML template file – “index.html” inside the “myApp/templates” directory.

index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Django Template Parameter Passing</title>
</head>
<body>
    <h1>Welcome to My Django App</h1>
    <p>{{ message }}</p>
</body>
</html>
```

Templates basics, passing data to templates

Create a view to render the template

In “myApp/views.py” create a view function that renders the “index.html” template and passes data to it.

views.py

```python
from django.shortcuts import render


# Create your views here.
def index(request):
    context = {
        'message': 'Hello, this is a message from the view!'
    }
    return render(request, template_name: 'index.html', context)

```

Templates basics, passing data to templates

Configure URL routing

In “myApp/urls.py” create a URL pattern for the view.

urls.py

```python
# myapp/urls.py
from django.urls import path
from .views import index

urlpatterns = [
    path('', index, name='index'),
]

```

Templates basics, passing data to templates

Configure URL routing

Include the “myApp/urls.py” in the main project “myproject/urls.py”

urls.py

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

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

Templates basics, passing data to templates

Run the Django development server

```
python manage.py runserver
```

Navigate to - http://127.0.0.1:8000/

Templates basics, passing data to templates

Passing a dictionary

Modify the view - “myApp/views.py” to pass a dictionary

views.py

```python
from django.shortcuts import render


# Create your views here.
def index(request):
    context = {
        'message': 'Hello, this is a message from the view!',
        'person': {
            'name': 'John Doe',
            'age': 30,
            'city': 'New York'
        }
    }
    return render(request, template_name: 'index.html', context)
```

Templates basics, passing data to templates

ICTACADEMY

- Update the template to display the dictionary items

index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Django Template Parameter Passing</title>
</head>
<body>
    <h1>Welcome to My Django App</h1>
    <p>{{ message }}</p>
    <p>Name: {{ person.name }}</p>
    <p>Age: {{ person.age }}</p>
    <p>City: {{ person.city }}</p>
</body>
</html>
```

Templates basics, passing data to templates

ICTACADEMY

- Update the template to display the dictionary items

index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Django Template Parameter Passing</title>
</head>
<body>
    <h1>Welcome to My Django App</h1>
    <p>{{ message }}</p>
    <p>Name: {{ person.name }}</p>
    <p>Age: {{ person.age }}</p>
    <p>City: {{ person.city }}</p>
</body>
</html>
```

Templates basics, passing data to templates

- Update the template to display the dictionary items

Templates basics, passing data to templates

TRY IT

Case Study: Django Blog Application

Task 1: Set Up the Django Project and Application.

Task 2: Define the “Post” Model.

- title: CharField with a max length of 200.
- content: TextField.
- created_at: DateTimeField with auto_now_add=True

Task 3: Create Sample posts through the Django admin interface.

Task 4: Create Views for the post.

Task 5: Configure URL Routing.

Task 6: Create Templates.

Task 7: Run the Development Server.

Question?

Thank you