Project – 10

Django

Submitted By:

Nishigandha Patil

INDEX

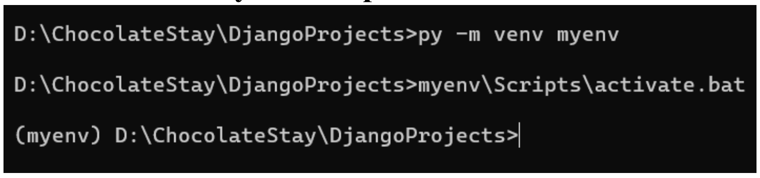
INTRODUCTION TO DJANGO

Django is a Python-based back-end server-side web framework that allows you to quickly create efficient web applications. It provides built-in features for everything including Django Admin Interface, default database – SQLlite3, etc. It follows the **Model-View-Template** (MVT) pattern.

➢ **Why Django:**

1. **Rapid Development:** Django provides a high-level, clean, and pragmatic design that speeds up the development process.
2. **Built-in Features:** It comes with built-in features for database models, admin interface, authentication, and more, reducing the need for external libraries.
3. **Security:** Django prioritizes security, offering protection against common web vulnerabilities and promoting best practices.
4. **Scalability:** Suitable for projects of all sizes, from small applications to large, complex systems.
5. **Community and Ecosystem:** Django has a vibrant and supportive community, extensive documentation, and a rich ecosystem of reusable apps.
6. **Versatility:** Works with various databases, supports different deployment options, and can be used for different types of web applications.

➢ **Prerequisites:**

1. **Install Python:** Go to https://www.python.org/downloads/ download latest version.
2. **PIP:** Use a Python’s Package Manager to install any packages/frameworks/modules.
3. **Virtual environment (Optional but Recommended):** It's good practice to work within a virtual environment to isolate your project dependencies.
   Create venv: python -m venv myenv
   Activate venv: myenv\Scripts\activate
   
   ```
   D:\ChocolateStay\DjangoProjects>py -m venv myenv

   D:\ChocolateStay\DjangoProjects>myenv\Scripts\activate.bat

   (myenv) D:\ChocolateStay\DjangoProjects>
   ```
4. **Install Django:** You can install Django globally as well as in virtual environment using the command: **pip install django**

➢ **Django Model-View-Template (MVT) pattern:**

1. **Model:** Provides data from the database.
2. **View:** A request handler that returns the relevant template and content - based on the request from the user.
3. **Template:** A text file (like an HTML file) containing the layout of the web page, with logic on how to display the data.

User request a webpage (URL). Request goes to the views and then it fetches data from model and placed that data into templates then it produces the final HTML output, which is sent as a response to the user's browser.

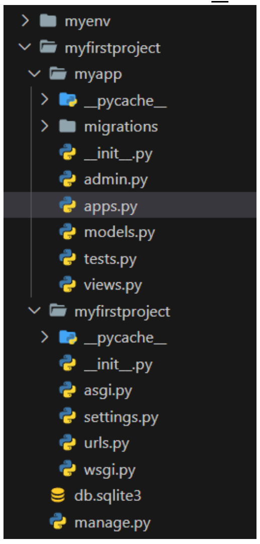
CREATE PROJECT

Step 1: Create Django Project
Open VS code → Open Folder → Open Terminal → Use command: **django-admin startproject "project_name"** (Example: django-admin startproject myfirstproject)

Step 2: Create Django App
Terminal → **python manage.py startapp "app_name"**
(Example: python manage.py startapp myapp.

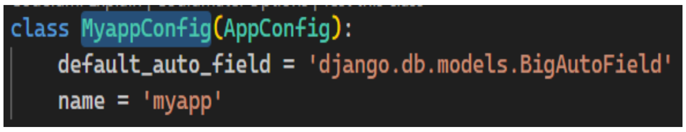
Step 3: Run Server
Terminal → python manage.py runserver → It will create **db.sqlite3** file.

Step 4: App Configuration
Open 'apps.py' → Copy Class name **"MyappConfig"** → Go to settings.py of your project → In INSTALLED_APPS add 'myapp. apps. MyappConfig'



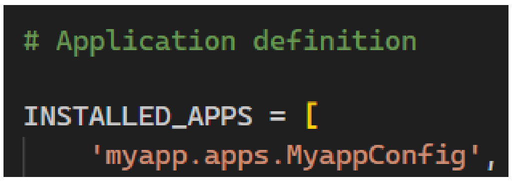
```
> myenv
v myfirstproject
  v myapp
    > __pycache__
    > migrations
      __init__.py
      admin.py
      apps.py
      models.py
      tests.py
      views.py
  v myfirstproject
    > __pycache__
      __init__.py
      asgi.py
      settings.py
      urls.py
      wsgi.py
    db.sqlite3
    manage.py
```

Apps.py



```
class MyappConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'myapp'
```

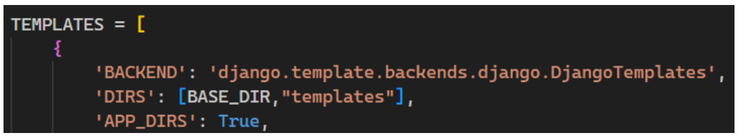
Settings.py: *app_name.apps.class_name*



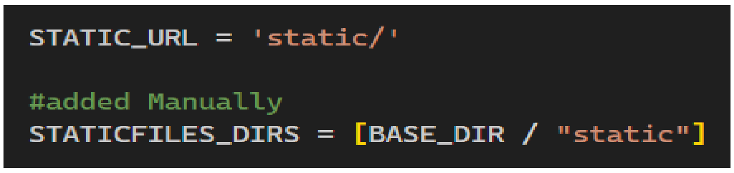
```
# Application definition

INSTALLED_APPS = [
    'myapp.apps.MyappConfig',
```

Step 5: Templates and Static DIR
Create templates and static folders in your Django Project → Open 'settings.py' and add **DIRS** for both TEMPLATES and STATIC.



```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR,"templates"],
        'APP_DIRS': True,
```



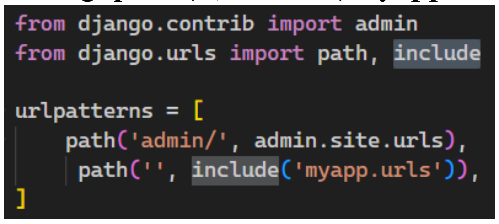
```
STATIC_URL = 'static/'

#added Manually
STATICFILES_DIRS = [BASE_DIR / "static"]
```

Create templates for your project in templates folder (e.g. index.html)
Create static file in static folder (e.g. style.css)

Step 6: Project Configuration

Go to **urls.py** file of your project (myfirstproject) → import **include** → Add app's URLS by writing: **path ('', include('myapp.urls')),** → Copy the entire code.



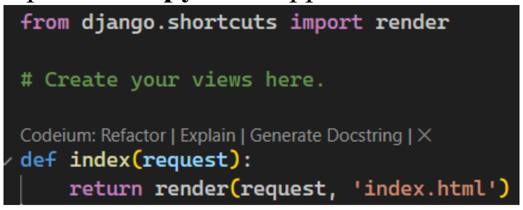
```
from django.contrib import admin
from django.urls import path, include

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

Create **urls.py** file in your **app** directory (myapp) → Paste the copied code.

Step 7: Create views

Open **views.py** from app → Create function to render **index.html** from templates folder.



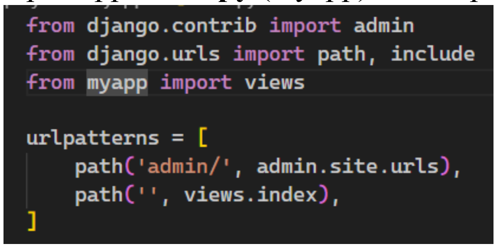
```
from django.shortcuts import render

# Create your views here.

Codeium: Refactor | Explain | Generate Docstring | ×
def index(request):
    return render(request, 'index.html')
```

Step 8: Create URLs

Open app's **urls.py** (myapp) → add path for index view- **path ('' , views.index).**

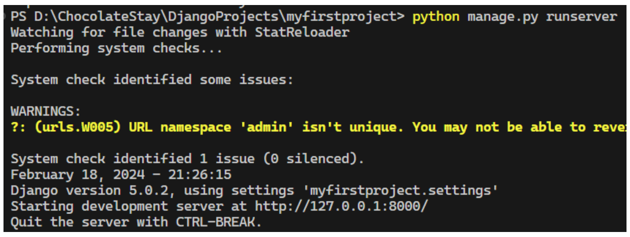


```
from django.contrib import admin
from django.urls import path, include
from myapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.index),
]
```

Step 9: Run server

python manage.py runserver → Follow the URL (http://127.0.01:8000/) → **index.html** file will render on browser.

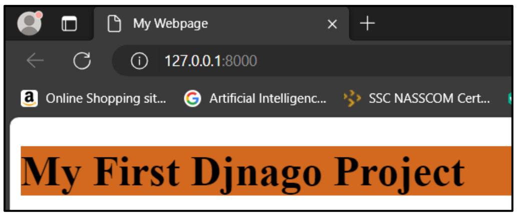


```
PS D:\ChocolateStay\DjangoProjects\myfirstproject> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified some issues:

WARNINGS:
?: (urls.W005) URL namespace 'admin' isn't unique. You may not be able to reve

System check identified 1 issue (0 silenced).
February 18, 2024 - 21:26:15
Django version 5.0.2, using settings 'myfirstproject.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```



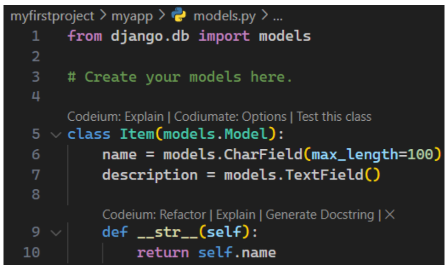
My First Djnago Project

MODELS

A Django model is a table in your database.

Step 1: Create Model

Go to **models.py** of your app (myapp) → Add an 'Item' table by creating a Item class, and describe the table fields in it:



myfirstproject > myapp > models.py > ...

```python
from django.db import models

# Create your models here.

Codeium: Explain | Codiumate: Options | Test this class
class Item(models.Model):
    name = models.CharField(max_length=100)
    description = models.TextField()

    Codeium: Refactor | Explain | Generate Docstring | ×
    def __str__(self):
        return self.name
```

Step 2: Create Database Tables

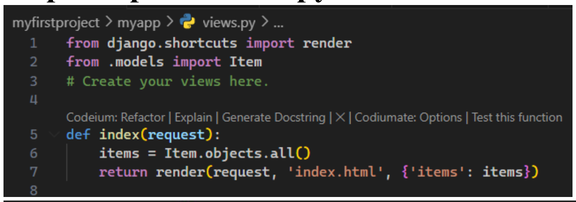
Run the following commands to apply your models and create database tables:

python manage.py makemigrations

python manage.py migrate

Django creates a file describing the changes and stores the file in the /migrations/ folder.

Step 3: Update views.py and index.html

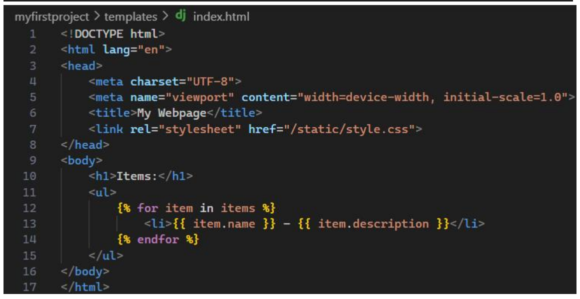


myfirstproject > myapp > views.py > ...

```python
from django.shortcuts import render
from .models import Item
# Create your views here.

Codeium: Refactor | Explain | Generate Docstring | × | Codiumate: Options | Test this function
def index(request):
    items = Item.objects.all()
    return render(request, 'index.html', {'items': items})

```

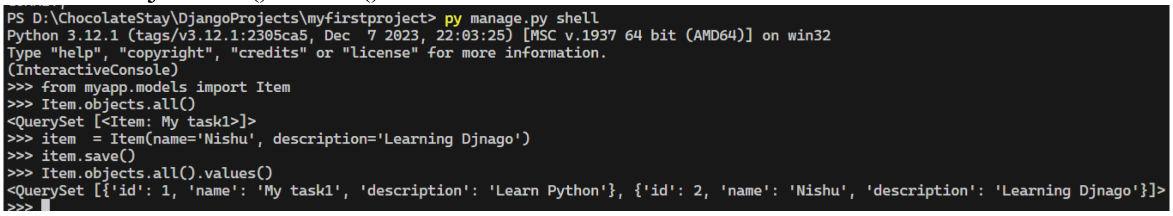


myfirstproject > templates > index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My Webpage</title>
    <link rel="stylesheet" href="/static/style.css">
</head>
<body>
    <h1>Items:</h1>
    <ul>
        {% for item in items %}
            <li>{{ item.name }} - {{ item.description }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

Step 4: Add Records by using following commands:

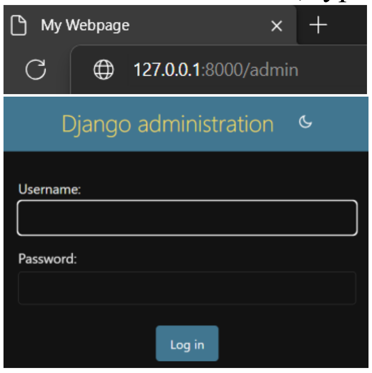
- py manage.py shell
- from myapp.models import Item
- Item.objects.all()
- item = Item(name='Nishu', description='Learning Djnago')
- item.save()
- Item.objects.all().values()



```
PS D:\ChocolateStay\DjangoProjects\myfirstproject> py manage.py shell
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023, 22:03:25) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from myapp.models import Item
>>> Item.objects.all()
<QuerySet [<Item: My task1>]>
>>> item  = Item(name='Nishu', description='Learning Djnago')
>>> item.save()
>>> Item.objects.all().values()
<QuerySet [{'id': 1, 'name': 'My task1', 'description': 'Learn Python'}, {'id': 2, 'name': 'Nishu', 'description': 'Learning Djnago'}]>
>>>
```

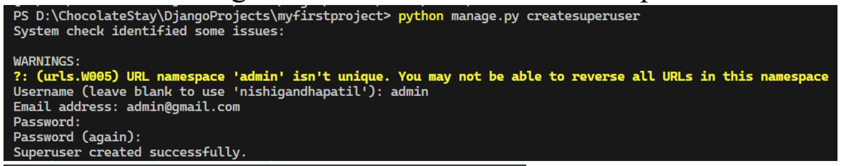
DJANGO ADMIN

Django Admin is a really great tool in Django, it is actually a CRUD* user interface of all your models!
To enter the admin user interface, start the server: python manage.py runserver
In the browser window, type 127.0.0.1:8000/admin/ in the address bar.



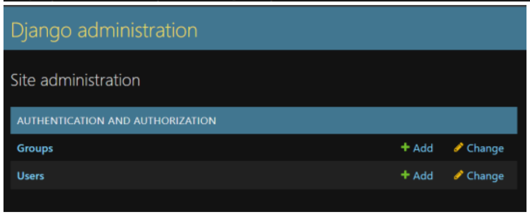
➢ Create User:

Use command: **python manage.py createsuperuser** → Enter username, email, password. → Run Server → Log In with created username and password on admin page.



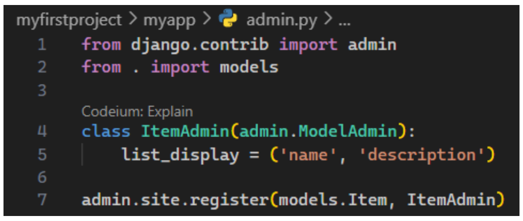
```
PS D:\ChocolateStay\DjangoProjects\myfirstproject> python manage.py createsuperuser
System check identified some issues:

WARNINGS:
?: (urls.W005) URL namespace 'admin' isn't unique. You may not be able to reverse all URLs in this namespace
Username (leave blank to use 'nishigandhapatil'): admin
Email address: admin@gmail.com
Password:
Password (again):
Superuser created successfully.
```



➢ Include Models:

Go to admin.py → Add following code→ Go to admin site, you will see items model there → Click on it to see records.

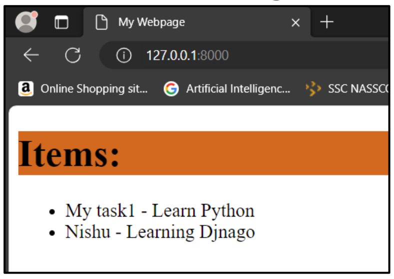


```python
from django.contrib import admin
from . import models

class ItemAdmin(admin.ModelAdmin):
    list_display = ('name', 'description')

admin.site.register(models.Item, ItemAdmin)
```

➢ Final HTML Page:



Items:

- My task1 - Learn Python
- Nishu - Learning Djnago

DJANGO SYNTAX

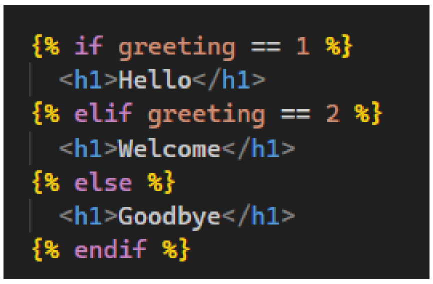
- **Template Variables**

In Django templates, you can render variables by putting them inside {{ }} brackets:

- **Template Tags**

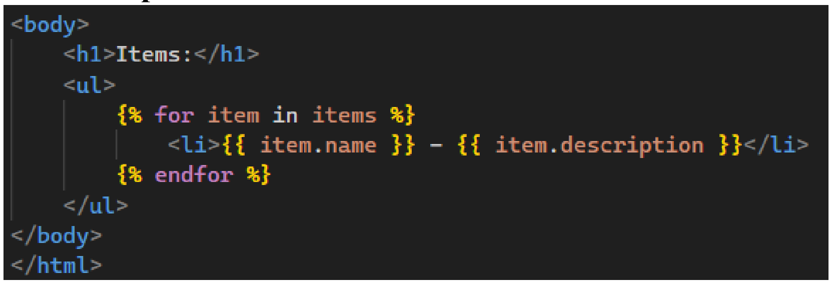
In Django templates, you can perform programming logic like executing **if** statements and **for** loops.
These keywords, **if** and **for**, are called **"template tags"** in Django.
To execute template tags, we surround them in **{% %}** brackets.

- **if else:**



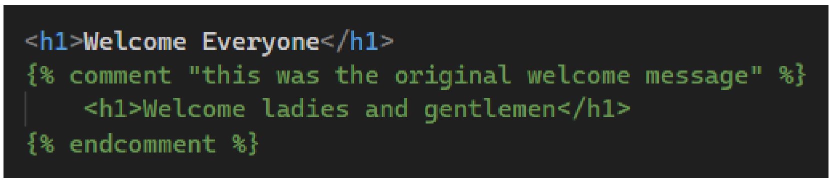
```
{% if greeting == 1 %}
  <h1>Hello</h1>
{% elif greeting == 2 %}
  <h1>Welcome</h1>
{% else %}
  <h1>Goodbye</h1>
{% endif %}
```

- **for loop:**



```
<body>
    <h1>Items:</h1>
    <ul>
        {% for item in items %}
            <li>{{ item.name }} - {{ item.description }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

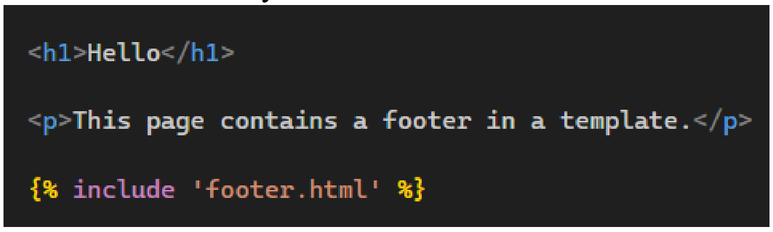
- **Comments:**



```
<h1>Welcome Everyone</h1>
{% comment "this was the original welcome message" %}
    <h1>Welcome ladies and gentlemen</h1>
{% endcomment %}
```

- **Include**

The include tag allows you to include a template inside the current template.
This is useful when you have a block of content that is the same for many pages.



```
<h1>Hello</h1>

<p>This page contains a footer in a template.</p>

{% include 'footer.html' %}
```

➢ **Summary:**

Actions	**Commands**
Create project	django-admin startproject “project_name”
Create app	python manage.py startapp “app_name”
Make migrations	python manage.py makemigrations python manage.py migrate
Run server	python manage.py runserver
Create user	python manage.py createsuperuser
Open shell	python manage.py shell

➢ **Conclusion:**

Django is a high-level web framework for Python that simplifies web development by providing a structured and modular approach. It follows the Model-View-Template (MVT) pattern, emphasizing clean code organization and separation of concerns. With a robust Object-Relational Mapping (ORM) system, Django enables developers to define data models, interact with databases, and rapidly build web applications. Its built-in features, including an admin interface, security measures, and a versatile template system, contribute to faster development and code efficiency. Supported by a vibrant community and extensive documentation, Django is a popular choice for developers building scalable and maintainable web applications.