Django Framework

Virtual Environment:

A virtual environment is a self-contained directory that houses the Python interpreter and all the packages required for a specific project. Virtual environments are used to isolate different projects from one another and to manage dependencies effectively.

Created from following command:

```
python -m venv <environment_name>
```

Example:

```
python -m venv venv
```

Activating virtual environment

```
venv\Scripts\activate
```

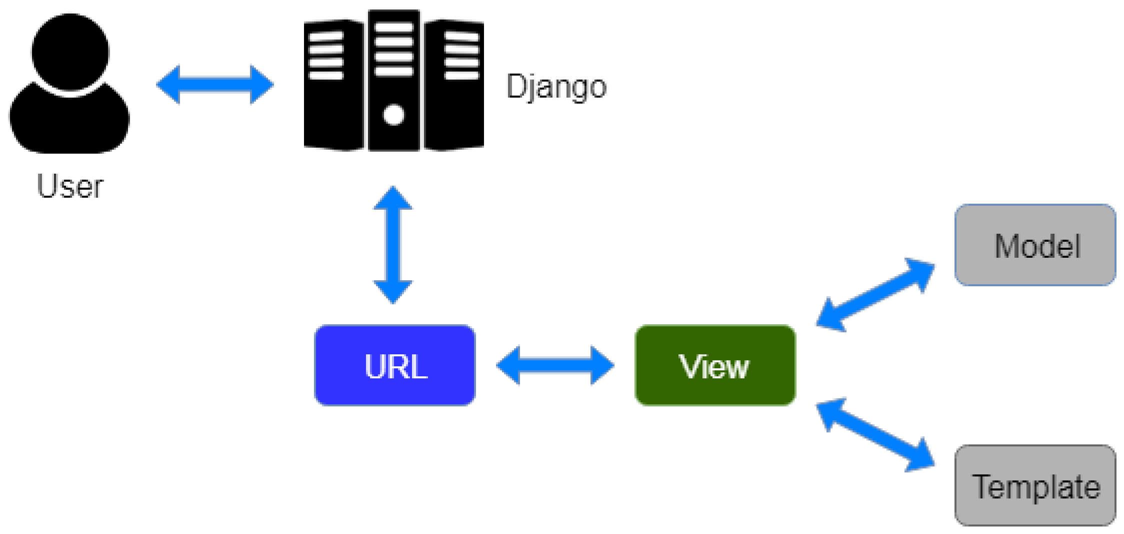
PIP

PIP (Python Package Index) is a package manager for Python. It's used to install and manage packages, which are sets of code that can be used in your projects. PIP makes it easy to install, upgrade, and remove packages, and also manages package dependencies.

Command	Explanation
pip install package_name	Used to install package
pip freeze	Used to check the packages installed
pip freeze > requirements.txt	Export package installed to requirements.txt file
pip freeze -r requirements.txt	Install the packages defined in requirements.txt file

MVT Architecture

In Django, the architectural pattern used is called the Model-View-Template (MVT) architecture. Let's look at each component of the MVT architecture and their relationships:



Model:

The Model represents the data and the business logic of the application. It defines the structure of the database tables and the data access methods. In Django, models are Python classes that inherit from django.db.models.Model. Each model class represents a database table, and its attributes define the table fields.

View:

The View handles the presentation logic of the application. It receives user requests, processes them, and returns an appropriate response. In Django, views are Python functions or classes responsible for handling HTTP requests and returning HTTP responses. Views interact with the Model to fetch data and process it as required before passing it to the Template for rendering.

Template:

The Template defines the presentation layer of the application. It is responsible for rendering the data received from the View in a format suitable for the user interface. Templates are

HTML files with additional template tags and filters provided by Django to display dynamic data.

Relationships in MVT Architecture:

The View receives an HTTP request from the user and processes it. It interacts with the Model to fetch the required data from the database.

- The Model represents the data and business logic. It interacts with the database to store and retrieve data.
- The Template receives the processed data from the View and renders it to produce an HTML response.
- The View passes the rendered HTML response back to the user's browser.

Start Django Project

- First lets create a new folder with name: django
- Now open terminal/cmd and go to that directory using cd command: cd django

Step 1: Create and activate virtual environment

- **python -m venv venv**
- **venv\Scripts\activate**

* Note: your command prompt should be similar to this:

```
(venv) C:\Users\user\Desktop\django>
```

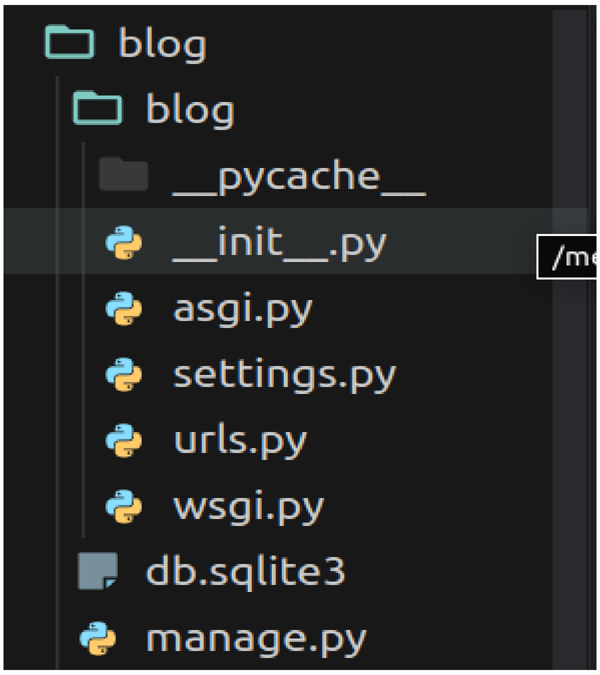
Step 2: Install django in virtual environment:

```
(venv) C:\Users\user\Desktop\django>pip install django
```

Step 3: Create Django Project:

- `(venv) C:\Users\user\Desktop\django>django-admin startproject blog`

This command will create a folder with the name blog. Your django folder will have following structure:



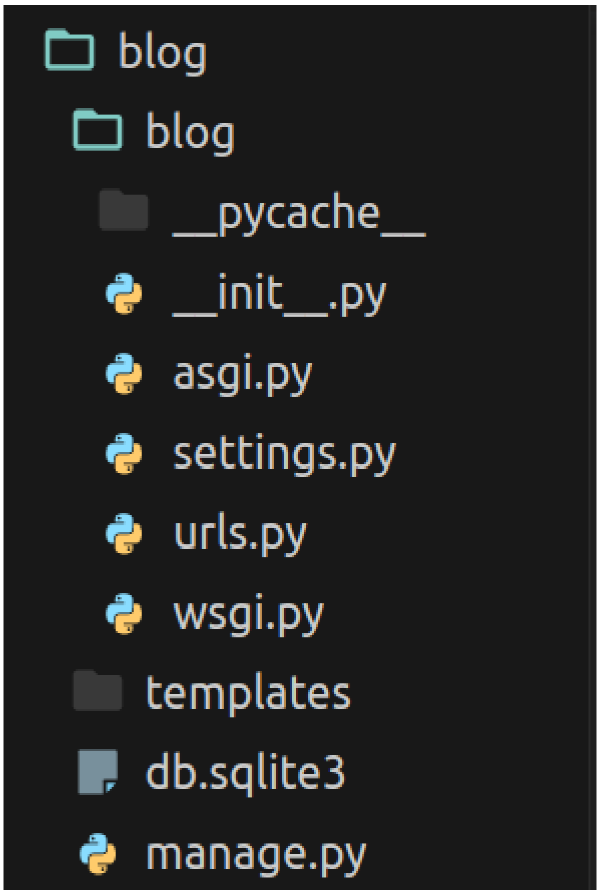
```
blog
  blog
    __pycache__
    __init__.py
    asgi.py
    settings.py
    urls.py
    wsgi.py
  db.sqlite3
  manage.py
```

Step 4: Run project

- Initially you will be in django folder: `(venv) C:\Users\user\Desktop\django>`
- Goto project folder blog. (venv) C:\Users\user\Desktop\django> cd blog
- To run the project run following command:

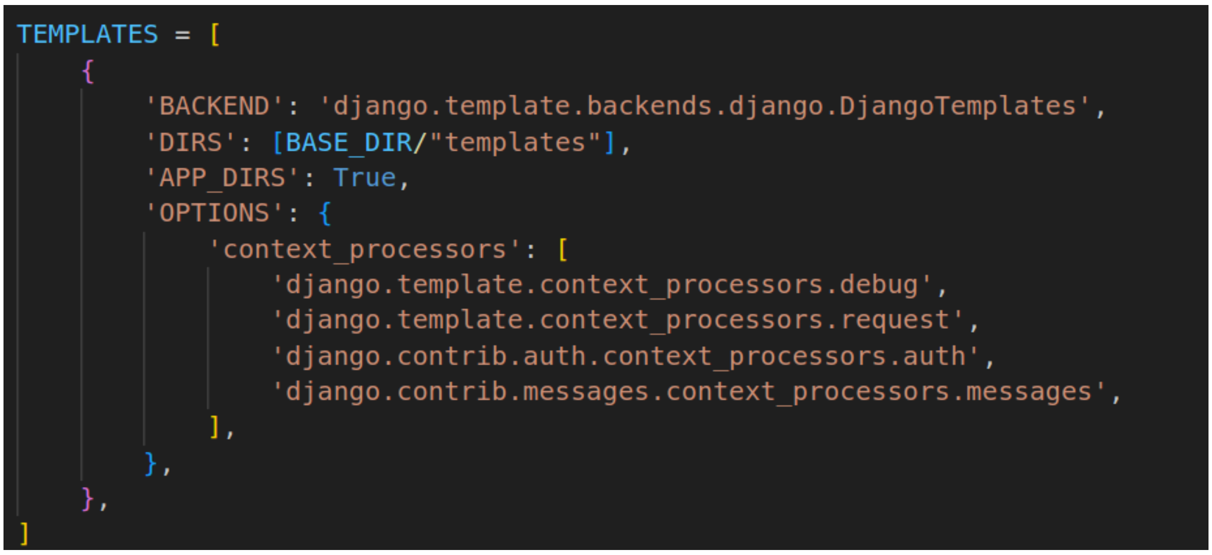
```
(venv) C:\Users\user\Desktop\django\blog> python manage.py runserver
```

Step 5: Create templates folder inside the blog folder:


```
blog
  blog
    __pycache__
    __init__.py
    asgi.py
    settings.py
    urls.py
    wsgi.py
  templates
  db.sqlite3
  manage.py
```

Step 6: Configure settings.py file for templates

- Goto settings.py file and search for TEMPLATES variable
- Search for dirs keys and add value **[BASE_DIR/ “templates”]** in the dirs key.

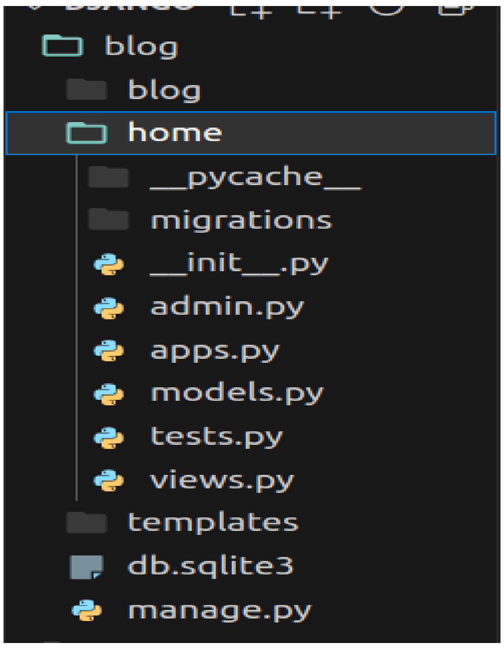

```python
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR/"templates"],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Step 7: Create an app name home

Generally App in django refers to the feature of the project for example Post feature, Comment feature, and more. App is used to separate the feature such that it can be used for other projects also. In order to create the app in django, you first need to be inside the django project.

```
(venv) C:\Users\user\Desktop\django\blog> python manage.py startapp home
```

This will create the folder named **home** inside the blog project. The home folder will consist of:



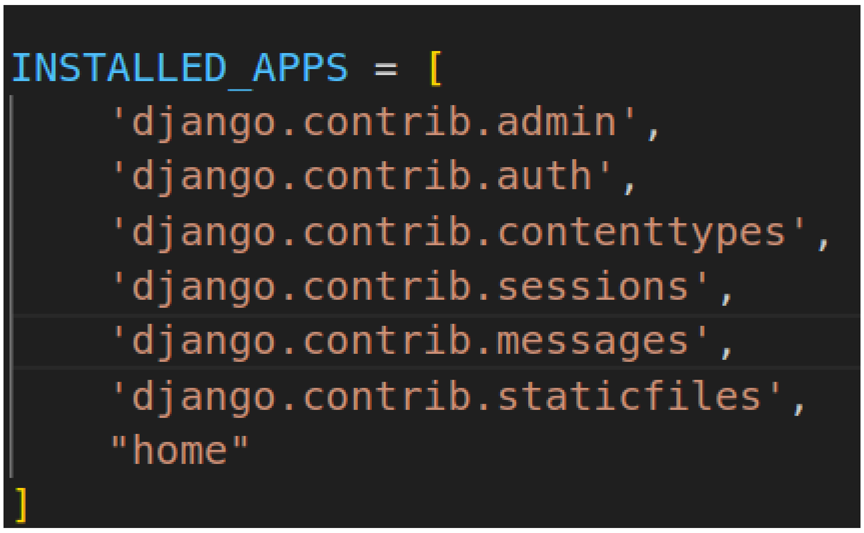
```
blog
  blog
  home
    __pycache__
    migrations
    __init__.py
    admin.py
    apps.py
    models.py
    tests.py
    views.py
  templates
  db.sqlite3
  manage.py
```

In models.py, we will create Models that will interact with databases.

In admin.py, we will configure the admin site of the django

In views.py, we will create views that will render the template to the browser.

Note: For every app you create make sure you list



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

Step 8: Create view inside the home app views.py

- First create an index.html file inside the templates folder you created at step 5.
- Then create a view inside view.py of home app/folder.

```python
def home(request):

    return render(request,"index.html")
```

Step 9: Add url in urls.py of the blog/blog folder.

- First import views you have created in home/views.py file inside blog/blog/urls.py
  `from home.views import home`
- Now add path in urlpatterns list of blog/blog/urls.py

```python
urlpatterns = [

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

    path("",home,name="home")

]
```

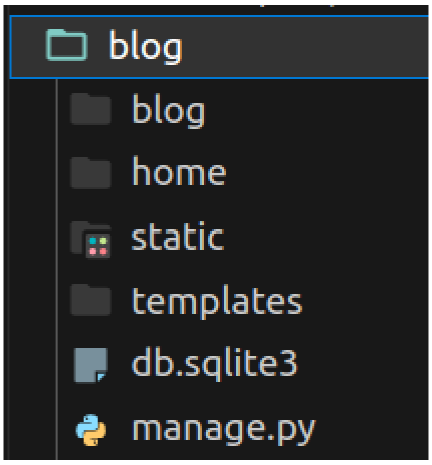
- Now, run the server using:

  ```
  (venv) C:\Users\user\Desktop\django\blog> python manage.py runserver
  ```

Static Files:

In django static files refers to the external css, js, image and other media files. These files are generally separated from templates. In order to include these files in our template we need to configure **settings.py** file of our project. Generally we create static folder in the project.

Step 1: Create static folder in project



In settings.py find STATIC_URL='/static/' and add following code after that line

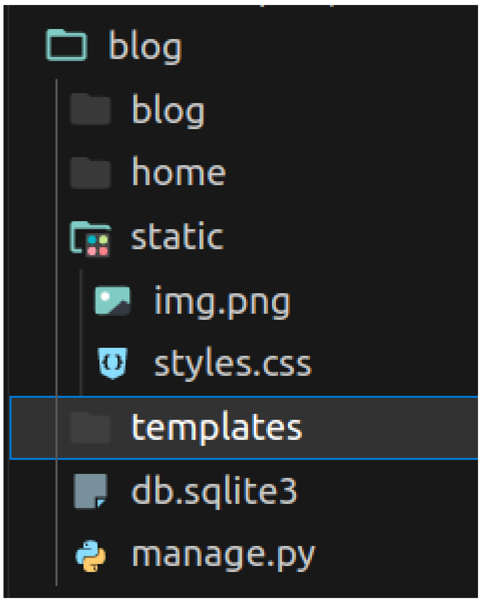
```
STATIC_URL = 'static/'

STATICFILES_DIRS = [BASE_DIR/"static"]
```

- In order to use static files in template we need to add **{% load static %}** at top of the template.
- To insert the static file in the template we need to use **{% static 'relative/path/to/static/file' %}**

Example:

Create styles.css inside static folder



Add following code to the html file for inserting styles.css file styling /design in that html file.

```
<link rel="stylesheet" href="{% static 'style.css' %}"/>
```

Template Tags and Filters

In Django, template tags and filters are powerful built-in features that enhance the capabilities of templates. They allow you to perform dynamic operations, manipulate data, and control the presentation of your web pages.

{{}} Double curly braces

In Django templates, {{ ... }} (double curly braces) is the syntax used for template variables. It is the primary way to insert dynamic content into your HTML templates. The content inside the double curly braces is evaluated as a Python expression, and its result is displayed on the rendered web page.

{%%} curly braces with percent signs

In Django templates, {% ... %} (curly braces with percent signs) is the syntax used for template tags. Template tags are used to perform logic, control flow, and include other templates within the Django template. They allow you to add programming logic directly in the template, which helps in creating dynamic and interactive web pages.

Here are the tags that are mainly used inside curly braces with percent signs:

Control Flow:

Template tags can be used to control the flow of the template rendering process. They allow you to execute different parts of the template based on conditions.

Example :

Views:

```python
def index(request):
    return render(request, "index.html", context={"name":"Harry"})
```

Template:

```
{% if name %}
    <p>Welcome, {{ name }}!</p>
{% else %}
    <p>Welcome, Anonymous User.</p>
```

```
{% endif %}
```

Loops:

Template tags can also be used to loop over lists and dictionaries, enabling you to display repetitive content.

Example:

Views:

```
def index(request):

    students = [

        "ram",

        "hary",

        "shyam",

        "krishna"

    ]

    return render(request, "index.html", context={"students":students})
```

Templates:

```

<ul>

{% for student in students %}

    <li>{{ student }}</li>

{% endfor %}
```

```
</ul>
```

Including Other Templates:

Template tags allow you to include other templates within a template. This helps in reusing common components across different templates.

Example:

Template1: css.html

```
<link rel="stylesheet"

href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.min.css"></link>

<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"

      rel="stylesheet"

integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"

      crossorigin="anonymous">
```

Template 2: index.html

```
{% load static %}

<!DOCTYPE html>

<html lang="en">

    <head>

        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
                   <meta  name="viewport"  content="width=device-width,
initial-scale=1.0">
        <title>
        </title>
        {% include "css.html" %}
    </head>
    <body>
       <h1> Hey </h1>
    </body>
</html>
```

Template Inheritance:

Template inheritance helps you to create a base skeleton for all the similar pages. We first create a base template that contains all the common elements. For eg: **Navigation bar** and **Footer** may be the same for all the pages in the website. In order to not repeat ourselves we create a base template or use template inheritance.

Template inheritance revolves around two tags {% extends %} and {% block … %}.

- Extends tag
  Most commonly used to inherit base templates. It is written at the top of the page where you want to inherit the base template. Base template generally means html.
  Syntax: **{% extends 'path/to/base.html' %}**
- Block Tag
  Generally block is used to define the certain portion of the template. Most of the time we define an empty block ({% block <block_name>%}{% endblock %}) in base template and override the content of that block in the derived template.
  Syntax: **{% block block_name%}...contents…{% endblock %}**

Example:

Base.html (Base Template)

```html
{% load static %}

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.m
in.css"                                                        rel="stylesheet"
integrity="sha384-4bw+/aepP/YC94hEpVNVgiZdgIC5+VKNBQNGCHeKRQN+PtmoHDEXu
ppvnDJzQIu9" crossorigin="anonymous">

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bun
dle.min.js"
integrity="sha384-HwwvtgBNo3bZJJLYd8oVXjrBZt8cqVSpeBNS5n7C8IVInixGAoxmn
lMuBnhbgrkm" crossorigin="anonymous"></script>

<title>{% block title %}{% endblock%}</title>

</head>

<body>

<main>

{% block content %}{% endblock %}

</main>

</body>

</html>
```

Index.html

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

{% block title %} Home {% endblock title %}

{% block content %}

<h1>Hello Students </h1>

{% endblock content %}
```

The resultant index.html becomes:

```
<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.m
in.css"                                                  rel="stylesheet"
integrity="sha384-4bw+/aepP/YC94hEpVNVgiZdgIC5+VKNBQNGCHeKRQN+PtmoHDEXu
ppvnDJzQIu9" crossorigin="anonymous">

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bun
dle.min.js"
integrity="sha384-HwwvtgBNo3bZJJLYd8oVXjrBZt8cqVSpeBNS5n7C8IVInixGAoxmn
lMuBnhbgrkm" crossorigin="anonymous"></script>

<title> Home </title>

</head>

<body>
```

```
<main>

<h1>Hello Students</h1>

</main>

</body>

</html>
```

Database:

It is basically a collection of structured data. It is generally used to store the data of an application or any other purposes. These data are mostly accessed and managed using Database management systems.

Database Management System

Database management system generally means the application or system that is used to maintain, manage, manipulate and access the data from the databases. This system makes it possible to create, update, delete and read data in databases.

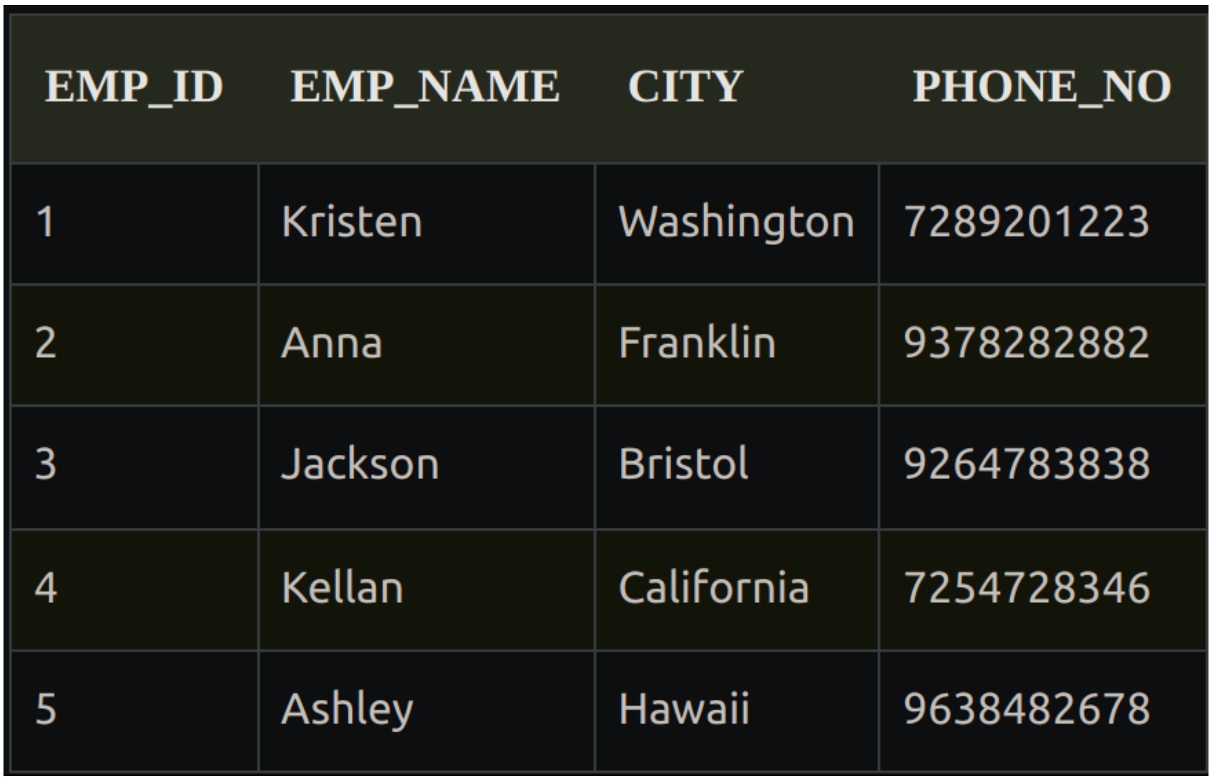
We will be using sqlite3 and postgres for our django projects.

Table:

Table in a database management system is a schema of rows and columns of data. Each table has a unique column that will uniquely represent a row of data, this column's data are called primary keys.

Lets See below example of Employee table:

Here EMP_ID, EMP_NAME, CITY and PHONE_NO are the columns of the table and below these columns are rows of data. Here EMP_ID will always uniquely define the row of data, so EMP_ID is the primary key in this table.

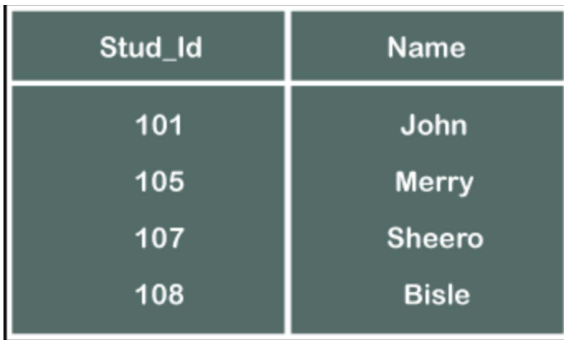


EMP_ID	EMP_NAME	CITY	PHONE_NO
1	Kristen	Washington	7289201223
2	Anna	Franklin	9378282882
3	Jackson	Bristol	9264783838
4	Kellan	California	7254728346
5	Ashley	Hawaii	9638482678

The other main concept of Table in database systems is Foreign key. This key represents the relationship between two or more tables. It utilizes Primary key of the tables to create a link between tables.

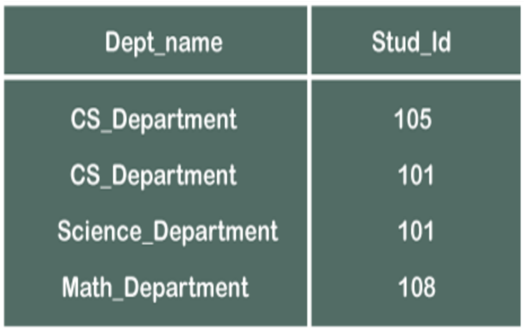
For example:

Let's say we have two tables, one is a student and another is a department. Each department can have multiple students. Here we have a certain kind of relationship between two tables. To define this relationship we will use foreign key concept.



Stud_Id	Name
101	John
105	Merry
107	Sheero
108	Bisle

Student Table



Dept_name	Stud_Id
CS_Department	105
CS_Department	101
Science_Department	101
Math_Department	108

Department Table

Here in the Student table, Stud_id column will act as the primary key. The same Stud_id column in Department table will act as foreign key to create relationship between these two table

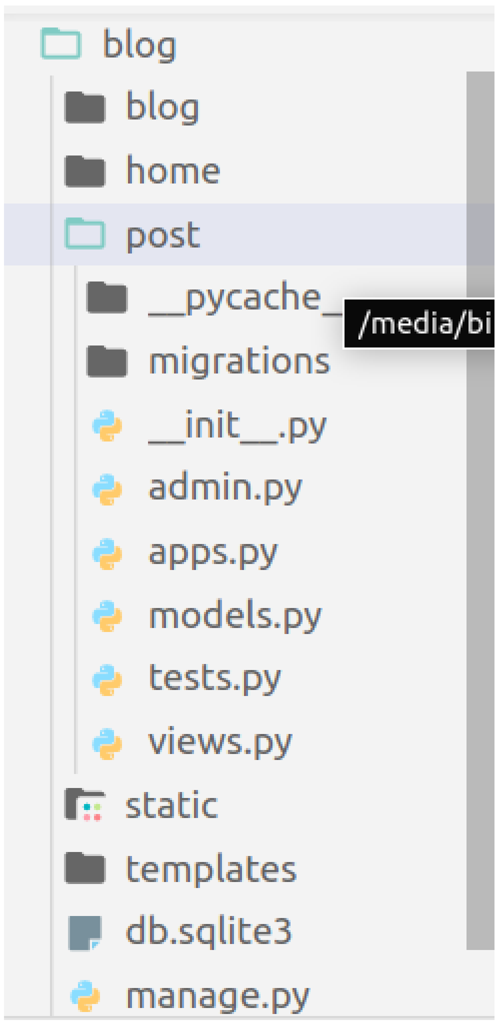
Models:

Models are the classes that interact with databases. It is used to create tables and manipulate the data of the table.

Now lets create one more app named 'post' in our django project

```
(venv) C:\Users\user\Desktop\django\blog>python manage.py startapp post
```

After running this command, you will get following structure in project:



```
blog
  blog
  home
  post
    __pycache__
    migrations
    __init__.py
    admin.py
    apps.py
    models.py
    tests.py
    views.py
  static
  templates
  db.sqlite3
  manage.py
```

Now add “post” in INSTALLED_APP of the settings.py file of the blog folder.

```
INSTALLED_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    "home",

    "post"

]
```

Now go to models.py file of the post folder, and type:

```
from django.db import models

class Post(models.Model):

    author = models.CharField(max_length=200)

    title = models.CharField(max_length=200)

    content = models.TextField()

    created_at = models.DateTimeField(auto_now_add=True)

    updated_at = models.DateTimeField(auto_now=True)

    active = models.BooleanField(default=True)


    def __str__(self) -> str:

        return self.title
```

Remember this, after changing anything in the class of the models.py file, you need to run following commands for it to take effect in the database.

```
(venv) C:\Users\user\Desktop\django\blog>python manage.py makemigrations
```

```
(venv) C:\Users\user\Desktop\django\blog>python manage.py migrate
```

Makemigrations command will create a migration file for each change in the models.py file classes.

Queries

Till now we have done static page loading, template configuration, template loading, url routing, making model and migrating that model to database. Now, let's dive into queries. One of the main reasons for using databases in the project is to create dynamic content that changes its data based on the data in the database. In order to use data in a database, we need to understand how to manipulate this data. By manipulation, we mean creating, updating, deleting and reading data from the database tables. In order to do this operation in django, it provides ORM (Object relational mapping). ORM basically means we are manipulating data in a table using an Object oriented Pattern which we have read Previously. Now let's dive in django ORM.

Previously we have created a Post model and migrated it to the database. This Post model will act as a medium for database manipulation as a well. Generally What we do using ORM is CRUD. Talking about CRUD, CRUD means Create, Read, Update and Delete.

Step 1:

Lets dive deep into CRUD with shell/cmd

```
(venv) C:\Users\user\Desktop\django\blog>python manage.py shell
```

Create